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

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: msrll.exe

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

**Abstract**  

**Laboratory Setup**  

**Hardware**  

**Networking**  

**Software**  

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

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

**Properties of the Malware Specimen**  

*Operating Systems Effected:*  

*Strings Embedded in File:*  
- Fig. A - Interesting strings packed msrl.exe  
- Fig. B - Interesting strings unpacked msrl.exe  

**Behavioral Analysis**  

*Preparation for Infection*  
- Fig. 1 - Regshot  
- Inf ection  
- Fig. 2 - Regshot Compare Results  
- Fig. 3 - FileMon Results  
- Fig. 4 - RegMon Results  
- Fig. 5 - TDI Mon Results  
- Fig. 6 - Snort Capture 2  
- Fig. 7 - Hosts File Modification  
- Fig. 8 - Snort Capture 3  
- Fig. 9 - Snort Capture 4  

**Code Analysis**  

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

---

Page 2 of 48
Fig. 11 - IDA Pro Flowchart
Commands
Analysis Wrap-Up
List of Resources
Software Tools
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 msrll.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](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  

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](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.yeah.net)  
[http://regshot.ist.md](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](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](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](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 UltraEdit-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 (Not Active)

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

**ASPACKDIE**
Version: 1.41
By: y0da
http://y0da.cjb.net

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

**IDA Pro**
Version: 4.6
By: DataRescue
http://www.datarescue.com

Description:
IDA Pro is a disassembler and debugger. I used IDA Pro to sift through the msrll.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](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](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](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>.adata</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>?rmmmod</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>
</tbody>
</table>
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 already loaded
00001416 00401416 0 %s: %s err %u
000015B5 004015B5 0 %s:%s not found
000015C5 004015C5 0 %s: unloading %s
000016AE 004016AE 0 [%u]: %s hinst:%x
00001712 00401712 0 unloading %s
000017A0 004017A0 0 %s: invalid_addr: %s
000017B5 004017B5 0 %s [port]
000018E8 004018E8 0 finished %s
00001A40 00401A40 0 %s <ip> <port> <t_time> <delay>
00001B32 00401B32 0 sockopt: %u
00001B3E 00401B3E 0 sendto err: %u
00001B4D 00401B4D 0 sockraw: %u
00001B59 00401B59 0 syn: done
00001FBC 00401FBC 0 %s <ip> <duration> <delay>
00002096 00402096 0 sendto: %u
000020A2 004020A2 0 jolt2: done
00002260 00402260 0 %s <ip> <p size> <duration> <delay>
00002356 00402356 0 Err: %u
0000235E 0040235E 0 smurf done
000025DE 004025DE 0 &err: %u
00002753 00402753 0 ping
00002759 00402759 0 udph
0000275E 0040275E 0 syn
00002763 00402763 0 smurf
0000276A 0040276A 0 jolt
000028B0 004028B0 0 PONG: %s
0000299D 0040299D 0 %s!%s@%s
00002A3D 00402A3D 0 %s!%s
00002B7D 00402B7D 0 NICK %s
00002C56 00402C56 0 MODE
00002E34 00402E34 0 ?bu
00002F28 00402F28 0 irc.pre
000032AC 004032AC 0 %s
000032B0 004032B0 0 %s
000032B5 004032B5 0 %s
000032B9 004032B9 0 %s
000032BF 004032BF 0 %s
000032C0 004032C0 0 %s
000032C7 004032C7 0 %s
000032CC 004032CC 0 %s
000032D2 004032D2 0 %s
000032D9 004032D9 0 %s
000032E1 004032E1 0 NICK %s
000032EB 004032EB 0 NICK
000032F0 004032F0 0 %s %s
0000345C 0040345C 0 CmP
000036B0 004036B0 0 irc.chan
00003775 00403775 0 %s %s
...
© SANS Institute 2000 - 2005

Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46

00005760 00405760 0 irc.pre
00005781 00405781 0 Set an irc sock to perform %s command on
000057AB 004057AB 0  Type
000057B3 004057B3 0  %csklist
000057BC 004057BC 0  to view current sockets, then
000057DC 004057DC 0  %cdccsk
000057E4 004057E4 0  <%>
00005BB4 00405BB4 0  %s: dll loaded
00005BC3 00405BC3 0  %s: %d
000059E1 004059E1 0  said %s to %s
000059EF 004059EF 0  usage: %s <target> "text"
00005A74 00405A74 0  %s not on %s
00005AB1 00405AB1 0  usage: %s <nick> <chan>
00005B18 00405B18 0  PASS
00005B20 00405B20 0  %s logged in
00005BA2 00405BA2 0  sys: %s bot: %s
00005BB2 00405BB2 0  performance counter not avail
00005C2B 00405C2B 0  usage: %s <cmd>
00005C3B 00405C3B 0  %s free'd
00005D40 00405D40 0  unable to free %s
00005DAE 00405DAE 0  later!
00005DB4 00405DB4 0  %s not on %s
00005E1E 00405E1E 0  syscall failed: code %u
00005F8F 00405F8F 0  %s: %s
00005F96 00405F96 0  %s: somefile
000060D4 004060D4 0  host: %s ip: %s
0000612C 0040612C 0  WIN%u (u:%u)%s%s mem:(%u/%u) %u%% %s %s
00006282 00406282 0  9x
000065CB 004065CB 0  %s: %s (%u)
00006708 00406708 0  NICK
000067E8 004067E8 0  %s[%u] %s
000067F2 004067F2 0  %s removed
000067FD 004067FD 0  couldn't find %s
00006816 00406816 0  %s already in list
0000682A 0040682A 0  usage: %s +/- <host>
000069EB 004069EB 0  jtram.conf
000069F6 004069F6 0  %s /t %s
000069FF 004069FF 0  jtr.home
00006AE0 00406AE0 0  %s%s
00006A0E 00406A0E 0  %s: possibly failed: code %u
Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46
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

