Global Information Assurance Certification Paper

LABORATORY SETUP

The laboratory is set up on a sole laptop system. The system is a Dell Latitude, Intel Pentium III-850 MHz with 528 MBs of Ram. The host operating system is Windows 2000 with Service Pack 4 and is current with all patches. No other program other than the operating system and the tools used in the analysis are installed. The purpose of this system is to host 2 operating systems in a virtual environment in order to analyze the code.

I will be using the VMware Workstation 4.05 build-6030 upon which are installed two operating systems.

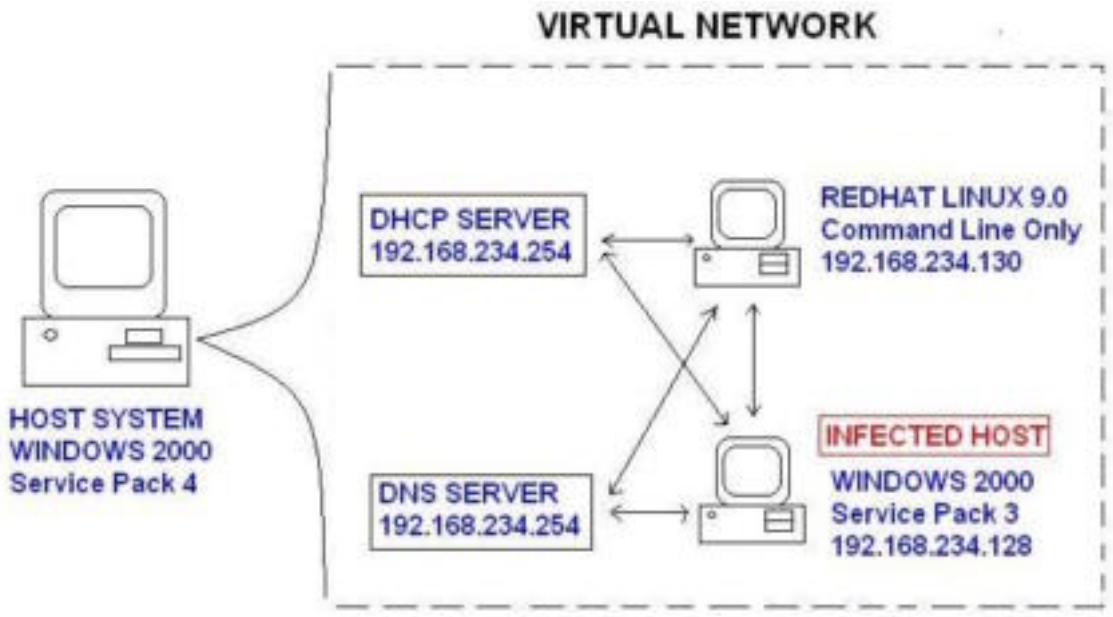
The first operating system installed is Windows 2000 with Service Pack 3. The system has been apportioned 192 MBs of RAM as well as 3 GBs of hard drive space. Networking on this virtual machine has been setup as Host-only and connects when powered on. The IP address is 192.168.234.128. In this system, various analysis tools have been installed, however I will explain these in more detail at the end of this section. The purpose of this virtual system is to be used as a victim and to be infected with the code to be studied. Please note that throughout the analysis, I will refer to this system as the "Virtual Host" or the "Infected Host".

The second operating system installed is Red Hat Linux 9.0. This virtual system is command line only and was obtained from the REM CD that came with the course. The system has been apportioned 64 MBs of RAM and is installed as disk file upon the C drive of the host system. Networking has been setup as Host-only and connects when powered on. The IP address is 192.168.234.130. The purpose of this system is to monitor the network traffic that may be sent by the code and to provide services that may be needed. Throughout the analysis, I will refer to this system as the "Virtual Linux" system.

There is another benefit to using a Linux machine in this virtual environment. If necessary, additional instances of Linux can be started by pressing the [ALT] key and one of the [F] keys. For example, a second console can be started by holding [ALT] and then pressing [F2].

It should be noted that VMware also includes a virtual DHCP server (192.168.234.254) and a virtual DNS Server (192.168.234.1) to provide networking services.

The laboratory has been installed on a cleanly formatted hard drive. Before installing the VMware system, the host operating system was updated with the latest Microsoft patches. After these patches were downloaded and installed, the machine was disconnected from the network before proceeding further. Due to the fact that this analysis will examine live malicious code, this system will not be connected to any network. Below is a diagram of the setup used.



Below are the tools that I will be using with the analysis which are installed on the "Infected Host":

BinText v 3.0: A small utility capable of extracting text such as ASCII, Unicode and other strings from a file. This tool will be used to examine the code in the initial analysis in order to gather information. It works by scanning the binary for ASCI characters. The results of the scan can give valuable clues about the capabilities of the code. [1]

Filemon v6.07: This is an application that monitors and displays all file system activity on a system in real-time. This tool is used when the code is first executed and can detail what the code opens, reads, writes and deletes as well as timestamp each activity. [2]

OLLYDBG 1.10: This program is a 32-bit assembler level analyzing debugger for Windows. This program is very useful for performing binary code analysis and will be used during code analysis. This is the 'microscope' for examining at the byte level the inner workings of the code on the infected system. Furthermore, this tool has the ability to modify or tweak the code as well as stepping through the complicated procedures. [3]

Regmon v6.06: This is a registry-monitoring tool and logs which applications are accessing the registry as well as which keys are being used. This program will be used when analyzing the behavior of the code on the infected system and shows how it interacts with the registry. [4]

Regshot v1.61e5 final: This utility allows the user to take 'before' and 'after' snapshots of the registry and specified folders. The results are then compared to detect differences between the two. Furthermore, the program can be configured

to monitor file creation within key system folders. The program is used while studying the behavior of the code on the infected system and answers the questions “What did this code do in the registry?” and “What file or folder did this code create?”. [5]

TDIMon v1.0: This is an application that allows TCP and UDP activity to be monitored on the local system. If the code performs any network activity, this tool will log and timestamp details of the code’s activity. This will be used on the infected virtual host during behavior analysis. [6]

Md5sum.exe: This is a freeware program used to take the fingerprint of a file by calculating its md5 checksum. This is an open source, command-line tool and will be used to verify that the code has not been modified in any way. It will be used again later in the analysis process to verify that the code has not changed or morphed itself. [7]

Below are the tools that I will be using with the analysis which are installed on the Virtual RedHat Linux system:

Netcat: This program is a feature-rich networking utility that can accept and forward traffic on designated ports, tunnel traffic, as well as to conduct remote scanning. This tool will be used during behavior analysis to listen on ports on the virtual Linux system. Doing so will aid the analysis by capturing data that may be sent from the infected host. [8]

SNORT: This program is a highly configurable intrusion detection system and will be used when analyzing the code’s behavior. Snort works by capturing all traffic on a network and will display and or save that data. It can be further configured to examine network traffic and alert whenever it detects a pattern recognized by one of Snort’s rules. [9]

.

PROPERTIES OF MALWARE SPECIMEN

The file that I will analyze was obtained from SANS and was zipped in a file called ‘msrll.zip’. Once unzipped, the file was an executable with an ‘.exe’ extension named ‘msrll.exe’. The file size is 41,984 bytes.

The next step to take is to calculate the MD5 hash of the file. The purpose of the hash is to detect if the file changes or modifies itself. If the MD5 hash remains the same, then we know that code has not changed. To get the MD5 hash, I ran this command at the command prompt on the Virtual Host:

```
C:\>md5sum msrll.exe     ← command
84acfe96a98590813413122c12c11aaa *msrll.exe   ← results
```

Next, I used the BinText program to extract strings from the file using the default settings. To use BinText, double click on the program, then browse to the file and click ‘Open’. Next, we will leave the ‘Filter’ tab at its default settings and simply click go. Below I’ve included the notable strings minus the chaff.

File pos	Mem pos	ID	Text
0000004D	0040004D	0	!This program cannot be run in DOS mode.
00000178	00400178	0	.text
000001A0	004001A0	0	.data
000001F0	004001F0	0	.idata
00000218	00400218	0	.aspack
00000240	00400240	0	.adata
00009271	0051D071	0	VirtualAlloc
0000927E	0051D07E	0	VirtualFree
00009641	0051D441	0	kernel32.dll
0000964E	0051D44E	0	ExitProcess
0000965A	0051D45A	0	user32.dll
00009665	0051D465	0	MessageBoxA
00009671	0051D471	0	wsprintfA
0000967B	0051D47B	0	LOADER ERROR
00009688	0051D488	0	The procedure entry point %s could not be located in the dynamic link library %s
000096D9	0051D4D9	0	The ordinal %u could not be located in the dynamic link library %s
0000A16C	0051DF6C	0	kernel32.dll
0000A17B	0051DF7B	0	GetProcAddress
0000A18C	0051DF8C	0	GetModuleHandleA
0000A19F	0051DF9F	0	LoadLibraryA
0000A274	0051E074	0	advapi32.dll
0000A281	0051E081	0	msvcrt.dll
0000A28C	0051E08C	0	msvcrt.dll
0000A297	0051E097	0	shell32.dll
0000A2A3	0051E0A3	0	user32.dll
0000A2AE	0051E0AE	0	version.dll
0000A2BA	0051E0BA	0	wininet.dll
0000A2C6	0051E0C6	0	ws2_32.dll
0000A313	0051E113	0	AdjustTokenPrivileges
0000A32B	0051E12B	0	_itoa
0000A333	0051E133	0	__getmainargs
0000A343	0051E143	0	ShellExecuteA
0000A353	0051E153	0	DispatchMessageA
0000A366	0051E166	0	GetFileVersionInfoA
0000A37C	0051E17C	0	InternetCloseHandle
0000A392	0051E192	0	WSAGetLastError

Looking at the results of the BinText results, it appears that the code runs on the windows platform. Notice that there are several references to .dll files that only are used on the windows platform such as “user32.dll” [10] and “shell32.dll” [11].

The code appears to use Aspack for packing since it is referenced in line 00000218. Aspack is a win32 executable file compressor and will work on files that will be used on Windows platforms.[12] The possibility that this code uses

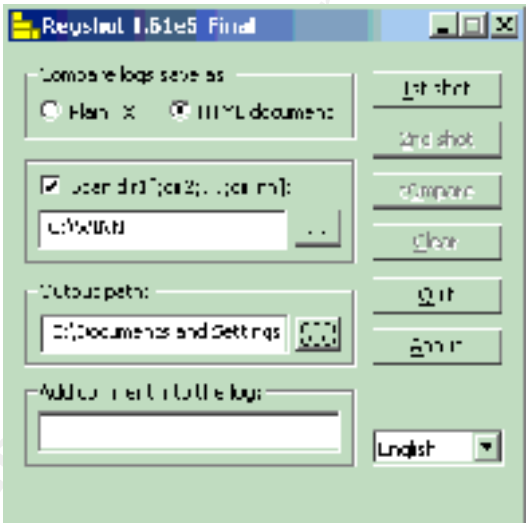
Aspack is further affirmation that this code is designed to run on Windows platforms.

The references at lines 0000A281 and 0000A28C indicates how the code was written. 'Msvcrt.dll' is a runtime library used in C++ programming, indicative that the code was written in the C programming language. [13]

BEHAVIORAL ANALYSIS

Monitoring file system access

The first step that I take is to make a snapshot of the virtual Windows 2000 system using the VMware program. That way, I have a clean working state that I can revert back to in order to undo what the malicious code has done. The next step I took was to make a snapshot of the registry as well as the 'C:\WINNT' and 'C:\' directories using the Regshot tool. This is the 'before' picture and it is done immediately before executing the code on the Virtual Host. To do this, double-click Regshot, enter "C:\WinNT" into the "Scan dir" text box, and then browse and setup the "Output path". I prefer to view the results as a HTML document and have selected the appropriate radio button. Finally, click the "1st shot" button to take the before picture.



Now that the environment has been set, I executed the code and let it run for about 30 seconds. The first thing that I noticed is that once I executed the code, the 'msrll.exe' file disappeared from my desktop where it was saved. The second thing that I noticed is that the code runs in the Process section of the Task Manager under the same name.

I stopped the code in the task manager window and performed the second shot of the registry and system folders by clicking the "2nd shot" button in Regshot. Next I compared the files by clicking the "compare" button and exported the results as 'msrllRegShot.htm' and saved the file.

The results showed that there were a total of 67 total changes; however, for the sake of brevity I will note the more significant changes.

Values added:20

Code created a new service called “Rll enhanced drive” and associated it with the “msrll.exe”. This service will be started when the system is started.
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ImagePath:
"C:\WINNT\System32\mfm\msrll.exe"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ObjectName: "LocalSystem"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\ImagePath:
"C:\WINNT\System32\mfm\msrll.exe"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\ObjectName: "LocalSystem"

Values modified:32

Files added:2

C:\WINNT\system32\mfm\jtram.conf ← Code created a new file
C:\WINNT\system32\mfm\msrll.exe ← Code copied itself to new folder

Files [attributes?] modified:4

C:\WINNT\system32\config\software
C:\WINNT\system32\config\software.LOG
C:\WINNT\system32\config\system
C:\WINNT\system32\config\SYSTEM.ALT

Folders added:3

C:\WINNT\system32\mfm ← Code created a new folder
C:\WINNT\system32\mfm\.
C:\WINNT\system32\mfm\..

Total changes:67

I noticed that the code copied itself into the newly created ‘C:\WINNT\system32\mfm’ folder. The first question that I have is if whether or not the new copy is an exact replica or did it morph in any way. The second question is what exactly is in the ‘jtram.conf’ file.

I quickly ran the test again, ensuring first that I reverted back to the clean state of the laboratory by pressing the “Revert” button of the VMware console. Once again, when executed the code disappeared from the desktop and was moved into the ‘C:\WINNT\system32\mfm’ folder. Using the md5sum utility, I verified that the file is indeed an exact copy by comparing the md5 checksum.

Next, I took a peek at the ‘jtram.conf’ using notepad and it appears that the contents are encrypted and are not in plain text. Nothing stood out as being significant that would indicate what was in the file. Here is a small sample:

```
sf8RAGx3zDX3mu8iQ7vwtIdpeEqNAjka5x/ZzadmtYW1hH1A+w==
```

In order to monitor file access by the code, I prepared the laboratory environment again by reverting to the clean snapshot of the virtual machine and starting the Filemon program. Once the program was running, I cleared the logs and immediately double-clicked on the code. I allowed the code to run for 30

seconds and then paused the logging feature of Filemon. A look at the logs reveals the following details:

The code accesses the following files on the system: usp10.dll, ws2_32.dll, and WS2HELP.DLL. Then it creates the folder 'c:\winnt\system32\mfm' and copies itself to it. Information is then updated in the software.log and it then accesses shell32.dll. The code then opens clbcatq.dll, cscui.dll, cscdll.dll, msi.dll, reopens msrll.exe in the new location, opens libssl32.dll, jtram.conf, msafd.dll, wshtcpip.dll, index.dat, rasman.dll, tapi32.dll, rtutils.dll, sensapi.dll, rsabase.dll, serenv.dll, crypt32.dll, and msasn1.dll.

The file jtram.conf is opened and written to several times. I also noticed that the code several times tries to open 'c:\dev\random' that doesn't exist. That type of file normally appears on a UNIX based system and is a random number generator that can be used for cryptographic purposes. I have included a snippet of the results from Filemon.

```
89  1:20:05 PM  msrll.exe:464  CREATE C:\WINNT\System32\mfm  SUCCESS
Options: Create Directory  Access: All
129  1:20:05 PM  msrll.exe:464  WRITE  C:\WINNT\System32\mfm\msrll.exe
SUCCESS  Offset: 0 Length: 41984
459  1:20:22 PM  msrll.exe:116  CREATE  C:\WINNT\system32\mfm\jtram.conf
SUCCESS    Options: OverwriteIf  Access: All
622  1:20:22 PM  msrll.exe:116  WRITE  C:\WINNT\system32\mfm\jtram.conf
SUCCESS    Offset: 0 Length: 53
633  1:20:22 PM  msrll.exe:116  WRITE  C:\WINNT\system32\mfm\jtram.conf
SUCCESS  Offset: 53 Length: 53
648  1:20:22 PM  msrll.exe:116  WRITE C:\WINNT\system32\mfm\jtram.conf
SUCCESS  Offset: 106 Length: 53
```

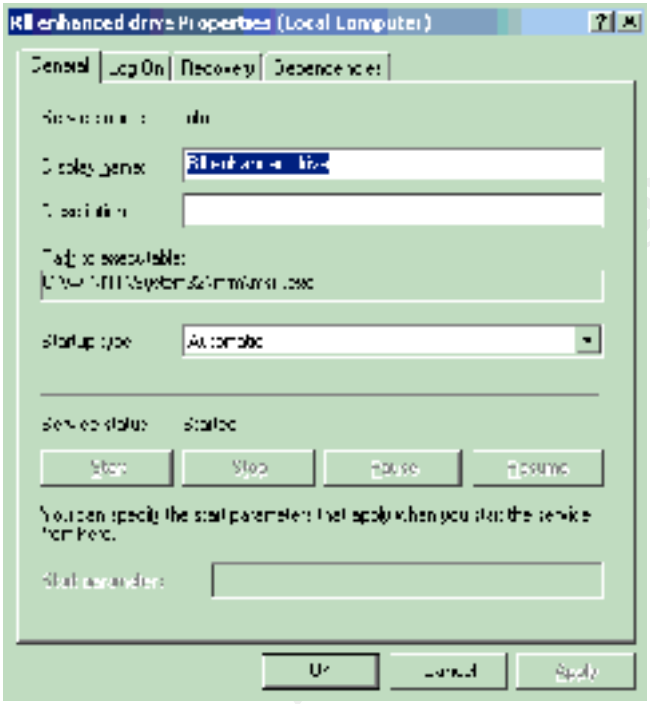
Monitoring registry access

To monitor access to the registry, I reverted back to the original configuration of the Windows 2000 virtual machine using VMware's 'revert' command. Next I prepared the Regmon program by starting it and clearing (CTRL + x) the logs. I immediately ran the executable code and let it run for 30 seconds. The results of the regmon.log file are as follows:

The regmon.log confirms that that the code added several keys which confirmed the results of the Regshot program. The code added a key to the registry that allowed the Trojan to restart when the computer boots up as a service called "Rll enhanced drive".

```
879  4.10308768  services.exe:212  SetValue
HKLM\System\CurrentControlSet\Services\mfm\ImagePath   SUCCESS
        "C:\WINNT\System32\mfm\msrll.exe"
880  4.10317568  services.exe:212  SetValue
HKLM\System\CurrentControlSet\Services\mfm\DisplayName  SUCCESS  "Rll enhanced drive"
```

I rebooted the virtual machine and confirmed that a new service with that name was running by checking the “Services” list in “Computer Management”. Also noted was that once the system was rebooted, the “Rll enhanced drive” service is automatically restarted. It also locks the msrll.exe file as it is run to keep it from being manually shut down.



Monitoring network connections

I ran the test again in order to determine what this code does with regards to the virtual network, however this time before I did it I performed the “netstat –an” command and took note of the ports that were open. After the test, I ran the command again and noted that there was a difference. These lines were added to the results.

```
TCP    0.0.0.0:113      0.0.0.0:0        LISTENING
TCP    0.0.0.0:2200     0.0.0.0:0        LISTENING
```

It looks like the code is now listening on TCP ports 113 and 2200. TCP Port 113 is the identity authentication server that would probably be used by the system to identify connecting systems with a service called “IDENT”. According to RFC 1413 [14], this protocol is used to determine the identity of a user on a particular TCP connection.

It is unknown what port 2200 is at this time, but it is indicative that the code opened up a backdoor on the system.

I also captured the traffic from the virtual Linux system by logging in as root and starting snort to capture the traffic using the following command:

```
snort –vd | tee /tmp/sniff.log
```

This command says to run the snort command in verbose mode (-v) and dump (-d) the application layer data and pipe it to a file in the 'tmp' directory named 'sniff.log'.

Examining this log, I can see that the infected host (192.168.234.128) began immediately querying the DNS server for the IP address of host 'collective7.zxy0.com'. There was a reply from the virtual server (192.168.234.1) that there was no information for that hostname.

```
11/17-02:50:01.037020 192.168.234.128:1067 -> 192.168.234.1:53
UDP TTL:128 TOS:0x0 ID:699 IpLen:20 DgmLen:66
Len: 38
00 01 01 00 00 01 00 00 00 00 00 00 0B 63 6F 6C  .............col
6C 65 63 74 69 76 65 37 04 7A 78 79 30 03 63 6F  lective7.zxy0.co
6D 00 00 01 00 01                                m.....
```

Using the TDIMon tool, I confirmed that these activities took place and further confirmed that it was "msrll.exe" as opposed to another service that sent this traffic. To use the TDImon program, double click TDImon shortcut. You can filter (CTRL+L) what traffic that you want to catch. When ready to begin the capture, clear (CTRL+x) the log and start the capture (CTRL+ e).

```
9     0.83396214  msrll.exe:920813E0DE8  TDI_SET_EVENT_HANDLER
        TCP:0.0.0.0:2200
110 3.07633540  services.exe:21281530328   TDI_SEND_DATAGRAM
        UDP:0.0.0.0:1062 192.168.234.1:53
112  3.07696062  services.exe:21281530328  TDI_SEND_DATAGRAM
        UDP:0.0.0.0:1062  192.168.234.1:53
129  3.14521259msrll.exe:920  81534E48  TDI_SET_EVENT_HANDLER
        TCP:0.0.0.0:113
```

I reverted back to the clean state and then added to the hosts file on the virtual host machine an entry to redirect all traffic for 'collective7.zxy0.com' to the Linux system, which is 192.168.234.130. I then pinged the hostname to make sure that it resolved to the Linux box.

```
192.168.234.130 collective7.zxy0.com   ← added to the end of "hosts" file
```

The hosts file is located at c:\winnt\system32\drivers\etc. It is a file that was used extensively before DNS to resolve IP addresses with domain names. This worked until the number of domains grew to the point that the hosts file became too cumbersome to maintain giving rise to DNS servers. On small networks, a hosts file can be useful for mapping IP addresses to specific systems. [15]

I ran the code again and monitored it using snort on the Linux machine, running the same command as before. This time there was a new activity. The code began trying to connect to the Linux machine on port 6667. This is indicative that the code is looking for an IRC server[16]. IRC stands for Internet Relay Chat and is highly popular in the hacker community as a chatting tool that allows users to chat in various channels. [17]

I also detected a couple of interesting packets that should be noted. Not only was the code trying to connect to port 6667 on the Linux machine, it also was attempting to connect on TCP ports 8080 and 9999.

```
11/25-23:19:37.819353 192.168.234.128:1025 -> 192.168.234.130:6667
TCP TTL:128 TOS:0x0 ID:15 IpLen:20 DgmLen:48 DF
******S* Seq: 0x44A3CBDB  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

11/26-02:19:28.816895 192.168.234.128:1064 -> 192.168.234.130:8080
TCP TTL:128 TOS:0x0 ID:697 IpLen:20 DgmLen:48 DF
******S* Seq: 0x15AE1B92  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

11/26-02:18:07.964892 192.168.234.128:1063 -> 192.168.234.130:9999
TCP TTL:128 TOS:0x0 ID:691 IpLen:20 DgmLen:48 DF
******S* Seq: 0x14D4FA13  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

Once again, I used VMware's useful 'revert' feature to return it to a clean state. On the Linux virtual system, I started the IRC server using the following commands:

```
[root@localhost root ]# su  - ircd
[ircd@localhost ircd]$ ./ircd
[ircd@localhost ircd]$ exit
[root@localhost root ]# ps –u ircd
  PID TTY          TIME CMD
1655 ?        00:00:00  ircd
[root@localhost root ]#  irc
```

The 'su' command is to run a shell with substitute user and group IDs. The 'ps' command is to list the process status and of course, 'irc' is to run the IRC server.

Next, in order to find out why the code was attempting to connect to ports 8080 and 9999, I started 2 more instances of the virtual Linux system using the key commands '[ALT] – F2' and '[ALT] – F3' respectfully.

On the second Linux console, I setup Netcat to listen on port 8080 and output the results to the screen using the following command:

```
nc –p 8080 –l –n
```

This command says to use the Netcat program (nc) to listen (-l) on port (-p) 8080 and to not (-n) resolve the hostnames.

On the third Linux console, I setup Netcat to listen on port 9999 and output the results to the screen using the following command:

```
nc –p 9999 –l –n
```

I started a fourth Linux console ([ALT] – F4) and set the system to run Snort using the same command that I described earlier.

Now I'm ready to run the test again. I modified the host file on the Windows 2000 virtual system to redirect traffic destined for 'collective7.zxy0.com' to the virtual Linux system. I double-clicked 'msrll.exe' to run the code, however this time I will allow the code to continue running.

I noticed immediately that the code began its work. Watching the snort output, I saw that the code joined the IRC server in the '#mils' channel using the name "CIlfgVjmO". Because of the amount of traffic involved in joining an IRC channel, for the sake of brevity, I am only including a small portion of the snort logs that shows the infected host has logged on:

```
11/30-06:31:38.275148 192.168.234.130:6667 -> 192.168.234.128:1065
TCP TTL:64 TOS:0x0 ID:5035 IpLen:20 DgmLen:212 DF
***AP*** Seq: 0xA9E17423  Ack: 0x16B6F872  Win: 0x16D0  TcpLen: 20
3A 43 49 6C 66 67 56 6A 6D 4F 21 77 6D 63 40 31     :CIlfgVjmO!wmc@1
39 32 2E 31 36 38 2E 32 33 34 2E 31 32 38 20 4A     92.168.234.128 J
4F 49 4E 20 3A 23 6D 69 6C 73 0D 0A 3A 6C 6F 63     OIN :#mils..:loc
61 6C 68 6F 73 74 2E 6C 6F 63 61 6C 64 6F 6D 61     alhost.localdoma
69 6E 20 33 35 33 20 43 49 6C 66 67 56 6A 6D 4F     in 353 CIlfgVjmO
20 3D 20 23 6D 69 6C 73 20 3A 43 49 6C 66 67 56     = #mils :CIlfgV
6A 6D 4F 20 72 6F 6F 74 20 0D 0A 3A 6C 6F 63 61     jmO root ..:loca
6C 68 6F 73 74 2E 6C 6F 63 61 6C 64 6F 6D 61 69     lhost.localdomai
6E 20 33 36 36 20 43 49 6C 66 67 56 6A 6D 4F 20     n 366 CIlfgVjmO
23 6D 69 6C 73 20 3A 45 6E 64 20 6F 66 20 2F 4E     #mils :End of /N
41 4D 45 53 20 6C 69 73 74 2E 0D 0A                 AMES list...
```

Netcat captured the following data from port 8080:

```
USER vewZOXgfPnWks localhost 0:gjhrRDAoHyTtDLbmAZWTOnJCEgilLJOjcWOjnXJTg
NICK uCvSxgIkKBO
```

Below is the log from snort:

```
11/30-06:29:32.580315 192.168.234.128:1064 -> 192.168.234.130:8080
TCP TTL:128 TOS:0x0 ID:708 IpLen:20 DgmLen:131 DF
***AP*** Seq: 0x1571CC5C  Ack: 0xA3F37A22  Win: 0x4470  TcpLen: 20
55 53 45 52 20 76 65 77 5A 4F 58 67 66 50 6E 57     USER vewZOXgfPnW
6B 73 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A     ks localhost 0 :
67 6A 68 72 52 44 41 6F 48 79 54 74 44 4C 62 6D     gjhrRDAoHyTtDLbm
41 5A 57 54 4F 6E 4A 43 45 67 69 6C 4C 4A 4F 6A     AZWTOnJCEgilLJOj
63 57 4F 6A 6E 58 4A 54 67 0A 4E 49 43 4B 20 75     cWOjnXJTg.NICK u
43 76 53 78 67 49 6B 4B 42 4F 0A                    CvSxgIkKBO.
```

And from port 9999:

```
USER cXrCTuqGWKQQ localhost 0 :OqrDQXDzxKfqMvdoOLWdiDPzTq
NICK LIppDdAniYir
```

Below is the log from snort:

```
11/30-06:28:30.585884 192.168.234.128:1063 -> 192.168.234.130:9999
TCP TTL:128 TOS:0x0 ID:697 IpLen:20 DgmLen:116 DF
***AP*** Seq: 0x1498C77B  Ack: 0x9FF634CC  Win: 0x4470  TcpLen: 20
55 53 45 52 20 63 58 72 43 54 75 71 47 57 4B 51       USER cXrCTuqGWKQ
51 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A 4F      Q localhost 0 :O
71 72 44 51 58 44 7A 78 4B 66 71 4D 76 64 6F 4F      qrDQXDzxKfqMvdoO
4C 57 64 69 44 50 7A 54 71 0A 4E 49 43 4B 20 4C      LWdiDPzTq.NICK L
49 70 70 44 64 41 6E 69 59 69 72 0A                  IppDdAniYir.
```

I quickly joined the #mils channel as root by using the command '/join' and performed the '/names' command. This command will list all the users who are in the room. The results were:

```
/join #mils          ← command to join the #mils chat room
*** root (~root@127.0.0.1) has joined channel #mils  ← root joins channel
*** #mils 1103717105     ← channel joined and timestamp
/names
Pub: #mils  ← name of public roomCilfgVjmO root  ← current users in room
```

Note: the timestamp is in Unix Time, which is the number of seconds since Jan 1, 1970 at 12:00 am. [18]

I tried various techniques to communicate with the code using various IRC commands, however I was unable to gain a response. The data that was sent to ports 9999 and 8080 appear to be more nicks and user names for the IRC channel.

I haven't forgotten how the code opened up an apparent backdoor on the host machine. I performed various attempts to connect to the infected system by using telnet and ftp commands from the Linux system, however I was unable to completely connect. As a last resort, I attempted to connect by using the netcat program and received an error.

```
[root@localhost root]#  nc –v 192.168.234.128 2200  ← command
192.168.234.128: inverse host lookup failed: Host name lookup failure
(UNKNOWN) [192.168.234.128] 2200 (?) open
```

The command is "nc" and it is to display connection information (-v) when connecting to 192.168.234.128 on port 2200. [19]

Lacking additional information, I am unable to further analyze the code's behavior.

CODE ANALYSIS

I used the BinText program earlier and detected that the code was packed with "Aspack". My curiosity got the better side of me and I decided to search and see

if I could locate an unpacker for this particular code. Using the Google search engine, I stumbled upon a website that contained a plethora of various unpacking utilities.[20] After some trial and error, I found that the utility called 'AspackDie 1.3d' unpacked 'msrll.exe' without any errors into a file called 'unpacked.ExE'. I used the following command:

```
C:\>AspackDie msrll.exe unpacked.ExE
```

Now that I had the code unpacked, I renamed the unpacked version to msrll.exe and replaced the original packed msrll.exe at c:\winnt\system32\mfm. I decided to use BinText again to see if I can harvest more information from the code. The results were astounding and I detected several interesting items that weren't noticed before. I saved the search results into a text file called 'strings.txt' for further review. There are 13 pages of strings, therefore I will only comment on the interesting snippets of information.

These items are modules used in Linux in installing, unloading, and listing modules. This code appears to be inserted into this code to provide Linux-like capabilities on the infected system.

```
00001326   00401326      0   ?insmod
0000132E   0040132E      0   ?rmmod
00001335   00401335      0   ?lsmod
```

This is further affirmation that the code is to use modules on the infected system.

```
00001399   00401399      0   %s: <mod name>
000013A8   004013A8      0   %s: mod list full
000013BA   004013BA      0   %s: err: %u
000013C6   004013C6      0   mod_init
000013CF   004013CF      0   mod_free
000013D8   004013D8      0   %s: cannot init %s
000013EB   004013EB      0   %s: %s loaded (%u)
000013FE   004013FE      0   %s: mod allready loaded
00001416   00401416      0   %s:%s err %u
000015B5   004015B5      0   %s:%s not found
000015C5   004015C5      0   %s: unloading %s
```

This indicates that this code could be used as a zombie. A zombie is a controlled computer that can be used to perform 'Distributed Denial of Service (DDoS) attacks. The '?ping' may be a reference to the "Ping of Death" attack. The '?jolt' may be a reference to a similar attack called the 'Jolt Attack'. The '?smurf' may be a reference to another popular attack called the 'Smurf Attack'. [21]

```
00002753   00402753      0   ?ping
00002763   00402763      0   ?smurf
0000276A   0040276A      0   ?jolt
```

This is interesting; this is exactly what the code forwarded to the Linux machine on ports 9999 and 8080.

```
00003C20   00403C20      0   USER %s localhost 0 :%s
```

```
00003C38   00403C38      0   NICK %s
```

This is the first reference to the 'jtram.conf' filed that was detected being placed into the 'c:\winnt\mfm\' file on the infected host.

```
000069EB   004069EB      0   jtram.conf
```

This appears to be the part of the code that contains the IRC bot. "mIRC" is a popular IRC tool that is freely available. [22]

```
000074C9   004074C9      0   mIRC v6.12 Khaled Mardam-Bey
```

These appear to be the commands that can be used to control the bot. I notice that some of these commands such as '?clone' and '?!fif' don't appear to be typical 'mIRC' commands. These could be commands created by the programmer to provide added functionality to the bot. The '?exec' command could be used to remotely execute files on the infected system.

```
0000934E   0040934E      0   ?clone
00009355   00409355      0   ?clones
0000935D   0040935D      0   ?login
00009364   00409364      0   ?uptime
0000936C   0040936C      0   ?reboot
00009374   00409374      0   ?status
0000937C   0040937C      0   ?jump
00009382   00409382      0   ?nick
00009388   00409388      0   ?echo
0000938E   0040938E      0   ?hush
00009394   00409394      0   ?wget
0000939A   0040939A      0   ?join
000093A9   004093A9      0   ?akick
000093B0   004093B0      0   ?part
000093B6   004093B6      0   ?dump
000093C6   004093C6      0   ?md5p
000093CC   004093CC      0   ?free
000093D7   004093D7      0   ?update
000093DF   004093DF      0   ?hostname
000093EE   004093EE      0   ?!fif
000093FE   004093FE      0   ?play
00009404   00409404      0   ?copy
0000940A   0040940A      0   ?move
00009415   00409415      0   ?sums
00009423   00409423      0   ?rmdir
0000942A   0040942A      0   ?mkdir
00009436   00409436      0   ?exec
00009440   00409440      0   ?kill
00009446   00409446      0   ?killall
0000944F   0040944F      0   ?crash
0000946E   0040946E      0   ?sklist
00009476   00409476      0   ?unset
```

```
0000947D   0040947D      0   ?uattr
00009484   00409484      0   ?dccsk
00009490   00409490      0   ?killsk
```

This is another reference to the 'jtram.conf' file and it appears that this is the part of the code that is used when writing to the file.

```
000099E0   004099E0      0   jtram.conf
000099EB   004099EB      0   jtr.*
000099F5   004099F5      0   DiCHFc2ioiVmb3cb4zZ7zWZH1oM=
00009A16   00409A16      0   conf_dump: wrote %u lines
```

This is interesting. The 'bot.port' could be referencing the backdoor that was established on TCP port 2200 on the infected system. Notice that 'bad pass' could be the code that is used when a bad password from a remote system is used when attempting to connect. If I was able to at least discover the proper logon procedure to the backdoor, it may be possible to reverse engineer and discover the correct password using the Ollydbg utility.

```
0000BB40   0040BB40      0   dcc.pass
0000BB49   0040BB49      0   bot.port
0000BB52   0040BB52      0   %s bad pass from "%s"@%s
0000BCC9   0040BCC9      0   %s: connect from %s
```

Now it is easy to see that this is the section of the code that determines what servers this particular bot will attempt to connect to. We can also see that that it will try to connect to ports 9999 and 8080. I do notice, however that there is an exclamation point after '9999'.

```
0000BD6E   0040BD6E      0   servers
0000BD80   0040BD80      0
collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.zxy0.com:8080
0000BDCA   0040BDCA      0   irc.chan
0000BDD3   0040BDD3      0   #mils
```

This may be the section of code that may be the logon and quitting procedures for the backdoor. I attempted the logon several times with this format and was unable to get a response.

```
0000C5BA   0040C5BA      0   %s : USERID : UNIX : %s
0000C6A4   0040C6A4      0   QUIT :FUCK %u
0000C742   0040C742      0   Killed!? Arrg! [%u]
0000C756   0040C756      0   QUIT :%s
```

This is the name of the service that starts up automatically when the infected system reboots.

```
0000C897   0040C897      0   RII enhanced drive
```

Finally, there are references to various types of hashing mechanisms, ciphers and encryption tools. I will only include a few here:

```
0000F140   0040F140      0       md5.c
```

0000FDBC	0040FDBC	0	rc6.c
0001114B	0041114B	0	Blowfish
00011157	00411157	0	RC2
0001115E	0041115E	0	RC5
00011165	00411165	0	RC6
0001116C	0041116C	0	Serpent
000111D0	004111D0	0	SHA-512
000111DB	004111DB	0	SHA-384
000111E6	004111E6	0	SHA-256
000111F1	004111F1	0	TIGER
0001124A	0041124A	0	Yarrow
00011254	00411254	0	SPRNG

Now is the time to use Ollydbg to show the internal workings of the code as it is run. To begin the process, I start the Ollydbg program. Next, I need to open the file [F3], browse to the executable code and select "Open". The code is now exposed and is possible to examine how it works.

The first thing that I want to examine is what is in the jtram.conf. The file is referenced on line 004099E0; however, a few lines after on lines 004099F5 and 00409A05 these ASCII characters are referenced:

DiCHFc2ioiVmb3cb4zZ7zWZH1oM=

This may be an encryption key to what is put into the jtram.conf. To confirm, I need to bypass this line as the code writes to jtram.conf. In the main window, I browse to the location using the scroll bar and locate the reference at line 00409A05. Next I right-click on the line select "Binary" and then "Fill with 00's". Hopefully when the code attempts to encrypt the contents for the jtram.conf file, the encryption will not work because the key will be nothing but 00's.

The next step is to delete the jtram.conf file in c:\winnt\system32\mfm folder and then restart the code [CTRL + F2] and then selecting "Yes". A few moments later, I noticed a fresh copy of jtram.conf suddenly appears in the c:\winnt\system32\mfm folder. I open it up with notepad and found several references to "collective7.zxy0.com". Furthermore, I detected that that as long as the code is running, jtram.conf is rewritten at least once an hour.

Next, I detected the reference in the code that specifies the port that is to be opened as a backdoor. I searched [Ctrl + B] for the binary ASCII string "2200" and landed on 0040BD52. To verify that this is the port, I changed the string to say "2201" instead. Right click on the line → Binary → edit. Then change the ASCII to 2201. I then stopped and restarted the code [CTRL-F2]. Moments later, I ran the netstat –an on the commandline of the infected host and saw that indeed a backdoor was now open on port 2201.

ANALYSIS WRAP UP

Codes capability

This code appears to be a bot that once executed, installs an IRC client that connects to a predefined IRC server. The code then opens a backdoor on TCP port 2200 that can allow a malicious user with the password to connect.

Once installed and connected to a remote IRC server, a malicious user can control the infected system as a type of 'zombie'. In the IRC realm, zombies are considered as property by hackers and can be the platforms in which Distributed Denial of Service (DDoS) attacks can be launched. The fact that there were strings associated with DDOS attacks (smurf, jolt, ping) in the code, it is most likely that this code fits within that category.

Defensive measures

There are several defensive measures that can be used to avoid getting this bot in the first place.

1. Always utilize an antivirus program, especially with systems that have access to the Internet. It is important to note, that antivirus programs should be kept up to date with the latest antivirus definitions. Simply installing the program will alone not do the trick as to numerous viruses and malicious codes are discovered everyday.
2. Be very cautious when downloading a file from an unknown source. Many seemingly harmless programs may contain a Trojan horse.
3. It is imperative to utilize firewalls. A properly configured firewall could have kept this bot from opening a backdoor or at the very least, alerted the system user. A properly configured firewall could have also kept the infected system from connecting out to a remote IRC server on TCP port 6667.

Use a Snort rule to detect the code

