



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Reverse-Engineering Malware: Malware Analysis Tools and Techniques (Forensics)"  
at <http://www.giac.org/registration/grem>

# **GIAC REVERSE ENGINEERING MALWARE (GREM) PRACTICAL V1.0**

Submitted by Lance Mueller  
October 22, 2004

© SANS Institute 2000 - 2005, A

## Statement of Purpose

3

## Definitions

3

### Malware

3

### Viruses

3

### Trojan Horse

4

### Worm

4

## Laboratory Setup

4

### Hardware

4

### Analysis tools & Software

4

### WinHex

4

### Md5Sum

5

### Snort

5

### Netstat

5

### PEInfo

5

### Ollydbg

5

[Bintext](#)

5

[TCPView](#)

6

[Filemon](#)

6

[Regmon](#)

6

[TDImon](#)

6

[Regshot](#)

6

[Netcat](#)

6

[VMware Workstation](#)

6

## **Properties of the Malware Specimen**

9

## **Behavioral Analysis**

14

[Executing the malware](#)

16

[Port Information](#)

17

[Registry Information](#)

19

[File System Information](#)

20

[Operating Systems](#)

21

[Network Communications](#)

21

## Code Analysis

26

JTRAM.CONF

26

Port 2200

28

IRCD CHANNEL #MILS

30

## Conclusion

31

Removal

31

## References

33

© SANS Institute 2000 - 2005

## Statement of Purpose

The Internet has become an indispensable resource and common way of life to many people in the world today. As technology has increased and the cost of equipment and Internet connectivity has decreased, the number of households and businesses online continues to grow each year.

As with many things in life, there are negative aspects associated with the positive aspects of having so many people connected to the Internet, specifically the number of targets available to malware authors. Even though technology has advanced and global connectivity has increased, online education is still behind the technology curve. Authors of malware take advantage of this lack of education and the existence of computer weaknesses to create programs that either do harm to a person's computer or obtain control of that computer for various reasons. According to a recent study done by the SANS Institute [1], an unprotected newly installed operating system exposed to the Internet has approximately 20 minutes to live before being attacked and infected with some type of malware.

The purpose of this paper is to demonstrate how to dissect a sample piece of malware and to explain its purpose and usage in order better understand how malware operates and infiltrates our computers worldwide. Taking a two-step approach I will perform a behavior analysis of the malware to better understand how it behaves on the infected operating system. I will then perform a code analysis of the malware to examine and better understand the malware functionality, capability and operation.

## Definitions

### Malware

Malware is a term that has been coined from the words "Malicious Software" and is used to describe software that has been produced for the purpose destroying, disrupting or controlling a computer system (Webopedia, <http://www.webopedia.com/TERM/m/malware.html> [2]).

### Viruses

A virus is a piece of code or program that is loaded onto your computer without your knowledge and performs some type of undesirable or destructive action<sup>1</sup>. Most viruses have a replication mechanism that causes the virus to spread or infect other computers or programs (Webopedia, <http://www.webopedia.com/TERM/v/virus.html> [2]).

### Trojan Horse

A Trojan horse is a program that masquerades as a legitimate program,

but really performs some other type of action, commonly destructive (Webopedia, [http://www.webopedia.com/TERM/T/Trojan\\_horse.html](http://www.webopedia.com/TERM/T/Trojan_horse.html) [2]).

## **Worm**

The term “worm” is use to refer to a virus or malware specimen that has a self-replication mechanism that does not require any human interaction. These types of viruses typically use system vulnerabilities or weak security habits to propagate from one machine to the next (Webopedia, <http://www.webopedia.com/TERM/w/worm.html> [2]).

## **Laboratory Setup**

In order to properly and safely examine malware specimens in a controlled laboratory environment, certain precautions should be taken in order to avoid contamination of your lab equipment and provide control mechanisms that ensure the malware specimen does not propagate outside of your lab environment and begin communicating with other computers outside of your control.

## **Hardware**

The laboratory environment used for this malware analysis consists of one 2 GHz Intel based computer, with 2 GB of RAM, and an IDE 80 GB hard drive. Since multiple VMware virtual machines were used a 19” flat screen monitor was used to provide more video real estate and the ability to organize the various VMware machines across the screen.