Block Chaining Modes:

- CFB
- OFB
- CTR

PRNG:

- Yarrow
- SPRNG
- RC4

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

---

%| %$ %T %h %p %4 %\ %t %d %}{ %g %X %< %D %H %i %L %8 %P %x %,

---

Page 20 of 48
00012341 00412341 0 %\ 
00012351 00412351 0 %h 
00012361 00412361 0 %B 
00012371 00412371 0 %j 
00012431 00412431 0 %t 
00012451 00412451 0 %D 
00012461 00412461 0 %p 
00012491 00412491 0 %P 
000124A1 004124A1 0 % 
000124B1 004124B1 0 %x 
000124F1 004124F1 0 %O 
00012501 00412501 0 %S 
00012511 00412511 0 %4 
00012521 00412521 0 %, 
00012591 00412591 0 %X 
000125A1 004125A1 0 %T 
000125E1 004125E1 0 %H 
00012641 00412641 0 %L 
00012671 00412671 0 %@ 
00012681 00412681 0 %l 
00012691 00412691 0 %d 
000126A1 004126A1 0 %(
000126C1 004126C1 0 %<
000130B0 004130B0 0 <ip> <total secs> <p size> <delay>
00013350 00413350 0 modem
00013358 00413358 0 Lan
0001335E 0041335E 0 Proxy
00013366 00413366 0 ??
0001336B 0041336B 0 none
00013390 00413390 0 m220 1.0 #2730 Mar 16 11:47:38 2004
000133D4 004133D4 0 unable to %s %s (err: %u)
00013420 00413420 0 unable to kill %s (%u)
00013437 00413437 0 %s killed (pid:%u)
00013470 00413470 0 AVICAP32.dll
0001347D 0041347D 0 unable to kill %u (%u)
00013494 00413494 0 pid %u killed
000134A2 004134A2 0 error!
000134A9 004134A9 0 ran ok
000134B0 004134B0 0 MODE %s +o %s
000134BF 004134BF 0 set %s %s
00013600 00413600 0 Mozilla/4.0
0001360C 0041360C 0 Accept: */*
0001361C 0041361C 0 <DIR>
0001362B 0041362B 0 Could not copy %s to %s
00013643 00413643 0 %s copied to %s
00013653 00413653 0 0123456789abcdef
00013664 00413664 0 %s unset
0001366D 0041366D 0 unable to unset %s
00013AD4 00413AD4 0 (s) %s
00013ADD 00413ADD 0 %s %s
00013B30 00413B30 0 #:
00013BA0 00413BA0 0 libss32.dll
00013BAD 00413BAD 0 libeay32.dll
00013BE0 00413BE0 0 <die|join|part|raw|msg>
0011B67A 0051B67A 0 AdjustTokenPrivileges 
0011B692 0051B692 0 CloseServiceHandle 
0011B6AA 0051B6AA 0 CreateServiceA 
0011B6BE 0051B6BE 0 CryptAcquireContextA
<table>
<thead>
<tr>
<th>Address</th>
<th>Offset</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011B6D6</td>
<td>0</td>
<td>CryptGenRandom</td>
</tr>
<tr>
<td>0011B6EA</td>
<td>0</td>
<td>CryptReleaseContext</td>
</tr>
<tr>
<td>0011B702</td>
<td>0</td>
<td>GetUserNamed</td>
</tr>
<tr>
<td>0011B712</td>
<td>0</td>
<td>LookupPrivilegeValueA</td>
</tr>
<tr>
<td>0011B72A</td>
<td>0</td>
<td>OpenProcessToken</td>
</tr>
<tr>
<td>0011B73E</td>
<td>0</td>
<td>OpenSCManagerA</td>
</tr>
<tr>
<td>0011B752</td>
<td>0</td>
<td>RegCloseKey</td>
</tr>
<tr>
<td>0011B762</td>
<td>0</td>
<td>RegCreateKeyExA</td>
</tr>
<tr>
<td>0011B776</td>
<td>0</td>
<td>RegSetValueExA</td>
</tr>
<tr>
<td>0011B78A</td>
<td>0</td>
<td>RegisterServiceCtrlHandlerA</td>
</tr>
<tr>
<td>0011B7AA</td>
<td>0</td>
<td>SetServiceStatus</td>
</tr>
<tr>
<td>0011B7BE</td>
<td>0</td>
<td>StartServiceCtrlDispatcherA</td>
</tr>
<tr>
<td>0011B7DE</td>
<td>0</td>
<td>AddAtomA</td>
</tr>
<tr>
<td>0011B7EA</td>
<td>0</td>
<td>CloseHandle</td>
</tr>
<tr>
<td>0011B7FA</td>
<td>0</td>
<td>CopyFileA</td>
</tr>
<tr>
<td>0011B806</td>
<td>0</td>
<td>CreateDirectoryA</td>
</tr>
<tr>
<td>0011B81A</td>
<td>0</td>
<td>CreateFileA</td>
</tr>
<tr>
<td>0011B82A</td>
<td>0</td>
<td>CreateMutexA</td>
</tr>
<tr>
<td>0011B83A</td>
<td>0</td>
<td>CreatePipe</td>
</tr>
<tr>
<td>0011B84A</td>
<td>0</td>
<td>CreateProcessA</td>
</tr>
<tr>
<td>0011B85E</td>
<td>0</td>