```
Alert tcp $HOME_NET any -> any 6667 (msg:"Possible IRC access (JOIN)"; flow:to_server,established; content:"JOIN"; classtype:misc-attack; sid:1000001; rev:7; tag:session,30,seconds;)
```

To explain how this rule works goes beyond the scope of this analysis. At its basics this rule is looking for TCP traffic in either direction to port 6667 with the keyword "JOIN". Any network traffic that meets these criteria could either be legitimate IRC traffic, if IRC programs are allowed on the network, or it could be an IRC bot. There is no guarantee that IRC Trojans and bots would use TCP port 6667, but in this analysis, it is clear that msrll.exe does. Chris Hanna has more information related to IRC bots and snort rules in his practical for the GSEC. [23]

Manually remove the code from the system

1. Start the system in safe mode. [24]

2. Browse to the c:\winnt\system32\mfm and delete msrll.exe and jtram.conf.
3. Remove the "mfm" folder from c:\winnt\system32.
4. Remove the following values from the registry:
    a. Click Start → Run
    b. Type "regedit"
    c. Navigate to the following keys and perform the following actions:

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm
In the left pane, right click and delete the "mfm" folder.

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm
In the left pane, right click and delete the "mfm" folder.

REFERENCES

[1] Foundstone, Inc. Strategic Security. "Free Tools". 2003 – 2005. URL: http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/proddesc/bintext.htm

[2] Russinovich, Mark, and Bryce Cogswell. "Utilities". Sysinternals Freeware. December 2004. URL: http://www.sysinternals.com/ntw2k/utilities.shtml

[3] Yuschuk, Oleh. "OllyDgb v1.10". 2000 – 2004. URL: http://home.t-online.de/home/Ollydbg/.

[4] Russinovich, Mark, and Bryce Cogswell. "Utilities". Sysinternals Freeware. December 2004. URL: http://www.sysinternals.com/ntw2k/utilities.shtml

[5] Webattack. "RegShot download and review – monitor for registry changes from SnapFiles". 1997 – 2004. URL: http://www.snapfiles.com/get/regshot.html

[6] Russinovich, Mark, and Bryce Cogswell. "Utilities". Sysinternals Freeware. December 2004. URL: http://www.sysinternals.com/ntw2k/utilities.shtml

[7] Etree.org. "etree.org | md5sum". 1998 – 2002. URL: http://www.etree.org/md5com.html

[8] Giacobbi, Giovanni. "The GNU Netcat – Official homepage". 2004. URL: http://netcat.sourceforge.net/

[9] Caswell, Brian, and Marty Roesch. "Snort.org". Sourcefire, Inc. 2005. URL: http://www.snort.org/

[10] "user32 – user32.dll – DLL Information". Uniblue Systems. 2000 – 2004. URL: http://www.liutilities.com/products/wintaskspro/dlllibrary/user32/

[11] "shell32 – shell32.dll – DLL Information". Uniblue Systems. 2000 – 2004. URL: http://www.liutilities.com/products/wintaskspro/dlllibrary/shell32/

[12] "Aspack Software – Best Choice Compression and Protection Tools for Software Developers". Aspack Software. 2004. URL: http://www.aspack.com/

[13] "The DLL zone – MSVCRT.DLL". The DLL Zone. 1998. URL: http://www.fortunecity.com/skyscraper/fortune/570/msvcrt.html

[14] St. Johns, M. "RFC 1413 (rfc1413) – Identification Protocol". US Department of Defense. February 1993. URL: http://www.faqs.org/rfcs/rfc1413.html

[15] "Simple DNS Plus". JH Software. 1999 – 2004. URL: http://www.jhsoft.com/help/index.html?df_hostsfile.htm

[16] http://www.iana.org/assignments/port-numbers. Internet Corporation for Assigned Names and Numbers. 2004. URL: http://www.iana.org/assignments/port-numbers

[17] Oikarinen, J, and D. Reed. "RFC1459, Internet Relay Chat Protocol". May 1993. URL: http://rfc.net/rfc1459.html

[18] Fogt, Robert. "Online Conversion – Unix time conversion". OnlineConversion.Com. 1997 – 2002. URL: http://www.onlineconversion.com/unix_time.htm

[19] Armstrong, Tom. "Netcat – The TCP/IP Swiss Army Knife". February 2001. URL: http://m.nu/program/util/netcat/netcat.html

[20] "Aaron's Homepage". URL: http://www.exetools.com/unpackers.htm

[21] "Denial of Service or Nuke Attacks". IRCHELP.ORG. 2004. URL: http://www.irchelp.org/irchelp/nuke/info.html.

[22] "Vonck, Tjerk. "mIRC – An Internet Relay Chat program". MIRC Co. 1995 – 2004. URL: www.mirc.com.

[23] Hanna, Christopher W. "Using Snort to Detect Rogue IRC Bot Programs". GSEC Practical Version 1.4c. October 8, 2004. URL: http://www.giac.org/practical/GSEC/Chris_Hanna_GSEC.pdf

[24] "Starting your computer in Safe mode". Symantec. November 8, 2004. URL: http://service1.symantec.com/SUPPORT/tsgeninfo.nsf/docid/2001052409420406?OpenDocument&src=sec_doc_nam