Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Reverse-Engineering Malware: Malware Analysis Tools and Techniques (Forens
at http://www.giac.org/registration/grem

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Reverse-Engineering Malware: Malware Analysis Tools and Techniques (Foren
at http://www.giac.org/registration/grem

Case Study: 2012 DC3 Digital Forensic Challenge Basic Malware Analysis Exercise

GIAC (GREM) Gold Certification

Author: Kenneth J. Zahn, kenneth.j.zahn@gmail.com
Advisor: Rick Wanner

Accepted: August 24, 2013

Abstract

The 2012 *DC3 Digital Forensic Challenge* included two malware analysis-related exercises, one described as "basic" and one described as "advanced." For each exercise, competing teams were provided with an ostensibly malicious—though ultimately innocuous—sample and asked to conduct an analysis befitting the sample's complexity. The author's challenge team, *Plan 9*, placed 2nd in the *Government-only* category, 3rd in the *US-only* category, and 5th in the overall competition. This paper will discuss an updated version of *Plan 9's* solution to the basic malware analysis exercise using a combination of goal-driven and process-driven approaches. It should be noted that one of *Plan 9's* goals in the competition was to use only freely available or open source tools to guarantee the portability of the exercise solutions. To improve the utility of this paper, the tools that were presented in the original solution have been updated to reflect their latest versions at the time of writing. Further, the solution has been expanded to include additional analysis tools that were not presented in the original exercise submission.

© 2013 The SANS Institute Author retains full rights.

1. Introduction

1.1. DC3 Digital Forensic Challenge Overview

The Department of Defense (DoD) Cyber Crime Center (DC3) provides digital forensic process standardization, analysis, and investigation support to the various agencies and military commands within the US DoD (DC3, 2013). As part of DC3's portfolio of forensics outreach initiatives, DC3's Futures Exploration Directorate holds an annual, world-wide competition called the *DC3 Digital Forensic Challenge*. Held since 2006, the 11-month-long challenge consists of approximately 30 digital forensics exercises of varying difficulties and point values. The most difficult of the exercises represent capability gaps in the digital forensic community, making their solutions of noted importance (DC3, 2012a).

In the 2012 competition, 1,209 teams throughout the world registered for the competition in one of eight categories: *Civilian*, *Commercial*, *Government*, *Military*, *High School*, *Community College*, *Undergraduate*, and *Graduate* (DC3, 2012a). Team *Plan 9*, the author's team, placed 2nd in the *Government-only* category, 3rd in the *US-only* category, and 5th overall by completing 28 of the 35 exercises (DC3, 2012b). Two of the exercises given in the 2012 competition—*305: Basic Level Malware Analysis* and *404: Advanced Malware Analysis*—involved the analysis of malicious[1] software samples. Exercise 305 provided the competitors with a malicious dynamic-link library (DLL) file, and Exercise 404 provided the competitors with a malicious Windows executable (EXE) file.

The solution to the *Basic Level Malware Analysis* exercise is presented using contemporary malware analysis techniques. The malware analysis process employed in this exercise represents a hybridization of process-driven and goal-driven approaches.

[1] It should be noted that the payloads of the provided samples were not malicious, but the techniques used by the samples are similar to those used by genuine malware.

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

1.2. Malware Analysis Techniques

Currently, there are five general techniques used in malware analysis: basic static or *surface* analysis, basic dynamic or *behavioral* analysis, static code analysis, dynamic code analysis, and volatile memory analysis.

- **Surface analysis** examines the structural properties and file attributes of a malware sample (e.g. true file type (useful if the file extension was changed), size, file hash values, file and section headers, strings, contained objects, packing mechanisms) without viewing assembly or machine-level instructions (Sikorski & Honig, 2012). Surface analysis can provide information artifacts—such as IP addresses, Internet domain names, and command parameters—that prove useful in subsequent analysis steps.
- **Behavioral analysis** observes the actions taken by a malware sample while it is running. Certain key actions taken by the malware sample, such as adding/modifying/deleting Windows Registry keys, dropping files on the file system, and establishing communications with a command-and-control server, may serve as indicators of compromise (IOC) for the particular sample (Mandiant, 2011). The IOC's observed by the analyst during this phase may then be used to produce signatures for intrusion detection and prevention systems. Because behavioral analysis requires executing the malware on a live machine, it is critical to implement appropriate risk mitigations (e.g. using a stand-alone, virtualized test environment or a sandbox) to avoid infecting production systems (Sikorski & Honig, 2012).
- **Static code analysis** examines the malware sample's executable instructions and internal data structures by loading the sample into a disassembler. Barring code that has been packed, encrypted, or otherwise obfuscated, all instructions present in the sample can be viewed. Although a time-consuming technique, static code analysis can give investigators full insight into the capabilities of the sample under examination (Sikorski & Honig, 2012).
- **Dynamic code analysis** allows the analyst to execute a malware sample instruction-by-instruction by loading it into a debugging application. Because malware samples may have obfuscated portions, it is sometimes necessary to

© 2013 The SANS Institute Author retains full rights.

execute the malware sample up to the completion of the de-obfuscation routine. Once execution is halted at that point in time, the sample in memory may be examined for de-obfuscated data structures or may be dumped to disk for additional static code analysis (Sikorski & Honig, 2012). Dynamic code analysis also reveals data values that are assigned at run time and not available at compile time.

- **Volatile Memory Analysis** involves the examination of volatile memory at a single point in time. Such analysis is accomplished first by dumping the volatile memory to a file and then by inspecting the contents offline using a specialized tool such as the Volatility Framework (Case, 2012).

1.3. Approaches to Malware Analysis

Based on the complexity of the malware sample and on the priority of the case, one or all of the techniques listed in Section 1.2 may be used in the malware analysis process. Some difficulty can arise when selecting the appropriate techniques for an investigation, as it is often necessary for the analyst to strike a balance between available resources and thoroughness of the solution. One of two generally accepted approaches may be taken when analyzing malware: the process-driven approach and the goal-driven approach. Either approach guides the analyst in selecting the appropriate techniques to use during an investigation.

1.3.1. Process-driven Approach to Malware Analysis

The process-driven approach strives to maintain the integrity of the process or procedure being executed. Following this approach results in the formation of well-documented, repeatable, and standardized processes. Because of these factors, this approach holds particular merit within accredited forensic laboratories where maintaining a standard of acceptability to ensure public trust is paramount (Barbara, 2006).

When applied to malware analysis, the process-driven approach ensures that all steps of the analysis process are executed in a repeatable, standardized manner. The one-size-fits-all solution may not always be appropriate, however, as malware samples can differ substantially from one another (e.g. function, language, obfuscation use). A

© 2013 The SANS Institute Author retains full rights.

© 2013 SANS Institute, Author retains full rights.

modified version of the general malware analysis process as defined by Lenny Zeltser is expressed in Figure 1 (Zeltser, 2013b).

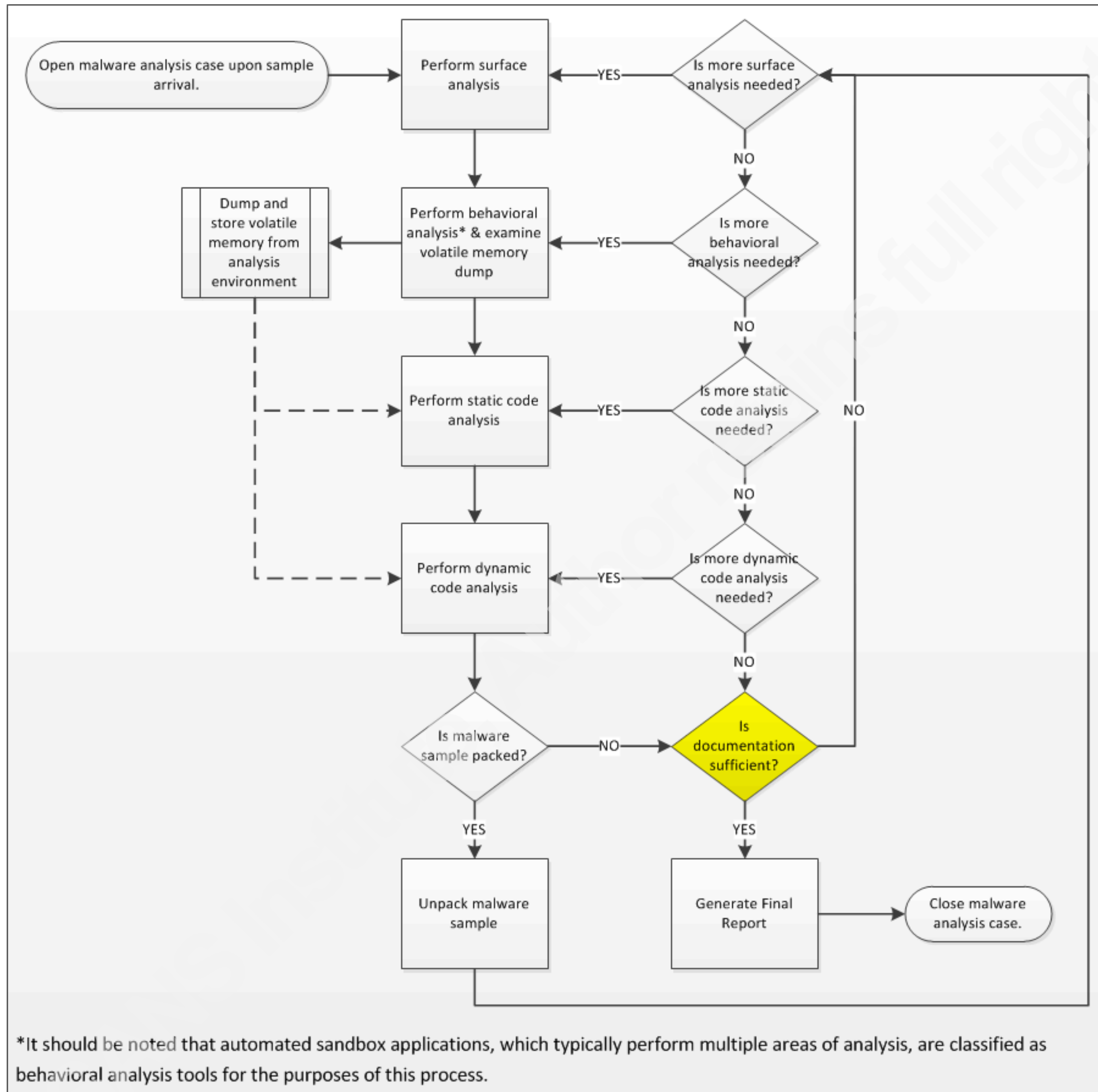


Figure 1: Process-driven approach to malware analysis

The critical decision point in the process-driven approach is highlighted in yellow. During an investigation, a malware analyst must pose the question, “Is the documentation sufficient to support the goals of the case?” Without having requirements for data in mind, the analyst would be hard-pressed to answer this question effectively.

© 2013 The SANS Institute Author retains full rights.

1.3.2. Goal-driven Approach to Malware Analysis

Unlike the process-driven approach, the goal-driven approach is results-oriented. That is, it places emphasis on the final outputs of a process (e.g. the aggregation of analysis data in the final report) and allows a process to be tailored to a particular case (Blackwell, 2013). Figure 2 effectively diagrams a general goal-driven approach to malware analysis using the mind map available on REMNux 4 (Zeltser, 2013a).

