



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 permitted without express written permission.

Interested in learning more?

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

Discovering Winlogoff.exe

GIAC Reverse Engineering Malware (GREM) Practical
V. 2.0

Submitted by Jennie Callahan
4 March 2005

© SANS Institute 2000 - 2005. Author retains full rights.

Table of Contents

Table of Figures	2
Abstract:	3
Laboratory Setup:	4
Properties of Malware Specimen:	6
Behavioral Analysis:	8
Code Analysis:	16
Analysis Wrap-Up:	24
References	26

Table of Figures

Figure 1: Virtual Network	4
Figure 2: GT2 Output	7
Figure 3: PEid Results	7
Figure 4: Process Explorer Results	9
Figure 5: TDIMon Results	11
Figure 6: TCPView Results	11
Figure 7: Ethereal DNS Requests	12
Figure 8: Snort DNS Requests	13
Figure 9: Netcat on Linux VM	14
Figure 10: IRCD on Linux VM	15
Figure 11: Ethereal TCP Stream	15
Figure 12: HView Edit	17
Figure 13: UPX Unpack	17
Figure 14: OllyDbg Breakpoint	21
Figure 15: OllyDbg "badpass"	21
Figure 16: OllyDbg NOP	22
Figure 17: Patched antivirus.exe	23
Figure 18: OllyDbg Packed Password	24

Abstract:

The following pages of this report were written to establish a working knowledge of the course material presented in Reverse Engineering Malware. An examiner often times will be faced with an unknown file, which they must determine the functionality of. Analysis is essentially dissected into two main sections, behavioral analysis and code analysis. The behavioral analysis demonstrates what the file does and the code analysis demonstrates why the file does it. In this report, the first aspect covered is the laboratory setup used for examination. Simple and standard file identification processes are next covered. The behavioral analysis and then the code analysis are the next two sections. The final section is the analysis wrap-up, in which a general overview of the examination findings was provided.

© SANS Institute 2000 - 2005, Author retains full rights.

Laboratory Setup:

When examining malware specimens or unknown files, it is important to conduct the analysis in an environment where the examiner has control of the file and the surroundings. For this exercise, an isolated network environment was created involving a host computer and two VMware virtual machines (Figure 1).

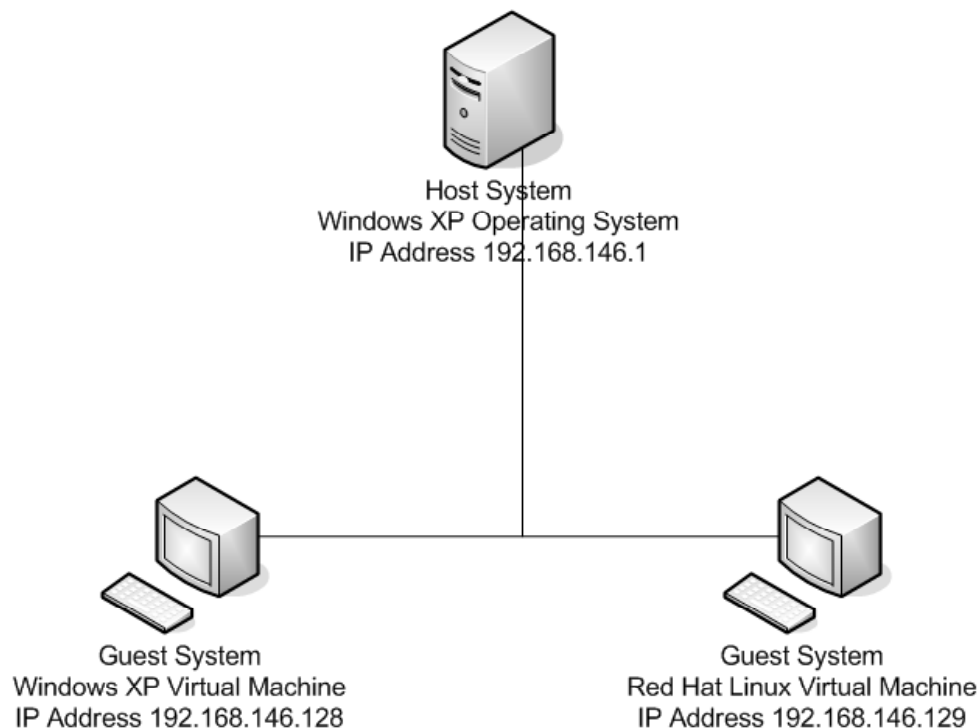


Figure 1: Virtual Network

The host computer was an Intel based laptop computer which contained a 1.7 GHz processor, 2 GB of RAM, and an 80 GB hard drive. The computer used the Microsoft Windows XP Operating System with Service Pack 2. The computer maintained current Microsoft updates and used Symantec AntiVirus and Firewall, which were also fully updated. The host system contained the software VMware Workstation, version 4.5.2 for Microsoft Windows, which was used to create the two VMware virtual machines. Additionally the computer used the software Screen Shot Deluxe version 4.0 which was used for all screen captures in this report. The host computer was designated as the virtual network gateway and assigned the IP address 192.168.146.1. The host computer was disconnected from any other network connections preventing access to the Internet and other networks not controlled within the controlled virtual network.

The first virtual machine contained the guest operating system Red Hat Linux version 9.0 (Kernel 2.4.20-8). The Red Hat operating system was accessed from console mode only and X Windows was not installed. The virtual machine was provided by the instructor during the Reverse Engineering Malware Course (2-23 Feb 05). The virtual machine contained a maximum size virtual hard drive of 2 GB, but only approximately 640 MB was used at the time of the examination. The virtual machine used 64 MB of RAM and also used a host only virtual network card which was configured to use DHCP. The virtual machine obtained its IP address, 192.168.146.129, from the host computer. Host only networking created a private network between the host and the guest operating systems. The virtual machine was provided with Snort version 2.0.4, IRCD version 2.8, and netcat version 1.10 preinstalled, which were the only programs used other than the operating system. Snort is an open source intrusion detection system which provides traffic analysis and packet capturing [1]. IRCD is a freely available IRC server program for Linux [2]. Netcat is a network utility designed to read and write data across a network [3].

The second virtual machine contained the guest operating system Microsoft Windows XP with Service Pack 1. The virtual machine maintained all Microsoft patches available with the service pack, but did not use antivirus and firewall software. The virtual machine contained a maximum size virtual hard drive of 4 GB, but only approximately 2.3 GB was used at the time of the examination. The virtual machine used 256 MB of RAM and used the physical CD-Rom device from the host machine. The virtual machine used a host only virtual network card which was configured to use DHCP. The virtual machine obtained its IP address, 192.168.146.128, from the host computer.

This Windows XP virtual machine was used to execute the unknown malware and numerous utilities were installed to analyze the process at various stages in the analysis process. IDA Pro version 4.7, written by DataRescue, is a disassembler and debugger which is registered for my use. A free version of the utility (version 4.3) is available for download from numerous locations on the Internet, but was provided to me by my instructor during the Reverse Engineering Malware Course. OllyDbg is another debugger that allows you to examine executable programs and view the internal code structure [4]. GT2 version 0.34 is a DOS based file detection utility which provides the identity of a file and file information. The GT2 program is available at the author's website [5]. PEid version 0.93, is a utility that detects common packers, cryptors and compilers for PE files (portable executables) [6]. UPX is the ultimate packer for executables version 1.25. The program is free and designed to compress and decompress executable files [7]. Bintext version 3.0, is a free utility which parses out text from inside of other files similar to the Unix "strings" command. Bintext is owned by Foundstone, Inc. and available from their website [8]. HView 2000 version 1.0 is a text editor that allows the user to view and edit files. HView is a freely available tool written by Tolo Oliver and is available for download at numerous websites [9]. Md5dum is a part of the Text Utilities

Suite (Windows version) which calculates the MD5 hash value of the specified file [10]. RegShot version 1.61e5 is a free registry compare utility that allows you to take a snapshot of your registry and then compare it with a second one; done after making system changes [11]. Process Explorer version 8.03 (XP Edition), TCPView version 2.34, and TDIMon version 1.0, were all freely available system monitor tools. Process Explorer shows the running process on a computer, the files and directories accessed by the file and TCP connections used by the file. TCPView shows all TCP and UDP connections the processes on a computer uses. TDIMon also shows TCP and UDP connections used from a file running on a computer [12]. The final tool used on the Windows XP virtual machine was Ethereal version 0.10.9. The program is a network protocol analyzer and is used in a similar manner to Snort in that it can capture and analyze packets sent across a network. Unlike Snort, Ethereal has a Windows GUI interface which allows for easy use and clearer displays of the analyzed traffic [13].

