Fight crime.
Unravel incidents... one byte at a time.

Copyright SANS Institute
Author Retains Full Rights

Interested in learning more?
Check out the list of upcoming events offering
"Advanced Incident Response, Threat Hunting, and Digital Forensics (FOR508)
GIAC Reverse Engineering Malware (GREM)
Practical Assignment
Version 1.0

Malware: msrl.exe

ILOT XII
James M. Balcik
12/8/2004
# Table of Contents

**Abstract**  
4

**Laboratory Setup**  
5

**Hardware**  
5

**Networking**  
5

**Software**  
5

*Windows XP SP1 Software Tools:*
- **Ghost**  
- **md5sum**  
- **BinText**  
- **Regshot**  
- **FileMon**  
- **RegMon**  
- **TDIMon**  
- **Process Explorer**  
- **AutoRuns**  
- **UltraEdit-32**  
- **nc (Netcat)**  
- **PESniffer**  
- **PEInfo**  
- **ASPACKDIE**  
- **IDA Pro**  
- **OllyDbg**

*Red Hat 9 Software:*
- **SNORT**
- **ircd**
- **irc**

**Properties of the Malware Specimen**  
11

*Operating Systems Effected:*
*Strings Embedded in File:*
- **Fig. A - Interesting strings packed msrll.exe**
- **Fig. B - Interesting strings unpacked msrll.exe**

**Behavioral Analysis**  
29

*Preparation for Infection*
- **Fig. 1 - Regshot**

*Infection*
- **Fig. 2 - Regshot Compare Results**
- **Fig. 3 - FileMon Results**
- **Fig. 4 - RegMon Results**
- **Fig. 5 - TDIMon Results**
- **Fig. 6 - Snort Capture 2**
- **Fig. 7 - Hosts File Modification**
- **Fig. 8 - Snort Capture 3**
- **Fig. 9 - Snort Capture 4**

**Code Analysis**  
38

*The Search for Authentication Code*
- **Fig. 10 - IDA Pro Text Search Results**
Abstract

The intent of this paper is to partially fulfill the requirements of the GREM certificate and to demonstrate my knowledge of “Reverse Engineering Malware”. This paper documents the tools and processes used to analyze the msrl.exe malware.
Laboratory Setup

Hardware

The laboratory hardware consists of an HP OmniBook XE3 laptop running with the following specs:

Intel Pentium III Processor
700 MHz
384MB Memory
10GB Hard Disk Drive
DVD/CD-ROM Drive
1.44MB 3.5" Floppy drive
10/100MB Integrated Network Interface
56Kbps Integrated Modem

Networking

The laboratory networking setup uses a VMware virtual network. The virtual network allows all network activity to be contained on the laptop between the virtual PC’s and the host system. To contain the malware fully the physical network interface on the laptop is not plugged in. The Vmware virtual network emulates a hub, which is convenient for sniffing network traffic.

Software

The key software in this laboratory is VMware Workstation 4.5.1. VMware allows you to run multiple operating systems on one physical PC by creating virtual PC’s that share the physical resources of the host system.

The laptop’s Microsoft Windows 2000 Server SP4 is the Host system that has VMware installed. There are 2 virtual PC’s, one running Microsoft Windows XP SP1 and one running Red Hat Linux 9. Each virtual PC has been preconfigured with software tools to analyze the malware. Also, each virtual PC is contained in a folder on the laptop which has been backed up using WinZip for later recovery of the base or clean system. This allows for quick restores to a clean state for each virtual PC. The host system has been imaged using Ghost to allow for complete system recovery of the host system and all virtual systems.

Windows XP SP1 Software Tools:
Ghost
Version: 7.5
By: Symantec
http://www.symantec.com

Description:
Symantec Ghost is a disk imaging software that can backup a entire disk to a image file for later recovery on that image. I used Ghost to image the entire laboratory laptop hard disk so that in the event of infection at the host level I could restore the entire system back to a clean state.

md5sum
Version: GNU textutils 2.0
By: Ulrich Drepper
http://www.gnu.org/software/textutils/textutils.html

Description:
md5sum will calculate the md5 hash of a file. Knowing the md5 hash of a file will allow you to do file comparisons to determine if the files are the exact same. If a file has changed even in the slightest way the md5 hash of the two files should not match therefore revealing that the file has been modified in some way. I used md5sum to do file comparisons on the different copies of the malware msrll.exe.

BinText
Version: 3.00
By: Foundstone Inc.
http://www.foundstone.com

Description:
BinText allows you to view the ASCII text, Unicode text, and resource strings contained in any file. By viewing the ASCII text, Unicode text, and resource strings in a binary file you can begin to get hints about its functionality, if it is packed or unpacked, and the memory addresses of interesting functions. BinText was used in my analysis of the different copies of msrll.exe malware file to gain hints on packing method, functionality, and memory addresses of certain interesting code.

Regshot
Version: 1.61e5 Final
By: TiANWEi
http://regshot.yeah.net
http://regshot.ist.md

Description:
Regshot allows you to take 2 snap shots of the registry on a system and compare them. When you compare the snap shots you will get a list displaying the keys and
values that have been added, deleted, and modified. You can save your snap shots for later comparison. This is useful when you want to figure out what changes a malware made to a system. Regshot was used to first take a snap shot of the system before infection with the msrll.exe malware. Another snap shot of the system was taken after the msrll.exe malware ran. The comparison shows all the changes to the registry keys and values. This helps to figure out what the malware did and what filtering to do in examination of other log files like those from FileMon and RegMon.

FileMon
Version: 6.12
By: Mark Russinovich and Bryce Cogswell of Sysinternals
http://www.sysinternals.com

Description:
FileMon monitors and displays file system changes. You can save the logged changes to a file for later review. This is useful in finding detailed file system access during a specific period of time like during infection with the malware. A key area it shows is attempts not just successful file access. Sometimes the errors are more revealing than the successful entries. FileMon was used to record all file access during the initial infection of the system with the msrll.exe malware.

RegMon
Version: 6.12
By: Mark Russinovich and Bryce Cogswell of Sysinternals
http://www.sysinternals.com

Description:
RegMon will monitor all registry activity and display it on screen. You can save the log to a file for later review. RegMon will show what programs are accessing the registry and what registry keys and values they are reading or writing. RegMon was used to monitor the registry while infecting the system with the msrll.exe malware.

TDIMon
Version: 1.0
By: Mark Russinovich of Sysinternals
http://www.sysinternals.com

Description:
TDIMon is used to monitor TCP and UDP traffic on the system. This can help with monitoring what the malware does with network communications. An example would be opening a port on the system to listen for connections. TDIMon was used to monitor TCP and UDP traffic during the infection of the system with the msrll.exe malware.
Process Explorer
Version: 8.52
By: Mark Russinovich of Sysinternals
http://www.sysinternals.com

Description:
Process Explorer displays processes that are running on the local system along with their PID, description, and company name if any related to the process. It also shows in the lower window pane open handles or dlls depending on what mode it is in. Process Explorer allows you to drill down on each process listed by double clicking on the process. This reveals a great deal of information about the running process like TCP/IP connections or the path to the program that created the process. Process Explorer was used to monitor process the msrll.exe malware created and to end them during certain points of analysis.

AutoRuns
Version: 5.01
By: Mark Russinovich and Bryce Cogswell of Sysinternals
http://www.sysinternals.com

Description:
AutoRuns shows all the registry entries that are running programs during startup of the system. This is a common way for malware to auto start on a system. AutoRuns was used to check for msrll.exe changes to the auto starting entries in the registry.

UltraEdit-32
Version: 10.10a
By: IDM Computer Solutions Inc.
http://www.ultraedit.com

Description:
UltraEdit-32 can edit text, html, hex, and program source code. UltraEdit-32 was used to view files in hex mode and to view saved log files to search using it’s advance search features.

nc (Netcat)
Version: 1.10
By: Hobbit
http://www.securityfocus.com/tools/139/scoreit
http://www.securityfocus.com/tools/137
http://netcat.sourceforge.net/

Description:
Netcat or nc is often called the network Swiss army knife because there are many uses for this tool. Netcat was used to transfer snort log files from the Red Hat 9 virtual
system to the Windows XP SP1 virtual system so that they could be viewed in Ultradeit-32 to allow for my preferred method of search and examining the file. Netcat was also used to setup listeners on the Linux system to capture any requests to certain ports from the infected system.

**PESniffer**
Version: 1.06  
By: SkymarShall/CST  
[http://start.at/skymarshall](http://start.at/skymarshall) (Not Active)

Description:  
PE-Sniffer can scan a file for various packed executable encodings like ASPack. PE-Sniffer was used to scan msrl.exe malware for the packed executable encodings.

**PEInfo**
Version: unknown  
By: Tom Liston  
Not available to public. This tool was obtained from the SANS Reverse Engineering Malware instructor lead on-line training cd-rom.

Description:  
PEInfo allows you to see the packed executable structure. By viewing the structure details you maybe given hints as to what method was used to pack the executable. PEInfo was used to figure out what packing method was used on msrl.exe malware.

**ASPACKDIE**
Version: 1.41  
By: y0da  
[http://y0da.cjb.net](http://y0da.cjb.net)

Description:  
ASPACKDIE is an ASPACK packed executable unpacker. ASPACKDIE was used to unpack the msrl.exe malware.

**IDA Pro**
Version: 4.6  
By: DataRescue  
[http://www.datarescue.com](http://www.datarescue.com)

Description:  
IDA Pro is a disassembler and debugger. I used IDA Pro to sift through the msrl.exe disassembled code in search for clues of its functionality. I especially liked the flowchart feature which help me find the different decisions branches in the code.
OllyDbg
Version: 1.10
By: Oleh Yuschuk
http://home.t-online.de/home/ollydbg

Description:
OllyDbg is a 32-bit debugger that runs on Windows. OllyDbg was used to analyze msrll.exe disassembled code while running msrll.exe within OllyDbg. This allowed me to set break points at key areas in the code to further understand the functioning of the malware. OllyDbg was also used to patch msrll.exe so that it didn’t require a proper password to authenticate.