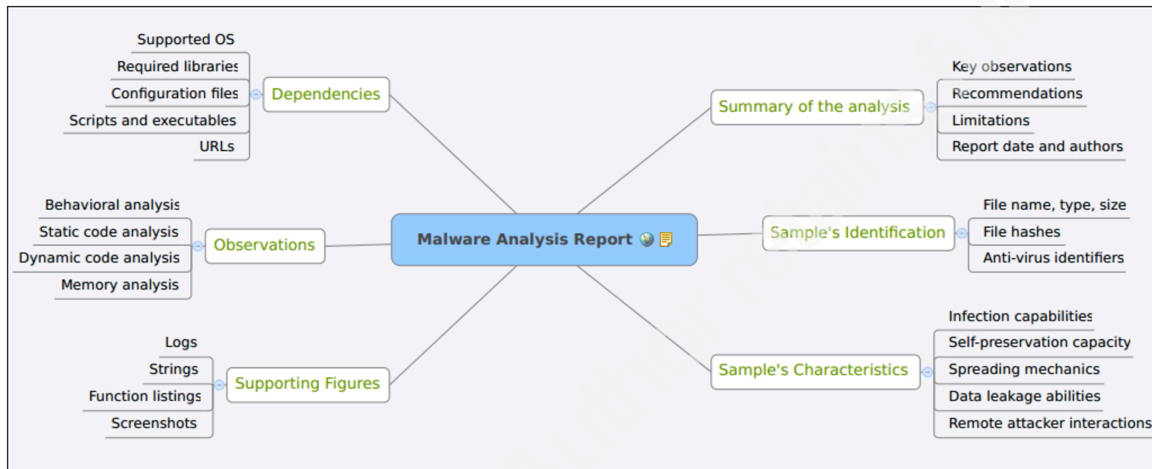


Figure 2: Goal-driven approach to malware analysis

The final output as described by the mind map is the malware analysis report. The six connected blocks represent the sections of the final report and the attributes of each block represent the data that populate those sections. The malware analyst is free to define the sections and supporting data points to create an analysis report that meets the requirements of the case. When the malware analyst extends the mind map by adding the steps taken to gather the information, the resultant diagram is called a *goal tree* (Blackwell, 2013).

As long as the data requirements for the final report are gathered prior to malware analysis, it is possible to create a traceable, repeatable process that is forensically sound (Blackwell, 2013). However, following this approach exactly can yield overly complex goal trees that may not be usable in all circumstances (e.g. within an accredited laboratory whose standard operating procedures dictate the usage of a single, standardized methodology).

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

1.3.3. Hybrid Approach Solution

Plan 9's approach to the basic malware analysis exercise was essentially a hybrid of the two aforementioned approaches. First, the goals were defined according to the data requirements of the exercise's problem statement. Second, the individual malware analysis techniques were applied in the context of the general malware analysis process to achieve the goals. The result was a series of simplified goal trees for each of the steps in the general malware analysis process. The overarching plan of execution is shown in Figure 3.

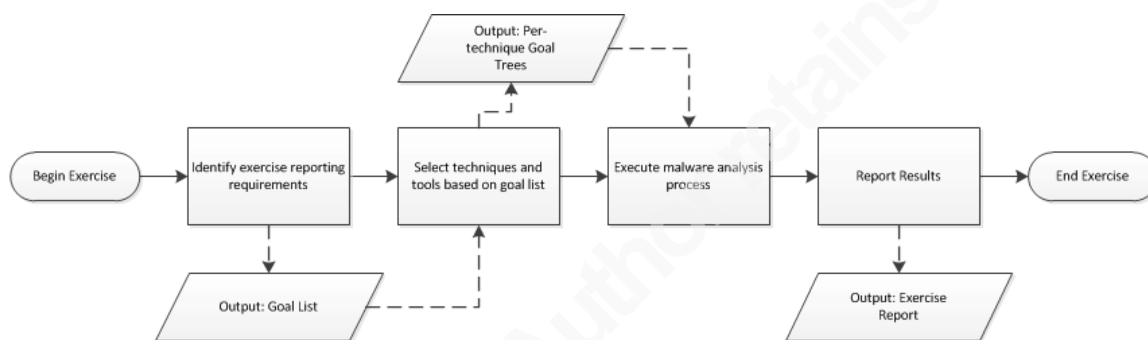


Figure 3: Exercise 305 execution plan

2. Malware Analysis Environment Setup

A virtual lab environment consisting of two logically separated virtual networks—one host-only network and one internal network—was used in the solution to the exercise. The host-only network environment consisted of a single virtual machine (VM) running Microsoft Windows XP with Service Pack 3. The host-only network environment was used solely for behavioral analysis using Cuckoo Sandbox. The internal network environment consisted of two VM's: one running Microsoft Windows with Service Pack 3 and one running REMNux 4. The internal network environment was used for all other analyses.

CentOS 6.4 was selected as the host operating system, for it is fully package-compatible with Red Hat Enterprise Linux 6. Oracle VirtualBox 4.2 was selected for the virtualization environment, as the package has many of the advanced capabilities of the commercial virtualization environments (such as the ability to take and manage multiple snapshots). Cuckoo Sandbox v.60 was extracted to the **Desktop** directory of the host

© 2013 The SANS Institute Author retains full rights.

© 2013 SANS Institute, Author retains full rights.

operating system's default user. Because Cuckoo Sandbox requires Python 2.7 or greater, it was necessary to perform a side-by-side install[2] with Python 2.6, the default package in CentOS 6.4.

The diagram for the virtual laboratory topology is shown in Figure 4.

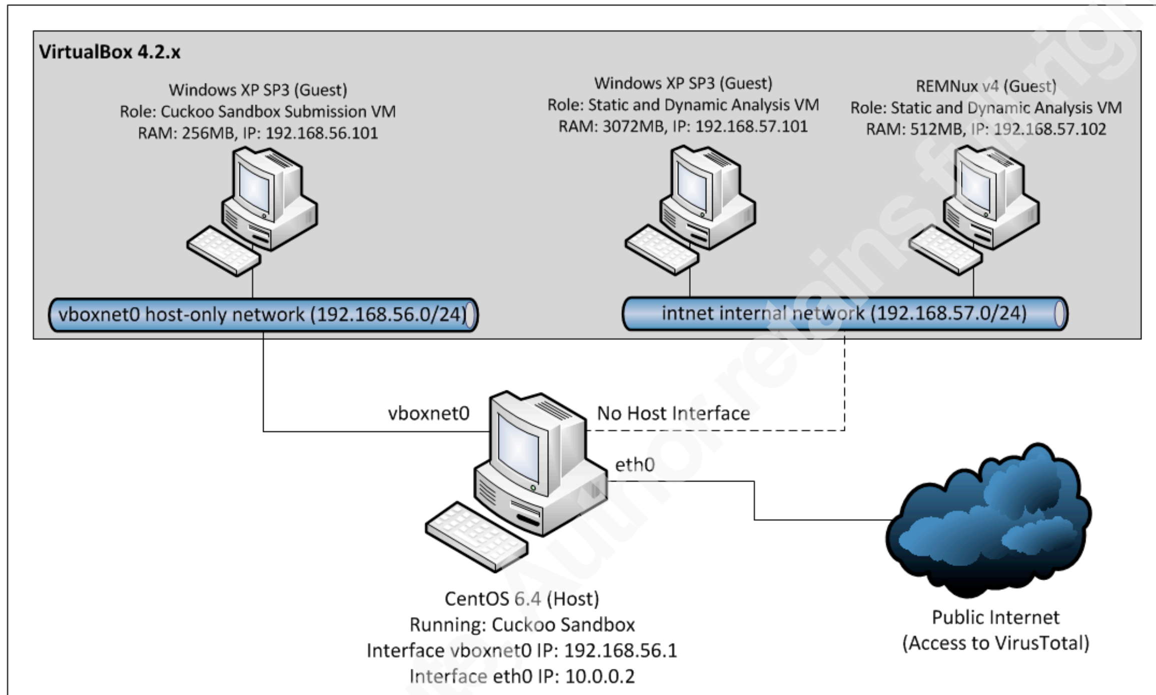


Figure 4: Virtual laboratory topology diagram

3. Basic Malware Analysis Exercise

3.1. Exercise Problem Statement Summary

The *2012 DC3 Digital Forensic Challenge* competitors were given a malware sample named **SvcHost.dll** and were asked to "develop and document a methodology to reverse engineer" it. The points awarded were based on the degree of success of the documented methodology. Specifically, the following observations were to be recorded:

- Programs or services installed, stopped, started, or modified
- Purpose of programs and services
- Location and method of installation
- Files created, deleted, or modified

[2] A guide to performing a side-by-side installation of Python 2.7/3.3 on CentOS 6.x is available here: http://toomuchdata.com/2012/06/25/how-to-install-python-2-7-3-on-centos-6-2/

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

- Registry settings created, deleted, or modified
- Network activity
- Obfuscation methods

3.2. Surface Analysis of SVccHost.dll

The surface analysis process used for this sample is shown in Figure 5.

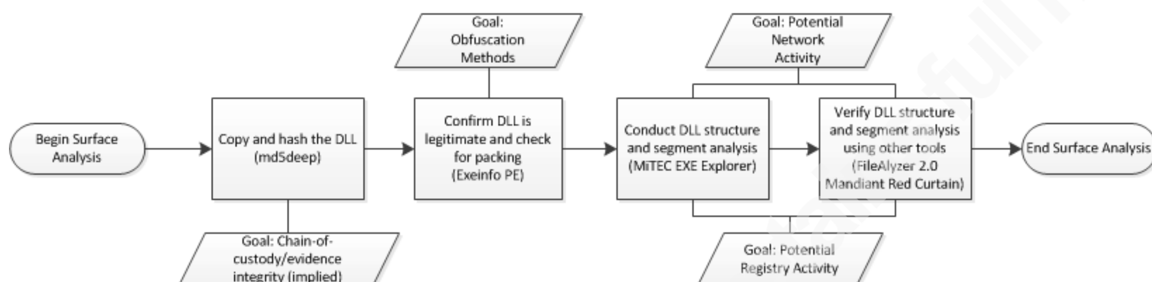


Figure 5: Surface analysis for DLL

3.2.1. Copy and Hash the DLL

The surface analysis process begins by copying the original **SVccHost.dll** file and then by hashing the copy using the md5deep tool. Working with a copy of the malware sample preserves the integrity of the original file in case the sample has self-alteration or self-deletion capability. The output of the md5deep utility is:

```
c6d984a3bba8360533c149fe4ee21e9d  C:\Exercise 305\Copy of SVccHost.dll
```

3.2.2. Confirm DLL is Legitimate and Check for Packing

In this step, the sample is analyzed with the Exeinfo PE[3] tool. While Exeinfo PE lists many attributes of a PE32 file, including the file and section header information, Exeinfo PE's most useful feature is the capability to recognize over 680 signatures of executable packers and compilers (as of December 2012) (A.S.L., 2013). An executable packer is an application used to compress—or *pack*—executable files to reduce their size on disk. A consequence of the packing process is the obfuscation of the original executable code, which may inhibit static code analysis. Thus, identifying the packer used to compress an executable is a critical first step in reversing the packing process.

Executing Exeinfo PE on **SVccHost.dll** reveals two useful pieces of information:

[3] In the original exercise submission, PEiD v.95 was used in this analysis step. PEiD is no longer actively supported, however. Exeinfo PE was selected as a replacement tool, for it offers form and function similar to PEiD and it remains in active development.

© 2013 The SANS Institute Author retains full rights.

1. The file is a legitimate 32-bit DLL (Win32 GUI subsystem) compiled using Borland Delphi 2006/2007; and
2. The DLL is *not* packed.

Figure 6: Exeinfo PE of SVccHost.dll

Selecting the '->' button next to the **EP Section** value generates the **Header Info** window. From this window, the examiner observes the directory entries for an Export Table, an Import Table, and *four* Resources.

Directory Info :	RVA	SIZE		
Export :	00018000	00000047	>>	(06) .edata
Import :	00017000	00000B8C	>>	(05) .idata
Resource :	0001B000	0007F400	0 % of exe	04

Figure 7: Exeinfo PE header information of SvccHost.dll

Because **SVccHost.dll** is a Win32 GUI DLL, the presence of an Import Table and an Export Table is expected. The presence of PE32 resources *may* or *may not* be indicative of potential maliciousness, as the resources might be nothing more than the file icon's graphic image or string tables. In this case, the size of the Resource section's size is given as 0x7F400— or 521,216— bytes, which is suspiciously large for a DLL.

Armed with this information, the analyst may proceed with the examination using PE32-specific tools.

3.2.3. Conduct DLL Structure and Section Analysis

Step 3 in the surface analysis process involves the examination of **SVccHost.dll** with MiTEC EXE Explorer. MiTEC EXE Explorer has the ability to parse the various

© 2013 The SANS Institute Author retains full rights.

headers and sections of a PE32 file and to perform a dump of embedded ASCII and UNICODE strings. MiTEC EXE Explorer may also be used to extract file objects that are embedded within a PE32 file's resources section (**.rsrc**).

Figure 8 displays the information found under MiTEC EXE Explorer's **Header** tab, which confirms that the DLL has a valid PE32 signature.