Each of the above listed programs was installed on the Windows XP host machine. Most tools were placed in a directory created for this examination entitled C:\Tools. IDAPro, Ethereal and GT2 were installed using the default installer settings, placing the utilities in the C:\Program Files directory. Using VMware, a snapshot was taken of the virtual machine to preserve the integrity of the clean install. Any changes made to the virtual machine could be undone by reverting back to the snapshot.

Properties of Malware Specimen:

For this analysis I was provided a file entitled winlogoff.exe. I transferred a copy of the file to my Windows XP virtual machine.

The first step I took in identifying the file was to use the tool GT2. GT2 can be accessed by right clicking on the file in question and clicking "Detect with GT2". A command prompt will open and the file will be analyzed using the gt2.exe file. This is a feature available with the installation. The GT2 program file may also be accessed by a command prompt and navigating to the program file directory. In this case I used the command C:\Program Files\GT2\gt2.exe C:\winlogoff.exe from a Windows command prompt located on the Windows XP virtual machine. The program provided a detailed listing of the winlogoff properties (Figure 2). The GT2 revealed the file size was 16384 bytes and that it was a DOS executable. The winlogoff file required a Windows based operating system to run on. Additionally, the GT2 program found the file was packed with UPX.

```

C:\Program Files\GT2>gt2 "c:\Documents and Settings\vmware\Desktop\winlogoff.exe"
gt2 0.34 (c) 1999-2004 by PHaX (coding@helger.com)
- c:\Documents and Settings\vmware\Desktop\winlogoff.exe (16384 bytes) - binary

Is a DOS executable
Size of header: 00000040h/64 bytes
File size in header: 00000490h/1168 bytes
Entrypoint: 00000040h/64
Overlay size: 00003B70h/15216 bytes
No relocation entries

PE EXE at offset 00000080h/128
Entrypoint: 00003BE0h / 15328
Entrypoint RVA: 000299E0h
Entrypoint section: 'GRM1'
Calculated PE EXE size: 00004000h / 16384 bytes
Image base: 00400000h
Required CPU type: 80386
Required OS: 4.00 - Win 95 or NT 4
Subsystem: Windows GUI
Linker version: 2.56
Stack reserve: 00200000h / 2097152
Stack commit: 00001000h / 4096
Heap reserve: 00100000h / 1048576
Heap commit: 00001000h / 4096
Flags:
Relocation info stripped from file
File is executable
Line numbers stripped from file
Local symbols stripped from file
Machine based on 32-bit-word architecture
Debugging info stripped from file in .DBG file

Processed with:
Found packer 'UPX 0.89.6 - 1.02 / 1.05 - 1.24 [PEI]'

C:\Program Files\GT2>_

```

Figure 2: GT2 Output

Since the GT2 program showed the winlogoff file was a packed executable. I decided to use the tool PEid to determine if additional information could be found regarding the file. PEid is a Windows GUI based tool. The user opens the program and then directs the program to the unknown file. PEid immediately scans the file and displays the results. The results of the PEid scan (Figure 3) provided the same information that GT2 had also listing the winlogoff executable as a UPX packed executable.

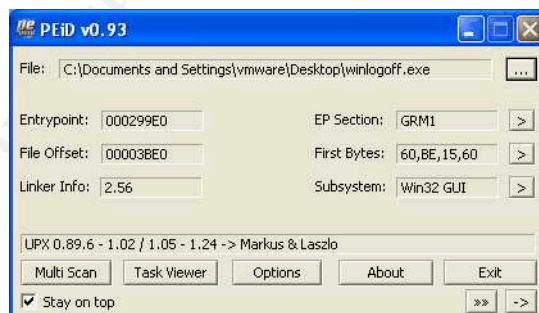


Figure 3: PEid Results

Neither the GT2 or PEid programs ran the winlogoff file or altered the file in anyway. The two utilities merely read the contents of the file and display the results in a user friendly format.

The next step in the analysis was to obtain and MD5 hash value of the file. In a command prompt inside of the Windows XP virtual machine, I entered the command C:\Tools\md5sum C:\winlogoff.exe. The md5sum program generated

the hash value 8b37148bc11c09a33224fa2ac6b25613.

The next step in the analysis process was to obtain a list of text embedded in the winlogoff file. Analyzing the text embedded within the file allows may provide the examiner with additional leads to follow up on. The text may contain the author, the true file name, version information, help information, and the proper usage of the file. For this step I used the program BinText, a Windows GUI based utility. After opening the BinText program file, I directed the program to the location of winlogoff and selected the "Go" button. The text contained within the winlogoff file was revealed and I exported the information to a file called strings_packed.txt. A review of the text revealed the program could not be run in DOS mode (meaning it required the Windows environment) and it contained the string "ThisIsNotThePasswordYouAreLookingFor". No other relevant information was discovered. This is due to the fact that the winlogoff file was packed using UPX. The compression of the file distorts the strings which are normally available when the file is decompressed. The winlogoff file must be decompressed and viewed again in BinText to obtain a new clearer version of the text embedded within the file. This process is conducted later in the analysis located in the "Code Analysis" section of this report.

Behavioral Analysis:

The next step in the analysis process is to see how the program behaves when it is run. The examiner must determine what the file does once it is executed. Typically, program files have the ability to run quietly in the background without making the full extent of the program's abilities known to the user. A malware specimen is more likely to hide its functionality from the casual observer since it is in its very nature to be malicious. In order to see what the casual observer does not, there are numerous tools to be launched before executing the malware specimen. The winlogoff file was described as a Windows executable file, so I maintained using the Windows XP virtual machine. In this analysis, I used the system monitor tools Process Explorer, RegShot, TDIMon, TCPView, and Ethereal. Although to some it may be considered a waste of time, I used each tool separately when launching the winlogoff file. The additional steps of launching one monitor tool at a time, executing winlogoff, reviewing the output, and reverting to the saved snapshot of Windows XP helped ensure I did not miss any of the file's properties.

The initial tool used was Process Explorer (XP). I opened the program file and it showed the currently running processes. I then launched the winlogoff file by double clicking it. The process winlogoff.exe appeared on the display window of Process Explorer highlighted in green indicating a new process was started. Winlogoff generated a new process entitled antivirus.exe and then winlogoff terminated displaying highlighted in red for termination. The program antivirus.exe continued to run (Figure 4). By right clicking on the antivirus.exe process and selecting properties or by viewing the bottom pane of the Process

Explorer window an examiner can see additional information pertaining to the files. The additional information includes the path of the file, *.dll files being used, TCP processes, and security of the file. In this case, no additional relevant information was discovered.

Process	PID	CPU	Description	Compa...
System Idle Process	0	88		
Interrupts	0	4	Hardware Interrupts	
DPCs	0		Deferred Procedu...	
System	4			
smss.exe	616		Windows NT Ses...	Microsoft...
csrss.exe	664		Client Server Run...	Microsoft...
winlogon.exe	688		Windows NT Log...	Microsoft...
services.exe	732	3	Services and Con...	Microsoft...
svchost.exe	928		Generic Host Pro...	Microsoft...
svchost.exe	1032		Generic Host Pro...	Microsoft...
wuauclt.exe	1748		Automatic Updates	Microsoft...
svchost.exe	1300		Generic Host Pro...	Microsoft...
svchost.exe	1368		Generic Host Pro...	Microsoft...
spoolsv.exe	1628		Spooler SubSyste...	Microsoft...
alg.exe	1916		Application Layer ...	Microsoft...
srvcy.exe	1980			
resetService.exe	2036			
VMwareService.exe	2024		VMware Tools Se...	VMware, ...
lsass.exe	760		LSA Shell (Export ...	Microsoft...
explorer.exe	1600	1	Windows Explorer	Microsoft...
VMwareTray.exe	1836		VMwareTray	VMware, ...
VMwareUser.exe	1844		VMwareUser	VMware, ...
procexp.exe	1268	3	Sysinternals Proc...	Sysintern...
antivirus.exe	512			

