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

Copyright SANS Institute
Author Retains Full Rights

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

Malware: A Look at Reverse Engineering
MSRLL.EXE

Lorna J. Hutcheson
Orlando SANS 2004
TABLE OF CONTENTS

Abstract ........................................................................................................... 4

Part 1: Laboratory Setup ............................................................................... 4
  Introduction ................................................................................................. 4
  Hardware Setup .......................................................................................... 4
  Networking Setup ....................................................................................... 4
  Software Resources .................................................................................... 5
    VMWare Workstation 4.5.1 ...................................................................... 5
    Linux Redhat ........................................................................................... 5
    Windows 98 ............................................................................................. 5
    Windows 2000 ........................................................................................ 5
    PEInfo ....................................................................................................... 5
    Ollydbg ..................................................................................................... 5
    Ethereal ..................................................................................................... 6
    SNORT ....................................................................................................... 6
    Filemon ..................................................................................................... 6
    RegMon ..................................................................................................... 6
    TDI Mon .................................................................................................... 6
    LordPE ....................................................................................................... 6
    Notepad ..................................................................................................... 7
    Regshot ..................................................................................................... 7
    Netcat ....................................................................................................... 7
    Process Explorer ...................................................................................... 7
    PSkill ......................................................................................................... 7
    MD5sum .................................................................................................... 7

Part 2: Properties of the Malware Specimen .............................................. 8
  Type of File .................................................................................................. 8
  Size of the File ............................................................................................ 8
  MD5 Hash of the File .................................................................................. 8
  Operating System it Runs on ..................................................................... 8
  Strings Embedded into it .......................................................................... 8

Part 3: Behavioral Analysis ......................................................................... 14
  Behavior Before Code Analysis .............................................................. 14
    Monitoring of File System Access ......................................................... 14
    Monitoring registry/configuration Access .............................................. 14
    Monitoring/Redirecting Network Connections ..................................... 15
    Monitoring Processes on the System .................................................... 15
  Behavior After Code Analysis ................................................................. 16
    The “Bot Army” ...................................................................................... 18

Part 4: Code Analysis .................................................................................. 19
  Unpacking/Unencrypting .......................................................................... 19
  Program Code Disassembly ..................................................................... 20
  Debugging .................................................................................................. 20

Table of Contents
  Page 2 of 24
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 5: Analysis Wrap-Up</td>
<td>23</td>
</tr>
<tr>
<td>Citation of Sources</td>
<td>24</td>
</tr>
<tr>
<td>Sites for Tools</td>
<td>24</td>
</tr>
</tbody>
</table>
Abstract
This practical will cover the reverse engineering of a malicious piece of code given to us to analyze. It will use the procedures taught in class and will follow the outline of the Table of Contents listed above.

Part 1: Laboratory Setup

Introduction
The laboratory environment was set up to safely analyze an unknown and potentially malicious piece of code called msrll.exe. In order to do so, the environment needed to be capable of exploring the full capabilities of the code, without allowing it to be released into the wild and potentially cause damage to systems. To facilitate this, the following laboratory configurations were designed to allow flexibility and ensure confinement of the unknown piece of software.

Hardware Setup
Only one computer was used as the test machine. This box is running a fully patched Windows XP Home edition and has a 2.70 GHz processor with one Gigabyte of memory. The unknown code was transferred to the box via a USB thumb drive. This was done to bypass the antivirus software running on the base system and allowed it to be copied directly to the VM image. Also available were CDROM and floppy drive capabilities.

Networking Setup
The base system was configured with Internet access and the Internet Connection Firewall turned on. There is a second Linksys firewall/router controlling access out of the live network. The box is running Norton Antivirus with the latest definitions. During testing, all ports to the internet were denied by blocking the box at the Linksys firewall/router. Once it was determined the ports in use I created a rule in the Linksys firewall/router to allow access on port 80 and port 443. This was to ensure that the malicious code could not get to the internet incase of a mishap during testing.

In order to create the closed network test environment, VMWare Workstation software was used to create the test network. Three images were used for testing: an unpatched Windows 2000; an unpatched Windows 98 and the Linux Redhat image given to us during the REM track. These images were configured in host only mode on bootup to disallow access outside of the VM environment to ensure containment of the malware. The boxes were configured using DHCP and network connectivity was checked between the boxes by ensuring that you could ping each system from the Windows 2000 image.
Software Resources

VMWare Workstation 4.5.1
The key part of the testing was the ability to use VMWare. The software provides the ability to run a completely isolated network with multiple operating systems all on one box. It reduces cost of testing by reducing the need for multiple test boxes and networking hardware. This can be found at http://www.vmware.com/.

Linux Redhat
The image was given during the Reverse Engineering Malware course at a SANS conference. It was used as one of the test operating systems and provided a box for the malicious code to attempt to connect to via an IRC channel since it already had an IRC server loaded on it. Linux can be obtained free from http://www.linux.org/.

Windows 98
The image was an unpatched Windows 98 operating system used as a secondary box on which to launch the malicious code. This was done for two reasons: one was to determine the effects of how the code interacted with multiple systems infected with the same thing in an enclosed environment and second to observe any differences on different operating platforms.

Windows 2000
The image was an unpatched Windows 2000 operating system and was used as the primary system to launch the code on and conduct the analysis on. On this image was also loaded all the tools necessary to conduct our analysis.

PEInfo
PEInfo is a tool developed by Tom Liston and allows for the breakdown of an executable to examine the PE header information and the structure of the windows executable. However, when dealing with a packed file, it first has to be unpacked before it can be of much use. The primary use of this tool was to analyze the strings of the file once it was unpacked. This is done by dragging the file over the window of PEInfo and dropping it. I obtained this tool from Tom Liston.

Ollydbg
This is a great tool and it’s free. Ollydbg is a disassembler with great functionality and it is relatively easy to use. Also downloaded was OllyDump which is a plugin that will allow you to dump code from memory. This tool was used to obtain a clean, unpacked version of the malicious code. Many scripts are available for Ollydbg that will allow you to find the Operational Entry Point (OEP) of the code in assembly. However, in the reality that one of these might not be available and the desire to learn how to find it manually, I enlisted the help of Tom Liston and his great programming skills. His techniques for finding the
OEP manually through Ollydbg will be described later in detail. This can be found at http://home.t-online.de/home/Ollydbg/.

**Ethereal**
Many sniffers exist, but Ethereal is my favorite. It is easy to use and allows you to look at the information in a concise fashion or provide an indepth look at the packets. This was used to sniff the network traffic on the Windows boxes. A requirement to run any sniffer is winpcap which allows the interface to go into promiscuous mode. The program as well as the winpcap drivers can be found at http://www.ethereal.com/.

**SNORT**
SNORT was also used as a sniffer. Since the Linux image already had snort installed, I used it to monitor traffic coming to the Linux box. It can be found at http://www.snort.org/.

**Filemon**
This a great tool provided free by Sysinternals and allows you to monitor changes as well as access to the file system. This tool was used when msrll.exe was launched to monitor what activity was taking place on the file system. This can be found at http://www.sysinternals.com/ntw2k/source/filemon.shtml.

**RegMon**
This tool is also provided free by Sysinternals and allows you to monitor registry accesses and changes. This tool was used when msrll.exe was launched to monitor what it was activity was taking place in the registry. This can be found at http://www.sysinternals.com/ntw2k/source/regex.shtml.

**TDIMon**
TDIMon is a tool that allows you to monitor the TCP and UDP activity on the system. TDI stands for Transport Driver Interface which is exactly what it is monitoring. This was used to help determine what activity msrll.exe might be doing. This can be found at http://www.sysinternals.com/ntw2k/freeware/tdimon.shtml.

**LordPE**
LordPE is a tool that was used during the analysis without useful results. The tool is intended to allow a look at processes that are running and how it interacts. It also allows to you dump a process from memory. This was used after launching msrll.exe to attempt to get an unpacked version of the code. However, the code that was obtained was still not readable so another method was used which was ollydbg. It can be found at http://mitglied.lycos.de/yoda2k/LordPE/info.htm.
Notepad
Notepad is a built in text editor that comes with Windows systems. It is used as a method to safely view files without launching them. This was used in many different forms throughout the analysis to look at files and view output from tools.

Regshot
Regshot is a great tool that allows you to do a before and after picture of the registry settings. This was run before msril.exe was launched and again immediately afterwards to help determine the modifications to the registry. The homepage for Regshot is http://regshot.ist.md/ however it was unavailable the last time I checked. It can be found easy by a simple Google query.

Netcat
This tool has been often called a “Swiss Army Knife” because of its many capabilities. You can use it to set up a listener on any port to allow connections to it or you can transfer files using it. It is very flexible and will be used to simulate any needed listening ports that the malware might want to connect to. This can be found at http://netcat.sourceforge.net.

Process Explorer
This is a tool from Sysinternals that allows you to monitor the processes running on the system as well as any handles they might have. You can also view all the information about the process such as the command line that calls it, security settings etc. This will be used to help monitor what the malware is doing. This can be found at http://www.sysinternals.com/ntw2k/freeware/proexp.shtml.

PSkill
This tool is a command line tool that will allow you to terminate a process by typing “pskill <PID>”. This will be used to quickly kill a process if needed. This can be found at http://www.sysinternals.com/ntw2k/freeware/pskill.shtml.

MD5sum
The tool is a command line executable that allows you generate an MD5 hash of a file. This will be used to hash the malware in question before and after it is launched to see if it is modifying itself or if it is the same as the original. The tool can be found at http://www.gnu.org/software/textutils/textutils.html.
Part 2: Properties of the Malware Specimen

Type of File

The malware file is a compressed executable that has been compressed with ASPack. This was determined by looking at the file in PEInfo. It was easy to look at the Sections area and find how the file was compressed as .aspack is listed. This is key to know because the file is unreadable for the strings in its current state and will require the file to be unpacked.

Size of the File

The file itself in its packed state is 41,984 bytes (41 KB). This was determined by looking at the file in PEInfo. If you click on the filename itself at the top of PEInfo, it will show you the file size. This was further verified by right clicking on the file and looking at the properties of it using Explorer. After the file was unpacked via Ollydbg and following the same procedure as described above, the file size was 1,182,720 bytes (1.12 MB).

MD5 Hash of the File

The MD5 hash of the file in its packed state is 84acfe96a98590813413122c12c11aaa. This was determined by using a command line tool called md5sum.exe. This tool is used by issuing the following command at the commandline “md5sum.exe msrll.exe”. I placed md5sum.exe in the directory where the malware was located to avoid modifying the path. The MD5 hash of the file was also taken after it was launched with the following results:

```
\84acfe96a98590813413122c12c11aaa *C:\WINNT\system32\mfm\msrll.exe.
```

This was done to ensure no modification of the file occurred after it was launched.

Operating System it Runs on

The malware is a Windows based executable. This was determined by using PEInfo, which breaks down the file structure of Windows based executables.

Strings Embedded into it

The strings embedded into it are not visible via PEInfo in its packed state. However, once the file was unpacked using Ollydbg, the strings were readily available using PEInfo and there were lots of strings. Here are the strings found:

<table>
<thead>
<tr>
<th>This program cannot be run in DOS mode</th>
<th>msg</th>
<th>GetExitCodeProcess</th>
<th>StartServiceCtrlDispatcherA</th>
</tr>
</thead>
<tbody>
<tr>
<td>.data</td>
<td>kb</td>
<td>GetFileSize</td>
<td>kernel32.dll</td>
</tr>
<tr>
<td>.aspack</td>
<td>sklist</td>
<td>GetFullPathNameA</td>
<td>AddAtomA</td>
</tr>
<tr>
<td>.adata</td>
<td>unset</td>
<td>GetLastError</td>
<td>CloseHandle</td>
</tr>
<tr>
<td>.newIID</td>
<td>hattr</td>
<td>GetModuleFileNameA</td>
<td>CopyFileA</td>
</tr>
<tr>
<td>?insmod</td>
<td>dccsk</td>
<td>GetModuleHandleA</td>
<td>CreateDirectoryA</td>
</tr>
<tr>
<td>?rmmod</td>
<td>con</td>
<td>GetProcAddress</td>
<td>CreateFileA</td>
</tr>
</tbody>
</table>
Part 2: Properties of the Malware Specimen

Page 9 of 24
Part 2: Properties of the Malware Specimen

Page 10 of 24
capGetDriverDescriptionA  cipher != NULL  WS2_32.DLL  _onexit
cpus:%u  hash != NULL  VirtualAlloc  _setmode
WIN%d(s %u)%%s mem(%u%%u) %u%% %s %s
prng != NULL  VirtualFree  _vsnprintf
%x %%u  LibTomCrypt 0.83  kern32.dll  atexit
%x bad args  Endianness: little (32-bit words)  ExitProcess  fclose
%x[%u] %s  Clean stack: disabled  user32.dll  fflush
%x removed  Ciphers built-in:  MessageBoxA  printf
couldnt find %s  Blowfish  _wsprintfA
%x added  VirtualFree  _vsnprintf
WS2_32.DLL  _onexit
hash != NULL  VirtualAlloc  _setmode
%u %s (u:%s)%s
mem:(%u/%u) %u%% %s %s
%u %s
%u %s
%u %s