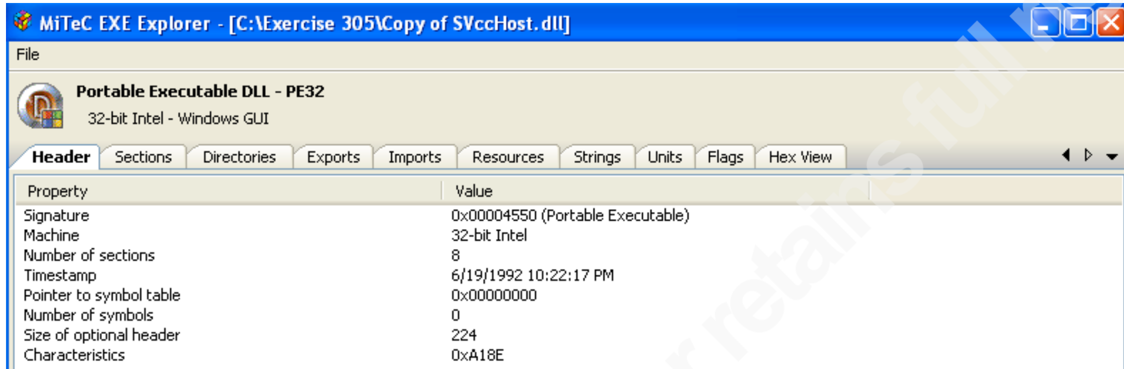


Figure 8: MiTEC EXE Explorer header information: SVccHost.dll

The purpose of a DLL is to house commonly used functions and other resources in a shared file to promote ease of updating and to reduce the size of compiled binaries (Microsoft, 2011). The functions shared by DLL's are called *exported* functions, as they are intended to be *imported* and called by compiled executables. While outside the scope of normal use, exported functions may also be called externally by using the Microsoft Windows **rundll32.exe** utility. Figure 9 displays the single function export named **Install** that was listed under the **Exports** tab.

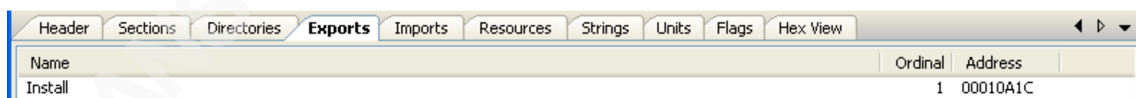


Figure 9: MiTEC EXE Explorer exports information: SVccHost.dll

Eighty-seven functions imported from five separate libraries are observed under the **Imports** tab. Table 1 names and describes each of the imported libraries.

Table 1: DLL's imported by SvccHost.dll

Imported DLL	Description
advapi32.dll	Provides access to advanced operating system functions such as the Service Manager and the Registry (Sikorski & Honig, 2012).
kernel32.dll	Provides access to core operating system functions such as memory management, I/O operations, and hardware interrupts (kernel32.dll, 2010).
oleaut32.dll	Provides access to object linking and embedding (OLE) functions

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

	(oleaut32.dll, 2010).
SHFolder.dll	Provides access to special Windows folders (SHFolder.dll, 2010)
user32.dll	Provides access to user interface components and message handling (Sikorski & Honig, 2012).

Table 2 names and describes a few of the more suspicious function imports.

Table 2: Suspicious imports found in SvccHost.dll

Imported Function(s)	Description
RegOpenKeyExW/RegCloseKey	Opens/closes the specified Registry key.
RegQueryValueExA	Queries the specified Registry key's value.
WriteFile	Writes data to disk.
TlsAlloc/TlsGetValue/TlsSetValue	Allocates/reads/writes data to the thread local storage area of a PE32 file.
WinExec	Passes commands to the Windows command shell.

Based on the function imports, the **SVccHost.dll** has the ability to access the Windows Registry, write files to disk, and execute commands through **cmd.exe**. Further, the sample has loaded functions that manipulate the thread local storage (TLS) area of the DLL file. Because TLS can be used to initialize data structures in threads of execution, the TLS area is accessed by the loader *prior* to calling the program's entry-point function. Thus, if callback functions exist within the TLS area, they will be executed before the program's **main()** or **DllMain()** functions. This trait makes the use of TLS callback functions ideal as an anti-debugging technique, as most debuggers break execution at the program's entry point by default (Sikorski & Honig, 2012). According to Exeinfo PE, however, there is no TLS table in the **SVccHost.dll** file.

The **Resources** tab lists the structures that are found in the **.rsrc** section of the PE32 file. According to the PE32 specification, the **.rsrc** section is a directory table whose entries point to data objects (e.g. files) embedded within the file (Microsoft, 2013c). Figure 10 depicts an object 512 kilobytes in size and bearing an MS-DOS file signature (PE32 files also possess the MS-DOS file signature) that is observed in the **RCDATA** segment of the **SVccHost.dll.** Because the **.rsrc** section is typically reserved for structures such as string tables and file icons, the presence of this executable file is highly suspicious.

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

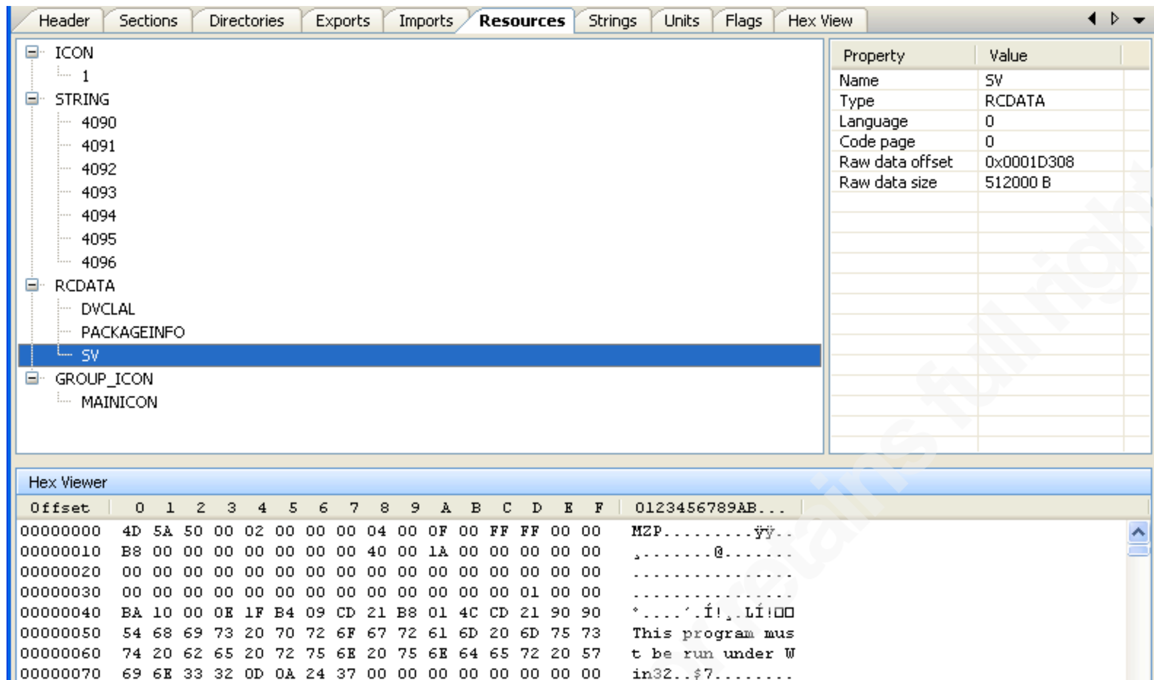


Figure 10: Embedded executable file found in SVccHost.dll

By right-clicking on the **SV** object and selecting **Save Resource…**, the **SV** object may be extracted and saved for later examination. At this point in time, only the hash value of the **SV** object need be recorded, as this value will be compared to the hash value of any files that are dropped by **SVCCHost.dll** during the behavioral analysis phase**.** Executing **md5deep** on the **SV** object yields the following hash information:

```
2b618d0aedfd9313a37d14b05a2b688a  C:\Exercise 305\SV
```

The **Strings** tab lists all of the strings (both ASCII and UNICODE are supported by toggling the tab at the bottom of the main application pane) found in the PE32 file. A few strings resembling low-level function names are present and displayed in Figure 11:

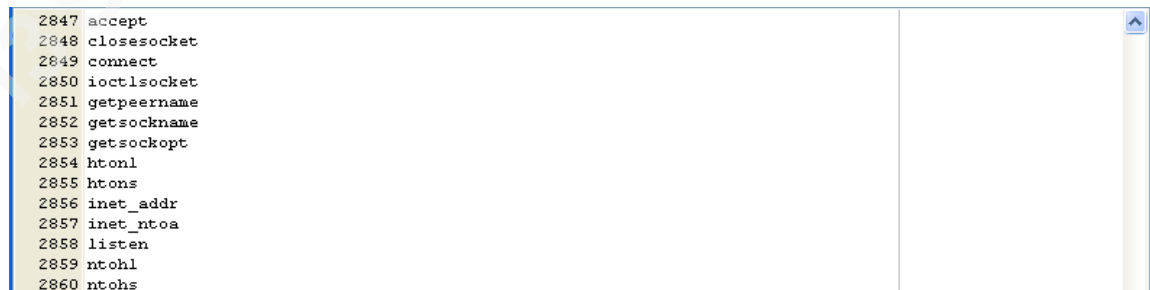


Figure 11: Possible low-level network functions

Further, the **Strings** tab reveals strings that appear to be shell commands intended to be launched through **cmd.exe** via the WinExec system call.

© 2013 SANS Institute, Author retains full rights.

© 2013 The SANS Institute Author retains full rights.

```
3121 /install
3122 cmd.exe /c net start Challenge /silent
3127 /uninstall
3128 cmd.exe /c net stop Challenge
```

Figure 12: Possible calls through WinExec

These findings support the notion that an embedded file—perhaps the aforementioned **SV** file object—exists within **SVccHost.dll.**

3.2.4. Verify Structure and Segment Analysis Using Other Tools

Because software analysis tools are not perfect (including those that are validated by accredited laboratories), it is recommended to use additional tools of similar function to verify the results of any given tool.

The first tool selected for verification purposes was FileAlyzer 2.0, a generic file scanner which can successfully parse the internal structures of PE32 files. FileAlyzer agreed with all but one of the findings of MiTEC's EXE Explorer. In particular, FileAlyzer 2.0 counted a total of 101 imported functions from 5 DLL's.

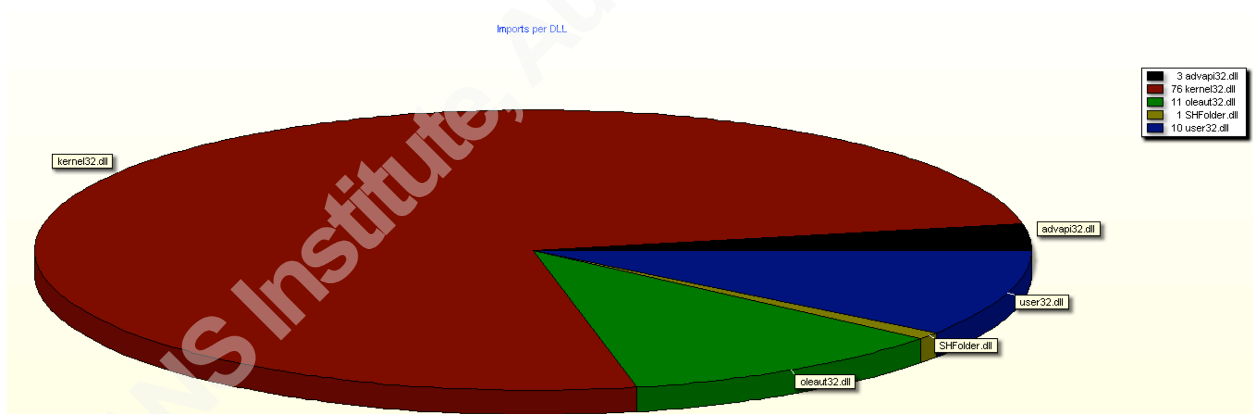


Figure 13: Function call count and distribution

In addition, FileAlyzer highlighted that the checksum of the PE32 file was zeroized, creating a checksum mismatch:

Checksum 00000000 does NOT match file checksum 000A0940

Because the checksum field is created and populated at compile time, the presence of a zeroized, mismatched checksum is indicative of deliberate tampering and may be used as an anti-analysis technique. A study conducted by Yibin Liao of the University of

© 2013 The SANS Institute Author retains full rights.