Figure 4: Process Explorer Results

I reverted to the saved snapshot and began the analysis over again, this time using the system monitor tool RegShot. Prior to executing the winlogoff.exe file, I opened the RegShot program file. I chose to view the comparison results as a text file, I chose to scan the entire root directory (C:\), and then directed the output path to my desktop. I took a snapshot of the registry at the time of the clean install and prior to the execution of the winlogoff.exe file by clicking on the 1st Shot button. I then executed the winlogoff.exe file and waited approximately 60 seconds before taking a second snapshot of the registry by clicking the 2nd Shot button. I terminated the antivirus.exe program and then clicked on the Compare button from RegShot. A text file was generated which showed the following registry key was added:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices

Additional review of the output from RegShot revealed the following values were added:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Microsoft
 Windows Antivirus Service: 61 6E 74 69 76 69 72 75 73 2E 65 78 65 00 00
 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices\Micros
 oft Windows Antivirus Service: 61 6E 74 69 76 69 72 75 73 2E 65 78 65 00 00

Finally, the RegShot comparison file also revealed the following file was added:

C:\WINDOWS\system32\antivirus.exe

I verified the changes had actually been made to the registry by manually looking for them in the native Windows program regedit. The information provided by RegShot confirmed the creation of the antivirus.exe program and revealed the files location. Additionally, RegShot revealed the registry entries which would force the antivirus.exe program to automatically restart when the computer was restarted and a user logged in. At this point I returned to one of my file identification tools, md5sum.exe. I used the utility to obtain a hash value of the antivirus.exe file, which revealed a matching hash value to the one obtained from winlogoff.exe. Winlogoff.exe generated a copy of itself and placed it in the C:\WINDOWS\system32 directory.

I reverted to the saved snapshot and began the analysis over again, this time using the system monitor tool TDIMon. I opened the TDIMon program file, paused the capture (Ctrl-E), and cleared the current capture results (Ctrl-X). When I was ready to execute winlogoff.exe, I restarted the TDIMon capture (Ctrl-E). I executed winlogoff.exe and let the program run for approximately 60 seconds. I paused the TDIMon capture and terminated the antivirus.exe program. I reviewed the display of the previous 60 seconds captured by TDIMon. The program revealed the winlogoff.exe program starting and closing, the antivirus program starting, and then a brief moment later the Windows program svchost.exe began sending information to the host computer 192.168.146.1 on port 53 (Figure 5). This was likely a DNS request (typical of UDP traffic trying to reach port 53) in an effort to resolve a host name. The host name look up was likely generated from the execution of the antivirus.exe file. No other file activity was noted during the 60 seconds, the suspicious file was executed.

© SANS Institute

TDIMon - SysInternals: http://www.sysinternals.com

File Edit Capture Options Help

#	Time	Process	Object	Request	Local	Remote	Result
6	0.002...	winlogoff.exe:11	81629B70	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
7	0.002...	winlogoff.exe:11	81629B70	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
8	0.002...	winlogoff.exe:11	81629B70	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
9	1.061...	winlogoff.exe:11	81629B70	IRP_MJ_CLEANUP	TCP:Control obj		SUCCESS
10	1.062...	winlogoff.exe:11	81629B70	IRP_MJ_CLOSE	TCP:Control obj		SUCCESS
11	1.062...	winlogoff.exe:11	815D4D80	IRP_MJ_CLEANUP	TCP:Control obj		SUCCESS
12	1.070...	winlogoff.exe:11	815D4D80	IRP_MJ_CLOSE	TCP:Control obj		SUCCESS
13	1.181...	antivirus.exe:11	8159EF90	IRP_MJ_CREATE	TCP:Control obj		SUCCESS
14	1.181...	antivirus.exe:11	8158D808	IRP_MJ_CREATE	TCP:Control obj		SUCCESS
15	1.182...	antivirus.exe:11	8159EF90	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
16	1.182...	antivirus.exe:11	8159EF90	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
17	1.182...	antivirus.exe:11	8159EF90	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
18	1.183...	antivirus.exe:11	8159EF90	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
19	1.183...	antivirus.exe:11	8159EF90	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
20	1.183...	antivirus.exe:11	8159EF90	IRP_MJ_DEVICE_CONTROL	TCP:Control obj		SUCCESS
21	1.226...	svchost.exe:1268	816B47B0	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
22	1.226...	svchost.exe:1268	816B47B0	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
23	1.226...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
24	1.226...	svchost.exe:1268	816B47B0	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
25	1.227...	svchost.exe:1268	816B47B0	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
26	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
27	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
28	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
29	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
30	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
31	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
32	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
33	1.235...	svchost.exe:1268	815E41A8	IRP_MJ_DEVICE_CONTROL	TCP:<none>		SUCCESS
34	1.242...	svchost.exe:1268	8169D208	TDI_SEND_DATAGRAM	UDP:0.0.0.0:1026	192.168.146.1:53	SUCCESS
35	2.234...	svchost.exe:1268	8169D208	TDI_SEND_DATAGRAM	UDP:0.0.0.0:1026	192.168.146.1:53	SUCCESS
36	3.235...	svchost.exe:1268	8169D208	TDI_SEND_DATAGRAM	UDP:0.0.0.0:1026	192.168.146.1:53	SUCCESS

Figure 5: TDiMon Results

I reverted to the saved snapshot and began the analysis over again, this time using the system monitor tool TCPView. I opened the TCPView program file and then launched the winlogoff.exe file. I waited approximately 60 seconds and then terminated the antivirus.exe file. During the entire time the programs were running, no TCP or UDP connections were noted as being generated by either winlogoff.exe or antivirus.exe (Figure 6).

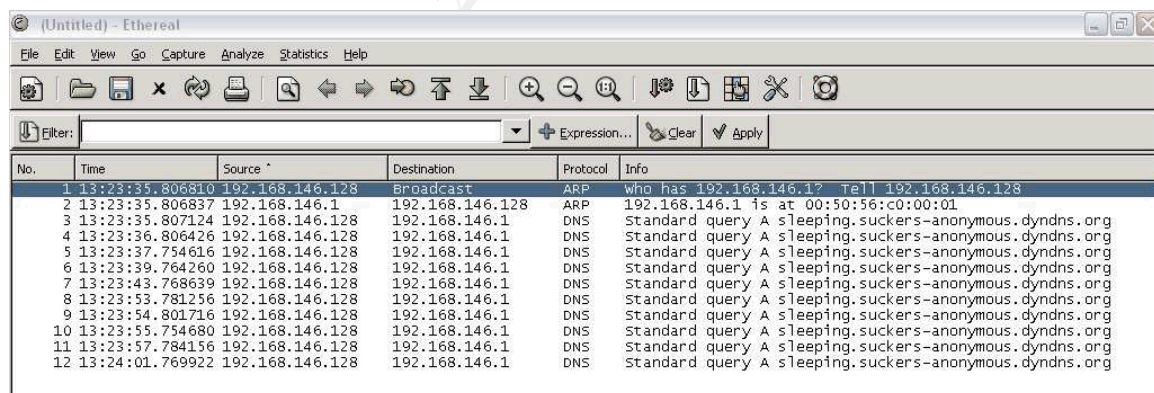
TCPView - Sysinternals: www.sysinternals.com

File Options Process View Help

Proc...	Protocol	Local Address	Remote Address	State
alg.exe:1988	TCP	xp-vm:3001	xp-vm:0	LISTENING
lsass.exe:744	UDP	xp-vm:lsakmp		
svchost.exe:1...	TCP	xp-vm:1025	xp-vm:0	LISTENING
svchost.exe:1...	TCP	xp-vm:3002	xp-vm:0	LISTENING
svchost.exe:1...	TCP	xp-vm:3003	xp-vm:0	LISTENING
svchost.exe:1...	UDP	xp-vm:3008		
svchost.exe:1...	UDP	xp-vm:nlp		
svchost.exe:1...	UDP	xp-vm:localdomain...		
svchost.exe:1...	UDP	xp-vm:1026		
svchost.exe:1...	UDP	xp-vm:3007		
svchost.exe:1...	TCP	xp-vm:5000	xp-vm:0	LISTENING
svchost.exe:1...	UDP	xp-vm:1900		
svchost.exe:1...	UDP	xp-vm:localdomain...		
svchost.exe:9...	TCP	xp-vm:epmap	xp-vm:0	LISTENING
System:4	TCP	xp-vm:microsoft-ds	xp-vm:0	LISTENING
System:4	TCP	xp-vm:1027	xp-vm:0	LISTENING
System:4	TCP	xp-vm:localdomain...	xp-vm:0	LISTENING
System:4	UDP	xp-vm:microsoft-ds		
System:4	UDP	xp-vm:localdomain...		
System:4	UDP	xp-vm:localdomain...		