Red Hat 9 Software:

SNORT
Version: 2.0.4
By: Martin Roesch
http://www.snort.org

Description:
SNORT is a network sniffer and an intrusion detection system or IDS. It is used here as a network sniffer to capture packets on the virtual network for analysis.

ircd
Version: 2.8/hybrid-6.3.1
http://www.ircd-hybrid.com

Description:
ircd is an IRC server daemon that runs on most UNIX based platforms. It is used here to run an irc server on the Red Hat 9 system to provoke additional behavior from the malware.

irc
Version: 20030709

Description:
irc is an IRC command line client for Unix/Linux. It is used here to interact with the ircd server and to further provoke and analyze the malware.
Properties of the Malware Specimen

Malware File: msrll.exe
File Type: executable
File Size: 41,984 bytes
MD5 Hash: 84acfe96a98590813413122c12c11aaa

Operating Systems Effected:
Microsoft Windows 9x, 2000, XP

Strings Embedded in File:

Fig. A shows the interesting strings found in the msrll.exe malware file before unpacking.

<table>
<thead>
<tr>
<th>File pos</th>
<th>Mem pos</th>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000004D</td>
<td>0040004D</td>
<td>0</td>
<td>!This program cannot be run in DOS mode.</td>
</tr>
<tr>
<td>00000178</td>
<td>00400178</td>
<td>0</td>
<td>.text</td>
</tr>
<tr>
<td>000001A0</td>
<td>004001A0</td>
<td>0</td>
<td>.data</td>
</tr>
<tr>
<td>000001F0</td>
<td>004001F0</td>
<td>0</td>
<td>.idata</td>
</tr>
<tr>
<td>00000218</td>
<td>00400218</td>
<td>0</td>
<td>.aspack</td>
</tr>
<tr>
<td>00000240</td>
<td>00400240</td>
<td>0</td>
<td>.edata</td>
</tr>
</tbody>
</table>

Fig. B shows the interesting strings found in the msrll.exe malware file after it was unpacked using aspackdie.

<table>
<thead>
<tr>
<th>File pos</th>
<th>Mem pos</th>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00400000</td>
<td>0</td>
<td>MZ</td>
</tr>
<tr>
<td>0000004D</td>
<td>0040004D</td>
<td>0</td>
<td>!This program cannot be run in DOS mode.</td>
</tr>
<tr>
<td>00000080</td>
<td>00400080</td>
<td>0</td>
<td>PE</td>
</tr>
<tr>
<td>00000178</td>
<td>00400178</td>
<td>0</td>
<td>.text</td>
</tr>
<tr>
<td>000001A0</td>
<td>004001A0</td>
<td>0</td>
<td>.data</td>
</tr>
<tr>
<td>000001C8</td>
<td>004001C8</td>
<td>0</td>
<td>.bss</td>
</tr>
<tr>
<td>000001F0</td>
<td>004001F0</td>
<td>0</td>
<td>.idata</td>
</tr>
<tr>
<td>00000218</td>
<td>00400218</td>
<td>0</td>
<td>.aspack</td>
</tr>
<tr>
<td>00000240</td>
<td>00400240</td>
<td>0</td>
<td>.adata</td>
</tr>
<tr>
<td>00000268</td>
<td>00400268</td>
<td>0</td>
<td>.newIID</td>
</tr>
<tr>
<td>0000130D</td>
<td>0040130D</td>
<td>0</td>
<td>PW</td>
</tr>
<tr>
<td>00001326</td>
<td>00401326</td>
<td>0</td>
<td>?insmod</td>
</tr>
<tr>
<td>0000132E</td>
<td>0040132E</td>
<td>0</td>
<td>?rmmod</td>
</tr>
<tr>
<td>00001335</td>
<td>00401335</td>
<td>0</td>
<td>?lsmod</td>
</tr>
<tr>
<td>00001399</td>
<td>00401399</td>
<td>0</td>
<td>%s: &lt;mod name&gt;</td>
</tr>
<tr>
<td>Time</td>
<td>Event</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000377B</td>
<td>0040377B</td>
<td>0   WHO %s</td>
<td></td>
</tr>
<tr>
<td>00003A45</td>
<td>00403A45</td>
<td>0   USERHOST %s</td>
<td></td>
</tr>
<tr>
<td>00003A52</td>
<td>00403A52</td>
<td>0   logged into %s(%s) as %s</td>
<td></td>
</tr>
<tr>
<td>00003B99</td>
<td>00403B99</td>
<td>0   nick.pre</td>
<td></td>
</tr>
<tr>
<td>00003BA2</td>
<td>00403BA2</td>
<td>0   %s=%04u</td>
<td></td>
</tr>
<tr>
<td>00003BAA</td>
<td>00403BAA</td>
<td>0   irc.user</td>
<td></td>
</tr>
<tr>
<td>00003BB3</td>
<td>00403BB3</td>
<td>0   irc.usereal</td>
<td></td>
</tr>
<tr>
<td>00003BBF</td>
<td>00403BBF</td>
<td>0   irc.real</td>
<td></td>
</tr>
<tr>
<td>00003BC8</td>
<td>00403BC8</td>
<td>0   irc.pass</td>
<td></td>
</tr>
<tr>
<td>00003BE0</td>
<td>00403BE0</td>
<td>0   tsend(): connection to %s:%u failed</td>
<td></td>
</tr>
<tr>
<td>00003C20</td>
<td>00403C20</td>
<td>0   USER %s localhost 0 :%s</td>
<td></td>
</tr>
<tr>
<td>00003C38</td>
<td>00403C38</td>
<td>0   NICK %s</td>
<td></td>
</tr>
<tr>
<td>000040BA</td>
<td>004040BA</td>
<td>0   PING</td>
<td></td>
</tr>
<tr>
<td>000040BF</td>
<td>004040BF</td>
<td>0   PRIVMSG</td>
<td></td>
</tr>
<tr>
<td>000040C7</td>
<td>004040C7</td>
<td>0   JOIN</td>
<td></td>
</tr>
<tr>
<td>000040D0</td>
<td>004040D0</td>
<td>0   QUIT</td>
<td></td>
</tr>
<tr>
<td>000040D5</td>
<td>004040D5</td>
<td>0   352</td>
<td></td>
</tr>
<tr>
<td>000040D9</td>
<td>004040D9</td>
<td>0   302</td>
<td></td>
</tr>
<tr>
<td>000040DD</td>
<td>004040DD</td>
<td>0   303</td>
<td></td>
</tr>
<tr>
<td>000040E1</td>
<td>004040E1</td>
<td>0   005</td>
<td></td>
</tr>
<tr>
<td>000040E5</td>
<td>004040E5</td>
<td>0   PART</td>
<td></td>
</tr>
<tr>
<td>000040EA</td>
<td>004040EA</td>
<td>0   KICK</td>
<td></td>
</tr>
<tr>
<td>000040EF</td>
<td>004040EF</td>
<td>0   353</td>
<td></td>
</tr>
<tr>
<td>000040F3</td>
<td>004040F3</td>
<td>0   433</td>
<td></td>
</tr>
<tr>
<td>000040F7</td>
<td>004040F7</td>
<td>0   324</td>
<td></td>
</tr>
<tr>
<td>000040FD</td>
<td>004040FD</td>
<td>0   t&amp;</td>
<td></td>
</tr>
<tr>
<td>00004100</td>
<td>00404100</td>
<td>0   trecv(): Disconnected from %s err:%u</td>
<td></td>
</tr>
<tr>
<td>0000446B</td>
<td>0040446B</td>
<td>0   NOTICE</td>
<td></td>
</tr>
<tr>
<td>00004472</td>
<td>00404472</td>
<td>0   %s %s :%s</td>
<td></td>
</tr>
<tr>
<td>0000447D</td>
<td>0040447D</td>
<td>0   %s</td>
<td></td>
</tr>
<tr>
<td>00004711</td>
<td>00404711</td>
<td>0   MODE %s -o+b %s *@%s</td>
<td></td>
</tr>
<tr>
<td>00004727</td>
<td>00404727</td>
<td>0   %s</td>
<td></td>
</tr>
<tr>
<td>000047E7</td>
<td>004047E7</td>
<td>0   MODE %s -bo %s %s</td>
<td></td>
</tr>
<tr>
<td>00004924</td>
<td>00404924</td>
<td>0   %s.key</td>
<td></td>
</tr>
<tr>
<td>00004AA8</td>
<td>00404AA8</td>
<td>0   sk%u %s is dead!</td>
<td></td>
</tr>
<tr>
<td>00004ABA</td>
<td>00404ABA</td>
<td>0   s_check: %s dead? pinging...</td>
<td></td>
</tr>
<tr>
<td>00004AD7</td>
<td>00404AD7</td>
<td>0   PING :ok</td>
<td></td>
</tr>
<tr>
<td>00004B28</td>
<td>00404B28</td>
<td>0   expect the worst</td>
<td></td>
</tr>
<tr>
<td>00004B39</td>
<td>00404B39</td>
<td>0   s_check: killing socket %s</td>
<td></td>
</tr>
<tr>
<td>00004B54</td>
<td>00404B54</td>
<td>0   irc.knick</td>
<td></td>
</tr>
<tr>
<td>00004B5E</td>
<td>00404B5E</td>
<td>0   jtr.%u%s.iso</td>
<td></td>
</tr>
<tr>
<td>00004B6B</td>
<td>00404B6B</td>
<td>0   ison %s</td>
<td></td>
</tr>
<tr>
<td>00004B74</td>
<td>00404B74</td>
<td>0   servers</td>
<td></td>
</tr>
<tr>
<td>00004B7C</td>
<td>00404B7C</td>
<td>0   s_check: trying %s</td>
<td></td>
</tr>
<tr>
<td>00004DAA</td>
<td>00404DAA</td>
<td>0   Ph9K@</td>
<td></td>
</tr>
<tr>
<td>00004ED5</td>
<td>00404ED5</td>
<td>0   PhkK@</td>
<td></td>
</tr>
<tr>
<td>00004F41</td>
<td>00404F41</td>
<td>0   ShtK@</td>
<td></td>
</tr>
<tr>
<td>00005052</td>
<td>00405052</td>
<td>0   %s.mode</td>
<td></td>
</tr>
<tr>
<td>000050A0</td>
<td>004050A0</td>
<td>0   MODE %s %s</td>
<td></td>
</tr>
<tr>
<td>00005078</td>
<td>00405078</td>
<td>0   ShRP@</td>
<td></td>
</tr>
<tr>
<td>000050DA</td>
<td>004050DA</td>
<td>0   Sh$1@</td>
<td></td>
</tr>
<tr>
<td>0000559F</td>
<td>0040559F</td>
<td>0   aop</td>
<td></td>
</tr>
<tr>
<td>000055A3</td>
<td>004055A3</td>
<td>0   mode %s +o %s</td>
<td></td>
</tr>
<tr>
<td>00005BB2</td>
<td>00405BB2</td>
<td>0   akick</td>
<td></td>
</tr>
<tr>
<td>000055B8</td>
<td>004055B8</td>
<td>0   mode %s +b %s %s</td>
<td></td>
</tr>
<tr>
<td>000055CA</td>
<td>004055CA</td>
<td>0   KICK %s %s</td>
<td></td>
</tr>
</tbody>
</table>
irc.pre
Set an irc sock to preform %s command on
Type
csklist
to view current sockets, then
cdcsk
%: dll loaded
%: %d
said %s to %s
usage: %s <target> "text"
% not on %s
% usage: %s <nick> <chan>
PAS
% logged in
% bot: %s
preformance counter not avail
% usage: %s <cmd>
% free'd
% unable to free %s
later!
% unable to %s errno:%u
service:%c user:%s inet connection:%c
contype:%s reboot privs:%c
% kill
% %-%u %s
% %s
% somefile
% host: %s ip: %s
capGetDriverDescriptionA
9x
% 2k
% XP
% XP++
cpus:%u
% CAM
WIN%: (u:%s)%s mem:(%u/%u) %u% %s
open
% %: %s (%u)
% NICK
% %s
% %s bad args
% %
% %
% akick
% OP
% KICK
% %u %s
% %s removed
% couldn't find %s
% % added
% % already in list
% usage: %s +/- <host>
jtram.conf
% t %s
jtr.home
% %s\\s\\s
%: possibly failed: code %u
0000880B 0040880B 0 usage %s <socks #>
000088D7 004088D7 0 leaves %s
000088E1 004088E1 0 :0 * :%s
000088EC 004088EC 0 hmm
00008A96 00408A96 0 joins: %s
00008BB7D 00408B7D 0 chat
00008BB9 00408BB9 0 resume
00008BB9 00408BB9 0 err: %u
00008BB9 00408BB9 0 DCC ACCEPT %s %s %s
00008BBAE 00408BBAE 0 dcc_resume: cant find port %s
00008BBCC 00408BBCC 0 send
00008BDD 00408BDD 0 dcc.dir
00008BEE 00408BEE 0 %s\%s\%s\%s
00008BDF 00408BDF 0 unable to open (%s): %u
00008BFD 00408BFD 0 resuming dcc from %s to %s
00008C19 00408C19 0 DCC RESUME %s %s %u
00009345 00409345 0 ?si
00009349 00409349 0 ?ssl
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
0000939D 0040939D 0 ?op
000093A4 004093A4 0 ?aop
000093A9 004093A9 0 ?akick
000093B0 004093B0 0 ?part
000093B6 004093B6 0 ?dump
000093BC 004093BC 0 ?set
000093C1 004093C1 0 ?die
000093C6 004093C6 0 ?md5p
000093CC 004093CC 0 ?free
000093D2 004093D2 0 ?raw
000093D7 004093D7 0 ?update
000093DF 004093DF 0 ?hostname
000093E9 004093E9 0 ?fif
000093EE 004093EE 0 ?!fif
000093F4 004093F4 0 ?del
000093F9 004093F9 0 ?pwd
000093FE 004093FE 0 ?play
00009404 00409404 0 ?copy
0000940A 0040940A 0 ?move
00009410 00409410 0 ?dir
00009415 00409415 0 ?sums
0000941B 0040941B 0 ?ls
0000941F 0040941F 0 ?cd
00009423 00409423 0 ?rmdir
0000942A 0040942A 0 ?mkdir
00009431 00409431 0 ?run
00009436 00409436 0 ?exec
Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46