Georgia reported that of the 5598 malicious PE32 files examined, 90% had zeroized checksums (2012).

The second tool selected for verification purposes was Mandiant Red Curtain, a Windows PE32 file scanner that provides entry point, entropy, digital signature, and segment information. The main application pane confirms the Borland Delphi signature (though Red Curtain disagrees with the version) and the **Details** pane confirms the function import count as 101.

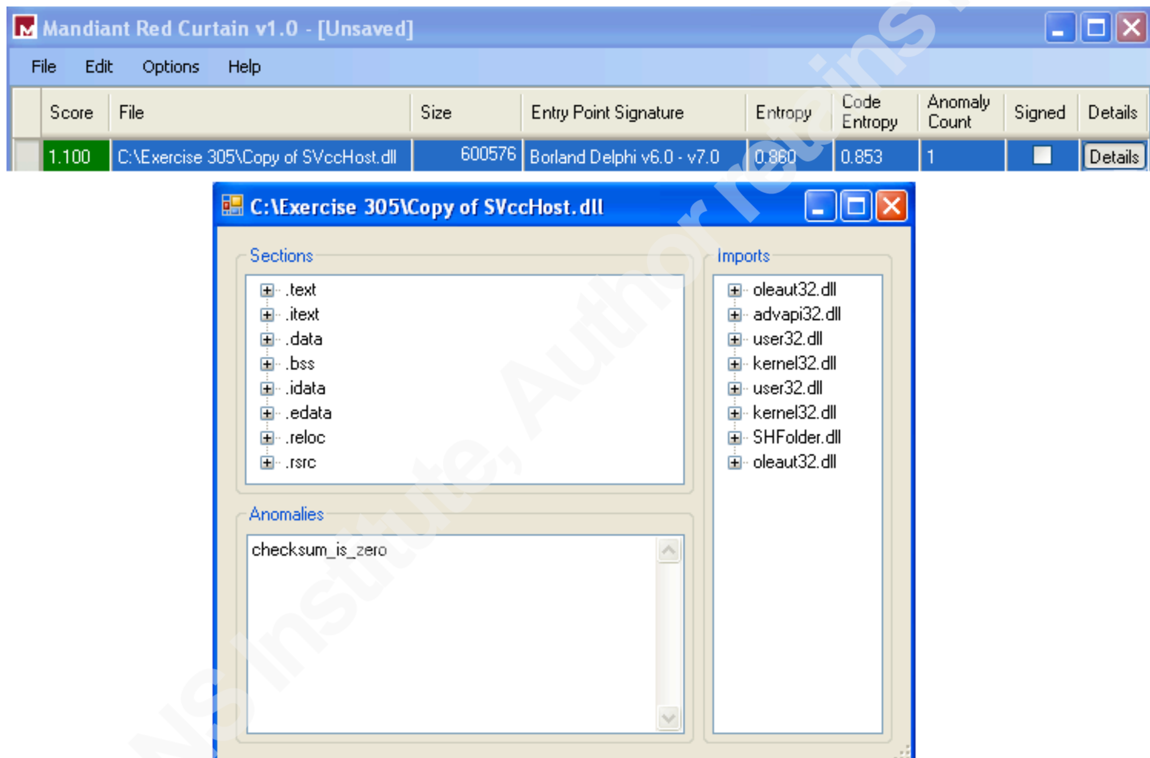


Figure 14: Mandiant Red Curtain output

In addition to the single file anomaly of a zero checksum, Red Curtain indicates that the malware sample is not digitally signed. This attribute can be used by Process Hacker to reduce system noise by filtering out the processes that are digitally signed.

3.3. Behavioral Analysis of SVccHost.dll

The behavioral analysis process used for this sample is given in Figure 15.

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute
Author retains full rights.

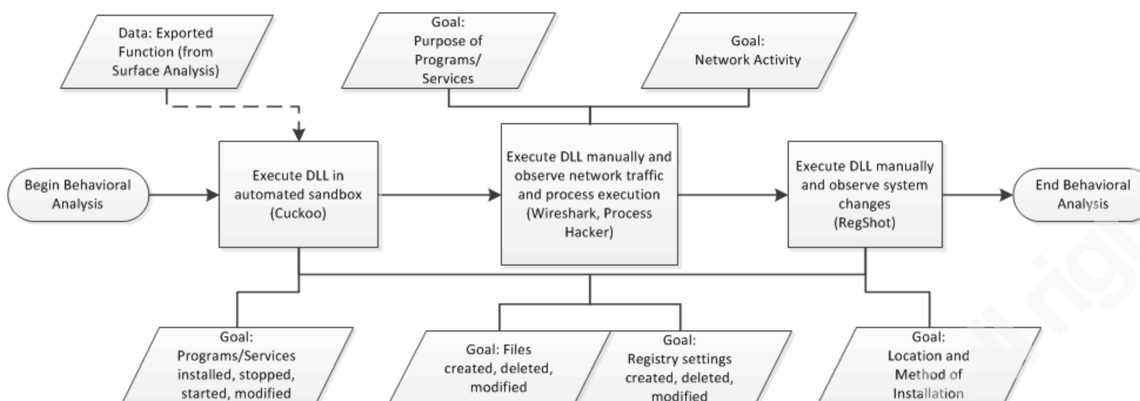


Figure 15: Behavioral analysis for DLL

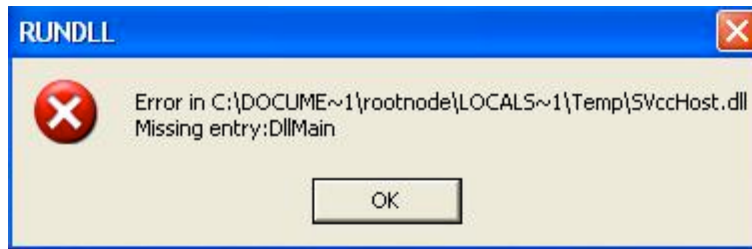
3.3.1. Execute DLL in Automated Sandbox

In the context of malware analysis, an automated sandbox is a tool that manages a separate—and usually virtual—environment within which malware samples are executed and monitored. Due to automation, the incorporation of such a tool can result in faster execution of the malware analysis process. Cuckoo Sandbox[4], an open source project, maintains both an online malware analysis service hosted at www.malwr.com, and a standalone, python-based, automated sandbox application. Once configured, Cuckoo Sandbox requires only two python scripts to be executed in order to analyze a malware sample: **cuckoo.py**, which manages the virtual environment, and **submit.py**, which submits the malware sample to the analysis VM.

If Cuckoo Sandbox's **submit.py** script is invoked with only the DLL's file name passed as a parameter, Cuckoo will default to loading the DLL into memory without explicitly calling an exported function. This would result in the invocation of the DLL's **DllMain()** function, if present. **DllMain()** is an optional entry-point function that the Windows loader automatically calls when it loads a DLL into random access memory (Microsoft, 2013a). Invoking **submit.py** with only the file name **SvccHost.dll** passed as a parameter results in the premature termination of the process and the display of the following error message from within the Cuckoo Sandbox VM:

[4] The use of Cuckoo Sandbox was not included in the original exercise submission. Because leveraging commercial automated sandboxes is common in today's malware analysis environment, Cuckoo Sandbox was included to offer an open source alternative.

© 2013 The SANS Institute Author retains full rights.



To have Cuckoo Sandbox execute the **Install()** function of **SVccHost.dll** observed during surface analysis, executing the following command from within the **cuckoo/utils** directory is necessary:

```
[root@CentOS]#python2.7 submit.py --package dll --options function=Install ../samples/SvccHost.dll
```

After an examination period of 22 seconds, Cuckoo Sandbox exited gracefully and generated a report. A quick examination of the **analysis.log** file confirmed that the analysis was terminated because the "process list was empty," indicating that all processes being traced by Cuckoo had exited.

Under the **Dropped Files** section of the report, a single file, **ChallengeSvc.exe**, is listed. Based on the MD5 hash listed, this file is identical to the **SV** object that was extracted from the **.rcsc** section of the **SVccHost.dll** file.

File name	ChallengeSvc.exe
File size	512000 bytes
File type	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
MD5	2b618d0aedfd9313a37d14b05a2b688a

Figure 16: Dropped file detected by Cuckoo Sandbox

In the **Behavior Summary** section, the files and Registry keys touched by **SVccHost.dll** are listed. The **Files** summary indicates that the **ChallengeSvc.exe** file was dropped into the **C:\Windows\system32** directory, which is the location of Windows's system files.

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

Files

- C:\DOCUME~1\Bob\LOCALS~1\Temp\SVccHost.dll
- C:\DOCUME~1\Bob\LOCALS~1\Temp\SVccHost.dll.123.Manifest
- C:\WINDOWS\system32\ChallengeSvc.exe
- C:\DOCUME~1
- C:\DOCUME~1\Bob
- C:\DOCUME~1\Bob\LOCALS~1
- C:\DOCUME~1\Bob\LOCALS~1\Temp
- C:\
- C:\DOCUME~1\Bob\LOCALS~1\Temp\net.*
- C:\DOCUME~1\Bob\LOCALS~1\Temp\net
- C:\WINDOWS\system32\net.*
- C:\WINDOWS\system32\net.COM
- C:\WINDOWS\system32\net.EXE

Figure 17: Cuckoo Sandbox files summary

The **Registry Keys** summary indicates that a service named **Challenge** was registered with the Windows Services list located within the **HKLM\SYSTEM\CurrentControlSet\Services** key.

Registry Keys

- HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\System
- HKEY_LOCAL_MACHINE\Software\Microsoft\Command Processor
- HKEY_CURRENT_USER\Software\Microsoft\Command Processor
- HKEY_CURRENT_USER\Software\Borland\Locales
- HKEY_LOCAL_MACHINE\Software\Borland\Locales
- HKEY_CURRENT_USER\Software\Borland\Delphi\Locales
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontSubstitutes
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\DnsCache\Parameters
- HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows NT\DnsClient
- HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\System\DNSClient
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Challenge
- HKEY_LOCAL_MACHINE\System\WPA\Starter

Figure 18: Cuckoo Sandbox registry keys summary

Finally, the **Processes** section lists all of the system calls made by the processes being tracked by Cuckoo during the examination. Based on a high-level trace of processes that were spawned, the flow chart given in Figure 19 was constructed.

© 2013 SANS Institute, Author retains full rights.

© 2013 The SANS Institute Author retains full rights.

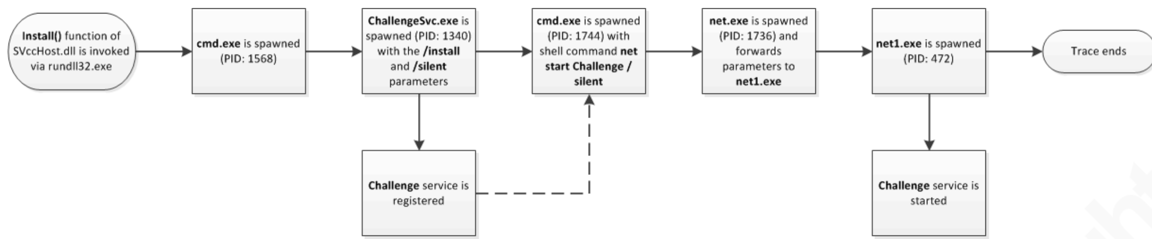


Figure 19: Process spawning flow chart

Under the **rundll32.exe** process (PID:1628[5]), two system calls are listed which create a handle to a file named "C:\Windows\system32\ChallengeSvc.exe" and then write data to that file handle. This is the series of instructions that drops the malware payload into the **C:\Windows\system32 directory**.

NtCreateFile	FileHandle => 0x000000bc DesiredAccess => 0xc0100080 FileName => C:\WINDOWS\system32\ChallengeSvc.exe CreateDisposition => 5 ShareAccess => 0	SUCCESS	0x00000000	
NtWriteFile	FileHandle => 0x000000bc Buffer =>	SUCCESS	0x00000000	8 times

Figure 20: System calls to drop ChallengeSvc.exe

Toward the end of the trace of the **rundll32.exe** process is a call to **CreateProcessInternalW()**, which spawns the **cmd.exe** process (PID: 1568) and executes the shell command "C:\WINDOWS\system32\ChallengeSvc.exe /install /silent".