<td>CreateToolhelp32Snapshot</td>
</tr>
<tr>
<td>0011B87A</td>
<td>0</td>
<td>DeleteFileA</td>
</tr>
<tr>
<td>0011B88A</td>
<td>0</td>
<td>DuplicateHandle</td>
</tr>
<tr>
<td>0011B89E</td>
<td>0</td>
<td>EnterCriticalSection</td>
</tr>
<tr>
<td>0011B8B6</td>
<td>0</td>
<td>ExitProcess</td>
</tr>
<tr>
<td>0011B8C6</td>
<td>0</td>
<td>ExitThread</td>
</tr>
<tr>
<td>0011B8D6</td>
<td>0</td>
<td>FileTimeToSystemTime</td>
</tr>
<tr>
<td>0011B8EE</td>
<td>0</td>
<td>FindAtomA</td>
</tr>
<tr>
<td>0011B8FA</td>
<td>0</td>
<td>FindClose</td>
</tr>
<tr>
<td>0011B906</td>
<td>0</td>
<td>FindFirstFileA</td>
</tr>
<tr>
<td>0011B91A</td>
<td>0</td>
<td>FindNextFileA</td>
</tr>
<tr>
<td>0011B92A</td>
<td>0</td>
<td>FreeLibrary</td>
</tr>
<tr>
<td>0011B93A</td>
<td>0</td>
<td>GetAtomNameA</td>
</tr>
<tr>
<td>0011B94A</td>
<td>0</td>
<td>GetCommandLineA</td>
</tr>
<tr>
<td>0011B95E</td>
<td>0</td>
<td>GetCurrentDirectoryA</td>
</tr>
<tr>
<td>0011B966</td>
<td>0</td>
<td>GetCurrentProcess</td>
</tr>
<tr>
<td>0011B98A</td>
<td>0</td>
<td>GetCurrentThreadId</td>
</tr>
<tr>
<td>0011B9A2</td>
<td>0</td>
<td>GetExitCodeProcess</td>
</tr>
<tr>
<td>0011B9BA</td>
<td>0</td>
<td>GetFileSize</td>
</tr>
<tr>
<td>0011B9CA</td>
<td>0</td>
<td>GetFullPathNameA</td>
</tr>
<tr>
<td>0011B9DE</td>
<td>0</td>
<td>GetLastError</td>
</tr>
<tr>
<td>0011B9EE</td>
<td>0</td>
<td>GetModuleFileNameA</td>
</tr>
<tr>
<td>0011BA06</td>
<td>0</td>
<td>GetModuleHandleA</td>
</tr>
<tr>
<td>0011BA1A</td>
<td>0</td>
<td>GetProcAddress</td>
</tr>
<tr>
<td>0011BA2E</td>
<td>0</td>
<td>GetStartupInfoA</td>
</tr>
<tr>
<td>0011BA42</td>
<td>0</td>
<td>GetSystemDirectoryA</td>
</tr>
<tr>
<td>0011BA5A</td>
<td>0</td>
<td>GetSystemInfo</td>
</tr>
<tr>
<td>0011BA6A</td>
<td>0</td>
<td>GetTempPathA</td>
</tr>
<tr>
<td>0011BA7A</td>
<td>0</td>
<td>GetTickCount</td>
</tr>
<tr>
<td>0011BA8A</td>
<td>0</td>
<td>GetVersionExA</td>
</tr>
<tr>
<td>0011BA9A</td>
<td>0</td>
<td>GlobalMemoryStatus</td>
</tr>
<tr>
<td>0011BBB2</td>
<td>0</td>
<td>InitializeCriticalSection</td>
</tr>
<tr>
<td>0011BACE</td>
<td>0</td>
<td>IsBadReadPtr</td>
</tr>
<tr>
<td>0011BADD</td>
<td>0</td>
<td>LeaveCriticalSection</td>
</tr>
<tr>
<td>0011BAF6</td>
<td>0</td>
<td>LoadLibraryA</td>
</tr>
<tr>
<td>0011BB06</td>
<td>0</td>
<td>MoveFileA</td>
</tr>
<tr>
<td>0011BB12</td>
<td>0</td>
<td>OpenProcess</td>
</tr>
<tr>
<td>0011BB22</td>
<td>0</td>
<td>PeekNamedPipe</td>
</tr>
</tbody>
</table>
0011BE3E 0051BE3E 0 strcat
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 WSASStartup
0011BFA2 0051BFA2 0 __WSAFDIsSet
0011BFBE 0051BFBE 0 accept
0011BFBE 0051BFBE 0 bind
0011BFC6 0051BFC6 0 closesocket
0011BFD6 0051BFD6 0 connect
0011BFE2 0051BFE2 0 gethostbyaddr
0011BFF2 0051BFF2 0 gethostbyname
0011C002 0051C002 0 gethostname
0011C012 0051C012 0 getsockname
0011C022 0051C022 0 htonl
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 msvcrt.dll
0011C220 0051C220 0 msvcrt.dll
0011C2F0 0051C2F0 0 SHELL32.DLL
0011C30C 0051C30C 0 USER32.dll
0011C320 0051C320 0 VERSION.dll
0011C340 0051C340 0 WININET.DLL
0011C3B4 0051C3B4 0 WS2_32.DLL
0011D071 0051D071 0 VirtualAlloc
0011D07E 0051D07E 0 VirtualFree
0011D441 0051D441 0 kernel32.dll
0011D44E 0051D44E 0 ExitProcess
0011D45A 0051D45A 0 user32.dll
0011D465   0051D465   0   MessageBoxA
0011D471   0051D471   0   wsprintfA
0011D47A   0051D47A   0   LOADER ERROR
0011D488   0051D488   0   The procedure entry point %s could not be located in the dynamic link library %s