...
Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46
Hashes built-in:
- SHA-512
- SHA-384
- SHA-256
- TIGER
- SHA1
- MD5
- MD4
- MD2

Block Chaining Modes:
- CFB
- OFB
- CTR

PK Algs:
- RSA
- DH
- ECC

Compiler:
- WIN32 platform detected.
- GCC compiler detected.

Various others: BASE64 MPI HMAC /dev/random
Microsoft Base Cryptographic Provider v1.0
bits.c
buf != NULL
prng != NULL
-LIBGCCW32-EH-SJLJ-GTHR-MINGW32
key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46

...
0011B6D6  0051B6D6  0  CryptGenRandom 0011B6EA  0051B6EA  0  CryptReleaseContext 0011B702  0051B702  0  GetUserNameA 0011B712  0051B712  0  LookupPrivilegeValueA 0011B72A  0051B72A  0  OpenProcessToken 0011B73E  0051B73E  0  OpenSCManagerA 0011B752  0051B752  0  RegCloseKey 0011B762  0051B762  0  RegCreateKeyExA 0011B776  0051B776  0  RegSetValueExA 0011B78A  0051B78A  0  RegisterServiceCtrlHandlerA 0011B7AA  0051B7AA  0  SetServiceStatus 0011B7BE  0051B7BE  0  StartServiceCtrlDispatcherA 0011B7DE  0051B7DE  0  AddAtomA 0011B7EA  0051B7EA  0  CloseHandle 0011B7FA  0051B7FA  0  CopyFileA 0011B806  0051B806  0  CreateDirectoryA 0011B81A  0051B81A  0  CreateFileA 0011B82A  0051B82A  0  CreateMutexA 0011B83A  0051B83A  0  CreatePipe 0011B84A  0051B84A  0  CreateProcessA 0011B85E  0051B85E  0  CreateToolhelp32Snapshot 0011B87A  0051B87A  0  DeleteFileA 0011B88A  0051B88A  0  DuplicateHandle 0011B89E  0051B89E  0  EnterCriticalSection 0011B8B6  0051B8B6  0  ExitProcess 0011B8C6  0051B8C6  0  ExitThread 0011B8D6  0051B8D6  0  FileTimeToSystemTime 0011B8EE  0051B8EE  0  FindAtomA 0011B8FA  0051B8FA  0  FindClose 0011B906  0051B906  0  FindFirstFileA 0011B91A  0051B91A  0  FindNextFileA 0011B92A  0051B92A  0  FreeLibrary 0011B93A  0051B93A  0  GetAtomNameA 0011B94A  0051B94A  0  GetCommandLineA 0011B95E  0051B95E  0  GetCurrentDirectoryA 0011B976  0051B976  0  GetCurrentProcess 0011B98A  0051B98A  0  GetCurrentThreadId 0011B9A2  0051B9A2  0  GetExitCodeProcess 0011B9BA  0051B9BA  0  GetFileSize 0011B9CA  0051B9CA  0  GetFullPathNameA 0011B9DE  0051B9DE  0  GetLastError 0011B9EE  0051B9EE  0  GetModuleFileNameA 0011BA06  0051BA06  0  GetModuleHandleA 0011BA1A  0051BA1A  0  GetProcAddress 0011BA2E  0051BA2E  0  GetStartupInfoA 0011BA42  0051BA42  0  GetSystemDirectoryA 0011BA5A  0051BA5A  0  GetSystemInfo 0011BA6A  0051BA6A  0  GetTempPathA 0011BA7A  0051BA7A  0  GetTickCount 0011BA8A  0051BA8A  0  GetVersionExA 0011BA9A  0051BA9A  0  GlobalMemoryStatus 0011BAB2  0051BAB2  0  InitializeCriticalSection 0011BACE  0051BACE  0  IsBadReadPtr 0011BADD  0051BADD  0  LeaveCriticalSection 0011BAF6  0051BAF6  0  LoadLibraryA 0011BB06  0051BB06  0  MoveFileA 0011BB12  0051BB12  0  OpenProcess 0011BB22  0051BB22  0  PeekNamedPipe
0011BB32 0051BB32 0  Process32First
0011BB46 0051BB46 0  Process32Next
0011BB56 0051BB56 0  QueryPerformanceFrequency
0011BB72 0051BB72 0  ReadFile
0011BB7E 0051BB7E 0  ReleaseMutex
0011BB8E 0051BB8E 0  RemoveDirectoryA
0011BBA2 0051BBA2 0  SetConsoleCtrlHandler
0011BBBA 0051BBBA 0  SetCurrentDirectoryA
0011BBD2 0051BBD2 0  SetFilePointer
0011BEE6 0051BEE6 0  SetUnhandledExceptionFilter
0011BC06 0051BC06 0  Sleep
0011BC0E 0051BC0E 0  TerminateProcess
0011BC22 0051BC22 0  WaitForSingleObject
0011BC3A 0051BC3A 0  WriteFile
0011BC46 0051BC46 0  _itoa
0011BC4E 0051BC4E 0  __stat
0011BC56 0051BC56 0  __strdup
0011BC62 0051BC62 0  __strcmp
0011BC6C 0051BC6C 0  __setmainargs
0011BC7E 0051BC7E 0  __p__environ
0011BC8E 0051BC8E 0  __p__fmode
0011BC9E 0051BC9E 0  __set_app_type
0011BCB2 0051BCB2 0  __beginthread
0011BCC2 0051BCC2 0  __exit
0011BCCE 0051BCCE 0  __errno
0011BCEA 0051BCEA 0  __fileno
0011BCEC 0051BCEC 0  __fio
0011BCEE 0051BCEE 0  __onexit
0011BCFA 0051BCFA 0  __setmode
0011BD06 0051BD06 0  __vsnprintf
0011BD16 0051BD16 0  abort
0011BD1E 0051BD1E 0  atexit
0011BD2A 0051BD2A 0  atoi
0011BD32 0051BD32 0  clock
0011BD3A 0051BD3A 0  fclose
0011BD46 0051BD46 0  ffflush
0011BD52 0051BD52 0  fgets
0011BD5A 0051BD5A 0  fopen
0011BD62 0051BD62 0  fprintf
0011BD6E 0051BD6E 0  fread
0011BD76 0051BD76 0  free
0011BD7E 0051BD7E 0  fwrite
0011BD8A 0051BD8A 0  malloc
0011BD96 0051BD96 0  memcp
0011BDA2 0051BDA2 0  memset
0011BDAE 0051BDAE 0  printf
0011BDBA 0051BDBA 0  raise
0011BDC2 0051BDC2 0  rand
0011BDC7 0051BDC7 0  realloc
0011BDD6 0051BDD6 0  setvbuf
0011BDE2 0051BDE2 0  signal
0011BDEE 0051BDEE 0  sprintf
0011BDF0 0051BDF0 0  srand
0011BEE0 0051BEE0 0  strcat
0011BEE0 0051BEE0 0  strchr
0011BE1A 0051BE1A 0  strcmp
0011BE26 0051BE26 0  strcpy
0011BE32 0051BE32 0  strerror
0011BE3E 0051BE3E 0  strncat
0011BE4A 0051BE4A 0  strncmp
0011BE56 0051BE56 0  strncpy
0011BE62 0051BE62 0  strstr
0011BE6E 0051BE6E 0  time
0011BE76 0051BE76 0  toupper
0011BE82 0051BE82 0  ShellExecuteA
0011BE92 0051BE92 0  DispatchMessageA
0011BEA6 0051BEA6 0  ExitWindowsEx
0011BEB6 0051BEB6 0  GetMessageA
0011BEC6 0051BEC6 0  PeekMessageA
0011BED6 0051BED6 0  GetFileVersionInfoA
0011BEEE 0051BEEE 0  VerQueryValueA
0011BF02 0051BF02 0  InternetCloseHandle
0011BF1A 0051BF1A 0  InternetGetConnectedState
0011BF36 0051BF36 0  InternetOpenA
0011BF46 0051BF46 0  InternetOpenUrlA
0011BF5A 0051BF5A 0  InternetReadFile
0011BF6E 0051BF6E 0  WSAGetLastError
0011BF82 0051BF82 0  WSASocketA
0011BF92 0051BF92 0  WSANotify