CreateProcessInternalW	ApplicationName => CommandLine => cmd.exe /c C:\WINDOWS\system32\ChallengeSvc.exe /install /silent CreationFlags => 0x00000000 ProcessId => 1568 ThreadId => 1468 ProcessHandle => 0x000000cc ThreadHandle => 0x000000d0	SUCCESS	0x00000001

Figure 21: Call to spawn installation process

Through **cmd.exe**, the process **ChallengeSvc.exe** (PID: 1340) is spawned. The trace of **ChallengeSvc.exe** includes seven calls to the function **DeviceIOControl()**, which allows processes operating in user space to communicate with device drivers.

[5] Process Identifiers (PIDs) are dynamically generated when processes are spawned and thus can differ in subsequent examinations.

© 2013 The SANS Institute Author retains full rights.

Because device drivers reside in kernel space, malware can use this function to pass shell code to the kernel for execution (Sikorski & Honig, 2012).

DeviceIoControl	DeviceHandle => 0x000007d0 IoControlCode => 3735560 InBuffer => xd5>U\xee \xca \x88\xachC\xd1\xfa\xec\xb6\x85\ xb8H\x93\x99i\x8e\xda\x00\x00\x 00\x00\x00\x00\x00\x00\x00\x00\	SUCCESS	0x00000001
	<SNIP>		
	OutBuffer => $"Qk\xd3F\xee_\x1fY\x88\xf3\x1 2A4\xebi;\xa5\x9bn\xb1\x12\xef\ xd5\x95P\x84oh\x19\x06\x9e\x1f\ x01q\xbb\x857\xd2\xe5\x86j\x05+ QM\xcc\x8a\x99\xb2]\x9f\xcbl\x1 7)\xfc\xa6\xb7\x167\\xb5.eX\xc1 Ty\xfd\xc6\xb8\x89\xef\xe3-		

Figure 22: Multiple calls to DeviceIOControl()

ChallengeSvc.exe then opens the Windows Service Manager and registers itself as a Windows service.

OpenSCManagerA	MachineName => DatabaseName => DesiredAccess => 983103	SUCCESS	0x00159fa0
CreateServiceA	ServiceControlHandle => 0x00159fa0 ServiceName => Challenge DisplayName => Challenge DesiredAccess => 983551 ServiceType => 16 StartType => 2 ErrorControl => 1 BinaryPathName => C:\WINDOWS\system32\ChallengeSv c.exe ServiceStartName => Password =>	SUCCESS	0x00159ea0

Figure 23: Calls to install a Windows service named "Challenge"

ChallengeSvc.exe then invokes **cmd.exe,** which in turn invokes **net.exe**, which calls **net1.exe** to start the **Challenge** service and to configure the service to start automatically.

© 2013 SANS Institute Author retains full rights.

OpenSCManagerW	MachineName => DatabaseName => DesiredAccess => 2147483648	SUCCESS	0x000ab380
OpenSCManagerW	MachineName => DatabaseName => DesiredAccess => 1	SUCCESS	0x000aaf70
OpenServiceW	ServiceControlManager => 0x000aaf70 ServiceName => Challenge DesiredAccess => 20	SUCCESS	0x000ab500
StartServiceW	ServiceHandle => 0x000ab500 Arguments => ['/silent']	SUCCESS	0x00000001
	<SNIP>		
OpenSCManagerW	MachineName => DatabaseName => DesiredAccess => 1	SUCCESS	0x000ab640
OpenServiceW	ServiceControlManager => 0x000ab640 ServiceName => Challenge DesiredAccess => 132	SUCCESS	0x000ab500
ControlService	ServiceHandle => 0x000ab500 ControlCode => 4	SUCCESS	0x00000001

Figure 24: Calls to start Windows service "Challenge"

Because the analysis ran for only 22 seconds, Cuckoo Sandbox's packet capture facility was not able to capture any network traffic associated with the malware. In order to capture network traffic, manual execution of the malware and monitoring with Wireshark were necessary.

3.3.2. Execute DLL Manually and Observe Network Traffic and Process Execution

For manual behavioral analysis, the internal network environment was first configured with the default IP addressing scheme (192.168.57.0/24). FakeDNS, farpd, and inetsim were running on the REMNux VM and Process Hacker 2[6] was running on the Windows XP VM when the **'rundll32.exe "C:\Exercise 305\Copy of SvccHost.dll", Install'** command was executed. The Wireshark capture revealed that ICMP echo-request (ping) packets 1058 bytes in length (data payload size of 1016 bytes) were sent to host 157.166.226.26 every 15 seconds for 90 minutes.

[6] In the original exercise submission, Process Explorer (procexp) from Windows SysInternals was used in this analysis step.

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

```
  6 9.013803   192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=41229/3489, ttl=128
  7 24.031995  192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=41485/3490, ttl=128
  8 39.054096  192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=41741/3491, ttl=128
  9 54.075418  192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=41997/3492, ttl=128
 10 69.096949  192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=42253/3493, ttl=128
                                        <SNIP>
489 5341.67954 192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=527/3842, ttl=128
490 5356.70184 192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=783/3843, ttl=128
491 5371.72363 192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=1039/3844, ttl=128
492 5386.74421 192.168.57.101   157.166.226.26   ICMP   1058 Echo (ping) request   id=0x9406, seq=1295/3845, ttl=128
```

Figure 25: Wireshark packet capture *without* ICMP echo-replies

Performing a reverse DNS lookup using the nslookup tool revealed that 157.166.226.26 resolved to host www.cnn.com:

```
[root@centos]#nslookup
> 157.166.226.26
Server:         4.2.2.2
Address:        4.2.2.2#53

Non-authoritative answer:
26.226.166.157.in-addr.arpa     name = www.cnn.com.
```

Because no DNS queries were observed attempting to resolve host www.cnn.com, it must be the case that the sample was pinging a hard-coded IP address. Given the current network configuration, it would not be possible to respond to echo-request packets sent to hard coded IP addresses outside the current network without inserting a layer 3 handler. For simplicity, the network configuration was altered instead.

For the second run, the IP address of the Windows XP VM was set to 157.166.226.1 and the IP address of the REMNux VM was set to 157.166.226.26 (the target of the echo-request packets). This configuration allowed the REMNux VM to reply to the echo request packets sent by the sample. The Wireshark capture revealed that ICMP echo-request packets 1058 bytes in length (data payload size of 1016 bytes) were sent to host 157.166.226.26 every <u>10 seconds</u> for <u>60 minutes</u>.

```
  4 9.012357   157.166.226.1    157.166.226.26   ICMP   1058 Echo (ping) request   id=0xb000, seq=41229/3489, ttl=128
  5 9.012373   157.166.226.26   157.166.226.1    ICMP   1058 Echo (ping) reply     id=0xb000, seq=41229/3489, ttl=64
  8 19.025165  157.166.226.1    157.166.226.26   ICMP   1058 Echo (ping) request   id=0xb000, seq=41485/3490, ttl=128
  9 19.025207  157.166.226.26   157.166.226.1    ICMP   1058 Echo (ping) reply     id=0xb000, seq=41485/3490, ttl=64
 10 29.038628  157.166.226.1    157.166.226.26   ICMP   1058 Echo (ping) request   id=0xb000, seq=41741/3491, ttl=128
 11 29.038667  157.166.226.26   157.166.226.1    ICMP   1058 Echo (ping) reply     id=0xb000, seq=41741/3491, ttl=64
 12 39.054102  157.166.226.1    157.166.226.26   ICMP   1058 Echo (ping) request   id=0xb000, seq=41997/3492, ttl=128
 13 39.054144  157.166.226.26   157.166.226.1    ICMP   1058 Echo (ping) reply     id=0xb000, seq=41997/3492, ttl=64
                                        <SNIP>
974 3584.15200 157.166.226.1    157.166.226.26   ICMP   1058 Echo (ping) request   id=0xb000, seq=1551/3846, ttl=128
975 3584.15212 157.166.226.26   157.166.226.1    ICMP   1058 Echo (ping) reply     id=0xb000, seq=1551/3846, ttl=64
978 3594.16590 157.166.226.1    157.166.226.26   ICMP   1058 Echo (ping) request   id=0xb000, seq=1807/3847, ttl=128
979 3594.16601 157.166.226.26   157.166.226.1    ICMP   1058 Echo (ping) reply     id=0xb000, seq=1807/3847, ttl=64
```

Figure 26: Wireshark packet capture *with* ICMP echo-replies

© 2013 SANS Institute, Author retains full rights.

© 2013 The SANS Institute Author retains full rights.

While the sample was communicating with the REMNux VM, Process Hacker, a process monitoring tool, was running in the Windows XP VM. Process Hacker observed that a service named **Challenge** was created and that the service running as a process named **ChallengeSvc.exe**. The **Handles** tab of the **Properties** window for **ChallengeSvc.exe** process revealed that at the aforementioned 10 and 15 second intervals, the process dynamically created file handles to two device objects: **\Device\Afd** and **\Device\RawIP**.

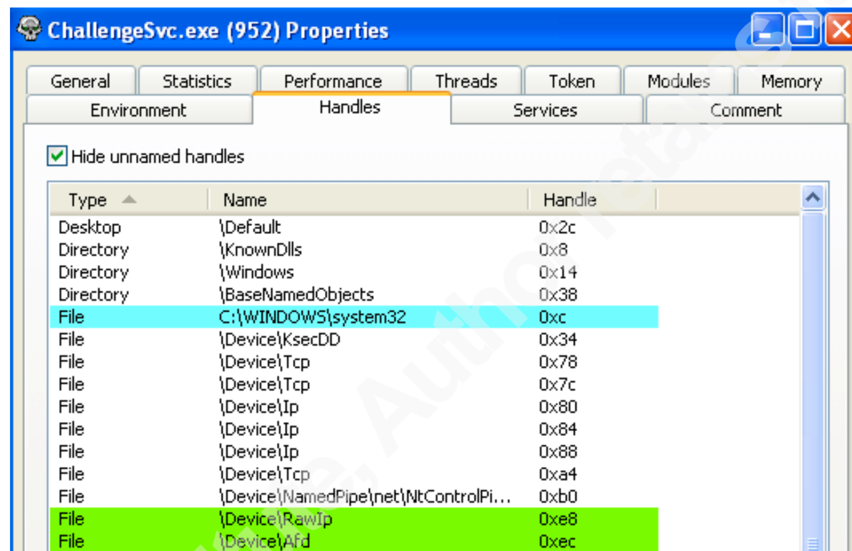


Figure 27: RawIP & Afd device activity observed by Process Hacker

The Ancillary Function Driver (\Device\Afd) serves as the management layer of the Windows Sockets (WinSock) interface and sits just above the RawIP device in the WinSock architecture (Mandt, 2012). This evidence indicates that the process was crafting raw ICMP echo-request packets.

3.3.3. Execute DLL Manually and Observe System Changes

The last step in the behavioral analysis process was observing the file system and Registry changes made by the sample using RegShot. RegShot reveals that the sample performs the following actions:

1. The sample drops a file named **ChallengeSvc.exe** to the **C:\Windows\system32** directory.

© 2013 SANS Institute, Author retains full rights.
© 2013 The SANS Institute Author retains full rights.

```
----------------------------------
Files added: 5
----------------------------------
C:\WINDOWS\Prefetch\CHALLENGESVC.EXE-0A9A13E4.pf
C:\WINDOWS\Prefetch\NET.EXE-01A53C2F.pf
C:\WINDOWS\Prefetch\NET1.EXE-029B9DB4.pf
C:\WINDOWS\Prefetch\RUNDLL32.EXE-33EF81C9.pf
C:\WINDOWS\system32\ChallengeSvc.exe
```

Figure 28: Added files list as detected by RegShot

2. The sample creates a service named **Challenge** which executes under the **LocalSystem** account.

```
HKLM\SYSTEM\ControlSet001\Services\Challenge\Type: 0x00000010
HKLM\SYSTEM\ControlSet001\Services\Challenge\Start: 0x00000002
HKLM\SYSTEM\ControlSet001\Services\Challenge\ErrorControl: 0x00000001
HKLM\SYSTEM\ControlSet001\Services\Challenge\ImagePath: "C:\WINDOWS\system32\ChallengeSvc.exe"
HKLM\SYSTEM\ControlSet001\Services\Challenge\DisplayName: "Challenge"
HKLM\SYSTEM\ControlSet001\Services\Challenge\ObjectName: "LocalSystem"
```

Figure 29: Service creation detected by RegShot

In addition, the Windows Firewall service is stopped upon the installation of the **Challenge** service. The corresponding Registry keys for Windows Firewall are not touched, meaning that the firewall service will start upon reboot if set to **Automatic**. This effectively hides the downing of the Windows Firewall from Registry monitoring tools, but not from the Windows Security Center notifying service:

Figure 30: Windows Security Center reports downing of firewall

3.4. Is the Documentation Sufficient?

The answer to this question is no, as three issues arose during surface and behavioral analysis that warrant additional examination. Upon the resolution of these three issues, however, the documentation may be judged as sufficient.