Part 2: Properties of the Malware Specimen

Page 11 of 24
Part 2: Properties of the Malware Specimen
The first step in starting to analyze the malware was to look at the strings for any clues about the capabilities of the malware and what it might do. One of the first things that popped out were references to “IRC” in some of the strings. So my initial assumption was it was an IRC bot of some sort. In light of this, I looked for strings that might be used as IRC commands. I found several that were proceeded by a ? and looked like good candidates such as “?login”, “?uptime”, “?join”, “?die” etc. Other strings that caught my attention were things like “irc.pass” and “dcc.pass” as possible ways for IRC password validation. The string “bot.port” seemed to indicate either a port that might be listening on the infected machine or that you could use it to specify the port that bot was connecting on so it was something to look at later. I also noticed the strings “Ciphers built-in” and “Hashes built-in” so I figured I would have to look for their usage and might make password identification difficult. “Mozilla/4.0” and strings such as “InternetCloseHandle” and “InternetOpenUrlA” might mean web server of some sort was being used. Also found were references to one server (collective7.zxy0.com) but two different ports were specified: 9999 and 8080. The other didn’t specify a port, so it looked like it might be a webserver connection. There were two strings that seemed similar that I found that I thought might be useful “$1$KZLPLKDF$W8kI8Jr1X8DOHZsmn9qq0” and “$1$KZLPLKDF$s55isA1ITvamR7bjAdBiX”. I did not know what they were for, but I wanted to explore them further. Also mentioned were “Ping” “UDP” “Smurf”, “Jolt2”, and “SYN”. This was evident that it possessed to capability to do denial of service (DOS) attacks.
Part 3: Behavioral Analysis

Behavior Before Code Analysis
In preparation of launching of msrll.exe on the Windows 2000 image, there were several tools set up in advance to monitor the behavior of it. Filemon, Regmon and TDImon were all launched, the captures paused and the contents cleared. This was to attempt to capture only what was being used by the msrll.exe. Also launched were LordPE to monitor the processes and RegShot to keep track of the registry. Ethereal was also run to sniff traffic coming from the system. The malware was then executed in the controlled environment described above. The tools and their output are described below in the analysis of the malware, however it is important to note that only the key pieces of the analysis are listed. Each of these tools generate a lot of data that has to be analyzed to determine what is relevant in understanding the malware.

Monitoring of File System Access
Filemon showed some very key things that are listed and discussed below.

We see that the malware created a directory named mfm in the system32 folder

```
177  3:08:45 PM  msrll.exe:252CREATE  C:\WINNT\system32\mfm
          SUCCESS  Options: Create Directory  Access: All
```

Next the malware copied itself to the directory, which was verified via an MD5 hash to be the same as the orginal file.

```
224  3:08:45 PM  msrll.exe:252CREATE
          C:\WINNT\system32\mfm\msrll.exe
          SUCCESS  Options: OverwriteIf Sequential  Access: All
```

Then we see the malware creates a file calld jtram.conf

```
869  3:09:04 PM  msrll.exe:1044 CREATE
          C:\WINNT\system32\mfm\jtram.conf
          SUCCESS  Options: OverwriteIf Access: All
```

Monitoring registry/configuration Access
For the registry I was looking for things created and/or modified that might be something that tells us what and how its doing it. Here are some of the interesting findings from Regmon and RegShot.

Regmon shows us that it is putting its self in to run as a service.

```
1462  101.79209788 SERVICES.EXE:212 CreateKey
          HKLM\System\CurrentControlSet\Services\mfm
          SUCCESS  Key: 0xE13704E0
```

Part 3: Behavioral Analysis
Page 14 of 24
Regmon also shows that several keys were created dealing with Cryptograph such as the one below. Probably to ensure that it had the crypto capabilities that it was wanting.

```
249  100.15181874 Filemon.exe:680 CreateKey
       HKLM\SOFTWARE\Microsoft\Cryptography\RNG   SUCCESS   Key: 0xE1D14FE0
```

RegShot showed us more about the service that it set itself up to run as and the name of the service was “Rll enhanced drive.”

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ImagePath: "C:\WINNT\system32\mfm\msrll.exe"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ObjectName: "LocalSystem"
```

**Monitoring/Redirecting Network Connections**

Before launching the program, I used Ethereal to start a packet capture to see if it was doing anything. One of the things noted after it was run were the DNS queries to the collective7.zxy0.com that we noted in the strings. Since it was looking for them, I modified the host file to point to my VM image of Linux that I was going to use as the server. As soon as it got the DNS name resolved, it attempted connections to the server on ports 6667, 9999 and 8080. These connections were sent a RST since I have nothing listening on them yet.

In order to give it something to connect to, I used the VM image of Linux that was given to us in class. On it is an IRC server that was ready to be used. Before starting the IRC server, I fired up Snort since it was already on the Linux box to capture the data that was being sent to us. Now, I launched the IRC server and waited. The packets showed connection attempts to a channel called #mils. So on the IRC channel I joined the IRC channel and very shortly a “user” popped up in it with me. The bot was now connected. The nickname used was not one that was readable and subsequent connections while analyzing the bot showed that it was a randomly generated nickname. I decided to also set up a netcat listener on each of the other ports and see what would happen. The bot connects to the netcat listener in the same fashion it appears that they connected to the IRC channel. Once a connection is established and the bot logs in there are no more attempts to connect to the other ports. The final thing I attempted was to issue commands to the bot from the IRC server that I was logged into. On the Linux image, I attempted all of the commands that I found listed and attempted to use them to get the bot to respond. I was unsuccessful in my attempts.

**Monitoring Processes on the System**

I used Task Manager to check the processes running on the system and I found that msrll.exe was indeed running now as a process. I tried to kill it from the process list and found that I was denied. I then went to look at the services and...
found I could not kill it from there either. I opened Sysinternals Process Explorer and found it there. I was able to kill it using Process Explorer or the command line tool PSkill also from Sysinternals.

The next thing I wanted to know, was what ports were being used by msrll.exe. I used Process Explorer again to find the msrll.exe process. Once selected I right clicked to look at the properties. From there, I could see the connection to the IRC server, but I also saw that it was listening on port 2200 as well. To watch what was happening I launched Ethereal again and since Mozilla was mentioned, I used it and attempted to connect to the bot on port 2200. I was unsuccessful and kept getting RSTs from the bot host. I then attempted to pass it things from the strings that looked like it could be connected to such as 
http://192.168.6.129:2200/jtr.home and http://192.168.6.129:2200/jtr.bin. I got nothing back on the first one, but on the second one I got a file that I downloaded. The file contained a “#.” I tried connecting the same way via SSL since it was mentioned and I received a connection to port 2200. The packet captures showed that I was sent a “#” in the data, but nothing appeared on my screen. I tried sending commands, but nothing would leave my browser based on packet captures. I then tried a telnet session since it appeared to attempt to shovel me a prompt with the “#.” I was rewarded with a “#.” on my screen but after typing “?login testuser” and “?pwd testpwd” I was disconnected. So now I knew I had three avenues to attempt to figure out how to get in: the IRC channel, port 2200 via telnet and using Mozilla.

**Behavior After Code Analysis**

Once I was able to bypass the authentication (see the code analysis section below) and talk to the bot directly, I attempted the commands to see what I was able to do with them. Of the three ways that I tried to get the bot to respond, the only one that was successful was via telnet. Here are the results of what I found for the commands that could be used and what I observed.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>?si</td>
<td>This command gives the system information about the computer that you connect to via telnet. “WIN2k (u:Administrator) mem©176/255) 30% GenuineIntel Intel(R) Celeron(R) CPU 2.70 GHz”</td>
</tr>
<tr>
<td>?ssl</td>
<td>This returned “?ssl: -1” I believe that this tells you whether SSL is being used or not</td>
</tr>
<tr>
<td>?clone</td>
<td>It showed the following: &quot;usage ?clone: server[:port] amount&quot;. I tested this by typing “?clone 192.168.6.129:2200 1kb and received **bot.port: connect from 192.168.6.129”</td>
</tr>
<tr>
<td>?clones</td>
<td>This showed usage of “[NETWORK][all] &lt;die</td>
</tr>
<tr>
<td>?login</td>
<td>?login requires username then &lt;ENTER&gt; and password then &lt;ENTER&gt;</td>
</tr>
<tr>
<td>?uptime</td>
<td>Shows how long the system has been up and how long the bot has been up in h/m/s</td>
</tr>
<tr>
<td>?reboot</td>
<td>This command reboots the system you are connected to and responds with &quot;later!&quot;</td>
</tr>
<tr>
<td>?status</td>
<td>Shows information about the computer the bot is on: &quot;service:y user:SYSTEM inet connection:y contype:Lan reboot privs:y&quot;</td>
</tr>
<tr>
<td>?jump</td>
<td>I got no response from this command</td>
</tr>
</tbody>
</table>

Part 3: Behavioral Analysis
Page 16 of 24
Part 3: Behavioral Analysis

Page 17 of 24
Unsure of its use, but when used with a file namespecified such as notepad I received the following: ***chdir: C:\winnt\system32\mfm -> C:\winnt\system32\mfm (0)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?con</td>
<td>Unsure of its use, but when used with a file namespecified such as notepad I received the following: ***chdir: C:\winnt\system32\mfm -&gt; C:\winnt\system32\mfm (0)</td>
</tr>
<tr>
<td>?killsk</td>
<td>Said it couldn’t kill the socket and specified a socket number</td>
</tr>
<tr>
<td>?insmod</td>
<td>Installs loadable modules</td>
</tr>
<tr>
<td>?rmmod</td>
<td>Removes Loadable modules</td>
</tr>
<tr>
<td>?lsmod</td>
<td>Lists loadable modules</td>
</tr>
</tbody>
</table>

**The “Bot Army”**

I infected a windows 98 box so that I now had a bot “army” to play with. I tried to get the commands to work that seemed to allow the access to all of the bots such as ?clones. I was unable to get my bot army” to respond to this command. I also tried to figure out how to get it to launch a DDOS attack that was referenced, but I was unable to do this as well.
Part 4: Code Analysis

Unpacking/Unencrypting

The first attempt to get the dumped code was with LordPE. However, I was not able to obtain an unpacked copy of the process that was decrypted. There are several tools that will dump an aspacked executable; however, I chose not to use them as the results can produce modified results and I wanted to make sure I had a clean copy. In order to get the unpacked code so that we can look at it, we are going to use Ollydbg to dump the code once it is unpacked in memory. The steps below will describe how to get the unpacked code from memory. The description in class showed an UPX packed file and used Ollydbg to dump it from memory. However, it did not discuss how to find the Original Entry Point (OEP) and the process to dump any type of file. It only showed what the code in Assembly would look like for the end of an UPX packed file. There are scripts that will do this for you for any type of packed file such as those found at [http://ollyscript.apsvans.com/](http://ollyscript.apsvans.com/). However, I wanted to learn the manual method of how it worked and with help from Tom Liston, he showed how to do this for any type file and what to look for. I will describe the process below that will work to find the OEP and allow you to dump the contents of msrll.exe.

Before firing up Ollydbg, make sure that Ollydump has been downloaded and placed in the Ollydbg directory. This is a free plugin that can be found at [http://dd.x-eye.net/file/](http://dd.x-eye.net/file/). To get started, first open Ollydbg and load msrll.exe. Now we want to press ALT M to open a memory map of the file. We are looking for the PE header information so that we can find the location in memory of the base image. Once you find the first PE header, click on it and a new window will popup. In the new window scroll down until you find “image base” in the right column. We find the image base is at 400000 and this was recorded for use later. Next, return to the CPU main thread module and you should be at the first command “PUSHAD”. At this point we are going to push F8 to set over the entire subroutine. We want to be at the end of it, not stepping into it. We arrive at a call to msrll.0051d00a. We want to follow this in a dump, but we want to follow it in memory so that we can see the code in memory of this call. Right click over the ESP register and select “Follow in Dump”. Now in the bottom left pane, highlight the first four bytes by holding the left mouse button down and selecting them. We only want the first four bytes before the next jump since this is what was pointed to in that call. Now right click and select breakpoint, hardware on access, dword. A hardware breakpoint does not modify the memory contents. Now comes the fun part, as we are going to run the code up to that point by pressing F9. We should be at a command “jnz short msrll.0051d3ba” in the main cpu thread window. We will press F8 until we hit a “RETN” command, which should be the end of the decompression function. Now we will step INTO it by pressing F7. We are now at the OEP which is very key. Write down the address of the OEP which was 401240 and press CTRL A which will analyze the code. You should now see code. To dump the code, we need that first number we