0011BFA2 0051BFA2 0  ___WSAPI32
0011BFB2 0051BFB2 0  accept
0011BFBE 0051BFBE 0  bind
0011BFC6 0051BFC6 0  closesocket
0011BFD6 0051BFD6 0  connect
0011BFE2 0051BFE2 0  gethostbyname
0011BFF2 0051BFF2 0  gethostname
0011C002 0051C002 0  getsockname
0011C02A 0051C02A 0  htons
0011C032 0051C032 0  inet_addr
0011C03E 0051C03E 0  inet_ntoa
0011C04A 0051C04A 0  ioctlsocket
0011C05A 0051C05A 0  listen
0011C066 0051C066 0  ntohl
0011C06E 0051C06E 0  recv
0011C076 0051C076 0  select
0011C082 0051C082 0  send
0011C08A 0051C08A 0  sendto
0011C096 0051C096 0  setsockopt
0011C0A6 0051C0A6 0  shutdown
0011C0B2 0051C0B2 0  socket
0011C0FC 0051C0FC 0  ADVAPI32.DLL
0011C1FC 0051C1FC 0  KERNEL32.dll
0011C21C 0051C21C 0  msvcr90.dll
0011C2E0 0051C2E0 0  msvcr90.dll
0011C2F0 0051C2F0 0  SHELL32.DLL
0011C30C 0051C30C 0  USER32.dll
0011C320 0051C320 0  VERSION.dll
0011C340 0051C340 0  WININET.DLL
0011C3B4 0051C3B4 0  WSZ 32.DLL
0011D071 0051D071 0  VirtualAlloc
0011D07E 0051D07E 0  VirtualFree
0011D441 0051D441 0  kernel32.dll
0011D44E 0051D44E 0  ExitProcess
0011D45A 0051D45A 0  user32.dll
001202FE 005202FE 0 ExitThread
0012030B 0052030B 0 FileTimeToSystemTime
00120322 00520322 0 FindAtomA
0012032E 0052032E 0 FindClose
0012033A 0052033A 0 FindFirstFileA
0012034B 0052034B 0 FindNextFileA
0012035B 0052035B 0 FreeLibrary
00120369 00520369 0 GetAtomNameA
00120378 00520378 0 GetCommandLineA
0012038A 0052038A 0 GetCurrentDirectoryA
001203A1 005203A1 0 GetCurrentProcess
001203B5 005203B5 0 GetCurrentThreadId
001203CA 005203CA 0 GetExitCodeProcess
001203DF 005203DF 0 GetFileSize
001203ED 005203ED 0 GetModuleFileNameA
00120400 00520400 0 GetLastError
0012040F 0052040F 0 GetModuleHandleA
00120424 00520424 0 GetProcAddress
00120437 00520437 0 GetProcAddress
00120448 00520448 0 GetStartupInfoA
0012045A 0052045A 0 GetSystemDirectoryA
00120470 00520470 0 GetSystemInfo
00120480 00520480 0 GetTempPathA
0012048F 0052048F 0 GetTickCount
0012049E 0052049E 0 GetVersionExA
001204AE 005204AE 0 GlobalMemoryStatus
001204DF 005204DF 0 IsBadReadPtr
001204EE 005204EE 0 LeaveCriticalSection
001204F0 005204F0 0 LoadLibraryA
00120514 00520514 0 MoveFileA
00120520 00520520 0 OpenProcess
0012052E 0052052E 0 PeekNamedPipe
0012053E 0052053E 0 Process32First
0012054F 0052054F 0 Process32Next
0012055F 0052055F 0 QueryPerformanceFrequency
0012057B 0052057B 0 ReadFile
00120586 00520586 0 ReleaseMutex
00120595 00520595 0 RemoveDirectoryA
001205A8 005205A8 0 SetConsoleCtrlHandler
001205C0 005205C0 0 SetCurrentDirectoryA
001205D7 005205D7 0 SetFilePointer
001205E8 005205E8 0 SetUnhandledExceptionFilter
00120606 00520606 0 Sleep
0012060E 0052060E 0 TerminateProcess
00120621 00520621 0 WaitForSingleObject
00120637 00520637 0 WriteFile
00120641 00520641 0 msvcrt.dll
0012064E 0052064E 0 __itoa
00120656 00520656 0 __stat
0012065E 0052065E 0 __mbsdup
00120668 00520668 0 __strcmpi
00120671 00520671 0 msvcrt.dll
0012067E 0052067E 0 __getmainargs
0012068E 0052068E 0 __p__env
0012069D 0052069D 0 __p__fmode
001206AA 005206AA 0 __set_app_type
001206BB 005206BB 0 _beginthread
001206CA 005206CA 0 _cexit
001206D3 005206D3 0 _errno
001206DC 005206DC 0 _fileno
001206E6 005206E6 0 _iob
001206ED 005206ED 0 _onexit
001206F7 005206F7 0 _setmode
00120702 00520702 0 _vsnprintf
0012070F 0052070F 0 abort
00120717 00520717 0 atexit
00120720 00520720 0 atoi
00120727 00520727 0 clock
0012072F 0052072F 0 fclose
00120738 00520738 0 fflush
00120741 00520741 0 fgets
00120749 00520749 0 fopen
00120751 00520751 0 fprintf
0012075B 0052075B 0 fread
00120763 00520763 0 free
0012076A 0052076A 0 fwrite
00120773 00520773 0 malloc
0012077C 0052077C 0 memset
00120785 00520785 0 memcpy
0012078E 0052078E 0 printf
00120797 00520797 0 raise
0012079F 0052079F 0 rand
001207A6 005207A6 0 realloc
001207B0 005207B0 0 setvbuf
001207BA 005207BA 0 signal
001207C3 005207C3 0 sprintf
001207CD 005207CD 0 strncat
001207D5 005207D5 0 _mbscat
001207DF 005207DF 0 strchr
001207E8 005207E8 0 strcmp
001207F1 005207F1 0 _mbscopy
001207FB 005207FB 0 strerror
00120806 00520806 0 strncat
00120810 00520810 0 strncmp
0012081A 0052081A 0 strncmp
00120824 00520824 0 stristr
0012082D 0052082D 0 time
00120834 00520834 0 toupper
0012083C 0052083C 0 shell32.dll
0012084A 0052084A 0 ShellExecuteA
00120858 00520858 0 USER32.dll
00120865 00520865 0 DispatchMessageA
00120878 00520878 0 ExitWindowsEx
00120888 00520888 0 GetMessageA
00120896 00520896 0 PeekMessageA
001208A3 005208A3 0 version.dll
001208B1 005208B1 0 GetFileVersionInfoA
001208C7 005208C7 0 VerQueryValueA
001208D6 005208D6 0 wininet.dll
001208E4 005208E4 0 InternetCloseHandle
001208FA 005208FA 0 InternetGetConnectedState
00120916 00520916 0 InternetOpenA
00120926 00520926 0 InternetOpenUrlA
00120939 00520939 0 InternetReadFile
0012094A 0052094A 0 ws2_32.dll
Behavioral Analysis

Preparation for Infection

First I use md5sum to get the md5 hash of the msrll.exe malware file by running the following command:

```
c:\malware\exe>md5sum msrll.exe
84acfe96a9859081341312c12c11aaa *msrll.exe
```

The reason I did this first is so I have a baseline to compare to after executing the msrll.exe malware. If the malware modifies msrll.exe or copies itself somewhere else we will be able to verify if the file is the exact same as the original.

I then ran Regshot to get a snap shot of the registry on the clean system. Fig. 1 shows the options used to get the 1st snap shot.

![Regshot 1.61e5 Final](image)

I click 1st shot and select shot and save to get a copy of the registry while the system is clean to compare later with the infected version.

Next I open FileMon, RegMon, and TDIMon and stop them from capturing and clear the display.

Process Explorer is then opened to show the processes running on the system.

On the VMware virtual Red Hat 9 system I start the sniffer by typing in the following
command

```
snort -vd | tee /tmp/sniffer1.log
```

This command will use snort to capture network traffic to the file sniffer1.log in the tmp directory.

**Infection**

Now everything is ready to infect the VMware Windows XP SP1 system. I start capturing on FileMon, RegMon, and TDIMon. I then execute msrll.exe malware by double clicking on it. After waiting about 30 seconds I view the processes using Process Explorer and see msrll.exe running under the parent process explorer. By highlighting the process and hitting the del key I kill the malware process.

Quickly I stop capturing on FileMon, RegMon, and TDIMon. Also, I switch over to the VMware Red Hat 9 system and hit CTRL+C to end the snort capture.

I run Regshot again this time clicking on 1st shot and selecting load. I browse to the saved registry file from the first time I ran Regshot on the clean system. Next I click on 2nd shot and select shot and save. Now we can click on compare and see the results shown in Fig. 2.

Fig. 2

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\mfm\Security\Security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 00 01 00 00 00 20 80 14 00 FF 01 0F 00 01 01 00 00 00 00 01 00 00 00 00 00 00 00 05 12 00 00 00 00 18 00 FF 01 0F 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 8D 01 02 00 01 01 00 00 00 00 00 05 0B 00 00 00 00 00 18 00 FD 01 02 00 01 02 00 00 00 00 00 05 00 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 01 01 00 00 00 00 05 12 00 00 00
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\mfm\Type: 0x00000120
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\mfm\Start: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\mfm\ErrorControl: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\mfm\ImagePath: "C:\WINDOWS\System32\msrll.exe"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\mfm\ObjectName: "LocalSystem"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\Security\Security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 00 01 00 00 00 20 80 14 00 FF 01 0F 00 01 01 00 00 00 00 05 12 00 00 00 00 18 00 FF 01 0F 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 8D 01 02 00 01 01 00 00 00 00 00 05 0B 00 00 00 00 00 18 00 FD 01 02 00 01 02 00 00 00 00 00 05 00 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 01 01 00 00 00 00 05 12 00 00 00
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\Type: 0x00000120
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\Start: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\ErrorControl: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\ImagePath: "C:\WINDOWS\System32\msrll.exe"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\ObjectName: "LocalSystem"
```
In examining the interesting results from the Regshot compare it looks like a service was created by the name of “RELL enhanced drive”. This service was confirmed to exist by opening the services window on the system and locating the service with that name. It was setup to start automatically on every boot up and ran with logon of local system. It was running the executable located in c:\windows\system32\mfm\msrll.exe. I verified the file existed there by navigating explorer to that location where I found msrll.exe and jtram.conf files.

I ran md5sum on msrll.exe in the c:\windows\system32\mfm location and compared the hash to the original msrll.exe file hash. The hashes matched so the file was an exact copy of the one I ran earlier. I then opened jtram.conf in UltraEdit-32. The jtram.conf file was not understandable and was probably encrypted.

I examined the FileMon log and found some interesting entries listed in Fig. 3.

Fig. 3

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>File Path</th>
<th>Options</th>
<th>Success Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:51:07</td>
<td>msrll.exe</td>
<td>CREATE</td>
<td>C:\WINDOWS\System32\mfm</td>
<td>SUCCESS</td>
<td>Options: Create Directory</td>
</tr>
<tr>
<td>2:51:08</td>
<td>msrll.exe</td>
<td>CREATE</td>
<td>C:\WINDOWS\System32\mfm\msrll.exe</td>
<td>SUCCESS</td>
<td>Options: OverwriteIf Sequential</td>
</tr>
<tr>
<td>2:51:08</td>
<td>msrll.exe</td>
<td>WRITE</td>
<td>C:\WINDOWS\System32\mfm\msrll.exe</td>
<td>SUCCESS</td>
<td>Options: OverwriteIf Sequential</td>
</tr>
<tr>
<td>2:51:29</td>
<td>msrll.exe</td>
<td>CREATE</td>
<td>C:\WINDOWS\system32\mfm\jtram.conf</td>
<td>SUCCESS</td>
<td>Options: OverwriteIf Sequential</td>
</tr>
</tbody>
</table>

The FileMon log shows the creation of the directory mfm in c:\windows\system32 along with the creation of the files msrll.exe and jtram.conf.

Examination of the RegMon log files didn’t reveal much more than what Regshot did. One interesting entry found in the RegMon log is listed in Fig. 4.

Fig. 4

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>File Path</th>
<th>Options</th>
<th>Success Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>659.29827310</td>
<td>msrll.exe</td>
<td>SetValue</td>
<td>HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed</td>
<td>SUCCESS</td>
<td>E0 7F F7 64 D4 66 A1 09 ...</td>
</tr>
</tbody>
</table>

This entry has something to do with cryptography.

Examination of TDIMon shows TCP activity being setup on the system. Entries in the TDIMon log show ports 2200 and 113 in use. TDIMon interesting log entries are displayed in Fig. 5.

Fig. 5

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>Address</th>
<th>Success Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:51:27</td>
<td>msrll.exe</td>
<td>IRP_MJ_CREATE</td>
<td>TCP:0.0.0.0:2200</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>2:51:27</td>
<td>msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER</td>
<td>TCP:0.0.0.0:2200</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>2:51:27</td>
<td>msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER</td>
<td>TCP:0.0.0.0:2200</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>2:51:27</td>
<td>msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER</td>
<td>TCP:0.0.0.0:2200</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>2:51:27</td>
<td>msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER</td>
<td>TCP:0.0.0.0:2200</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>2:51:27</td>
<td>msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER</td>
<td>TCP:0.0.0.0:2200</td>
<td>SUCCESS</td>
</tr>
</tbody>
</table>
I also ran AutoRuns just to check if any auto run registry entry had been added, but all looked well there.

Next I examined the snort capture file sniffer1.log. This capture file was completely empty. So it appears that no network traffic took place during the initial infection of the system.
I decided to run the service created “Rll Enhanced Drive”, but before I do I ready snort for another network capture with the command:

```
snort –vd | tee /tmp/sniffer2.log
```

I also start Process Explorer to monitor what processes are running.

Now I start the service and leave it run another 30 seconds. While I’m waiting for the 30 seconds I examine Process Explorer and see msrll.exe process running under parent service services.exe. By double clicking on the msrll.exe process in Process Explorer I get a window with various tabs to display information about the process. I select the TCP/IP tab and see that it is listening on ports 113 and 2200. After about 30 seconds pass I kill the process and switch to the VMware Red Hat 9 system and end the snort capture.

The listening ports 113 and 2200 correlates to the TDIMon log entries. This malware is listening on these ports.

Examination of the snort capture sniffer2.log shows the interesting entries displayed in Fig. 6.

```
UDP TTL:128 TOS:0x0 ID:786 IpLen:20 DgmLen:66
Len: 38
00 0D 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.....
```

This packet shows a DNS request coming from 192.168.62.129 port 1026 to 192.168.62.1 port 53 for the domain name collective7.zxy0.com. This DNS request is not answered however because 192.168.62.1 is the host computer that is running VMware and is not running a DNS server. 192.168.62.129 is the VMware virtual Windows XP SP1 system that is the infected system.

Since I have no DNS server I add to the hosts file on the infected system the entry listed in Fig. 7. The hosts file on the infected system is in `c:\windows\system32\drivers\etc\hosts`.

```
192.168.62.128 collective7.zxy0.com
```

Fig. 6

Fig. 7
I put the address 192.168.62.128 in because I want to redirect any traffic going from
the infected host 192.168.62.129 to my VMware Red Hat 9 system 192.168.62.128 so I
can capture the requests the infected system is sending to collective7.zxy0.com.

I start another snort capture on the Red Hat 9 system using the following command:

```sh
snort –vd | tee /tmp/sniffer3.log
```

I also start Process Explorer to monitor the msrll.exe process.

I restart the “Rll Enhanced Drive” service and wait about 30 seconds. While waiting the
30 seconds I use Process Explorer to view the process msrll.exe TCP/IP activity. I
observe it listening on ports 113 and 2200 like before, but during the wait I see it send
a connection request to 192.168.62.128:6667 and then stop. I then see another
connection request to 192.168.62.128:9999 and then stop. And another connection
request to 192.168.62.128:8080 and then stop. The 30 seconds are past so I kill the
msrll.exe process using Process Explorer and stop the snort capture on the Red Hat 9
system.

Examination of the snort capture file sniffer3.log I found the interesting entries listed in
Fig. 8.

Fig. 8

```
+================================+=
TCP TTL:128 TOS:0x0 ID:1401 IpLen:20 DgmLen:48 DF
******S* Seq: 0x8C1BF6D9  Ack: 0x0  Win: 0xFAF0  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
+================================+=
TCP TTL:128 TOS:0x0 ID:1403 IpLen:20 DgmLen:48 DF
******S* Seq: 0x8C8951C2  Ack: 0x0  Win: 0xFAF0  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
+================================+=
TCP TTL:128 TOS:0x0 ID:1406 IpLen:20 DgmLen:48 DF
******S* Seq: 0x8CF5F5FD8  Ack: 0x0  Win: 0xFAF0  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
+================================+=
```
Here there are connection requests from the infected system to the Red Hat 9 system on ports 6667, 9999, and 8080. The obvious port is 6667 which possibly could be an irc connection attempt. Port 9999 I'm not sure of, but listed as Distinct in the IANA list. Port 8080 is usually an alternate port for port 80 or proxy. These connection attempts listed in the snort capture correlate to the observed connection attempts in Process Explorer’s TCP/IP section for the msrll.exe process.

Since port 6667 is normally irc port I decide to run an irc server on the Red Hat 9 system that the infected system already thinks is the collective7.zxy0.com. Also the observed connection attempts to 192.168.62.128 on ports 9999 and 8080 prompt me to run netcat to record the requests on the Red Hat 9 system.

I start the irc server which is configured to listen on port 6667 by issuing the following commands on the Red Hat 9 system:

```
su – ircd
./ircd
exit
```

I then start the irc client in another session on the Red Hat 9 system by issuing the command:

```
irc
```

I also start two more sessions on the Red Hat 9 system and issue the following commands:

```
nc –l –p 9999 >/tmp/port9999.txt
nc –l –p 8080 >/tmp/port8080.txt
```

These commands start netcat listening on ports 9999 and 8080. Whatever is sent to these ports will be written to the appropriate file port9999.txt or port8080.txt in the /tmp directory. This is in the hope to capture what type of requests are being sent to these ports.

Snort capture is also started again with the following command:

```
snort –vd | tee /tmp/sniffer4.log
```

Again we start the msrll.exe malware by starting the “Rll Enhanced Drive” service and wait about 30 seconds. After 30 seconds I kill the msrll.exe process and stopped the snort capture along with the two netcat listeners on the Red Hat 9 system.

First I examine the snort capture sniffer4.log and find the interesting sections displayed
in Fig. 9.

Fig. 9

```
TCP TTL:128 TOS:0x0 ID:1449 IpLen:20 DgmLen:53 DF
***AP*** Seq: 0x992158CC  Ack: 0xB8F21707  Win: 0xFAA3  TcpLen: 20
4A 4F 49 4E 20 23 6D 6L 73 20 3A 0A           JOIN #mils :
```

Here it can be seen that the malware joined the channel #mils.

Next I examined the netcat capture files port9999.txt and port8080.txt and found them to be empty.

In the irc client I join the #mils channel by issuing the command:

```
/join #mils
```

This puts my client in the #mils channel.

I then run the malware again by starting the “Rll Enhanced Drive” service and wait in the irc client for the malware to join the #mils channel. After a few minutes it does join the #mils channel verifying that irc is in play with this malware.

I then try connecting to port 2200 on the infected system from the Red Hat 9 system using netcat command:

```
nc 192.168.62.129 2200
```

Netcat connects and a prompt appears #: that is awaiting my command. At this point I don’t really know what it wants so I just hit enter then CTL+C to drop the connection.

I try to connect to port 113 on the infected system from the Red Hat 9 system by issuing the following command:
nc 192.168.62.129 113

Netcat connects but no prompt just a blinking cursor. I hit enter and nothing happens so I type x and press enter. This disconnects me with the following text:

```
x : USERID : UNIX : GPRdvDe
```

At this point I am done with my behavioral analysis since I cannot invoke anymore behavior.

The summary of what is known at this point is that the malware installs itself into the c:\windows\system32\mfm directory as msrll.exe and an encrypted configuration file jtram.conf. It sets up a Windows service called “Rll Enhanced Drive” that runs at Windows startup the msrll.exe file in the mfm directory. The malware then attempts to connect to a server on the internet called collective7.zxy0.com on port 6667, 9999, and 8080. The connection to port 6667 is an irc connection and the malware joins the channel #mils. The malware also listens on ports 2200 and 113 which can be connected to using netcat. The listener on port 2200 displays a prompt #: and the port 113 doesn’t display any prompt.
Code Analysis

The code analysis starts with examining the BinText listing of the msrll.exe file. Fig. A on page 11 displays the BinText output for the msrll.exe file. Here the segment .aspack leads me to believe that the executable has been packed with aspack. The rest of the strings are not understandable which also indicates that the file is packed.

PE-Sniffer is run on the msrll.exe file, but the scans do not reveal the packing technique used. I then load the file in PEInfo which displays the section aspack further indicating that aspack was used.

I load aspackdie with the msrll.exe and a file is created called unpacked.exe and seems to have unpacked the msrll.exe. A quick examination of the strings using BinText reveals more understandable strings. It looks like it worked, but an execute test should be done. I copy the file to the c:\windows\system32\mfm directory and rename the original msrll.exe to msrll.exe.org and then rename unpacked.exe to msrll.exe. I start the service while monitoring the irc channel #mils with the irc client on the Red Hat system and sure enough the malware joins the channel after a few minutes. It looks now like I have an unpacked version of the malware that still works.

Examination of the interesting strings reveals what looks like commands that start with a question mark (?). Commands like ?si, ?jolt, ?uptime, and ?login. Some of these commands look like denial of service attacks while others seem to be giving information about the current state of the system. The command ?login leads me to believe that some sort of authentication needs to be done.

While running the malware I tried some of the commands in the irc connection to the #mils channel. Every command I typed didn’t elicit any response from the malware. I then try the following in the irc client:

```
?login
?login malware
? login a b
```

None of these generated a response from the malware. I tried these same commands using the netcat connection to port 2200 on the infected system from the Red Hat 9 system and got similar results.

The Search for Authentication Code

I’m going to cut to the chase here since there were many failed attempts at finding the authentication code. I used mainly the interesting strings found by BinText, see Fig. B on page 11, IDA Pro and OllyDbg to search the assembly instructions for possible areas where the malware is processing the authentication (?login).
I stumbled on the following string:

```
% bad pass from \"%s\"@%s
```

This looked like a place in the code you would go if your login failed.

I loaded the unpacked version of msrll.exe into IDA Pro and performed a text search looking for the string “bad pass”. The search had two hits, one at memory address 0040BB52 and another at memory address 0040BC6F displayed in Fig. 10.

---

Fig. 10

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>text:0040BB52</code></td>
<td><code>asBadPassFromS @ db %s bad pass from \&quot;%s\&quot;@%s</code></td>
</tr>
<tr>
<td><code>text:0040BC6F</code></td>
<td><code>push offset asBadPassFromS ; \&quot;%s bad pass from \&quot;%s\&quot;@%s\&quot;</code></td>
</tr>
</tbody>
</table>

The instruction located at address 0040BC6F looked to be where bad pass would be pushed to the stack preparing to display on the screen possibly. This became the area of the code I focused on. I wanted to find how you ended up at this instruction and where the decision was made to branch to this section of the code. To find this branch I used IDA Pro to view a flowchart of the malware instructions. Clicking on view + graphs + flow chart I was able to view the code in a more visual way. I searched the flow chart manually until I found the instruction set that matched what was at address 0040BC6F. This was in a section labeled loc_40BC5A seen in Fig. 11. Going up the flowchart the instructions that decide what branch to jump to is in the section labeled 0040BB6. In this section you can see the following instructions:

```assembly
    test   eax, eax
    jz     short loc_40BC5A
```

It’s easy to see in the flowchart that if the result of the test is true we are going to jump to the section of code that contains the bad pass string which is where I’m assuming I don’t want to go if I want to get authenticated. If the result is false the code jumps to address 0040BBEB which might be where you go if you get successfully authenticated.
I switch over to the code view and search for address 004BBD6 since this is the label for the beginning of the instructions that include the test and jz instructions. I find that the test instruction is at address 0040BBE7 which is where I decide I want to set a breakpoint, but not in IDA Pro. For this job I load msrll.exe into OllyDbg.

In OllyDbg I go to address 0040BBE7 and press F2 to set a breakpoint. Next I start ircd and irc on the Red Hat 9 system. In irc I join the #mils channel. Back in OllyDbg I hit F9 to run msrll.exe malware. Then I switch back to irc and wait for the malware to join the #mils channel. After a few minutes I see the malware join the #mils channel. I want to trigger the break point with the ?login command so I try the following commands:

```
?login malware
?login a b
```

Neither of these triggered the break point. I examined the code around 0040BBE7. The beginning of this section started at address 0040BBD6 which you can see in Fig. 11 above. Following the code I couldn't find any hints as to what the problem was.
I decided to connect using netcat on the Red Hat 9 system to port 2200 on the infected system. I issued the following command:

```
nc 192.168.62.129 2200
```

Netcat connected and I was prompted with a #:_.. Next I issued the following commands:

```
?login malware <enter>
<enter>
<enter>
x <enter>
```

At the point I typed (x <enter>) I triggered the breakpoint in OllyDbg. Now I knew I was sitting at the TEST EAX, EAX instruction. From the flowchart research I knew I needed the result to be false. The EAX register contained 00000000 as its value which means the next instruction, JE SHORT msrll.0040BC5A, would be taken. This would be a true condition. So I right clicked on the EAX register and selected increment changing the EAX register's value to 00000001. This would now result in a false condition. I pressed F9 and continued running the malware.

Back on the Red Hat 9 system all I had was a cursor, but I was still connected. I typed in the following command:

```
?uptime <enter>
```

This returned the following line:

```
sys: 01h 49m 04s bot; 33m 09s
```

I was now authenticated, but just to make sure I issued the following command:

```
?status <enter>
```

This also returned information. It looks like it worked. I want to try all the commands to see what happens, but before I do that I'm going to patch the malware to bypass the authentication.

The instruction (JE SHORT, msrll.0040BC5A) needs to be replaced with a NOP so that this jump cannot happen since taking the jump is a true condition and I want a false condition to get authenticated.

To patch the malware I clicked on the JE SHORT, msrll.0040BC5A instruction in OllyDbg and hit the space bar. This opened an assembler window with an entry box that I typed in NOP. I made sure the “Fill with NOP’s” box was checked in order to fill the
replaced instruction space properly. I then click assemble and cancel. This replaced the JE instruction with two NOP instructions. To save this patched version of the malware I right clicked on the assembler pane, select “Copy to executable”, select “All modifications”, clicked “Copy all”, and then a new disassembler pane opened. I right click on the pane and select “Save file”. This prompted me for the file name to save as. I used the name msrll-patched.exe and saved it.

Now I had a patched version of the program. I saved this version in place of the current msrll.exe in the c:\windows\system32\mfm directory.

**Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>?insmod</td>
<td>Install loadable modules</td>
<td>?insmod: &lt;mod name&gt;</td>
</tr>
<tr>
<td>?rmmod</td>
<td>Remove loadable modules</td>
<td>?rmmod: &lt;mod name&gt;</td>
</tr>
<tr>
<td>?lsmod</td>
<td>List loadable modules</td>
<td>(No response)</td>
</tr>
<tr>
<td>?ping</td>
<td>ping DoS attack</td>
<td>?ping &lt;ip&gt; &lt;total secs&gt; &lt;p size&gt; &lt;delay&gt; [port]</td>
</tr>
<tr>
<td>?udp</td>
<td>udp DoS attack</td>
<td>?udp &lt;ip&gt; &lt;total secs&gt; &lt;p size&gt; &lt;delay&gt; [port]</td>
</tr>
<tr>
<td>?syn</td>
<td>syn DoS attack</td>
<td>?syn &lt;ip&gt; &lt;port&gt; &lt;t_time&gt; &lt;delay&gt;</td>
</tr>
<tr>
<td>?smurf</td>
<td>smurf DoS attack</td>
<td>?smurf &lt;ip&gt; &lt;p size&gt; &lt;duration&gt; &lt;delay&gt;</td>
</tr>
<tr>
<td>?jolt</td>
<td>jolt DoS attack</td>
<td>?jolt &lt;ip&gt; &lt;duration&gt; &lt;delay&gt;</td>
</tr>
<tr>
<td>?si</td>
<td>Displays system information</td>
<td>WINXP (u:James) mem:(52/127) 58% Genuine Intel (null)</td>
</tr>
<tr>
<td>?ssl</td>
<td>Something to do with ssl</td>
<td>?ssl: -1</td>
</tr>
<tr>
<td>?login</td>
<td>Login command must be ?login &lt;enter&gt; user &lt;enter&gt; password &lt;enter&gt;</td>
<td>(No response on unsuccessful login)</td>
</tr>
<tr>
<td>?uptime</td>
<td>Displays system and bot uptime stats</td>
<td>sys: 58m 16s bot: 27m 14s</td>
</tr>
<tr>
<td>?reboot</td>
<td>Reboots the system</td>
<td>later!</td>
</tr>
<tr>
<td>?status</td>
<td>Displays status info</td>
<td>service:N user:James inet connection:Y contype: Lan reboot privs:Y</td>
</tr>
<tr>
<td>?jump</td>
<td>?</td>
<td>(No response)</td>
</tr>
<tr>
<td>?nick</td>
<td>Change nick on irc channel that bot is connected to, but you must first select the irc sock to perform the command on</td>
<td>Set an irc sock to perform ?nick command on Type .sklist to view current sockets, then .dccsk &lt;#&gt;</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Output</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td><code>?echo</code></td>
<td>Echoes what you type after <code>?echo</code> to the screen - if you typed <code>?echo hello &lt;enter&gt;</code> the response would be what is in the response column to the right</td>
<td><code>hello</code></td>
</tr>
<tr>
<td><code>?hush</code></td>
<td>?</td>
<td>no file name in x</td>
</tr>
<tr>
<td><code>?wget</code></td>
<td>When you type <code>?wget &lt;enter&gt;</code> there is no response, but if you type <code>?wget x &lt;enter&gt;</code> the response will be what is in the response column to the right</td>
<td></td>
</tr>
<tr>
<td><code>?join</code></td>
<td>Irc join command to join other channels</td>
<td>(No output to screen - need to use the <code>?sklist</code> and <code>?dccsk</code> commands to find and connect to an irc sock first)</td>
</tr>
<tr>
<td><code>?op</code></td>
<td>Irc command to become channel operator</td>
<td><code>?op bad args</code></td>
</tr>
<tr>
<td><code>?aop</code></td>
<td>Not sure what this does, but you can add or remove a host by the <code>?aop + &lt;host&gt;</code> or <code>?aop - &lt;host&gt;</code></td>
<td>usage: <code>?aop +/- &lt;host&gt;</code></td>
</tr>
<tr>
<td><code>?akick</code></td>
<td>Not sure what this one does either, but has similar syntax as <code>?aop</code></td>
<td>usage: <code>?akick +/- &lt;host&gt;</code></td>
</tr>
<tr>
<td><code>?part</code></td>
<td>Irc command to leave a channel</td>
<td>(No output to screen)</td>
</tr>
<tr>
<td><code>?dump</code></td>
<td>?</td>
<td>(No response)</td>
</tr>
<tr>
<td><code>?set</code></td>
<td>Shows the <code>jtram.conf</code> contents plus can change settings by issuing the <code>?set &lt;setting&gt; &lt;value&gt;</code></td>
<td>set jtr.bin msrll.exe set jtr.home mfm set bot.port 2200 set jtr.id run5 set irc.quit set servers collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.zxy0.com:8080 set irc.chan #mils set pass $1$KZLPLKdf$W8kI8Jr1X8DOHZsmIp9qq0 set dcc.pass $1$KZLPLKdf$55isA1TtvamR7bjAdBziX</td>
</tr>
<tr>
<td><code>?die</code></td>
<td>Kills msrll.exe process</td>
<td>(No output to screen)</td>
</tr>
<tr>
<td><code>?md5p</code></td>
<td>Displays the salt and md5 hash of whatever is typed in as <code>&lt;pass&gt;</code> parameter</td>
<td><code>?md5p &lt;pass&gt; &lt;salt&gt;</code></td>
</tr>
<tr>
<td><code>?free</code></td>
<td>?</td>
<td>usage: <code>?free &lt;cmd&gt;</code></td>
</tr>
<tr>
<td><code>?raw</code></td>
<td>?</td>
<td>(No output to screen)</td>
</tr>
<tr>
<td><code>?update</code></td>
<td>Possibly a command to update bot</td>
<td><code>?update &lt;url&gt; &lt;id&gt;</code></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>?hostname</td>
<td>Displays hostname and ip address</td>
<td>host: xxx.localdomain ip: 192.168.62.129</td>
</tr>
<tr>
<td>?fif</td>
<td>?</td>
<td>(No response)</td>
</tr>
<tr>
<td>?ffif</td>
<td>?</td>
<td>(No response)</td>
</tr>
<tr>
<td>?del</td>
<td>Delete a file</td>
<td></td>
</tr>
<tr>
<td>?pwd</td>
<td>Display the current directory</td>
<td>c:\windows\system32\mfm</td>
</tr>
<tr>
<td>?play</td>
<td>?</td>
<td>(null): somefile</td>
</tr>
<tr>
<td>?copy</td>
<td>Copy a file</td>
<td></td>
</tr>
<tr>
<td>?move</td>
<td>Move a file</td>
<td></td>
</tr>
<tr>
<td>?dir</td>
<td>Display directory</td>
<td>(Displays directory listing of current directory)</td>
</tr>
<tr>
<td>?sums</td>
<td>Display the md5 hashes for all files in current directory</td>
<td>(Displays a file listing with it’s md5 hash value next to it)</td>
</tr>
<tr>
<td>?ls</td>
<td>Displays directory listing of current directory</td>
<td>(Displays directory listing of current directory)</td>
</tr>
<tr>
<td>?cd</td>
<td>Changes directory</td>
<td></td>
</tr>
<tr>
<td>?rm</td>
<td>Removes a directory</td>
<td></td>
</tr>
<tr>
<td>?mkdir</td>
<td>Makes a new directory</td>
<td></td>
</tr>
<tr>
<td>?run</td>
<td>Run a program (hidden) - syntax: ?run c:\windows\system32\notepad.exe &lt;enter&gt;</td>
<td>(Example: ?run c:\windows\system32\notepad.exe)</td>
</tr>
<tr>
<td>?exec</td>
<td>irc command exec</td>
<td>(No response)</td>
</tr>
<tr>
<td>?ps</td>
<td>Display all processes running and their PID's</td>
<td>(lists active processes running on infected system)</td>
</tr>
<tr>
<td>?kill</td>
<td>Kill a process - ?kill 1448 &lt;enter&gt;</td>
<td>pid 1448 killed</td>
</tr>
<tr>
<td>?killall</td>
<td>?</td>
<td>(No response)</td>
</tr>
<tr>
<td>?crash</td>
<td>Crashes system</td>
<td>(No output to screen)</td>
</tr>
<tr>
<td>?dcc</td>
<td>irc command dcc direct connections to remote clients</td>
<td>(No response)</td>
</tr>
<tr>
<td>?get</td>
<td>?</td>
<td>(No response)</td>
</tr>
<tr>
<td>?say</td>
<td>irc say command - I believe this message would be said non-privately</td>
<td>usage: ?say &lt;target&gt; &quot;text&quot;</td>
</tr>
<tr>
<td>?msg</td>
<td>irc command to send private message to nick or list of nicks</td>
<td>usage: ?msg &lt;target&gt; &quot;text&quot;</td>
</tr>
<tr>
<td>?kb</td>
<td>?</td>
<td>?kb &lt;nick&gt; &lt;chan&gt;</td>
</tr>
<tr>
<td>?sklist</td>
<td>Display current socks</td>
<td>(A display numbering the different socks and connection information like ip address, nick, and irc chan)</td>
</tr>
<tr>
<td>?unset</td>
<td>Un sets a set command refer to ?set command</td>
<td>(Example: ?unset pass &lt;enter&gt;)</td>
</tr>
<tr>
<td>?uattr</td>
<td>?</td>
<td>usage: ?uattr &lt;nick&gt; &lt;chan&gt;</td>
</tr>
<tr>
<td>?dccsk</td>
<td>Set the irc sock to use</td>
<td>usage: ?dccsk &lt;socks #&gt;</td>
</tr>
</tbody>
</table>
Analysis Wrap-Up

Once the malware is executed on a system, it will copy itself to the c:\%systemroot%\system32\mfm directory. It will then create jtram.conf file in the same directory. The jtram.conf file contains the encrypted configuration settings of the bot. The malware sets up a service, Rll enhanced drive, which starts automatically when the system is booted and runs with local system authority. The bot then attempts to connect to the irc server collective7.zxy0.com first on port 6667 then port 9999 and then port 8080. Once the bot is connected to the irc server it joins the #mils channel with a randomly generated nick. At this point the bot is awaiting orders from the bot commander/creator.

Analysis shows this bot is capable of receiving a connection on port 2200 using telnet or netcat. Connecting to this port presents a prompt #:_ awaiting authentication using the ?login command. Authentication allows you to execute numerous commands. These commands can setup denial of service attacks, run programs hidden to the user, update the bot, send irc related commands to the irc server from the bot system, get information on the infected system resources and configuration, kill processes, transfer files and change the bot configuration.

This bot army might have been created to sell or trade for something in return, to attack a specific website, to speed the spread of a future virus or worm, to steal financial information, to harvest email address, to spam, and the list goes on. This bot appears to have upgradeability build in so its purpose could change.

The first defensive tactic is to use a firewall to block outgoing traffic on port 6667, 9999, and 8080. If other outgoing ports are not being used they should be blocked as well to prevent the bot from reporting in. Next block incoming traffic to port 2200 as well as any other ports that are not required. Now you’re left with the existing infected systems to detect and clean. To find the infected systems first run antivirus software, but if that doesn’t detect it then you could run a port scanner on the network like nmap and look for systems listening on port 2200 or other odd ports. You could manually go to each one of the suspect systems and kill the msrll.exe process and remove the c:\%systemroot%\system32\mfm directory and files from the system. You’d also have the rll enhanced drive service to deal with by at least setting it to manual startup instead of automatic. You could also script the removal of the files since nothing should live in the mfm directory. The following batch file presents as an example. It would not matter if this ran on a system that wasn’t infected; it would just not delete
anything.