3.4.1. The Presence of a TLS Table in the Dropped File

Surface analysis was performed on the dropped file **ChallengeSvc.exe** after the behavioral analysis of **SvccHost.dll** was conducted. The only additional observation was the presence of a Thread Local Storage (TLS) callback table:

© 2013 SANS Institute, Author retains full rights.
© 2013 The SANS Institute Author retains full rights.

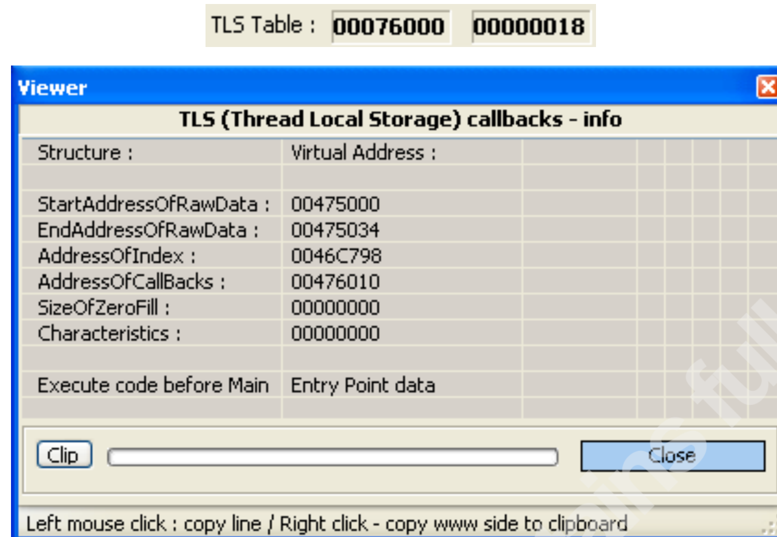


Figure 31: TLS detected in dropped file ChallengeSvc.exe

The TLS area would normally be flagged for examination during dynamic code analysis under the process-driven approach, for it can contain anti-debugging callback functions. Because the documenting of anti-debugging techniques was not a defined goal of this examination, however, the TLS area *was not* examined.

3.4.2. The Calls Made to DeviceIOControl()

During behavioral analysis, Cuckoo Sandbox reported a series of calls made to **DeviceIOControl()** by the **ChallengeSVC.exe** process. Because **DeviceIOControl()** may be used by malware to directly communicate with device drivers residing in kernel space, the calls require additional investigation. To discover the source of the **DeviceIOControl()** calls, API Monitor v2[7], a tool with a fine-grained system call tracing facility, was selected.

API Monitor is a GUI-based tool containing definitions for more than 13,000 application programming interface (API) calls (Batra, 2012). API Monitor can trace the API calls at the thread level, trace the input and output buffers of each call, and can filter results based on API type.

In the **API Filter** window, **Devices**, **Internet**, and **Networking** were selected as the API types to trace. The **Install()** function was then manually invoked from the

[7] API Monitor was not included in the original exercise solution. API Monitor was added to the solution to provide verification of the results produced by Cuckoo Sandbox.

© 2013 The SANS Institute Author retains full rights.

Windows command line using the **rundll32.exe** facility. Thirty seconds after execution, the processes recorded by Cuckoo were observed in the **Monitored Processes** window:

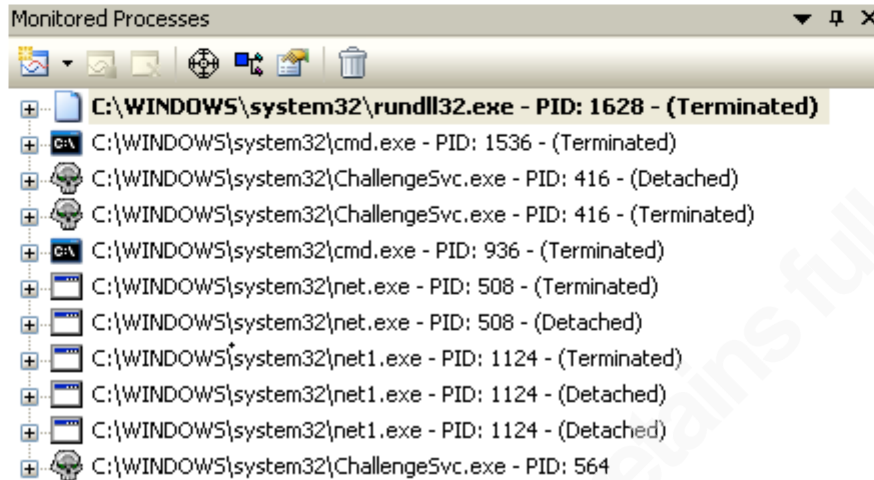


Figure 32: Processes monitored by API Monitor

Examination of the first **ChallengeSvc.exe** (PID: 416) uncovered the suspect calls to **DeviceIOControl()**:

4	ChallengeSvc.exe	gethostname (0x024948c8, 250)	0
5	DNSAPI.dll	NdrClientCall2 (0x76f22ea8, 0x76f22c60, ...)	{ Pointer = NULL, Simple = 0 }
6	DNSAPI.dll	RpcStringBindingComposeW (NULL, "ncalrpc", NULL, "DNSResolver", "Security=Impersonation Dynamic False", 0x0...	RPC_S_OK
7	DNSAPI.dll	RpcBindingFromStringBindingW ("ncalrpc:[DNSResolver,Security=Impersonation Dynamic False]", 0x0012ebbc)	RPC_S_OK
8	DNSAPI.dll	RpcStringFreeW (0x0012ebc0)	RPC_S_OK
9	ADVAPI32.dll	DeviceIoControl (0x000007b0, 3735560, 0x77e46318, 256, 0x0012e850, 256, 0x0012e848, NULL)	TRUE
10	ADVAPI32.dll	DeviceIoControl (0x000007b0, 3735560, 0x77e46318, 256, 0x0012e850, 256, 0x0012e848, NULL)	TRUE
11	ADVAPI32.dll	DeviceIoControl (0x000007b0, 3735560, 0x77e46318, 256, 0x0012e850, 256, 0x0012e848, NULL)	TRUE
12	ADVAPI32.dll	DeviceIoControl (0x000007b0, 3735560, 0x77e46318, 256, 0x0012e850, 256, 0x0012e848, NULL)	TRUE
13	ADVAPI32.dll	DeviceIoControl (0x000007b0, 3735560, 0x77e46318, 256, 0x0012e850, 256, 0x0012e848, NULL)	TRUE
14	ADVAPI32.dll	DeviceIoControl (0x000007b0, 3735560, 0x77e46318, 256, 0x0012e850, 256, 0x0012e848, NULL)	TRUE
15	ADVAPI32.dll	DeviceIoControl (0x000007b0, 3735560, 0x77e46318, 256, 0x0012e850, 256, 0x0012e848, NULL)	TRUE
16	DNSAPI.dll	RpcBindingFree (0x0012eba8)	RPC_S_OK

Figure 33: Trace of DeviceIOControl() calls

The call tree above traced the first seven **DeviceIOControl()** invocations to the **NdrClientCall2()** function, which was called by the **gethostname()** function. In fact, *all* of the **DeviceIOControl()** calls that were recorded were traced back to **gethostname()**. Because the **gethostname()** function's only purpose is to determine the hostname of the local machine, it is not judged to be malicious (Microsoft, 2013b). As a result, **DeviceIOControl()** may be omitted from the report.

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

3.4.3. Obfuscation of Strings

During behavioral analysis, the **ChallengeSvc.exe** process was observed pinging host 157.166.226.26 by hard-coded IP address. However, the string “157.166.226.26” was not observed during surface analysis of either **SVccHost.dll** or **ChallengeSvc.exe**. This discrepency implied that the sample was employing some type of obfuscation in order to conceal the IP address.

In order to reveal the obfuscated text, the XORSearch tool was executed on the **ChallengeSvc.exe** file using “157.166.226.26” as the search string. XORSearch is a command line-based tool developed by Didier Stevens that accepts an input string and will conduct a brute force search of the specified file. The tool’s brute force search includes executing the following operations on each byte of the file (Stevens, 2013):

- Exclusive-or (XOR) for keys 0x00 through 0xFF
- Rotating left/right for keys 1 through 7 (ROL/ROR)
- Rotating English alphabet characters for keys 1 through 25 (ROT)
- Shifting left for keys 1 through 7

Once executed, the XORSearch tool returned a single result: a file XOR’ed with key **0xFF**:

```
C:\Exercise 305>XORSearch.exe -s ChallengeSvc.exe 157.166.226.26
Found XOR FF position 69D64: 157.166.226.26...t.~;..
```

When the resulting file **ChallengeSvc.exe.XOR.FF** was loaded into the BinText application, a series of de-obfuscated strings was revealed:

	File pos	Mem pos	ID	Text
A	00000006947C	00000006947C	0	Debugger Message:
A	000000069498	000000069498	0	Current Time is:
A	00000006967A	00000006967A	0	t<t+F
A	000000069691	000000069691	0	t<EO]
A	000000069864	000000069864	0	\SYSTEM\CurrentControlSet\Services\
A	000000069890	000000069890	0	Windows Security Application - Provides Challenge/Password Authentication Support
A	0000000698EC	0000000698EC	0	Description
A	000000069988	000000069988	0	SharedAccess
A	000000069C24	000000069C24	0	FileList
A	000000069D64	000000069D64	0	157.166.226.26
A	000000069F4C	000000069F4C	0	Ch.log
A	000000069F5C	000000069F5C	0	Current Time is:

Figure 34: Strings found by XOR'ing file with 0xFF

The identification of obfuscation methods *was* a defined goal of this exercise, thus this was flagged as a critical finding and added to the final report.

© 2013 The SANS Institute Author retains full rights.

4. Conclusion

The DC3 Digital Forensic Challenge provides students, enthusiasts, and professionals with relevant and realistic exercises to hone their computer forensics skills. Much like real-world computer forensic investigations, the exercises require competitors to develop and document viable and repeatable investigative methodologies—not merely to state their findings.

In the 2012 competition, participants were given two exercises that tested their malware analysis skills. Using the repeatable hybrid approach outlined in this paper, *Plan 9* successfully completed *Exercise 305: Basic Level Malware Analysis* using only surface and behavioral analysis techniques.

The solution to the exercise is summarized in Table 3.

Table 3: Results of Analysis

Objective	Finding
Programs or services installed, stopped, started, or modified	Service started: Challenge Service stopped: Windows ICS/Firewall
Purpose of programs and services	**SVccHost.dll:** Drop ChallengeSvc.exe into C:\Windows\system32 and execute by passing "/install" as a command line switch. **ChallengeSvc.exe:** Register itself as Challenge service. **Challenge Service:** Ping host 157.166.226.26 either: 1. Every 15 seconds for 90 minutes if no reply was received. 2. Every 10 seconds for 60 minutes if reply was received.
Location and method of installation	ChallengeSvc.exe was dropped in the C:\Windows\system32 directory and installed as the **Challenge** service
Files created, deleted, or modified	C:\Windows\system32\ChallengeSvc.exe created.
Registry settings created, deleted, or modified	See Appendix B for filtered list.
Network activity	Ping host 157.166.226.26 either: 1. Every 15 seconds for 90 minutes if no reply was received. 2. Every 10 seconds for 60 minutes if reply was received.
Obfuscation methods	Hard-coded IP string "157.166.226.26" was encoded using XOR, key=**0xFF**

© 2013 The SANS Institute Author retains full rights.