Figure 6: TCPView Results

I reverted to the saved snapshot and began the analysis over again, this time using the system monitor tool Ethereal. I opened the Ethereal program and began capturing by clicking on Capture and Start from the menu bar. I configured the program to listen in promiscuous mode and to update and scroll the real time capture. Promiscuous mode on a network card allows the card to capture all traffic on the network, even if the information is not bound for the particular computer the sniffer is running on. I clicked the Okay button and the capture began. At this point, I introduced the Linux virtual machine to my virtual network. I started the Snort packet capture by using the command `snort -vd | tee /tmp/sniffer.log`. This command began the sniffer process and displayed the results to both my screen and to a file called `sniffer.log` for later review. This sniffer was also configured for promiscuous mode, but I did not configure any of the settings. Snort was preinstalled and configured to work in the classroom environment and also worked for the purpose of this examination. Both sniffer programs were used to ensure no network traffic was missed; however, the results were the same after the `winlogoff` file was executed. The Windows XP virtual machine, assigned IP address 192.168.146.128, began sending requests to the host computer, assigned IP address 192.168.146.1, to determine what computer had the domain name `sleeping.sucker-anonymous.dyndns.org` on port 53 (Figures 7 and 8). This was a typical DNS request indicating the program `antivirus.exe` was attempting to access the `sleeping.sucker` computer and needed to know what address to find it at. The host computer was not setup as a DNS server and the information could not be found; however, the DNS requests from the virtual machine continued. After approximately two minutes I terminated the `antivirus.exe` program.



No.	Time	Source *	Destination	Protocol	Info
1	13:23:35.806810	192.168.146.128	Broadcast	ARP	who has 192.168.146.1? Tell 192.168.146.128
2	13:23:35.806837	192.168.146.1	192.168.146.128	ARP	192.168.146.1 is at 00:50:56:c0:00:01
3	13:23:35.807124	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
4	13:23:36.806426	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
5	13:23:37.754616	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
6	13:23:39.764260	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
7	13:23:43.768639	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
8	13:23:53.781256	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
9	13:23:54.801716	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
10	13:23:55.754680	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
11	13:23:57.784156	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org
12	13:24:01.769922	192.168.146.128	192.168.146.1	DNS	Standard query A sleeping.suckers-anonymous.dyndns.org

Figure 7: Ethereal DNS Requests


```

Len: 55
00 07 01 00 00 01 00 00 00 00 00 00 08 73 6C 65 .....sle
65 70 69 6E 67 11 73 75 63 6B 65 72 73 2D 61 6E eping.suckers-an
6F 6E 79 6D 6F 75 73 06 64 79 6E 64 6E 73 03 6F onymous.dyndns.o
72 67 00 00 01 00 01 rg....

=====

02/25-02:22:45.845182 192.168.146.128:1026 -> 192.168.146.1:53
UDP TTL:128 TOS:0x0 ID:176 IpLen:28 DgmLen:83
Len: 55
00 07 01 00 00 01 00 00 00 00 00 00 08 73 6C 65 .....sle
65 70 69 6E 67 11 73 75 63 6B 65 72 73 2D 61 6E eping.suckers-an
6F 6E 79 6D 6F 75 73 06 64 79 6E 64 6E 73 03 6F onymous.dyndns.o
72 67 00 00 01 00 01 rg....

=====

02/25-02:22:47.851186 192.168.146.128:1026 -> 192.168.146.1:53
UDP TTL:128 TOS:0x0 ID:177 IpLen:28 DgmLen:83
Len: 55
00 07 01 00 00 01 00 00 00 00 00 00 08 73 6C 65 .....sle
65 70 69 6E 67 11 73 75 63 6B 65 72 73 2D 61 6E eping.suckers-an
6F 6E 79 6D 6F 75 73 06 64 79 6E 64 6E 73 03 6F onymous.dyndns.o
72 67 00 00 01 00 01 rg_

```

Figure 8: Snort DNS Requests

The next step in the analysis was to determine what the antivirus.exe program wanted with the computer assigned the domain name sleeping.sucker-anonymous.dyndns.org. The host computer could have been configured as a DNS server and responded with a given IP address when the DNS request was given. Also, the computer could have been connected to the Internet and the real IP address of the sleeping.sucker computer could have been provided. Neither of these is a good idea as it prevents the examiner from keeping control of the examined file and the DNS server is an unnecessary step. In the Windows XP virtual machine file system, there is a host file which is reviewed by the computer in an effort to resolve domain names before attempting to access a DNS server. In this case I edited the file C:\WINDOWS\system32\drivers\etc\hosts to reflect that the computer assigned IP address 192.168.127.129 was assigned the domain name sleeping.sucker-anonymous.dyndns.org. The IP address assigned was the IP address to my Linux virtual machine. This allowed the traffic being sent by the antivirus program to be contained within the virtual network and under the control of the examiner. After making the changes to the hosts file, I restarted Ethereal and Snort to obtain clean packet captures and then I restarted the antivirus.exe program. At this point the Windows XP virtual machine read the host file and saw that the computer using the sleeping.sucker DNS was the Linux virtual machine. Ethereal and Snort both captured network traffic from the Windows XP virtual machine attempting to connect to the Linux virtual machine on port 8080. Port 8080 is typically used for web services. No web server or any service for that matter was listening on port 8080 of the Linux virtual machine. While the antivirus.exe file was still running, I started the netcat program on the Linux virtual machine. I used the command nc -l -p 8080, which set the netcat program listening on port 8080 for incoming connections. Incoming information was displayed on the screen. The screen showed the words NICK and USER which words are typically associated with IRC. The nick and user appeared to be assigned the name ugkk (Figure 9). This name appeared to be randomly

generated, but further attempts to login in using the antivirus.exe would later confirm this. Only netcat was operating on port 8080, which is nothing like an IRC server. No other information was available at this stage as Ethereal had only shown the same text displayed from netcat.

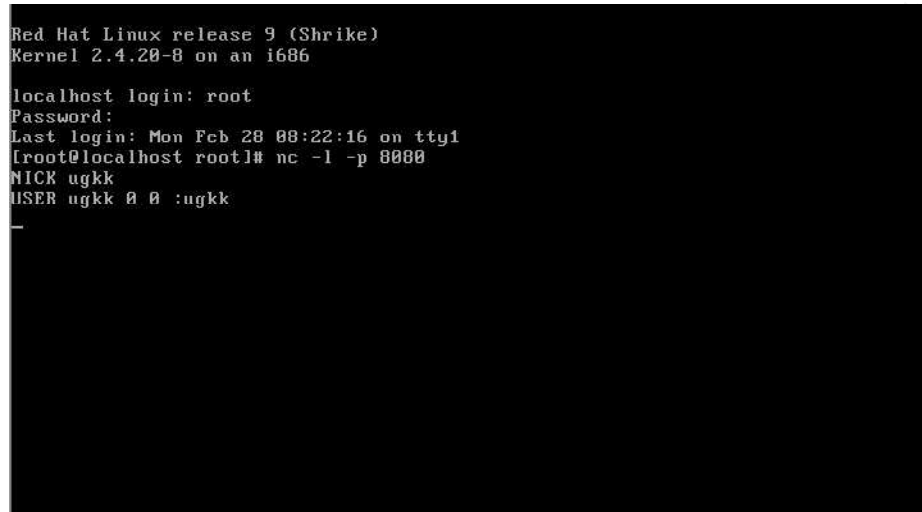
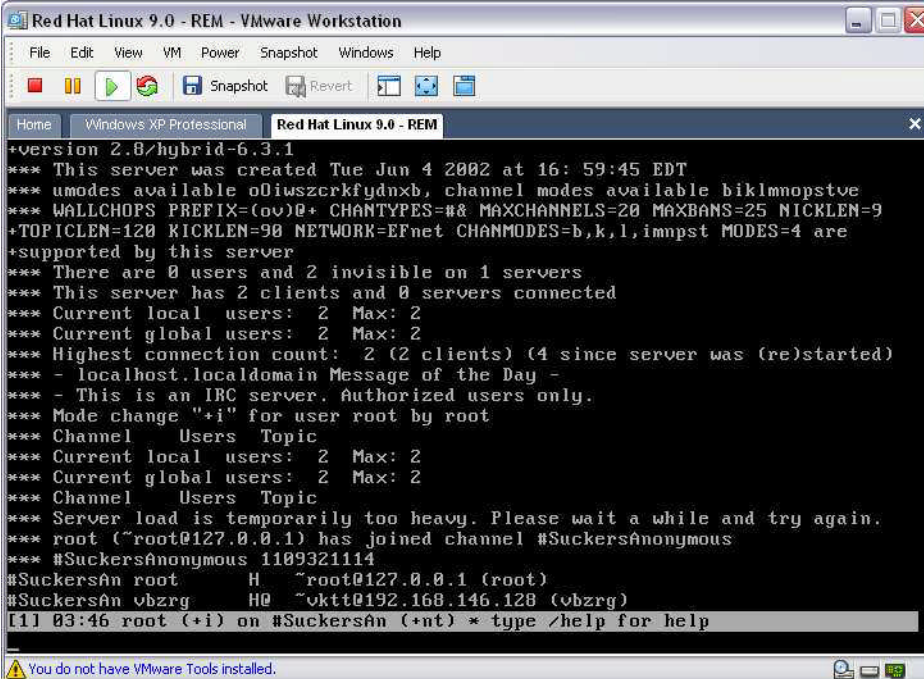
A screenshot of a terminal window with a black background and white text. The text shows the system booting into Red Hat Linux release 9 (Shrike) with kernel 2.4.20-8 on an i686. It then shows a root login on localhost. The user enters the password, and the system shows the last login time as Mon Feb 28 08:22:16 on tty1. The user then runs the command 'nc -l -p 8080' to start a netcat listener. The prompt changes to 'NICK ugkk' and then 'USER ugkk 0 0 :ugkk'.