0011D4D9   0051D4D9   0   The ordinal %u could not be located in the dynamic link library %s
0011DF6C   0051DF6C   0   kernel32.dll
0011DF7B   0051DF7B   0   GetProcAddress
0011DF8C   0051DF8C   0   GetModuleHandleA
0011DF9F   0051DF9F   0   LoadLibraryA
0011E074   0051E074   0   advapi32.dll
0011E081   0051E081   0   msvcrtdll
0011E08C   0051E08C   0   msvcrtdll
0011E097   0051E097   0   shell32.dll
0011E0A3   0051E0A3   0   user32.dll
0011E0AE   0051E0AE   0   version.dll
0011E0BA   0051E0BA   0   wininet.dll
0011E0C6   0051E0C6   0   ws2_32.dll
0011E113   0051E113   0   AdjustTokenPrivileges
0011E12B   0051E12B   0   _itoa
0011E133   0051E133   0   __getmainargs
0011E143   0051E143   0   ShellExecuteA
0011E153   0051E153   0   DispatchMessageA
0011E166   0051E166   0   GetFileVersionInfoA
0011E17C   0051E17C   0   InternetCloseHandle
0011E192   0051E192   0   WSAGetLastError
001200C8   005200C8   0   advapi32.dll
001200D7   005200D7   0   AdjustTokenPrivileges
001200EF   005200EF   0   CloseServiceHandle
00120104   00520104   0   CreateServiceA
00120115   00520115   0   CryptAcquireContextA
0012012C   0052012C   0   CryptGenRandom
0012013D   0052013D   0   CryptReleaseContext
00120153   00520153   0   GetUserNameA
00120162   00520162   0   LookupPrivilegeValueA
0012017A   0052017A   0   OpenProcessToken
0012018D   0052018D   0   OpenSCManagerA
0012019E   0052019E   0   RegCloseKey
001201AC   005201AC   0   RegCreateKeyExA
001201BE   005201BE   0   RegSetValueExA
001201CF   005201CF   0   RegisterServiceCtrlHandlerA
001201ED   005201ED   0   SetServiceStatus
00120200   00520200   0   StartServiceCtrlDispatcherA
0012021C   0052021C   0   kernel32.dll
0012022B   0052022B   0   AddAtomA
00120236   00520236   0   CloseHandle
00120244   00520244   0   CopyFileA
00120250   00520250   0   CreateDirectoryA
00120263   00520263   0   CreateFileA
00120271   00520271   0   CreateMutexA
00120280   00520280   0   CreatePipe
0012028D   0052028D   0   CreateProcessA
0012029D   0052029D   0   CreateToolhelp32Snapshot
001202B9   005202B9   0   DeleteFileA
001202C7   005202C7   0   DuplicateHandle
001202D9   005202D9   0   EnterCriticalSection
001202F0   005202F0   0   ExitProcess
<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Memory Address</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>001202FE</td>
<td>005202FE</td>
<td>ExitThread</td>
</tr>
<tr>
<td>0012030B</td>
<td>0052030B</td>
<td>FileTimeToSystemTime</td>
</tr>
<tr>
<td>00120322</td>
<td>00520322</td>
<td>FindAtomA</td>
</tr>
<tr>
<td>0012032E</td>
<td>0052032E</td>
<td>FindClose</td>
</tr>
<tr>
<td>0012033A</td>
<td>0052033A</td>
<td>FindFirstFileA</td>
</tr>
<tr>
<td>0012034B</td>
<td>0052034B</td>
<td>FindNextFileA</td>
</tr>
<tr>
<td>0012035B</td>
<td>0052035B</td>
<td>FreeLibrary</td>
</tr>
<tr>
<td>00120369</td>
<td>00520369</td>
<td>GetAtomNameA</td>
</tr>
<tr>
<td>00120378</td>
<td>00520378</td>
<td>GetCommandLineA</td>
</tr>
<tr>
<td>0012038A</td>
<td>0052038A</td>
<td>GetCurrentDirectoryA</td>
</tr>
<tr>
<td>001203A1</td>
<td>005203A1</td>
<td>GetCurrentProcess</td>
</tr>
<tr>
<td>001203B5</td>
<td>005203B5</td>
<td>GetCurrentThreadId</td>
</tr>
<tr>
<td>001203CA</td>
<td>005203CA</td>
<td>GetExitCodeProcess</td>
</tr>
<tr>
<td>001203DF</td>
<td>005203DF</td>
<td>GetFileSize</td>
</tr>
<tr>
<td>00120400</td>
<td>00520400</td>
<td>GetLastError</td>
</tr>
<tr>
<td>0012040F</td>
<td>0052040F</td>
<td>GetModuleFileNameA</td>
</tr>
<tr>
<td>00120424</td>
<td>00520424</td>
<td>GetModuleHandleA</td>
</tr>
<tr>
<td>00120437</td>
<td>00520437</td>
<td>GetProcAddress</td>
</tr>
<tr>
<td>00120448</td>
<td>00520448</td>
<td>GetStartupInfoA</td>
</tr>
<tr>
<td>0012045A</td>
<td>0052045A</td>
<td>GetSystemDirectoryA</td>
</tr>
<tr>
<td>00120470</td>
<td>00520470</td>
<td>GetSystemInfo</td>
</tr>
<tr>
<td>00120480</td>
<td>00520480</td>
<td>GetTempPathA</td>
</tr>
<tr>
<td>0012048F</td>
<td>0052048F</td>
<td>GetTickCount</td>
</tr>
<tr>
<td>0012049E</td>
<td>0052049E</td>
<td>GetVersionExA</td>
</tr>
<tr>
<td>001204AE</td>
<td>005204AE</td>
<td>GlobalMemoryStatus</td>
</tr>
<tr>
<td>001204C3</td>
<td>005204C3</td>
<td>InitializeCriticalSection</td>
</tr>
<tr>
<td>001204DF</td>
<td>005204DF</td>
<td>IsBadReadPtr</td>
</tr>
<tr>
<td>001204EE</td>
<td>005204EE</td>
<td>LeaveCriticalSection</td>
</tr>
<tr>
<td>00120505</td>
<td>00520505</td>
<td>LoadLibraryA</td>
</tr>
<tr>
<td>00120514</td>
<td>00520514</td>
<td>MoveFileA</td>
</tr>
<tr>
<td>00120520</td>
<td>00520520</td>
<td>OpenProcess</td>
</tr>
<tr>
<td>0012052E</td>
<td>0052052E</td>
<td>PeekNamedPipe</td>
</tr>
<tr>
<td>0012053E</td>
<td>0052053E</td>
<td>Process32First</td>
</tr>
<tr>
<td>0012054F</td>
<td>0052054F</td>
<td>Process32Next</td>
</tr>
<tr>
<td>0012055F</td>
<td>0052055F</td>
<td>QueryPerformanceFrequency</td>
</tr>
<tr>
<td>0012057B</td>
<td>0052057B</td>
<td>ReadFile</td>
</tr>
<tr>
<td>00120586</td>
<td>00520586</td>
<td>ReleaseMutex</td>
</tr>
<tr>
<td>00120595</td>
<td>00520595</td>
<td>RemoveDirectoryA</td>
</tr>
<tr>
<td>001205A8</td>
<td>005205A8</td>
<td>SetConsoleCtrlHandler</td>
</tr>
<tr>
<td>001205C0</td>
<td>005205C0</td>
<td>SetCurrentDirectoryA</td>
</tr>
<tr>
<td>001205D7</td>
<td>005205D7</td>
<td>SetFilePointer</td>
</tr>
<tr>
<td>001205EE</td>
<td>005205EE</td>
<td>SetUnhandledExceptionFilter</td>
</tr>
<tr>
<td>00120606</td>
<td>00520606</td>
<td>Sleep</td>
</tr>
<tr>
<td>0012060E</td>
<td>0052060E</td>
<td>TerminateProcess</td>
</tr>
<tr>
<td>00120621</td>
<td>00520621</td>
<td>WaitForSingleObject</td>
</tr>
<tr>
<td>00120637</td>
<td>00520637</td>
<td>WriteFile</td>
</tr>
<tr>
<td>00120641</td>
<td>00520641</td>
<td>msvcrtdll</td>
</tr>
<tr>
<td>0012064E</td>
<td>0052064E</td>
<td>_itoa</td>
</tr>
<tr>
<td>00120656</td>
<td>00520656</td>
<td>_stat</td>
</tr>
<tr>
<td>0012065E</td>
<td>0052065E</td>
<td>_mbsdup</td>
</tr>
<tr>
<td>00120668</td>
<td>00520668</td>
<td>_strc mpi</td>
</tr>
<tr>
<td>00120671</td>
<td>00520671</td>
<td>msvcrtdll</td>
</tr>
<tr>
<td>0012067E</td>
<td>0052067E</td>
<td>__getmainargs</td>
</tr>
<tr>
<td>0012068E</td>
<td>0052068E</td>
<td>__p__environ</td>
</tr>
<tr>
<td>0012069D</td>
<td>0052069D</td>
<td>__p__fmode</td>
</tr>
<tr>
<td>001206AA</td>
<td>005206AA</td>
<td>__set_app_type</td>
</tr>
<tr>
<td>001206BB</td>
<td>005206BB</td>
<td>_beginthread</td>
</tr>
</tbody>
</table>
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 memmove
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 strdup
001207DF 005207DF 0 strerror
001207E8 005207E8 0 strcpy
001207F1 005207F1 0 _mbscpy
001207FB 005207FB 0 strlen
00120806 00520806 0 strncat
00120810 00520810 0 strncmp
0012081A 0052081A 0 strncpy
00120824 00520824 0 strstr
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.

![Fig. 1](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\ControlSet001\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 1C 00 01 00 00 00 02 80 14 00 FF 01 0F 00 01 01 00 00 00 00 00 00 10 00 00 00 00 05 12 00 00 00 00 00 18 00 FF |
| 01 0F 00 01 02 00 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 06 BD 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 00 00 00 00 00 00 00 00 00 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 00 01 01 00 00 00 00 05 12 00 00 00 |
| HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\Type: 0x00000120 |
| HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\Start: 0x00000002 |
| HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ErrorControl: 0x00000002 |
| HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ImagePath: "C:\WINDOWS\System32\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\Security\Security: 01 00 14 80 90 00 00 00 9C |
| 00 00 00 14 00 00 00 30 00 00 00 02 00 1C 00 01 00 00 00 02 80 14 00 FF 01 0F 00 01 01 00 00 00 00 00 00 10 00 00 00 00 05 12 00 00 00 00 00 18 00 FF |
| 01 0F 00 01 02 00 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 06 BD 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 00 00 00 00 00 00 00 00 00 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 00 01 01 00 00 00 00 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 “Rll 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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:51:07 PM</td>
<td>Msrll.exe</td>
<td>CREATE</td>
<td>C:\WINDOWS\System32\mfm SUCCESS Options: Create Directory Access: All</td>
</tr>
<tr>
<td>2:51:08 PM</td>
<td>Msrll.exe</td>
<td>CREATE</td>
<td>C:\WINDOWS\System32\mfm\msrll.exe SUCCESS Options: OverwriteIf Sequential Access: All</td>
</tr>
<tr>
<td>2:51:08 PM</td>
<td>Msrll.exe</td>
<td>WRITE</td>
<td>C:\WINDOWS\System32\mfm\msrll.exe SUCCESS Offset: 0 Length: 41984</td>
</tr>
<tr>
<td>2:51:29 PM</td>
<td>Msrll.exe</td>
<td>CREATE</td>
<td>C:\WINDOWS\system32\mfm\jtram.conf SUCCESS Options: OverwriteIf Access: All</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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>659.29827310</td>
<td>Msrll.exe</td>
<td>SetValue</td>
<td>HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed SUCCESS</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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:51:27 PM</td>
<td>Msrll.exe</td>
<td>IRP_MJ_CREATE IRP_MJ_CREATE TCP:0.0.0.0:2200 SUCCESS Address Open</td>
<td></td>
</tr>
<tr>
<td>2:51:27 PM</td>
<td>Msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Error Event</td>
<td></td>
</tr>
<tr>
<td>2:51:27 PM</td>
<td>Msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Disconnect Event</td>
<td></td>
</tr>
<tr>
<td>2:51:27 PM</td>
<td>Msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Receive Event</td>
<td></td>
</tr>
<tr>
<td>2:51:27 PM</td>
<td>Msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Expedited Receive</td>
<td></td>
</tr>
<tr>
<td>2:51:27 PM</td>
<td>Msrll.exe</td>
<td>TDI_SET_EVENT_HANDLER TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Chained Receive</td>
<td></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 6C 65 63 74 69 76 65 37 04 7A 78 79 30 03 63 6F 6D 00 00 01 00 01
```

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
```
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:

```
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: 0x8CF5FCD8  Ack: 0x0  Win: 0xFAF0  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

© SANS Institute 2000 - 2005

Author retains full rights.
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](image)

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:

```plaintext
% 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](image)

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 0040BBD6. In this section you can see the following instructions:

```plaintext
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>?clone</td>
<td>’?’</td>
<td>usage ?clone: server[:port] amount</td>
</tr>
<tr>
<td>?clones</td>
<td>?</td>
<td>?clones: [NETWORK</td>
</tr>
<tr>
<td>?login</td>
<td>Login command must be</td>
<td>(No response on unsuccessful login)</td>
</tr>
<tr>
<td>?login</td>
<td>&lt;enter&gt; user &lt;enter&gt; password &lt;enter&gt;</td>
<td></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>Example</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>?echo</td>
<td>echos what you type after ?echo to the screen - if you typed ?echo hello &lt;enter&gt; the response would be what is in the response column to the right</td>
<td>hello</td>
</tr>
<tr>
<td>?hush</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>?wget</td>
<td>When you type ?wget &lt;enter&gt; there is no response, but if you type ?wget x &lt;enter&gt; the response will be what is in the response column to the right</td>
<td>no file name in x</td>
</tr>
<tr>
<td>?join</td>
<td>irc join command to join other channels</td>
<td>(No output to screen - need to us the ?sklist and ?dccsk commands to find and connect to an irc sock first)</td>
</tr>
<tr>
<td>?op</td>
<td>irc command to become channel operator</td>
<td>?op bad args</td>
</tr>
<tr>
<td>?aop</td>
<td>Not sure what this does, but you can add or remove a host by the ?aop + &lt;host&gt; or ?aop - &lt;host&gt;</td>
<td>usage: ?aop +/- &lt;host&gt;</td>
</tr>
<tr>
<td>?akick</td>
<td>Not sure what this one does either, but has similar syntax as ?aop</td>
<td>usage: ?akick +/- &lt;host&gt;</td>
</tr>
<tr>
<td>?part</td>
<td>irc command to leave a channel</td>
<td>(No output to screen)</td>
</tr>
<tr>
<td>?dump</td>
<td>?</td>
<td>(No response)</td>
</tr>
</tbody>
</table>
| ?set    | shows the jtram.conf contents plus can change settings by issuing the ?set <setting> <value> | set jtr.bin msrll.exe
set jtr.name 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$W8kl8Jr1X8DOHZsmlp9qq0
set dcc.pass $1$KZLPLKDf$55isA1fTvamR7bjAdBziX |
<p>| ?die    | Kills msrll.exe process | (No output to screen) |
| ?md5p   | Displays the salt and md5 hash of whatever is typed in as &lt;pass&gt; parameter | ?md5p &lt;pass&gt; &lt;salt&gt; |
| ?free   | ?         | usage: ?free &lt;cmd&gt; |
| ?raw    | ?         | (No output to screen) |
| ?update | Possibly a command to update bot | ?update &lt;url&gt; &lt;id&gt; |</p>
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>?hostname</code></td>
<td>Displays hostname and ip address</td>
<td>host: xxx.localdomain ip: 192.168.62.129</td>
</tr>
<tr>
<td><code>?fif</code></td>
<td>(No response)</td>
<td></td>
</tr>
<tr>
<td><code>?liff</code></td>
<td>(No response)</td>
<td></td>
</tr>
<tr>
<td><code>?del</code></td>
<td>Delete a file</td>
<td></td>
</tr>
<tr>
<td><code>?pwd</code></td>
<td>Display the current directory</td>
<td>c:\windows\system32\mfm</td>
</tr>
<tr>
<td><code>?play</code></td>
<td>(null): somefile</td>
<td></td>
</tr>
<tr>
<td><code>?copy</code></td>
<td>Copy a file</td>
<td></td>
</tr>
<tr>
<td><code>?move</code></td>
<td>Move a file</td>
<td></td>
</tr>
<tr>
<td><code>?dir</code></td>
<td>Display directory</td>
<td>(Displays directory listing of current directory)</td>
</tr>
<tr>
<td><code>?sums</code></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><code>?ls</code></td>
<td>Displays directory listing of current directory</td>
<td>(Displays directory listing of current directory)</td>
</tr>
<tr>
<td><code>?cd</code></td>
<td>Changes directory</td>
<td></td>
</tr>
<tr>
<td><code>?rmdir</code></td>
<td>Removes a directory</td>
<td></td>
</tr>
<tr>
<td><code>?mkdir</code></td>
<td>Makes a new directory</td>
<td></td>
</tr>
</tbody>
</table>
| `?run` | Run a program (hidden) - syntax: `?run c:\windows\system32\notepad.exe <enter>` | (Example: `?run c:\windows\system32\notepad.exe`)
`?run: ran ok (4022304)` |
| `?exec` | irc command `exec` | (No response) |
| `?ps` | Display all processes running and their PID's | (lists active processes running on infected system) |
| `?kill` | Kill a process - `?kill 1448 <enter>` | pid 1448 killed |
| `?killall` | ? | (No response) |
| `?crash` | Crashes system | (No output to screen) |
| `?dcc` | irc command `dcc` direct connections to remote clients | (No response) |
| `?get` | ? | (No response) |
| `?say` | irc say command - I believe this message would be said non-privately | usage: `?say <target> "text"` |
| `?msg` | irc command to send private message to nick or list of nicks | usage: `?msg <target> "text"` |
| `?kb` | ? | `?kb <nick> <chan>` |
| `?sklist` | Display current socks | (A display numbering the different socks and connection information like ip address, nick, and irc chan) |
| `?unset` | Unsets a set command refer to `?set command` | (Example: `?unset pass <enter>`)
This will remove the set pass parameter |
| `?uattr` | ? | usage: `?uattr <nick> <chan>` |
| `?dcsck` | Set the irc sock to use | usage: `?dcsck <socks #>` |
| ?con | irc command (server operator command) connect two servers together | (No response) |
| ?killsk | Kills a irc sock - Example: ?killsk 1 <enter> | closing 1 [collective7.zxy0.com:6667] |

**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
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.