5. References

A.S.L. (2013). Exeinfo PE by A.S.L. – packer – compression detector and data detector. Retrieved from: http://www.exeinfo.antserve.com/

Barbara, J. (2006). A Standard Level of Acceptability for Computer Forensics. Retrieved from ASTM International: http://www.astm.org/SNEWS/FEBRUARY_2006/barbara_feb06.html

Batra, R. (2012). API Monitor. Retrieved from: http://www.rohitab.com/apimonitor

Blackwell, C. et al. (2013). Implementation of Digital Forensics Investigations Using a Goal-Driven Approach for a Questioned Contract. *9th Annual IFIP WG 11.9 International Conference on Digital Forensics*. Retrieved from: http://eprints.port.ac.uk/9809/1/ifipwg1192013_submission_27.pdf

Case, A. (2012, December 18). Analyzing Malware in Memory [PDF document]. Retrieved from Lecture Notes on Web Log: http://blog.hackeracademy.com/wp-content/uploads/2012/12/THA-Deep-Dive-Analyzing-Malware-in-Memory.pdf

comctl32.dll. (2010). In Process Library. Retrieved from: http://www.processlibrary.com/en/directory/files/comctl32/21090/

Department of Defense Cyber Crime Center (DC3). (2013, May). DC3 Mission. Retrieved from: http://www.dc3.mil/dc3/dc3Mission.php

Department of Defense Cyber Crime Center (DC3). (2012). DC3 Digital Forensic Challenge 2012. Retrieved from: http://www.dc3.mil/challenge/2012/

Department of Defense Cyber Crime Center (DC3). (2012). 2012 DC3 Digital Forensics Challenge Final Report. Retrieved from: http://www.dc3.mil/challenge/2012/files/2012_DC3_Digital_Forensics_Challenge_Final_Report.pdf

kernel32.dll. (2010). In Process Library. Retrieved from: http://www.processlibrary.com /directory/files/kernel32/23314/

Liao, Yibin. (2012). PE-Header-Based Malware Study and Detection. Retrieved from the University of Georgia: http://www.cs.uga.edu/~liao/PE_Final_Report.pdf

Mandiant. (2011, August 11). Using Indicators of Compromise to Find Evil and Fight Crime [PDF Document]. *GFIRST 2011 Conference*. Retrieved from 2011 Presentations Online Web site: http://www.us-cert.gov/sites/default/files/gfirst/presentations/2011/Using_Indicators_of_Compromise.pdf

Mandt, T. (2012, February 17). A Deep Dive Into AFD. Message posted to:

© 2013 The SANS Institute Author retains full rights.

http://mista.nu/blog/2012/02/17/cve-2012-0148-a-deep-dive-into-afd/

Microsoft Corporation. (2011, September 24). Definition and Explanation of a .DLL File. Retrieved from Microsoft KB: http://support.microsoft.com/kb/87934

Microsoft Corporation. (2013, July 25). DllMain Entry Point. Retrieved from MSDN: http://msdn.microsoft.com/en-us/library/windows/desktop/ms682583(v=vs.85).aspx

Microsoft Corporation. (2013, July 11). gethostname function. Retrieved from MSDN: http://msdn.microsoft.com/en-us/library/windows/desktop/ms738527(v=vs.85).aspx

Microsoft Corporation. (2013, June 11). Common Control Functions. Retrieved from MSDN: http://msdn.microsoft.com/en-us/library/windows/desktop/hh298349(v=vs.85).aspx

Microsoft Corporation. (2013, February 6). Microsoft Portable Executable and Common Object File Format Specification v8.3. Retrieved from MSDN: http://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx

oleaut32.dll. (2010). In Process Library. Retrieved from: http://www.processlibrary.com /directory/files/oleaut32/23291/

SHFolder.dll. (2010). In Process Library. Retrieved from: http://www.processlibrary.com /directory/files/shfolder/23462/

Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis*. San Francisco, CA: No Starch Press.

Stevens, D. (2013). XORSearch. Retrieved from Didier Stevens's Blog: http://blog.didierstevens.com/programs/xorsearch

Zeltser, L. (2013). REM Report Template. REMNux version 4.

Zeltser, L. (2013). Reverse Engineering Malware Cheat Sheet. Retrieved from: http://zeltser.com/reverse-malware/reverse-malware-cheat-sheet.html

6. Acknowledgments

I would like to recognize the other members of team *Plan 9* for their hard work and dedication during the *2012 DC3 Digital Forensic Challenge*. A hardy "thank you" goes out to Mr. Bill Littleton, Mr. Gordon Martin, and Mr. Derek Smith for their great effort and teamwork.

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 SANS Institute, Author retains full rights.

© 2013 The SANS Institute Author retains full rights.

Appendix A – List of Malware Analysis Tools

The following table summarizes the analysis tools that were used in the exercise solution.

Tool Name/Version	Analysis Type	Function	Reference
API Monitor v2	Behavioral	API Call Tracing	www.rohitab.com/apimonitor (Rohitab Batra, 2012)
BinText v3.03	Surface	GUI-based string display	www.mcafee.com (McAfee, Inc., 2013)
Cuckoo Sandbox v.60	Behavioral	Automated sandbox	www.cuckoosandbox.org (Guarnieri, 2012)
Exeinfo PE	Surface	Packer and file header information	www.exeinfo.antserve.com (‘A.S.L.’, 2013)
FileAlyzer v2.0	Surface	PE32 header and structure information	www.safer-networking.org (Safer-Networking Ltd, 2013)
md5deep v4.3	Surface	MD5 hash utility	md5deep.sourceforge.net (Kornblum, 2012)
Mandiant Red Curtain v1.0.0.9	Surface	PE32 header and import information	www.mandiant.com (Mandiant, 2008)
MiTEC EXE Explorer v1.4.0	Surface	PE32 header and structure information	www.mitec.cz/exe.html (MiTEC, 2013)
PEiD v.95	Surface	Packer and file header information	www.peid.info (defunct) (‘snaker’, 2008)
Process Hacker v2	Behavioral	Real-time process information	processhacker.sourceforge.net (‘wj32’, 2013)
RegShot v1.9.0	Behavioral	Tracks changes made to Registry and file system	sourceforge.net/projects/ regshot (‘maddes’, 2013)
Windows SysInternals Suite	Behavioral	System monitoring suite	technet.microsoft.com/en-us/sysinternals/default (Microsoft, 2013)
Wireshark v1.6.2	Behavioral	Packet Capture and Protocol Analysis	www.wireshark.org (Wireshark Foundation, 2013)
XORSEARCH	Surface	File de-	blog.didierstevens.com/

© 2013 The SANS Institute Author retains full rights.

v1.8		obfuscation via ROT and XOR	programs (Stevens, 2013)

© 2013 SANS Institute, Author retains full rights.

© 2013 The SANS Institute Author retains full rights.

Appendix B – Registry Activity (Regshot)

Regshot 1.9.0 x86 ANSI
Comments:
Datetime: XXXX
Computer: XXXX
Username: XXXX

Keys added: 12

HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\Control
HKLM\SYSTEM\ControlSet001\Services\Challenge
HKLM\SYSTEM\ControlSet001\Services\Challenge\Security
HKLM\SYSTEM\ControlSet001\Services\Challenge\Enum
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\Contro
l
HKLM\SYSTEM\CurrentControlSet\Services\Challenge
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Security
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Enum

Values added: 41

HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\NextInstance:
0x00000001
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\Service:
"Challenge"
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\Legacy:
0x00000001
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\ConfigFla
gs: 0x00000000
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\Class:
"LegacyDriver"
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\ClassGUI
D: "{8ECC055D-047F-11D1-A537-0000F8753ED1}"
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\DeviceDes
c: "Challenge"
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\Control*
NewlyCreated*: 0x00000000

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_CHALLENGE\0000\Control\ActiveService: "Challenge"
HKLM\SYSTEM\ControlSet001\Services\Challenge\Type: 0x00000010
HKLM\SYSTEM\ControlSet001\Services\Challenge\Start: 0x00000002
HKLM\SYSTEM\ControlSet001\Services\Challenge\ErrorControl: 0x00000001
HKLM\SYSTEM\ControlSet001\Services\Challenge\ImagePath: "C:\WINDOWS\system32\ChallengeSvc.exe"
HKLM\SYSTEM\ControlSet001\Services\Challenge\DisplayName: "Challenge"
HKLM\SYSTEM\ControlSet001\Services\Challenge\ObjectName: "LocalSystem"
HKLM\SYSTEM\ControlSet001\Services\Challenge\Description: "Windows Security Application - Provides Challenge/Password Authentication Support"
HKLM\SYSTEM\ControlSet001\Services\Challenge\Security\Security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 1C 00 01 00 00 00 02 80 14 00 FF 01 0F 00 01 01 00 00 00 00 00 01 00 00 00 00 02 00 60 00 04 00 00 00 00 00 14 00 FD 01 02 00 01 01 00 00 00 00 00 05 12 00 00 00 00 00 18 00 FF 01 0F 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 8D 01 02 00 01 01 00 00 00 00 00 05 0B 00 00 00 00 00 18 00 FD 01 02 00 01 02 00 00 00 00 00 05 20 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 00 01 01 00 00 00 00 00 05 12 00 00 00
HKLM\SYSTEM\ControlSet001\Services\Challenge\Enum\0: "Root\LEGACY_CHALLENGE\0000"
HKLM\SYSTEM\ControlSet001\Services\Challenge\Enum\Count: 0x00000001
HKLM\SYSTEM\ControlSet001\Services\Challenge\Enum\NextInstance: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\NextInstance: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\Service: "Challenge"
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\Legacy: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\ConfigFlags: 0x00000000
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\Class: "LegacyDriver"
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\ClassGUID: "{8ECC055D-047F-11D1-A537-0000F8753ED1}"
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\DeviceDesc: "Challenge"
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\Control*NewlyCreated*: 0x00000000
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CHALLENGE\0000\Control\ActiveService: "Challenge"
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Type: 0x00000010
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Start: 0x00000002
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\ErrorControl: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\ImagePath: "C:\WINDOWS\system32\ChallengeSvc.exe"
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\DisplayName: "Challenge"

Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

HKLM\SYSTEM\CurrentControlSet\Services\Challenge\ObjectName: "LocalSystem"
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Description: "Windows Security Application - Provides Challenge/Password Authentication Support"
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Security\Security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 1C 00 01 00 00 00 02 80 14 00 FF 01 0F 00 01 01 00 00 00 00 00 01 00 00 00 00 02 00 60 00 04 00 00 00 00 00 14 00 FD 01 02 00 01 01 00 00 00 00 00 05 12 00 00 00 00 00 18 00 FF 01 0F 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 8D 01 02 00 01 01 00 00 00 00 00 05 0B 00 00 00 00 00 18 00 FD 01 02 00 01 02 00 00 00 00 00 05 20 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 00 01 01 00 00 00 00 00 05 12 00 00 00
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Enum\0: "Root\LEGACY_CHALLENGE\0000"
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Enum\Count: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Services\Challenge\Enum\NextInstance: 0x00000001

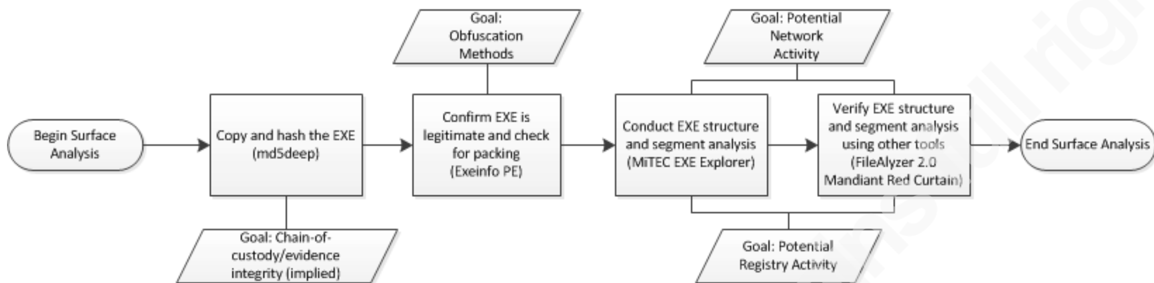
Kenneth J. Zahn, kenneth.j.zahn@gmail.com

© 2013 The SANS Institute Author retains full rights.

© 2013 SANS Institute, Author retains full rights.