Figure 9: Netcat on Linux VM

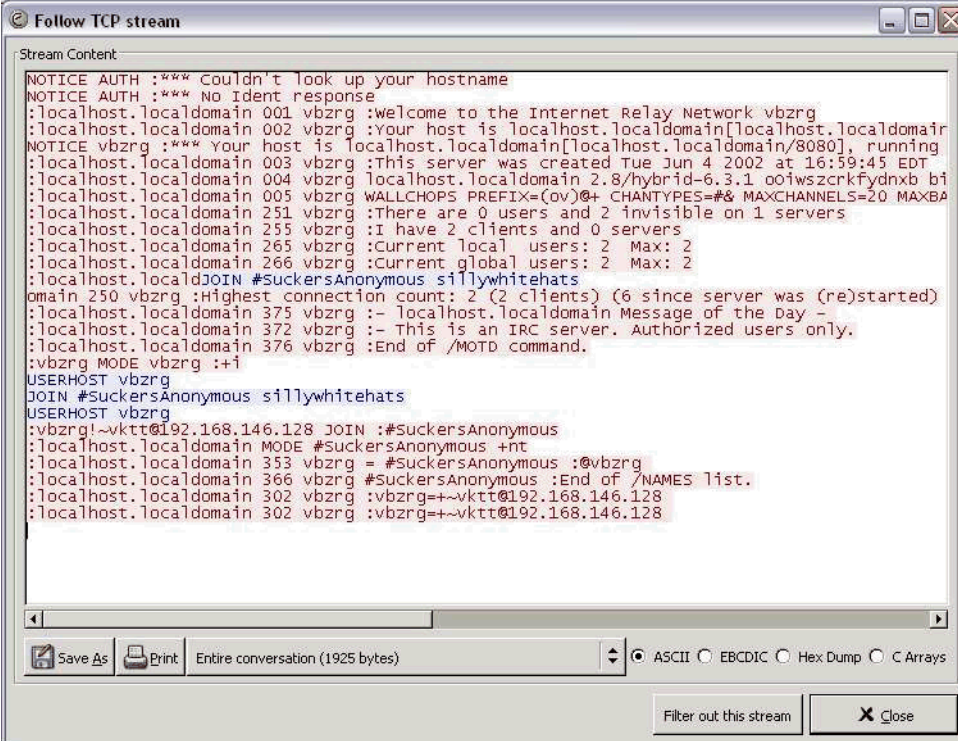
Assuming that the antivirus.exe program was searching for an IRC server on port 8080, I needed to provide the program access to one. The IRC server program IRCD was previously installed on the Linux virtual machine, but it was configured to listen for connections on ports 6666 and 6667, the common IRC server ports. I used the text editor “vi” to add port 8080 to the IRCD configuration file located at /usr/local/ircd/ircd.conf. Once the configuration file was saved, I started the IRC server by logging in as the user ircd (\$su – ircd) and starting the ircd service (\$./ircd). I exited from that user (\$exit) and as root I started the IRC client program (\$irc). I reverted back to the snapshot of my Windows XP virtual machine, started Ethereal and then executed the winlogoff.exe file again. This time, the antivirus.exe file caused the Windows XP virtual machine to successfully connect to the IRC server on the Linux virtual machine. The Windows XP virtual machine received a new nick and user name, vbzrg, confirming the nick creation was random (Figure 10). Ethereal has a feature which allows a user to single out specific streams captured. The user can right click on a particular TCP line in the display window and choose Follow TCP Stream. A secondary window opens displaying the specific TCP stream between the two designated computers. In this case, I followed the TCP stream of the Windows XP virtual machine (IP address 192.168.146.128) and the Linux virtual machine (IP address 192.168.146.129). The TCP display showed the traffic sent between the two virtual machines in order to connect to the IRC server (Figure 11). Based on the traffic analysis, the Windows XP virtual machine joined the channel #SuckersAnonymous using the channel password of “sillywhitehats”. I transferred back to my IRC client on the Linux virtual machine and logged into the same channel, using the same password. I

attempted numerous commands in order to interact with the Windows XP virtual machine, but all attempts resulted in no response.



```
+version 2.8/hybrid-6.3.1
*** This server was created Tue Jun 4 2002 at 16: 59:45 EDT
*** umodes available o0iwszcrkfydnxb, channel modes available biklmnopstve
*** WALLCHOPS PREFIX=(ov)@+ CHANTYPES=#& MAXCHANNELS=20 MAXBANS=25 NICKLEN=9
+TOPICLEN=120 KICKLEN=90 NETWORK=EFnet CHANMODES=b,k,l,impst MODES=4 are
+supported by this server
*** There are 0 users and 2 invisible on 1 servers
*** This server has 2 clients and 0 servers connected
*** Current local users: 2 Max: 2
*** Current global users: 2 Max: 2
*** Highest connection count: 2 (2 clients) (4 since server was (re)started)
*** - localhost.localdomain Message of the Day -
*** - This is an IRC server. Authorized users only.
*** Mode change "+i" for user root by root
*** Channel Users Topic
*** Current local users: 2 Max: 2
*** Current global users: 2 Max: 2
*** Channel Users Topic
*** Server load is temporarily too heavy. Please wait a while and try again.
*** root (~root@127.0.0.1) has joined channel #SuckersAnonymous
*** #SuckersAnonymous 1109321114
#SuckersAn root H ~root@127.0.0.1 (root)
#SuckersAn vbzrg HQ ~vkt@192.168.146.128 (vbzrg)
[11 03:46 root (+i) on #SuckersAn (+nt) * type /help for help
```

Figure 10: IRCD on Linux VM



```
Stream Content
NOTICE AUTH :*** Couldn't look up your hostname
NOTICE AUTH :*** No ident response
:localhost.localdomain 001 vbzrg :welcome to the Internet Relay Network vbzrg
:localhost.localdomain 002 vbzrg :Your host is localhost.localdomain[localhost.localdomain/8080], running
NOTICE vbzrg :*** Your host is localhost.localdomain[localhost.localdomain/8080], running
:localhost.localdomain 003 vbzrg :This server was created Tue Jun 4 2002 at 16:59:45 EDT
:localhost.localdomain 004 vbzrg :localhost.localdomain 2.8/hybrid-6.3.1 o0iwszcrkfydnxb bi
:localhost.localdomain 005 vbzrg WALLCHOPS PREFIX=(ov)@+ CHANTYPES=#& MAXCHANNELS=20 MAXBA
:localhost.localdomain 251 vbzrg :There are 0 users and 2 invisible on 1 servers
:localhost.localdomain 255 vbzrg :I have 2 clients and 0 servers
:localhost.localdomain 265 vbzrg :Current local users: 2 Max: 2
:localhost.localdomain 266 vbzrg :Current global users: 2 Max: 2
:localhost.localdomain JOIN #SuckersAnonymous sillywhitehats
domain 250 vbzrg :Highest connection count: 2 (2 clients) (6 since server was (re)started)
:localhost.localdomain 375 vbzrg :- localhost.localdomain Message of the Day -
:localhost.localdomain 372 vbzrg :- This is an IRC server. Authorized users only.
:localhost.localdomain 376 vbzrg :End of /MOTD command.
:vbzrg MODE vbzrg :+i
USERHOST vbzrg
JOIN #SuckersAnonymous sillywhitehats
USERHOST vbzrg
:vbzrg!~vkt@192.168.146.128 JOIN :#SuckersAnonymous
:localhost.localdomain MODE #SuckersAnonymous +nt
:localhost.localdomain 353 vbzrg = #SuckersAnonymous :@vbzrg
:localhost.localdomain 366 vbzrg #SuckersAnonymous :End of /NAMES list.
:localhost.localdomain 302 vbzrg :vbzrg=+~vkt@192.168.146.128
:localhost.localdomain 302 vbzrg :vbzrg=+~vkt@192.168.146.128
```