wrote down of the image base. To find the memory location of the code, you use this formula: OEP – Base Image. In our case its 1240 so now we select plugins from the top of the Ollydbg main window and then choose Ollydump and Dump debugged Process. You put 1240 in the modify box, and then select the “rebuild import” box and dump the code. Now you will have a dumped code that is unpacked and readable for later use.

Program Code Disassembly

Once the bot was connected, I wanted to find out as much as I could about what it was doing. This was done by attaching to the process using Ollydbg on the W2K image and doing the same procedures as I did above to dump the code into readable form. After this, I decided to use breakpoints at key places to try to determine which parts of the code are controlling what. It will be impossible to discuss all of them, since I set so many different ones. However, there are a few that will be discussed. My ultimate goal was to learn how the Bot was logging in to the IRC channel and to be able to control it. After this, I will discuss what I saw found from the code. Here are the techniques used to figure out where to set breakpoints.

Ollydbg has great functionality and flexibility. One feature is to right click on the Main CPU window (upper left window) and select “Search for” then choose “All referenced Text Strings”. A wonderful window will appear that shows you all of the readable text strings and where they are used in the code. From here you can select your breakpoints by highlighting the item and pressing F2. To view your break points and turn them on and off very quickly, use ALT-B to bring up a window that shows them all to you. As a side note, to turn off your hardware breakpoints that you set, you need to use the main tool bar and under DEBUG select “hardware breakpoints”. Another way of finding good places to set breakpoints are the commands that do comparisons such as “strncpy” and “strncpy”. If you select the procedure that does these, then hit CTRL-R you will bring up a window that shows you everytime that command is called and from where. You can then set breakpoints on these commands to see when they are used.

Debugging

To find where the password was being used, I set breakpoints on all commands referencing anything looking like it was associated with a password such as “dcc.pass” and “irc.pass”. I then attempted to launch commands from the IRC window and started by stepping through the code line by line from the breakpoint and watching what was happening. However, after hours of looking at the code, I realized that I was not seeing a password and could not figure out how irc.pass was working, which appeared to control the password to the IRC channel. I then attempted to login via the telnet session using the commands “?login testlogin” and “?pwd testpwd” (it was later determined that ?pwd was not for the password,
but rather for the “present working directory”) which kept triggering the dcc.pass
and then I would end up hitting these two lines of:
0040BC6A PUSH msrl.0040BB49 ASCII "bot.port"
0040BC6F PUSH msrl.0040BB52 ASCII "%s bad pass from "%s"@%s"

I knew I had found the login code for port 2200. During the process, I found one
of the hashes appear that were mentioned in the strings earlier and being passed
as a value: "$1$KZLPLKF$55isA1TvamR7bjAdBziX". In watching the code be
parsed, I saw the value passed and it was broke down by the $. $1$ was always
a constant value passed, then the KZLPLKDf and finally the last letters. These
were concatenated together. Later I realized that the ?login required a username
(appears to be ignored) to be entered and then a password to be entered. The
above string is the hash of the password that expected. I tested three passwords
and set a breakpoint at Address 0040D655 which is the final compare of the
password inputted and the actual password. To do this I used different
usernames with each of the passwords. Every password had its own unique
hash regardless of the username that I used. Since I was only seeing the hash,
which was determined to be stored encrypted in used I decided to see if I could
bypass the authentication by modifying the registers. I set a breakpoint at
0040BBD9 PUSH msrl.0040BB40 ASCII “dcc.pass” Here are the key lines of
code that will allow you to bypass the authentication.
0040BBD9 PUSH msrl.0040BB40 Arg2 = 0040BB40 ASCII “dcc.pass”
0040BBDE PUSH EDX Arg1
0040BBDF CALL msrl.00405872 msrl.00405872
0040BBE4 ADD ESP,10
0040BBE7 TEST EAX,EAX
0040BBE9 JE SHORT msrl.0040BC5A

If you read what this is doing, you see its passing “dcc.pass” as a variable to the
Call to msrl.405872 procedure. Then it adjusts the stack size with the “ADD
ESP,10” and finally the key to the puzzle is its checking EAX. “TEST EAX, EAX”
tell you to check the value of EAX and see if its 0 or 1 (False or True). If its True,
it takes the jump and you end up with a bad password, but if its false you get
authenticated. So, I forced it to bypass the jump by modifying the value of EAX
to be 1 (right click and then Increment). After pressing F9, I returned to my telnet
session and sure enough I was still running. To see if I was authenticated, I
typed ?si and got the information of the system. I was finally in. Once
authenticated I tested the ?md5p and got the following results “?md5p <pass>
<salt>“. So I put in a password of malware and a salt of test and got the
following results: "$1$test$02Ctxuyyv0OHIS01Hx6hS1". I know now that
KZLPLKDf is the salt being used, but I still had no way of determining the
password that was in use.

Another important file was created and used by the malware was jtram.conf. To
find out what this file was used for I set breakpoints in the code that referenced
the file. This file was written to on occasion, six lines at a time, usually after
several login and log outs by the bot due to me pausing the code and stepping through it. However, it was written to after using the ?set command for the passwords. At these times it appears that the passwords generated for the bot was written to it. However, the hashes do not match those passed as arguments so my guess is that they are the password hashes that have been encrypted and written to the file. From all indications, this file is read at initial startup and then not again. I believe this to be the case because I never saw it read during any login attempts.
Part 5: Analysis Wrap-Up

The malware that we were asked to analyze is an IRC bot. It connects to the IRC server called collective7.zxy0.com on one of three ports: 8080, 9999 and 6667. It also listens on port 2200 which appears to be the command port to talk to the bot directly and I believe to control the bot network, but I could not validate this. Based on observations of the bot and the commands, the owner of the bot network has full control over all of the infected the systems.

The malware has the capability to allow full control over each individual computer that has been infected. The owner of the PC no longer is the “true” owner of the system. Based upon the strings, it allows a connection to the bot via a web interface using Mozilla, but I couldn’t not figure out how to issue commands once connected, it denies connections from Internet Explorer. It has the capability as well to launch the following DOS attacks: JOLT2, Smurf, UDP, SYN and Ping floods, combined with all the Bots participating, the bot network would be able to launch a DDOS attack against a specific target. It appears that the bots can be controlled by specifying a specific network that certain bots are on or all of them participating due to the parameters of the “?clones” command. This program would be used by someone who wanted to have an “army” at their disposal. There are many people who would use this capability such as people out for revenge or someone with a bot Army for hire that wanted to make money to take out a target. It could also be by folks participating in IA warfare and to disrupt Internet connectivity. This could be devasting for our Armed Forces, especially in light of the use of the Internet for daily operations.

In order to help protect against this type of malware several things should be done. Users need to ensure they have antivirus software running and updated as this malware is detected by Norton as Backdoor.IRC.bot. Organizations can do this by using something like Norton’s Coporate Edition to manage the who organization. The firewall, whether at corporate or home, needs to block unneeded ports such as those being used here. If infected, it would stop the bot from connecting. Also, users need to log on with the most restrictive privileges. If they were infected, it appears the bot has the privileges of the logged on user. The system needs to be kept up will all patches to ensure that there is no way to be infected using a known exploit such as the .chm vulnerability. One of the most important things is user training. Users need to learn not to open email from someone they don’t know or to click on attachments they are unsure of. They also need to learn to be careful in downloading software from unknown websites as these can contain the malicious code.

To clean your system of this bot, you need to run your antivirus software and remove all references that it finds to it. Also, you can manually remove it by deleting the registry keys that were mentioned above for the service and under the Run key. The service can be stopped using pskill or Process Explorer, then the directory mfm and its contents can be deleted. This type of bot should be easy to prevent if the steps above are taken.
Citation of Sources
Liston, Tom. Conversation via Jabber on finding the OEP. September 2004


Sites for Tools
VMWare Workstation 4.5.1: http://www.vmware.com/
Linux Redhat: http://www.linux.org/
Ollydbg: http://home.t-online.de/home/Ollydbg/
Ethereal: http://www.ethereal.com/
SNORT: http://www.snort.org/
Filemon: http://www.sysinternals.com/ntw2k/source/filemon.shtml
RegMon: http://www.sysinternals.com/ntw2k/source/regmon.shtml
TDIMon: http://www.sysinternals.com/ntw2k/freeware/tdimon.shtml
LordPE: http://mitglied.lycos.de/yoda2k/LordPE/info.htm
Regshot: http://regshot.ist.md/ however it was unavailable the last time I checked. It can be found easy by a simple Google query.
PSkill: http://www.sysinternals.com/ntw2k/freeware/pskill.shtml
## Upcoming SANS Forensics Training

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