Appendix C – Surface Analysis of ChallengeSvc.exe

The surface analysis process used for this sample is given by the following diagram:



Copy and Hash the EXE

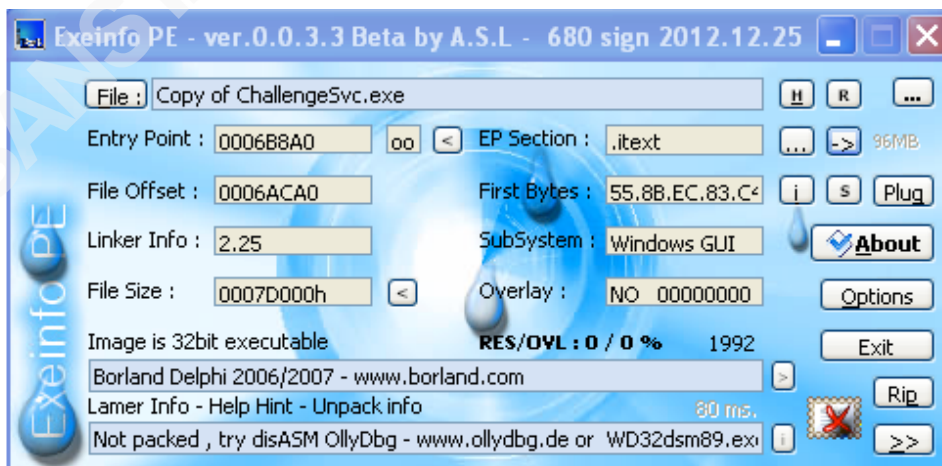
The output of the md5deep utility is:

```
2b618d0aedfd9313a37d14b05a2b688a  C:\Exercise 305\Copy of ChallengeSvc.exe
```

Confirm EXE is Legitimate and Check for Packing

Executing Exeinfo PE on **ChallengeSvc.exe** reveals two useful pieces of information:

3. The file is a legitimate 32-bit EXE (Win32 GUI subsystem) compiled using Borland Delphi 2006/2007, and
4. The EXE is *not* packed:



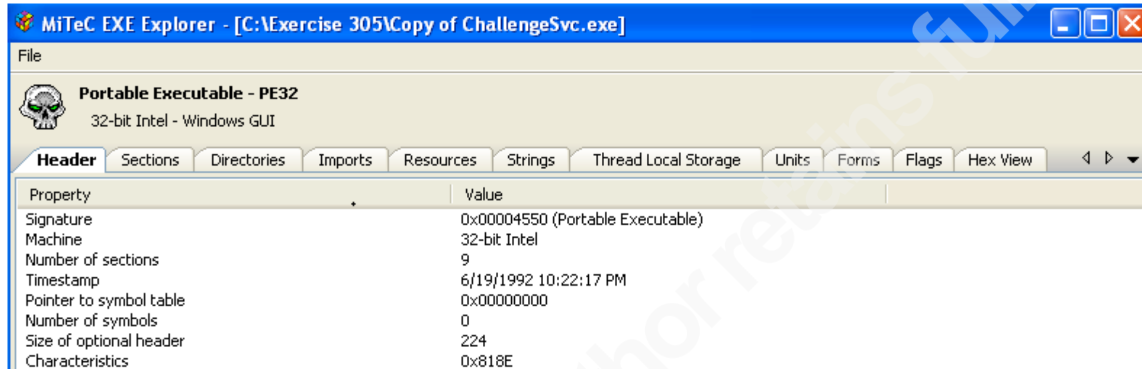
Selecting the '->' button next to the **EP Section** value generates the **Header Info** window. From this window, the examiner observes directory entries for an Import Table, *six* Resources and a Thread Local Storage (TLS) table:

© 2013 The SANS Institute Author retains full rights.

Import :	00072000	00002BDA	(05) .idata
Resource :	0007F000	00005800	0 % of exe 06
TLS Table :	00076000	00000018	(07) .rdata

Conduct EXE Structure and Section Analysis

The information found under MiTEC EXE Explorer's **Header** tab confirms that the EXE has a valid PE32 signature.



143 functions imported from eight separate libraries are observed under the **Imports** tab. The following table names and describes each of the imported libraries.

Imported DLL	Description
advapi32.dll	Provides access to advanced operating system functions such as the Service Manager and the Registry (Sikorski & Honig, 2012).
comctl32.dll	Provides access to user interface components (comctl32.dll, 2010).
gdi32.dll	Provides access to graphics display and manipulation functions (Sikorski & Honig, 2012).
kernel32.dll	Provides access to core operating system functions such as memory management, I/O operations, and hardware interrupts (kernel32.dll, 2010).
oleaut32.dll	Provides access to object linking and embedding (OLE) functions (oleaut32.dll, 2010).
SHFolder.dll	Provides access to special Windows folders (SHFolder.dll, 2010)
user32.dll	Provides access to user interface components and message handling (Sikorski & Honig, 2012).
version.dll	Provides functions to determine Windows versions (Microsoft, 2013d).

The following table names and describes a few of the more suspicious function imports.

© 2013 The SANS Institute Author retains full rights.

Imported Function(s)	Description
RegOpenKeyExW/RegCloseKey	Opens/closes the specified Registry key.
RegQueryValueExA	Queries the specified Registry key's value.
WriteFile	Writes data to disk.
WinExec	Passes commands to the Windows command shell.

Based on the function imports, the **ChallengeSvc.exe** has the ability to access the Windows Registry and write files to disk.

The **Resources** tab lists the structures that are found in the **.rsrc** section of the PE32 file. There are six separate resources, each of which is either a string table or a graphics icon.

The **Strings** tab lists all of the strings (both ASCII and UNICODE are supported by toggling the tab at the bottom of the main application pane) found in the PE32 file. A few strings resembling Windows service management calls are present:

```
3998 DisplayName
3999 Challenge
4000 AfterInstall
4001 ServiceAfterInstall
4002 OnExecute
4003 ServiceExecute
```

```
3021 StartServiceCtrlDispatcherA
3022 SetServiceStatus
3023 RegisterServiceCtrlHandlerA
3024 OpenServiceA
3025 OpenSCManagerA
3026 DeleteService
3027 CreateServiceA
3028 ControlService
3029 CloseServiceHandle
```

Further, the **Strings** tab reveals strings that are ICMP related:

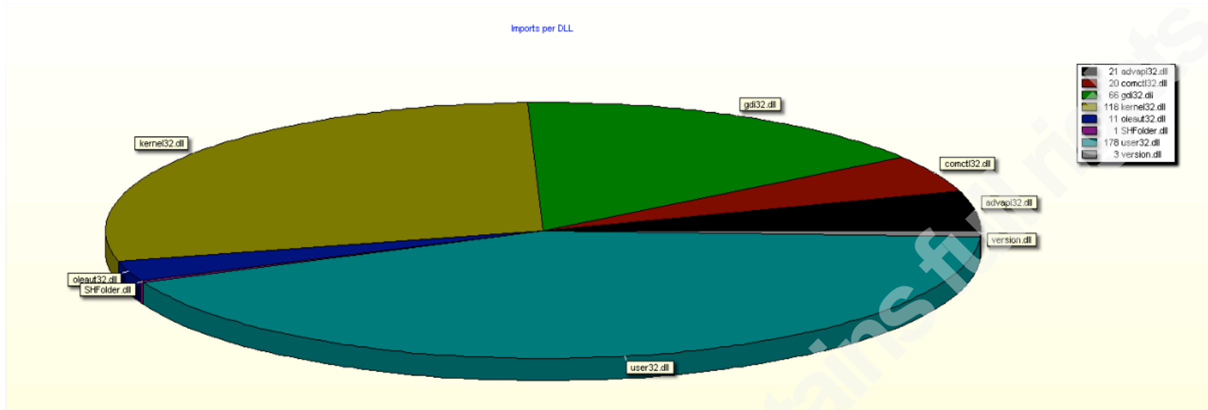
```
4006 TIdIcmpClient
4007 IdIcmpClient1
4008 Protocol
4009 ProtocolIPv6
4010 IPVersion
4011 Id_IPv4
4012 PacketSize
4013 OnReply
4014 IdIcmpClient1Reply
```


© 2013 SANS Institute Author retains full rights.

© 2013 The SANS Institute Author retains full rights.

Verify Structure and Segment Analysis Using Other Tools

FileAlyzer 2.0 counted a total of 418 imported functions from 8 DLL's:

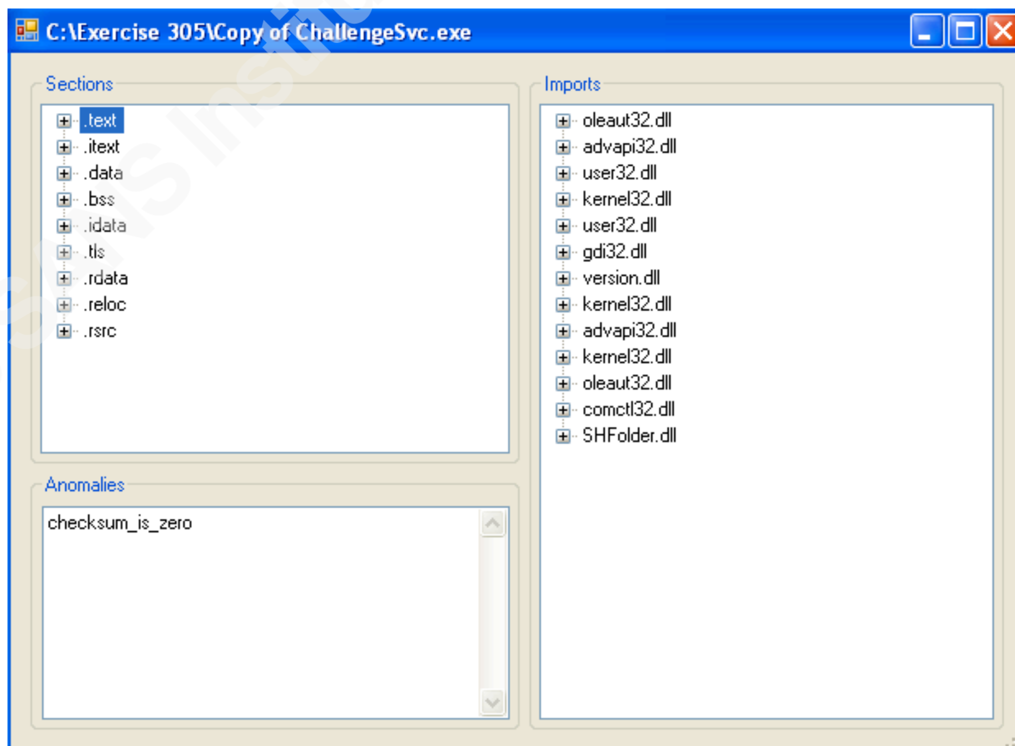


Mandiant Red Curtain confirms the Borland Delphi signature (again, Red Curtain disagrees with the version) and the **Details** pane confirms the function import count as 418:

Mandiant Red Curtain v1.0 - [Unsaved]

File Edit Options Help

Score	File	Size	Entry Point Signature	Entropy	Code Entropy	Anomaly Count	Signed	Details
1.140	C:\Exercise 305\Copy of ChallengeSvc.exe	512000	Borland Delphi v6.0 - v7.0	0.840	0.837	1		Details



© 2013 The SANS Institute Author retains full rights.

In addition to the single file anomaly of a zero checksum, Red Curtain indicates that the malware sample is not digitally signed.

© 2013 SANS Institute, Author retains full rights.

© 2013 The SANS Institute Author retains full rights.

Upcoming Training

Click Here to {Get CERTIFIED!}

SANS Security West 2014	San Diego, CA	May 08, 2014 - May 17, 2014	Live Event
Mentor Session - FOR 610	Columbia, MD	May 21, 2014 - Jul 23, 2014	Mentor
Digital Forensics & Incident Response Summit	Austin, TX	Jun 03, 2014 - Jun 10, 2014	Live Event
Community SANS Ottawa	Ottawa, ON	Jun 16, 2014 - Jun 21, 2014	Community SANS
SANSFIRE 2014	Baltimore, MD	Jun 21, 2014 - Jun 30, 2014	Live Event
SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	FOR610 - 201407,	Jul 14, 2014 - Aug 20, 2014	vLive
SANS Virginia Beach 2014	Virginia Beach, VA	Aug 18, 2014 - Aug 29, 2014	Live Event
SANS Baltimore 2014	Baltimore, MD	Sep 22, 2014 - Sep 27, 2014	Live Event
SANS DFIR Prague 2014	Prague, Czech Republic	Sep 29, 2014 - Oct 11, 2014	Live Event
SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	FOR610 - 201410,	Oct 13, 2014 - Nov 19, 2014	vLive
Community SANS Paris @ HSC - FOR610 (in French)	Paris, France	Nov 24, 2014 - Nov 28, 2014	Community SANS
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced