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)" at http://digital-forensics.sans.org/events/
GIAC Reverse Engineering Malware (GREM) Practical Assignment
Version 1.0

Malware: msril.exe

ILOT XII
James M. Balcik

12/8/2004
Fig. 11 - IDA Pro Flowchart

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 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 sniffer 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 msrl.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 msrl.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

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](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](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](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  

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

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>.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>?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>
</tbody>
</table>
Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46

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)
00001416 00401416 0 %s: %s not found
000015C5 004015C5 0 %s: unloading %s
000016AE 004016AE 0 [%u]: %s hinst:%x
00001712 00401712 0 unloading %s
000017B5 004017B5 0 %s: invalid_addr: %s
000018E8 004018E8 0 finished %s
00001A40 00401A40 0 %s <ip> <port> <t_time> <delay>
00001B32 00401B32 0 sockopt: %u
00001B4D 00401B4D 0 %s <ip> <p size> <duration> <delay>
00001C5E 00401C5E 0 %s: mod allready loaded
00001D8 00401D8 0 %s: cannot init %s
00001EB 00401EB 0 %s: %s loaded (%u)
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 ?udp
0000275E 0040275E 0 ?syn
00002763 00402763 0 ?smurf
0000276A 0040276A 0 ?jolt
00002820 00402820 0 PONG:%s
0000299D 0040299D 0 %s!%s@%s
00002B32 00402B32 0 irc.chan
00002B3D 00402B3D 0 %s__%s
00002BD7 00402BD7 0 irc.pre
00002BD8 00402BD8 0 NICK %s
00002CA8 00402CA8 0 irc.nick
00002CA9 00402CA9 0 NICK %s
00002CAE 00402CAE 0 %s
00002C56 00402C56 0 %s
00002C57 00402C57 0 %s
00002CA9 00402CA9 0 %s
00002CAE 00402CAE 0 %s
00002CF5 00402CF5 0 %s
00002CFA 00402CFA 0 %s
00002D13 00402D13 0 %s
00002D14 00402D14 0 %s
00002D15 00402D15 0 %s
00002D16 00402D16 0 %s
0000377B 0040377B 0  WHO %s
00003A45 00403A45 0  USERHOST %s
00003A52 00403A52 0  logged into %s(%s) as %s
00003B99 00403B99 0  nick.pre
00003BA2 00403BA2 0  %s-%04u
00003BAA 00403BAA 0  irc.user
00003BB3 00403BB3 0  irc.usereal
00003BBF 00403BBF 0  irc.real
00003B88 00403B88 0  irc.pass
00003BE0 00403BE0 0  tsend(): connection to %s:%u failed
00003C20 00403C20 0  USER %s localhost 0 :%s
00003C38 00403C38 0  NICK %s
000040BA 004040BA 0  PING
000040BF 004040BF 0  PRIVMSG
000040C7 004040C7 0  001
000040D0 004040D0 0  QUIT
000040D5 004040D5 0  352
000040D9 004040D9 0  302
000040DD 004040DD 0  303
000040E1 004040E1 0  005
000040E5 004040E5 0  PART
000040EA 004040EA 0  JOIN
000040E7 004040E7 0  QUIT
000040D4 004040D4 0  353
000040F3 004040F3 0  433
000040F7 004040F7 0  324
000040FD 004040FD 0  t&
00004100 00404100 0  trecv(): Disconnected from %s err:%u
0000446B 0040446B 0  NOTICE
00004472 00404472 0  %s %s :%s
0000447D 0040447D 0  %s
00004711 00404711 0  MODE %s -o+b %s *@%s
00004727 00404727 0  %s
0000477F 0040477F 0  MODE %s -bo %s %s
00004924 00404924 0  %s.key
00004AA8 00404AA8 0  sk#%u %s is dead!
00004AB2 00404AB2 0  s_check: %s dead? pinging...
00004AD7 00404AD7 0  PING :ok
00004B20 00404B20 0  s_check: send error to %s disconnecting
00004B28 00404B28 0  expect the worst
00004B39 00404B39 0  s_check: killing socket %s
00004B65 00404B65 0  irc.knick
00004B6B 00404B6B 0  jtr.%u%s.iso
00004B6F 00404B6F 0  ison %s
00004B74 00404B74 0  servers
00004B7C 00404B7C 0  s_check: trying %s
00004DAA 00404DAA 0  Ph9K@
00004ED5 00404ED5 0  PhkK@
00004F41 00404F41 0  ShtK@
00005052 00405052 0  %s.mode
0000505A 0040505A 0  MODE %s %s
00005078 00405078 0  SHRP@
000050DA 004050DA 0  Sh$i@
0000559F 0040559F 0  aop
000055A3 004055A3 0  mode %s +o %s
000055B2 004055B2 0  akick
000055B8 004055B8 0  mode %s +b %s %s
000055CA 004055CA 0  KICK %s %s
...
Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46