Figure 11: Ethereal TCP Stream

At this point I presumed the examined file was an IRC bot. An IRC bot is a

program that sits in an IRC channel and looks just like a normal user on the channel, but is usually idle until it's called upon to perform a particular function. [14]. Like many programs available in the world, the functions could be put to good or to bad uses. Based on previous experience with IRC bot files, the interaction to access the particular functions of this bot required a password used by the bot master. At this point, I did not have the password and had completed the behavioral analysis as far as I could.

Code Analysis:

As previously noted in the Behavioral Analysis section of this report, the programs PEid and GT2 identified the winlogoff file as being packed with UPX. While inside of my Windows XP virtual machine, I attempted to use the program IDA Pro which also recognized the file as being packed. I subsequently ceased my analysis using IDA Pro as the disassembler and debugger utility would not properly reveal the contents of the file while it was still packed.

The UPX program has a decompression switch built within the program. This is not true for all packing utilities and sometimes a search for a workable unpacker is necessary. From the Windows XP virtual machine, I attempted to use the decompression functionality of the UPX program file in a Windows command prompt by typing the command `upx -d c:\winlogoff.exe` from the program file's directory. The attempt to decompress resulted in error message relating the winlogoff file was modified, hacked, or protected.

I next chose to view the contents of the binary file winlogoff by using the utility HView2000 from the Windows XP virtual machine. The review of the contents of the file revealed no occurrence of the letters "UPX", which are typically resident in a UPX compressed file. I did however see the letters "GRM", where I believed the letters UPX should be. Using HView, I changed the four occurrences of GRM to UPX and saved the file as winlogoff_edit.exe (Figure 12). Once again I used the decompression functionality of the UPX program in an effort to unpack the file the newly edited file. This time the file was successfully unpacked. UPX revealed a Windows 32-bit portable executable file, which was now 57856 bytes in size (Figure 13).

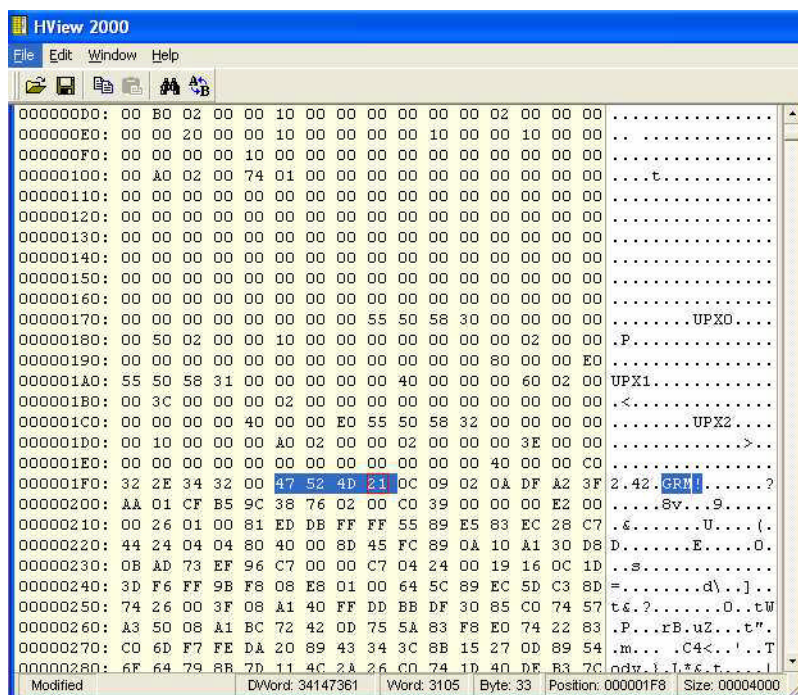


Figure 12: HView Edit

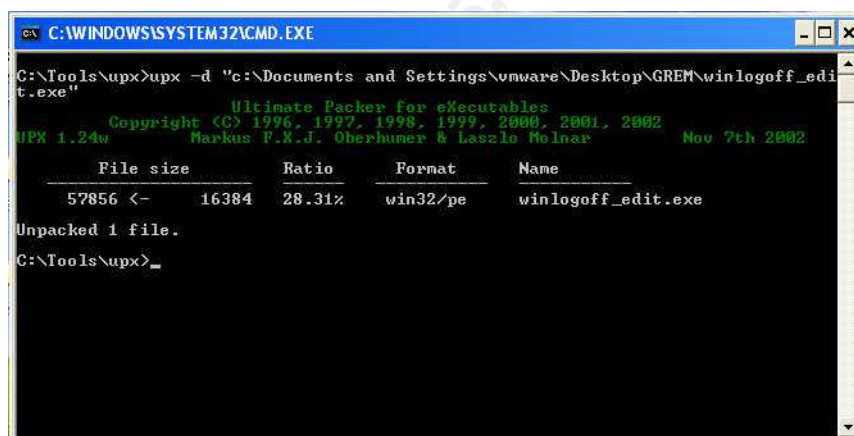


Figure 13: UPX Unpack

I returned to my file identification tool, BinText, and viewed the text strings not previously available from the packed version of winlogoff.exe from the Windows XP virtual machine. This time the strings revealed numerous clues as to the nature of the file I was examining. Key strings noted included the following:

```
0000C637 0040E037 0 SLoT bot 2.0 by Black Ninja
0000C657 0040E057 0 sillywhitehats
0000C666 0040E066 0 #SuckersAnonymous
0000C680 0040E080 0 sleeping.suckers-anonymous.dyndns.org
0000C6A6 0040E0A6 0 antivirus.exe
0000CA0D 0040E40D 0 Software\Microsoft\Windows\CurrentVersion\Run
0000CA3B 0040E43B 0 Software\Microsoft\Windows\CurrentVersion\RunServices
0000CCC1 0040E6C1 0 password accepted.
```

0000CE2C 0040E82C 0 SLoT bot 2.0 by SLoTH homepage <http://yeah-right/suckersanonymous>

The text strings now available revealed the file could be SLoT bot version 2.0 by SLoTH. The text revealed the IRC channel and password, the DNS, the secondary file name, and the registry entries noted from the behavioral analysis. Also noted was the string “password accepted” which presents a good starting point for the code analysis when using the disassemblers. Unfortunately, text analysis of a binary file is not always helpful. False strings can be written into a file, the same way I used HView to alter the letters GRM to UPX. Individuals creating the binaries can insert false leads or leave no information what so ever.

The same behavioral analysis steps taken above were repeated resulting using the winlogoff_edit.exe file. Each of the system monitor tools resulted in the same findings. This step was taken to ensure the unpacked file did not exhibit any new characteristics not previously seen in the packed version.

The next step was to return to the disassembler. I chose to use IDA Pro which had been installed on the Windows XP virtual machine and opened the file winlogoff_edit.exe. Once the file was completely loaded into the disassembler using the default settings, I searched for the string “password” as seen in the BinText results. I was brought to the string “password accepted” located at offset 0040354A. Several lines above the string “password accepted” there was a call for a string compare (strcmp) at offset 004034B3 just after the word “login”. This indicates the program will look for the string login from the user input and if it matches will continue with the subroutine. In between the login and the password accepted was another string compare which looked for the word “sillywhitehat”. This was likely the password used for the login process.

I decided to test the discovery from IDA Pro and launched the winlogoff_edit.exe program. I did not use IDA Pro to control the execution of the file, I just merely executed the file from its location on the Desktop of the Windows XP virtual machine. The execution of the unpacked version of winlogoff resulted in an unpacked version of antivirus.exe. I continued to let the program run and connect to the IRC Server which was still running on the Linux virtual machine. After the Windows XP virtual machine successfully joined the #SuckersAnonymous channel I returned to the IRC client and used the Linux virtual machine in an effort to interact with the bot. I used the command “login sillywhitehat” which resulted in the successful login to the victim computer. The program generated the text “password accepted” as previously discovered in BinText and IDA Pro. Attempts were made to login in with the wrong password as well, but no indication was given that that the login failed.

I returned to the Windows XP virtual machine and used IDA Pro to view the assembly language of the unpacked version of antivirus.exe. I scrolled through the information following the successful login. At offset 004039D6, I discovered

the beginning of what appeared to be a long list of commands which could be used with the program. The assembly language showed additional string compares looking for the particular keyword to execute that part of the program. Noted below most of the possible commands, there was what appeared to be a shortcut, one or two letters and a subsequent string compare, leading to the same portion of the program. Overall, there appeared to be 50 separate commands noted in IDA Pro. I returned the Linux virtual machine and tried 10 of the commands. The commands tested would not work without first entering in the login command accompanied by the proper password. The following is the results of the commands tested:

The command netinfo (ni) provided details on the connection type (dial-up, lan, etc), the local IP and the IP address from where the computer connected from.

The command threads (t) showed the words thread list. Presumably any threads would have been shown as well if they existed.

The command status (s) showed the text SLoT bot 2.0 ready and provided the uptime in days, hours, and minutes.

The command vmware resulted in a lost connection whereby all other commands became ineffective. The command may be used by the program to determine if it is being run in VMware. In this case, it was.

The command sysinfo (si) provided the cpu, ram, space total and free, os, and uptime from the computer which was running the bot.

The command id (i) showed only the text SLoT.

The command about (ab) showed the text SLoT bot 2.0 by SLoTH homepage <http://yeah-right/suckersanonymous> as previously noted from the BinText analysis.

The command aliases (al) resulted in the sign out of the computer using the bot. It is not likely this was supposed to happen, but it is what happened each time I ran the command.

The command open (o) opened a file on the computer running the bot. In this case, I opened notepad.exe on the Windows XP virtual machine.

The command visit (v) would send the computer running the bot to the specified URL. In this case, I sent it to <http://www.cnn.com>. There was no network connectivity, so the attempt failed.

The command remove (rm) disconnected the computer running the bot from IRC, stopped the program antivirus.exe, removed the registry entries

created by the program; however, the file C:\WINDOWS\system32\antivirus.exe remained.

Initially, no attempt to patch or debug the file was made since the password was clear in the antivirus.exe file when viewed with IDA Pro. However, if the password was not visible an examiner may view antivirus.exe in a debugger and take steps to discover the true password. Using the debugger utility, OllyDbg, an examiner would set a breakpoint (keyboard shortcut F2 will set a breakpoint) at offset 0040351E (Figure 14). This is the location of the string compare just after the word sillywhitehat. Setting the breakpoint will pause the program's execution when it reaches the breakpoint and allow the examiner to see what is going on with the program and to control the next step the program takes. The file should be executed from within the debugger. In this case the keyboard shortcut F9 could be used to accomplish the task; however, there are numerous ways to start the program. After the computer running the antivirus file has joined the IRC channel, the examiner can attempt to login in using a known incorrect password; in this case, badpass. On the computer using OllyDbg, the antivirus.exe program pauses at the breakpoint set at the string compare. Displayed in the lower right hand pane in OllyDbg, is the wrong password as well as the correct password (Figure 15). The examiner can then let the program finish running and then try to login again with the newly discovered correct password. Using the Windows XP virtual machine all actual attempts at these steps resulted in access violations to antivirus.exe. The antivirus file would stop running and would never actually access the IRC channel. On a hunch, I introduced a new temporary member to the virtual network. I created a virtual machine using the Microsoft Windows 2000 Server operating system. The virtual machine was configured to use a maximum of a 4 GB hard drive, 256 MB of RAM, and a host only network interface. The virtual machine contained no patches and the only additional tool installed was OllyDbg. The same steps noted above were taken using the Windows 2000 virtual machine and the debugging process successfully revealed the wrong password as well as the correct password.

© SANS Institute

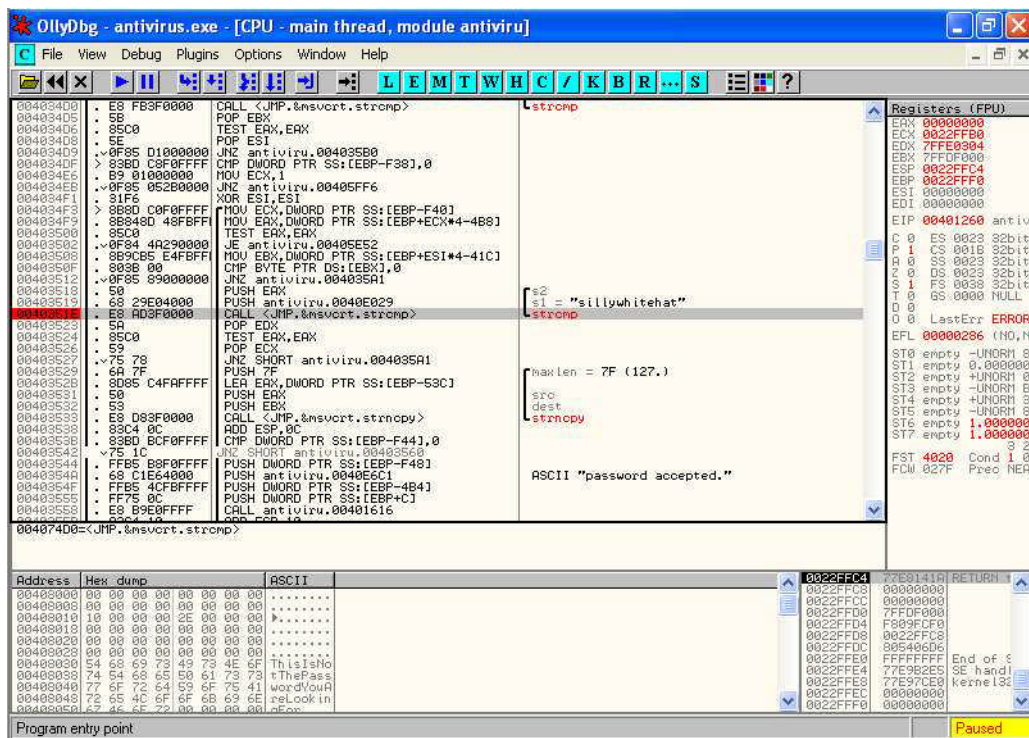


Figure 14: OllyDbg Breakpoint

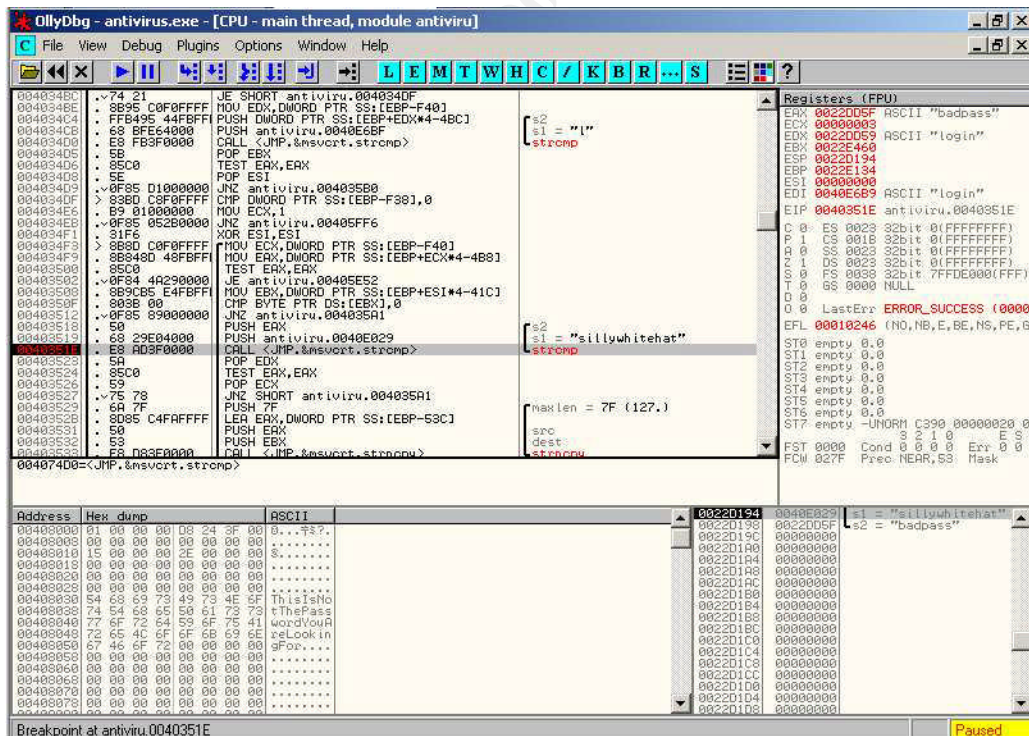


Figure 15: OllyDbg “badpass”