```plaintext
@echo off
c:
el /Q c:\windows\system32\mfm\*.*
el /Q c:\winnt\system32\mfm\*.*
attrib –r c:\windows\system32\mfm
attrib –r c:\winnt\system32\mfm
rd c:\windows\system32\mfm
rd c:\winnt\system32\mfm
```

Additional things you could do to prevent future attacks is to install a personal firewall on each system that can detect when unauthorized applications try to communicate on the network or Internet. Keep the antivirus software up to date. Monitor on regular bases the listening ports on each system. Create Snort IDS signatures to detect this activity.
List of Resources

Software Tools

vmware Workstation Product Web Site. 8 Dec. 2004

Symantec Ghost Product Web Site. 8 Dec. 2004
<http://sea.symantec.com/content/product.cfm?productid=9>

Free Software Foundation, Inc. md5sum Web Site. 8 Dec 2004
<http://www.gnu.org/software/textutils/textutils.html>

Foundstone, Inc. BinText Web Site. 8 Dec. 2004
<http://www.foundstone.com/proddesc/bintext.htm>

TiANWEi. Regshot Download Web Site. 8 Dec. 2004
<http://www.pcworld.com/downloads/file_description/0,fid,19540,00.asp>

Russinovich, Mark and Cogswell, Bryce. Filemon Web Site. 8 Dec. 2004
<http://www.sysinternals.com/ntw2k/source/filemon.shtml>

Russinovich, Mark and Cogswell, Bryce. Regmon Web Site. 8 Dec. 2004
<http://www.sysinternals.com/ntw2k/source/regmon.shtml>

Russinovich, Mark. TDIMon Web Site. 8 Dec. 2004
<http://www.sysinternals.com/ntw2k/freeware/tdimon.shtml>

Russinovich, Mark. Process Explorer Web Site. 8 Dec. 2004
<http://www.sysinternals.com/ntw2k/freeware/procexp.shtml>

Russinovich, Mark and Cogswell, Bryce. Autoruns Web Site. 8 Dec. 2004
<http://www.sysinternals.com/ntw2k/freeware/autoruns.shtml>

UltraEdit Web Site. 8 Dec. 2004
<http://www.ultraedit.com/index.php?name=Content&pa=showpage&pid=10>


y0da. ASPACKDIE Web Site. 8 Dec. 2004 <http://scifi.pages.at/yoda9k/proggies.htm>


Yuschuk, Oleh. OllyDbg Web Site. 8 Dec. 2004 <http://home.t-online.de/home/ollydbg/>


### Upcoming SANS Forensics Training

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS San Francisco Spring 2020</td>
<td>San Francisco, CA</td>
<td>Mar 16, 2020 - Mar 27, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Norfolk 2020</td>
<td>Norfolk, VA</td>
<td>Mar 16, 2020 - Mar 21, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Secure Singapore 2020</td>
<td>Singapore, Singapore</td>
<td>Mar 16, 2020 - Mar 28, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Seattle Spring 2020</td>
<td>Seattle, WA</td>
<td>Mar 23, 2020 - Mar 28, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Mentor Session - FOR508</td>
<td>Sao Paulo, Brazil</td>
<td>Mar 25, 2020 - Mar 28, 2020</td>
<td>Mentor</td>
</tr>
<tr>
<td>SANS Frankfurt March 2020</td>
<td>Frankfurt, Germany</td>
<td>Mar 30, 2020 - Apr 04, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS vLive - FOR508: Advanced Incident Response, Threat Hunting, and Digital Forensics</td>
<td>FOR508 - 202003,</td>
<td>Mar 31, 2020 - May 07, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>SANS 2020</td>
<td>Orlando, FL</td>
<td>Apr 03, 2020 - Apr 10, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Minneapolis 2020</td>
<td>Minneapolis, MN</td>
<td>Apr 14, 2020 - Apr 19, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Bethesda 2020</td>
<td>Bethesda, MD</td>
<td>Apr 14, 2020 - Apr 19, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS London April 2020</td>
<td>London, United Kingdom</td>
<td>Apr 20, 2020 - Apr 25, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels April 2020</td>
<td>Brussels, Belgium</td>
<td>Apr 20, 2020 - Apr 25, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Baltimore Spring 2020</td>
<td>Baltimore, MD</td>
<td>Apr 27, 2020 - May 02, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Bucharest May 2020</td>
<td>Bucharest, Romania</td>
<td>May 04, 2020 - May 09, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Security West 2020</td>
<td>San Diego, CA</td>
<td>May 06, 2020 - May 13, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Hong Kong 2020</td>
<td>Hong Kong, Hong Kong</td>
<td>May 11, 2020 - May 16, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Community SANS Scottsdale FOR508</td>
<td>Scottsdale, AZ</td>
<td>May 11, 2020 - May 16, 2020</td>
<td>Community SANS</td>
</tr>
<tr>
<td>SANS Amsterdam May 2020</td>
<td>Amsterdam, Netherlands</td>
<td>May 11, 2020 - May 18, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Antonio 2020</td>
<td>San Antonio, TX</td>
<td>May 17, 2020 - May 22, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Northern Virginia- Alexandria 2020</td>
<td>Alexandria, VA</td>
<td>May 17, 2020 - May 22, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Autumn Sydney 2020</td>
<td>Sydney, Australia</td>
<td>May 18, 2020 - May 23, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Nashville Spring 2020</td>
<td>Nashville, TN</td>
<td>May 26, 2020 - May 31, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Atlanta Spring 2020</td>
<td>Atlanta, GA</td>
<td>May 26, 2020 - May 31, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS London June 2020</td>
<td>London, United Kingdom</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Chicago Spring 2020</td>
<td>Chicago, IL</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS FOR508 Milan June 2020 (in Italian)</td>
<td>Milan, Italy</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
</tbody>
</table>