00006A2B 00406A2B 0 %s: possibly failed
00006A3F 00406A3F 0 %s: exec of %s failed err: %u
00006A90 00406A90 0 u.exf
00006CBC 00406CBC 0 jtr.id
00006CC3 00406CC3 0 %s: <url> <id>
00006E79 00406E79 0 %s
00006EBD 00406EBD 0 IRC
00006EC2 00406EC2 0 DCC
00006EC8 00406EC8 0 DATH
00006ED0 00406ED0 0 IATH
00006ED7 00406ED7 0 IREG
00006EED 00406EED 0 CLON
00006EE3 00406EE3 0 ICON
00006EE9 00406EE9 0 RNL
00006EEE 00406EEE 0 RBN
00006EF3 00406EF3 0 WSN
00006EF8 00406EF8 0 WCON
00006EF8 00406EF8 0 LSN
00006F03 00406F03 0 SSL
00006F08 00406F08 0 S>S
00006F40 00406F40 0 %u [fd:%u] %s:%u [%s%s] last:%u
00006F63 00406F63 0 |\=> [n:%s fh:%s] (\s)
00006F6D 00406F6D 0 |
00006F82 00406F82 0 |---[%s] (%u) %s
00006F96 00406F96 0 |  |-%[%s] [%s]
00006FAD 00406FAD 0 |=> (%s) (%.8x)
00007360 00407360 0 %s <pass> <salt>
000073C0 004073C0 0 %s <nick> <chan>
00007435 00407435 0 !%s!
000074BB 004074BB 0 PING %s
000074C0 004074C0 0 mIRC v6.12 Khaled Mardam-Bey
000074E7 004074E7 0 VERSION %s
dcc.pass
00007525 00407525 0 %s opened (%u)
0000766A 0040766A 0 %s %u-%s
00007672 00407672 0 wb
00007675 00407675 0 %s created
000076A0 004076A0 0 %u bytes from %s in %u seconds saved to %s
000076CB 004076CB 0 (%s %s): incomplete! %u bytes
000076EE 004076EE 0 couldn't open %s err:%u
00007700 00407700 0 (\s) %s: %s
0000770C 0040770C 0 (\s) urlopen failed
00007720 00407720 0 (\s): inetopen failed
000077BE 004077BE 0 no file name in %s
000077DB 004077DB 0 %s opened
00007E49 00407E49 0 %s %s to %s Ok
00007EE0 00407EE0 0 %0.2u/%0.2u/%0.2u %0.2u:%0.2u %15s %s
00007F09 00407F09 0 %s (err: %u)
00008085 00408085 0 %u bytes (%u)
000080F8 004080F8 0 %s %s :ok
00008165 00408165 0 unable to %s %s (err: %u)
000081F5 004081F5 0 %s %s
00008200 00408200 0 %s (\u.%u.%u.%u)
00008499 00408499 0 [\s]%[\s] %s
00008595 00408595 0 closing %u [%s:%u]
000085A8 004085A8 0 unable to close %s %s
000087E2 004087E2 0 using sock #u %s:%u (%s)
000087FD 004087FD 0 Invalid sock

Page 15 of 48
usage %s <socks #>
leaves %s
:j: % * : %
hmm
joins: %
ACCEPT
resume
err: %u
DCC ACCEPT %s %s %s
dcc_resume: cant find port %
send
dcc.dir
%s\%s\%s\%s
unable to open (%s): %u
resuming dcc from %s to %s
DCC RESUME %s %s %u
?si
?ssl
?clon e
?clones
?login
?uptime
?reboot
?status
?jump
?nick
?echo
?hush
?wget
?join
?op
?aop
?akick
?part
?dump
?set
?die
?md5p
?free
?raw
?update
?hostname
?fif
?]fif
?del
?pwd
?play
?copy
?move
?dir
?sums
?ls
?cd
?rm
?mkdir
?run
?exec
0000943C 0040943C 0   ?ps
00009440 00409440 0   ?kill
00009446 00409446 0   ?killall
0000944F 0040944F 0   ?crash
00009456 00409456 0   ?dcc
0000945B 0040945B 0   ?get
00009460 00409460 0   ?say
00009465 00409465 0   ?msg
0000946A 0040946A 0   ?kb
0000946E 0040946E 0   ?sklist
00009476 00409476 0   ?unset
0000947D 0040947D 0   ?uattr
00009484 00409484 0   ?dccsk
0000948B 0040948B 0   ?con
00009490 00409490 0   ?killsk
00009499 00409499 0   VERSION*
000094A3 004094A3 0   DCC
000094A8 004094A8 0   PING
000094AE 004094AE 0   IDENT
000094BE 004094BE 0   %ud %02uh %02um %02us
000094D4 004094D4 0   %02uh %02um %02us
000094E6 004094E6 0   %um %02us
000099E0 004099E0 0   jtram.conf
000099F1 004099F1 0   jtr.*
00009FF5 00409FF5 0   DiCHFc2ioiVmb3cb4z7zWZH1oM=
00009A16 00409A16 0   conf_dump: wrote %u lines
00009C8C 00409C8C 0   set
0000A270 0040A270 0   get of %s incomplete at %u bytes
0000A2B0 0040A2B0 0   get of %s completed (%u bytes), %u seconds %u cps
0000A2F0 0040A2F0 0   error while writing to %s (%u)
0000A373 0040A373 0   Php
0000A65C 0040A65C 0   chdir: %s -> %s (%u)
0000A750 0040A750 0   dcc_wait: get of %s from %s timed out
0000A790 0040A790 0   dcc_wait: closing [%u] %s:%u (%s)
0000A9DC 0040A9DC 0   SEND
0000A9E1 0040A9E1 0   GET
0000A9F0 0040A9F0 0   %s #%.2u %s %ucps %u%% [%sk%%u] %s
0000AA30 0040AA30 0   %u Send(s) %u Get(s) (%u transfer(s) total)
UP:%ucps DOWN:%ucps Total:%ucps
0000ACD0 0040ACD0 0   send of %s incomplete at %u bytes
0000AD10 0040AD10 0   send of %s completed (%u bytes), %u seconds %u cps
0000AF50 0040AF50 0   cant open %s (err:%u) pwd:{%s}
0000AF70 0040AF70 0   DCC SEND %s %u %u %u
0000B751 0040B751 0   %s %s
0000B757 0040B757 0   %s exited with code %u
0000B76E 0040B76E 0   %s\%s
0000B774 0040B774 0   %s: %s
0000B77B 0040B77B 0   exec: Error:%u pwd:%s cmd:%s
0000BB40 0040BB40 0   dcc.pass
0000BB49 0040BB49 0   bot.port
0000BB52 0040BB52 0   %s bad pass from "%s@%s
0000BCC9 0040BCC9 0   %s: connect from %s
0000BD33 0040BD33 0   jtr.bin
0000BD3B 0040BD3B 0   msrll.exe
0000BD45 0040BD45 0   jtr.home
Key fingerprint = AF19 FA27 2F94 998D FDB5 DE3D F8B5 06E4 A169 4E46