```bash
@echo off
cls
echo Ready to delete mfm directory...
pause
c:
del /Q c:\windows\system32\mfm\*.*
del /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
```
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>

IDA Pro 4.6 Product Web Site. 8 Dec. 2004
<http://www.datarescue.com/idabase/index.htm>

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


<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Dates</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Upcoming SANS Forensics Training</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SANS Brussels April Live Online 2020</td>
<td>Brussels, Belgium</td>
<td>Apr 20, 2020 - Apr 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS London April Live Online 2020</td>
<td>London, United Kingdom</td>
<td>Apr 20, 2020 - Apr 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Apr 27</td>
<td>Baltimore, MD</td>
<td>Apr 27, 2020 - May 02, 2020</td>
</tr>
<tr>
<td>Hunting, and Digital Forensics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>May 4</td>
<td></td>
<td>May 04, 2020 - May 09, 2020</td>
</tr>
<tr>
<td>Live Online - FOR500: Windows Forensic Analysis</td>
<td>United Arab Emirates</td>
<td>May 06, 2020 - Jun 12, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>Hunting, Analysis, and Incident Response</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis Tools and Techniques</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SANS Security West 2020</td>
<td>San Diego, CA</td>
<td>May 11, 2020 - May 16, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Hong Kong SE Asia Live Online 2020</td>
<td>Hong Kong, Hong Kong</td>
<td>May 11, 2020 - May 22, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Mid-May</td>
<td>Alexandria, VA</td>
<td>May 17, 2020 - May 22, 2020</td>
</tr>
<tr>
<td>Autumn Australia Live Online 2020</td>
<td>Sydney, Australia</td>
<td>May 18, 2020 - May 29, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Live Online - FOR508: Advanced Incident Response, Threat</td>
<td>United Arab Emirates</td>
<td>May 19, 2020 - Jun 06, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>Hunting, and Digital Forensics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Live Online - FOR578: Cyber Threat Intelligence</td>
<td>United Arab Emirates</td>
<td>May 27, 2020 - Jun 26, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Jun 1</td>
<td>IL</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
</tr>
<tr>
<td>SANSFIRE 2020</td>
<td>DC</td>
<td>Jun 13, 2020 - Jun 20, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Hunting, and Digital Forensics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SANS Asia Pacific Live Online 2020</td>
<td>Singapore</td>
<td>Jun 22, 2020 - Jun 27, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Hunting, Analysis, and Incident Response</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cyber Defence Australia Online 2020</td>
<td>Australia</td>
<td>Jun 22, 2020 - Jul 04, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Analysis Tools and Techniques</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Live Online - FOR508: Advanced Incident Response, Threat</td>
<td>United Arab Emirates</td>
<td>Jul 09, 2020 - Aug 17, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>Hunting, and Digital Forensics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hunting, Analysis, and Incident Response</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis Tools and Techniques</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DFIR Summit &amp; Training 2020</td>
<td>Virtual - US Eastern,</td>
<td>Jul 16, 2020 - Jul 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Jul 27</td>
<td>NC</td>
<td>Jul 27, 2020 - Aug 01, 2020</td>
</tr>
<tr>
<td>Live Online - FOR578: Cyber Threat Intelligence</td>
<td>United Arab Emirates</td>
<td>Jul 28, 2020 - Aug 27, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>South by Southeast Asia Online 2020</td>
<td>Singapore</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 3 MT</td>
<td>CA</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
</tr>
</tbody>
</table>