The antivirus.exe file could also be patched to where the file would accept any password. In order to patch the antivirus.exe file, the program was loaded into OllyDbg. I traversed to offset 00403527, which was where the jump instruction

atched (Fig. 1).
Windows 20

**0bq NOP**

```
*** cuvfpm (~xvguju@192.168.146.130) has joined channel #SuckersAnonymous
> login badpass
<cuvfpm> password accepted.
> ni
<cuvfpm> connection type: LAN (LAN Connection). local IP address:
+192.168.146.130. connected from: 192.168.146.130

[1] 04:10 @root (+i) on #SuckersAn (+nt) * type /help for help
```

Figure 17: Patched antivirus.exe

In my initial code analysis, I modified the unknown file, winlogoff.exe using a hex editor. While this time the method was successful, it will not always be so easy to decompress or unpack a packed executable. It is possible to obtain a decompressed version of the file from the compressed file using OllyDbg. It is important to know what type of packer is used and it may be necessary to do some Internet research to find out about the particular qualities of a packer. In this case, we know the file was packed using UPX. From the Windows XP virtual machine, I executed the unpacked version of winlogoff.exe allowing it to generate an unpacked version of antivirus.exe, which I promptly terminated. I then used OllyDbg to open the unpacked version of antivirus.exe. I allowed the antivirus program to run using the F9 keyboard shortcut and allowed the Windows XP virtual machine to join the IRC channel. Once it had joined, I returned to OllyDbg and accessed the Memory Map of the running antivirus.exe. The memory map showed three location of interest labeled GRM0, GRM1, and GRM2. GRM0 is where the unpacked version of the executable resides as the section begins at offset 00401000 and extends until offset 00426000. The unpacked assembly language earlier showed the relevant string compare was located at offset 0040351E. From the Memory Map screen, I right clicked and selected Dump in CPU. The dump opens up in a new window and the examiner must chose to disassemble the instructions by right clicking on the lower left hand pane, right clicking and selecting disassemble. In order to obtain the correct password similar steps as taken above are executed. I set a breakpoint at 0040351E by right clicking, choosing breakpoint and then choosing memory on access. No visual display is provided showing the breakpoint is set. I ran the file allowing the Windows XP virtual machine to login to IRC. I attempted to login with the wrong password and seen previously, OllyDbg reveals the see true password (Figure 18).

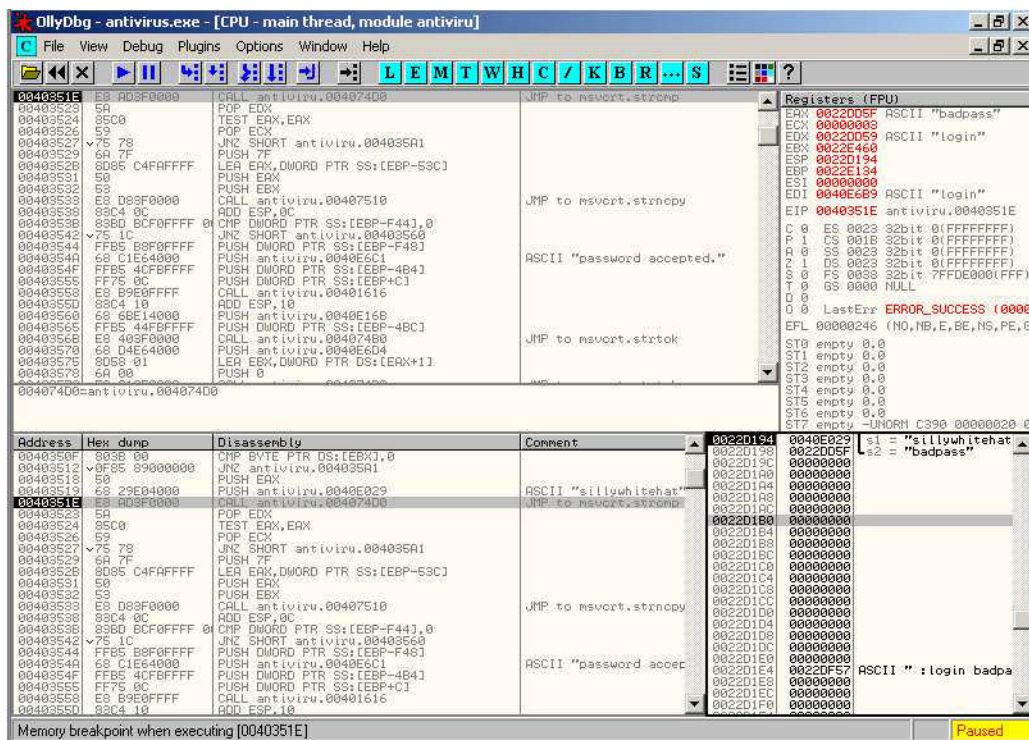


Figure 18: OllyDbg Packed Password

Analysis Wrap-Up:

This malware specimen is an IRC bot which when executed creates a copy of itself in the C:\%WINDIR%\system32\ called antivirus.exe. The malware creates registry entries enabling the file antivirus.exe to start each time the computer is restarted. Once installed, the malware attempts to connect to the domain name sleeping.suckers-anonymous.dyndns.org. If the host is available, the malware attempts to connect to an IRC server located at that address on port 8080. If an IRC server is running on port 8080, the malware obtains a nick which appears to be randomly generated. The malware then joins the IRC channel #SuckersAnonymous with the password "sillywhitehats". The malware waits in the IRC channel until accessed using the proper authentication method. In this case, the method is entering the command and password "login sillywhitehat".

The IRC bot is a common tool used by hackers. The program allows the bot master (the person controlling the computers running the bot program) to cause a lot of damage. Files can be accessed, programs executed, denial of service attacks can be initiated and all malicious activity can be executed without the knowledge of the average computer user.

In order to ensure a computer does not become the victim of an IRC bot program, the user should practice safe computing. They should enable antivirus and firewall programs. The user should not open executable files received from unknown sources. If they are a member of a managed network, the network

administrators should also implement antivirus and firewall utilities. Proper email filters can remove executable files from email messages, which prevents accidental execution from the unwary email reader.

© SANS Institute 2000 - 2005, Author retains full rights.

References

- [1] Caswell, Brian et al. Snort Homepage. 1 March 2005 <<http://www.snort.org>>.
- [2] IRCD Homepage. 1 March 2005 <<http://www.funet.fi/~irc/server/>>.
- [3] Netcat Download Site. 1 March 2005 <<http://www.securityfocus.com/tools/137>>.
- [4] Yuschuk, Oleh. OllyDbg Homepage. 1 March 2005 <<http://home.t-online.de/home/OllyDbg/>>.
- [5] PHaX. GT2 Homepage. 1 March 2005 <<http://philip.helger.com/gt/>>.
- [6] Jibz, et al. PEid Homepage. 1 March 2005 <<http://peid.has.it/>>.
- [7] Oberhumer, Markus, et al. UPX Homepage. 1 Mar 2005 <<http://upx.sourceforge.net/>>.
- [8] Foundstone, Inc. BinText Download Page. 1 March 2005 <<http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/freetools.htm>>
- [9] HView Download Page. 1 March 2005 <<http://www.softlookup.com>>.
- [10] Md5sum Download Page. 1 March 2005 <<http://gnuwin32.sourceforge.net/>>.
- [11] TiANWEi. RegShot Homepage. 1 March 2005 <<http://regshot.yeah.net>>.
- [12] Process Explorer, TDIMon, and TCPView Homepage. 1 March 2005 <<http://www.sysinternals.com>>.
- [13] Ethereal Homepage. 1 March 2005 <<http://www.ethereal.com>>.
- [14] Eggdrop IRC Bot Help Homepage. 1 March 2005 <<http://www.egghelp.org/whatis.htm>>.