...
0001118A 0041118A 0 Rijndael
00011196 00411196 0 XTEA
0001119E 0041119E 0 Twofish
000111AA 004111AA 0 CAST5
000111B3 004111B3 0 Noekeon
000111BF 004111BF 0 Hashes built-in:
000111D0 004111D0 0 SHA-512
000111DB 004111DB 0 SHA-384
000111E6 004111E6 0 SHA-256
000111F1 004111F1 0 TIGER
000111FA 004111FA 0 SHA1
00011202 00411202 0 MD5
00011209 00411209 0 MD4
00011210 00411210 0 MD2
00011218 00411218 0 Block Chaining Modes:
0001122E 0041122E 0 CFB
00011235 00411235 0 OFB
0001123C 0041123C 0 CTR
00011244 00411244 0 PRNG:
0001124A 0041124A 0 Yarrow
00011254 00411254 0 SPRNG
0001125D 0041125D 0 RC4
00011265 00411265 0 PK Algs:
0001126E 0041126E 0 RSA
00011275 00411275 0 DH
0001127B 0041127B 0 ECC
00011282 00411282 0 KR
00011289 00411289 0 Compiler:
00011293 00411293 0 WIN32 platform detected.
000112AF 004112AF 0 GCC compiler detected.
000112CA 004112CA 0 Various others: BASE64 MPI HMAC
00011313 00411313 0 /dev/random
00011430 00411430 0 Microsoft Base Cryptographic Provider v1.0
000114D2 004114D2 0 bits.c
000114D9 004114D9 0 buf != NULL
0001154A 0041154A 0 prng != NULL
00011A10 00411A10 0 -LIBGCCW32-EH-SJLJ-GTHR-MINGW32
00012091 00412091 0 %|
000120C1 004120C1 0 %$
000120E1 004120E1 0 %T
000120F1 004120F1 0 %h
00012111 00412111 0 %p
00012141 00412141 0 %4
00012151 00412151 0 %\n
00012161 00412161 0 %t
000121A1 004121A1 0 %d
000121B1 004121B1 0 %}
000121F1 004121F1 0 %@
00012201 00412201 0 %X
00012231 00412231 0 %<
00012241 00412241 0 %D
00012251 00412251 0 %H
000122B1 004122B1 0 %i
000122C1 004122C1 0 %L
000122D1 004122D1 0 %8
000122E1 004122E1 0 %P
000122F1 004122F1 0 %x
00012301 00412301 0 %,
<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011B6D6</td>
<td>051B6D6</td>
<td>CryptGenRandom</td>
</tr>
<tr>
<td>0011B6EA</td>
<td>051B6EA</td>
<td>CryptReleaseContext</td>
</tr>
<tr>
<td>0011B702</td>
<td>051B702</td>
<td>GetUser_Name</td>
</tr>
<tr>
<td>0011B712</td>
<td>051B712</td>
<td>LookupPrivilegeValueA</td>
</tr>
<tr>
<td>0011B72A</td>
<td>051B72A</td>
<td>OpenProcessToken</td>
</tr>
<tr>
<td>0011B73E</td>
<td>051B73E</td>
<td>OpenSCManagerA</td>
</tr>
<tr>
<td>0011B752</td>
<td>051B752</td>
<td>RegCloseKey</td>
</tr>
<tr>
<td>0011B762</td>
<td>051B762</td>
<td>RegCreateKeyExA</td>
</tr>
<tr>
<td>0011B776</td>
<td>051B776</td>
<td>RegSetValueExA</td>
</tr>
<tr>
<td>0011B78A</td>
<td>051B78A</td>
<td>RegisterServiceCtrlHandlerA</td>
</tr>
<tr>
<td>0011B7AA</td>
<td>051B7AA</td>
<td>SetServiceStatus</td>
</tr>
<tr>
<td>0011B7BE</td>
<td>051B7BE</td>
<td>StartServiceCtrlDispatcherA</td>
</tr>
<tr>
<td>0011B7DE</td>
<td>051B7DE</td>
<td>AddAtomA</td>
</tr>
<tr>
<td>0011B7EA</td>
<td>051B7EA</td>
<td>CloseHandle</td>
</tr>
<tr>
<td>0011B7FA</td>
<td>051B7FA</td>
<td>CopyFileA</td>
</tr>
<tr>
<td>0011B806</td>
<td>051B806</td>
<td>CreateDirectoryA</td>
</tr>
<tr>
<td>0011B81A</td>
<td>051B81A</td>
<td>CreateFileA</td>
</tr>
<tr>
<td>0011B82A</td>
<td>051B82A</td>
<td>CreateMutexA</td>
</tr>
<tr>
<td>0011B83A</td>
<td>051B83A</td>
<td>CreatePipe</td>
</tr>
<tr>
<td>0011B84A</td>
<td>051B84A</td>
<td>CreateProcessA</td>
</tr>
<tr>
<td>0011B85E</td>
<td>051B85E</td>
<td>CreateToolhelp32Snapshot</td>
</tr>
<tr>
<td>0011B87A</td>
<td>051B87A</td>
<td>DeleteFileA</td>
</tr>
<tr>
<td>0011B88A</td>
<td>051B88A</td>
<td>DuplicateHandle</td>
</tr>
<tr>
<td>0011B89E</td>
<td>051B89E</td>
<td>EnterCriticalSection</td>
</tr>
<tr>
<td>0011B8B6</td>
<td>051B8B6</td>
<td>ExitProcess</td>
</tr>
<tr>
<td>0011B8C6</td>
<td>051B8C6</td>
<td>ExitThread</td>
</tr>
<tr>
<td>0011B8D6</td>
<td>051B8D6</td>
<td>FILETIMEToSYSTEMTIME</td>
</tr>
<tr>
<td>0011B8EE</td>
<td>051B8EE</td>
<td>FindAtomA</td>
</tr>
<tr>
<td>0011B8FA</td>
<td>051B8FA</td>
<td>FindClose</td>
</tr>
<tr>
<td>0011B906</td>
<td>051B906</td>
<td>FindFirstFileA</td>
</tr>
<tr>
<td>0011B91A</td>
<td>051B91A</td>
<td>FindNextFileA</td>
</tr>
<tr>
<td>0011B92A</td>
<td>051B92A</td>
<td>FreeLibrary</td>
</tr>
<tr>
<td>0011B93A</td>
<td>051B93A</td>
<td>GetAtomNameA</td>
</tr>
<tr>
<td>0011B94A</td>
<td>051B94A</td>
<td>GetCommandLineA</td>
</tr>
<tr>
<td>0011B95E</td>
<td>051B95E</td>
<td>GetCurrentDirectoryA</td>
</tr>
<tr>
<td>0011B976</td>
<td>051B976</td>
<td>GetCurrentProcess</td>
</tr>
<tr>
<td>0011B98A</td>
<td>051B98A</td>
<td>GetCurrentThreadId</td>
</tr>
<tr>
<td>0011B9A2</td>
<td>051B9A2</td>
<td>GetExitCodeProcess</td>
</tr>
<tr>
<td>0011B9BA</td>
<td>051B9BA</td>
<td>GetFileSize</td>
</tr>
<tr>
<td>0011B9CA</td>
<td>051B9CA</td>
<td>GetFullPathNameA</td>
</tr>
<tr>
<td>0011B9DE</td>
<td>051B9DE</td>
<td>GetLastError</td>
</tr>
<tr>
<td>0011B9EE</td>
<td>051B9EE</td>
<td>GetModuleFileNameA</td>
</tr>
<tr>
<td>0011BA06</td>
<td>051BA06</td>
<td>GetModuleHandleA</td>
</tr>
<tr>
<td>0011BA1A</td>
<td>051BA1A</td>
<td>GetProcAddress</td>
</tr>
<tr>
<td>0011BA2E</td>
<td>051BA2E</td>
<td>GetStartupInfoA</td>
</tr>
<tr>
<td>0011BA42</td>
<td>051BA42</td>
<td>GetSystemDirectoryA</td>
</tr>
<tr>
<td>0011BA5A</td>
<td>051BA5A</td>
<td>GetSystemInfo</td>
</tr>
<tr>
<td>0011BA6A</td>
<td>051BA6A</td>
<td>GetTempPathA</td>
</tr>
<tr>
<td>0011BA7A</td>
<td>051BA7A</td>
<td>GetTickCount</td>
</tr>
<tr>
<td>0011BA8A</td>
<td>051BA8A</td>
<td>GetVersionExA</td>
</tr>
<tr>
<td>0011BA9A</td>
<td>051BA9A</td>
<td>GlobalMemoryStatus</td>
</tr>
<tr>
<td>0011BAB2</td>
<td>051BAB2</td>
<td>InitializeCriticalSection</td>
</tr>
<tr>
<td>0011BACE</td>
<td>051BACE</td>
<td>IsBadReadPtr</td>
</tr>
<tr>
<td>0011BADD</td>
<td>051BADD</td>
<td>LeaveCriticalSection</td>
</tr>
<tr>
<td>0011BAF6</td>
<td>051BAF6</td>
<td>LoadLibraryA</td>
</tr>
<tr>
<td>0011BB06</td>
<td>051BB06</td>
<td>MoveFileA</td>
</tr>
<tr>
<td>0011BB12</td>
<td>051BB12</td>
<td>OpenProcess</td>
</tr>
<tr>
<td>0011BB22</td>
<td>051BB22</td>
<td>PeekNamedPipe</td>
</tr>
</tbody>
</table>
The procedure entry point %s could not be located in the dynamic link library %s.
The ordinal %u could not be located in the dynamic link library %s.

kernel32.dll
msvcrt.dll
shell32.dll
user32.dll
version.dll
wininet.dll
ws2_32.dll

AdjustTokenPrivileges
_getmainargs
ShellExecuteA
DispatchMessageA
GetFileVersionInfoA
InternetCloseHandle
WSAGetLastError

AddAtomA
CloseHandle
GetUserNameA
LookupPrivilegeValueA
OpenProcessToken
OpenSCManagerA
RegCloseKey
RegCreateKeyExA
RegSetValueExA
RegisterServiceCtrlHandlerA
SetServiceStatus
StartServiceCtrlDispatcherA
kernel32.dll

AddServiceCtrlDispatcherA
AddAtomA
AllocateFile
AllocateProcess
AllocateSystemProcess
AllocateSystemToken
CloseHandle
CreateFileA
CreatePipe
CreateServiceA
CreateToolhelp32Snapshot
DeleteFileA
DuplicateHandle
EnterCriticalSection
ExitProcess
GetUserNameA
<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Description</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>0012039A</td>
<td>0052039A</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>001203ED</td>
<td>005203ED</td>
<td>GetFullPathNameA</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>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>001205E8</td>
<td>005205E8</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>__strcpyp</td>
</tr>
<tr>
<td>0012065E</td>
<td>0052065E</td>
<td>__mbsdup</td>
</tr>
<tr>
<td>00120668</td>
<td>00520668</td>
<td>__strcmpi</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>

Page 26 of 48
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
84acfe96a98590813413122c12c11aaa *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 1\textsuperscript{st} 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 2\textsuperscript{nd} shot and select shot and save. Now we can click on compare and see the results shown in Fig. 2.

![Fig. 2](image-url)
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.

```
2:51:07 PM msrll.exe:256 CREATE C:\WINDOWS\System32\mfm SUCCESS Options: Create Directory Access: All
2:51:08 PM msrll.exe:256 CREATE C:\WINDOWS\System32\mfm\msrll.exe SUCCESS Options: OverwriteIf Sequential Access: All
2:51:08 PM msrll.exe:256 WRITE C:\WINDOWS\System32\mfm\msrll.exe SUCCESS Offset: 0 Length: 41984
2:51:29 PM msrll.exe:952 OPEN C:\WINDOWS\system32\mfm\jtram.conf FILE NOT FOUND Options: Open Access: All
2:51:29 PM msrll.exe:952 CREATE C:\WINDOWS\system32\mfm\jtram.conf SUCCESS Options: OverwriteIf Access: All
```

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.

```
659.29827310 msrll.exe:256 SetValue HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed SUCCESS
```

This entry has something to do with cryptography.

Examination of TDIMon shows TCP activitiy 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.

```
2:51:27 PM msrll.exe:952 80D06A38 IRP_MJ_CREATE TCP:0.0.0.0:2200 SUCCESS Address Open
2:51:27 PM msrll.exe:952 80D06A38 TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Error Event
2:51:27 PM msrll.exe:952 80D06A38 TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Disconnect Event
2:51:27 PM msrll.exe:952 80D06A38 TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Receive Event
2:51:27 PM msrll.exe:952 80D06A38 TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Expedited Receive
2:51:27 PM msrll.exe:952 80D06A38 TDI_SET_EVENT_HANDLER TCP:0.0.0.0:2200 SUCCESS Chained Receive
```

Page 32 of 48
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 mrrl.exe process running under parent service services.exe. By double clicking on the mrrl.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
```

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
```

© SANS Institute 2000 - 2005

Author retains full rights.
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
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```
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

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](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 0040BBBD6. In this section you can see the following instructions:

```
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 0040BEBEB 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 004BBBE7 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 004BBBE7 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 004BBBE7. The beginning of this section started at address 004BBD6 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 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 a entry box that I typed in NOP. I made sure the “Fill with NOP’s” box was checked in order to fill the

---

Page 42 of 48

© SANS Institute 2000 - 2005
Author retains full rights.
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</td>
<td>(No response on unsuccessful login)</td>
</tr>
<tr>
<td>?login</td>
<td>?login &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>Echoes 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 (No output to screen - need to use the ?sklist and ?dccsk commands to find and connect to an irc sock first)</td>
<td></td>
</tr>
<tr>
<td>?op</td>
<td>Irc command to become channel operator ?op bad args</td>
<td></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 (No output to screen)</td>
<td></td>
</tr>
<tr>
<td>?dump</td>
<td>?</td>
<td>(No response)</td>
</tr>
<tr>
<td>?set</td>
<td>Shows the jtram.conf contents plus can change settings by issuing the ?set &lt;setting&gt; &lt;value&gt;</td>
<td>set jtr.bin msrl.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 $1$KZLPLKDr58kI8Jr1X8DOHZsmlp9qq0 set dcc.pass $1$KZLPLKDf$55isA1fTvamR7bjAdBziX</td>
</tr>
<tr>
<td>?die</td>
<td>Kills msrl.exe process (No output to screen)</td>
<td></td>
</tr>
<tr>
<td>?md5p</td>
<td>Displays the salt and md5 hash of whatever is typed in as &lt;pass&gt; parameter</td>
<td>?md5p &lt;pass&gt; &lt;salt&gt;</td>
</tr>
<tr>
<td>?free</td>
<td>?</td>
<td>usage: ?free &lt;cmd&gt;</td>
</tr>
<tr>
<td>?raw</td>
<td>?</td>
<td>(No output to screen)</td>
</tr>
<tr>
<td>?update</td>
<td>Possibly a command to update bot ?update &lt;url&gt; &lt;id&gt;</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<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>?fif</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 listing of current directory</td>
<td></td>
</tr>
<tr>
<td><code>?pwd</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>?rmidir</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>
<tr>
<td><code>?run</code></td>
<td>Run a program (hidden) - syntax: <code>?run c:\windows\system32\notepad.exe &lt;enter&gt;</code></td>
<td>(Example: <code>?run c:\windows\system32\notepad.exe</code> <code>?run: ran ok (4022304)</code></td>
</tr>
<tr>
<td><code>?exec</code></td>
<td>irc command exec</td>
<td>(No response)</td>
</tr>
<tr>
<td><code>?ps</code></td>
<td>Display all processes running and their PID's</td>
<td>(Lists active processes running on infected system)</td>
</tr>
<tr>
<td><code>?kill</code></td>
<td>Kill a process - <code>?kill 1448 &lt;enter&gt;</code></td>
<td>pid 1448 killed</td>
</tr>
<tr>
<td><code>?killall</code></td>
<td>(No response)</td>
<td></td>
</tr>
<tr>
<td><code>?crash</code></td>
<td>Crashes system</td>
<td>(No output to screen)</td>
</tr>
<tr>
<td><code>?dcc</code></td>
<td>irc command dcc direct connections to remote clients</td>
<td>(No response)</td>
</tr>
<tr>
<td><code>?get</code></td>
<td>(No response)</td>
<td></td>
</tr>
<tr>
<td><code>?say</code></td>
<td>irc say command - I believe this message would be said non-privately</td>
<td>usage: <code>?say &lt;target&gt; &quot;text&quot;</code></td>
</tr>
<tr>
<td><code>?msg</code></td>
<td>irc command to send private message to nick or list of nicks</td>
<td>usage: <code>?msg &lt;target&gt; &quot;text&quot;</code></td>
</tr>
<tr>
<td><code>?kb</code></td>
<td></td>
<td><code>?kb &lt;nick&gt; &lt;chan&gt;</code></td>
</tr>
<tr>
<td><code>?sklist</code></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><code>?unset</code></td>
<td>Un sets a set command refer to <code>?set command</code></td>
<td>(Example: <code>?unset pass &lt;enter&gt;</code> (This will remove the set pass parameter)</td>
</tr>
<tr>
<td><code>?attr</code></td>
<td></td>
<td>usage: <code>?attr &lt;nick&gt; &lt;chan&gt;</code></td>
</tr>
<tr>
<td><code>?dcsck</code></td>
<td>Set the irc sock to use</td>
<td>usage: <code>?dcsck &lt;socks #&gt;</code></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.

```bash
@echo off
c
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
```

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>Training Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Special Investigations 2020</td>
<td>United Arab Emirates</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>South by Southeast Asia Online 2020</td>
<td>Singapore</td>
<td>Aug 03, 2020 - Aug 14, 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>
<tr>
<td>SANS Reboot - NOVA 2020 - Live Online</td>
<td>Arlington, VA</td>
<td>Aug 10, 2020 - Aug 15, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>FOR498 Australia Live Online 2020</td>
<td>Australia</td>
<td>Aug 10, 2020 - Aug 15, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 17 ET</td>
<td>DC</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
</tr>
<tr>
<td>Cyber Defence APAC Live Online 2020</td>
<td>Singapore</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS FOR508 Sydney August 2020</td>
<td>Sydney, Australia</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Summer Forensics Europe 2020</td>
<td>United Arab Emirates</td>
<td>Aug 17, 2020 - Aug 28, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 24 MT</td>
<td>CA</td>
<td>Aug 24, 2020 - Aug 29, 2020</td>
</tr>
<tr>
<td>SANS Virginia Beach 2020 - Live Online</td>
<td>Virginia Beach, VA</td>
<td>Aug 30, 2020 - Sep 04, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Virginia Beach 2020</td>
<td>Virginia Beach, VA</td>
<td>Aug 30, 2020 - Sep 04, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Japan Bi-Lingual Live Online</td>
<td>Japan</td>
<td>Aug 31, 2020 - Sep 05, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Philippines 2020</td>
<td>Manila, Philippines</td>
<td>Sep 07, 2020 - Sep 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London September 2020 - Live Online</td>
<td>London, United Kingdom</td>
<td>Sep 07, 2020 - Sep 12, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS London September 2020</td>
<td>London, United Kingdom</td>
<td>Sep 07, 2020 - Sep 12, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Baltimore Fall 2020</td>
<td>Baltimore, MD</td>
<td>Sep 08, 2020 - Sep 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Baltimore Fall 2020 - Live Online</td>
<td>Baltimore, MD</td>
<td>Sep 08, 2020 - Sep 13, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Threat Hunting &amp; Incident Response Summit &amp; Training 2020</td>
<td>Virtual - US Eastern,</td>
<td>Sep 10, 2020 - Sep 19, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Munich September 2020 - Live Online</td>
<td>Munich, Germany</td>
<td>Sep 14, 2020 - Sep 19, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Munich September 2020</td>
<td>Munich, Germany</td>
<td>Sep 14, 2020 - Sep 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Network Security 2020 - Live Online</td>
<td>Las Vegas, NV</td>
<td>Sep 20, 2020 - Sep 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Australia Spring 2020 - Live Online</td>
<td>Australia</td>
<td>Sep 21, 2020 - Oct 03, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Australia Spring 2020</td>
<td>Australia</td>
<td>Sep 21, 2020 - Oct 03, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Northern VA - Reston Fall 2020 - Live Online</td>
<td>Reston, VA</td>
<td>Sep 28, 2020 - Oct 03, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS San Antonio Fall 2020</td>
<td>San Antonio, TX</td>
<td>Sep 28, 2020 - Oct 03, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Antonio Fall 2020 - Live Online</td>
<td>San Antonio, TX</td>
<td>Sep 28, 2020 - Oct 03, 2020</td>
<td>CyberCon</td>
</tr>
</tbody>
</table>