## **Analysis tools & Software**

Various software tools will be needed throughout the malware analysis and were loaded onto the laboratory machine used for the analysis. The following software tools were installed and utilized during the analysis of the malware.

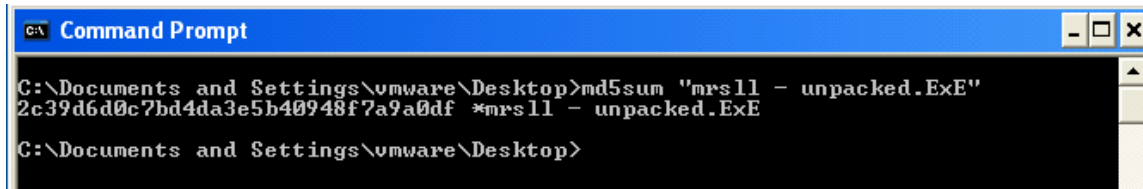
### **WinHex [3]**

Winhex is a shareware Hex Editor program that I used during the code analysis phase to examine the contents in Hex and look for known indicators of compressed files. The program also has a “calculate digest” feature which can be used to calculate a hash for any file opened in the hex editor.

### **Md5Sum [4]**

Md5Sum is Win32 port of the UNIX md5 command line hash utility. From

the command line, you can specify a filename and get the resulting Md5 hash from the contents of that file. I used this program during the initial properties portion of the analysis to record the initial MD5 hash of the malware. Additionally, I used it to verify the integrity of various executables during the analysis

A screenshot of a Windows Command Prompt window. The title bar is blue and says "C:\ Command Prompt". The command prompt shows the following text:

```
C:\Documents and Settings\vmware\Desktop>md5sum "mrs11 - unpacked.Exe"  
2c39d6d0c7bd4da3e5b40948f7a9a0df *mrs11 - unpacked.Exe  
C:\Documents and Settings\vmware\Desktop>
```

Figure 1 -Example of MD5sum.exe

### Snort [\[5\]](#)

Snort is an Intrusion Detection System (IDS), but also acts as a raw packet capturing program. During the behavior stages of this analysis, I used Snort to capture raw data packets being sent and received by the malware to better understand its capabilities

### Netstat [\[6\]](#)

Netstat is a executable included with various Windows installations and is used to see the current network communication information. I used this tool during the behavioral analysis to see and verify any communications coming from or going to my test machine.

### PEInfo [\[7\]](#)

PEInfo is a utility which analyzes an Executable program and provides information about the executable such as the Portable Executable (PE) header information.

### Olllydbg [\[8\]](#)

During the code analysis phase, it was necessary to examine the actual code as it exists in its compiled state. Additionally, it may become necessary to watch the malware specimen operate at various states of execution. Olllydbg is a disassembler and debugger that allows you to examine executable programs and view the internal code structure. Olllydbg can also display various variables and functions in real-time as the program runs.

### Bintext [\[9\]](#)

Bintext is a utility that will scan through a loaded file and extract recognizable strings by pre-set filter settings. This tool was used during the code analysis stage to extract recognizable commands or strings such as domain names, IP addresses, passwords and usernames.



**TCPView [\[10\]](#)**

TcpView is a utility by Sysinternals and is used to monitor port states and conditions. This utility is used during the behavioral phase to see when ports are used for communication and/or which ports are controlled by the malware.

**Filemon [\[10\]](#)**

Filemon is a utility that monitors all file system access and during the behavior stages was be used to monitor file system access by the malware.

**Regmon [\[10\]](#)**

Regmon is a utility that monitors all registry access and was be used during the behavior analysis to see what registry keys are being read from, written to and created.

**TDImon [\[10\]](#)**

TDImon is a utility that monitors all network access and communications. This utility was used during the behavioral stage. It does not capture the actual network traffic payload, but does capture the header information used during the communication in order to show what ports and IP address are being accessed by various programs.

**Regshot [\[11\]](#)**

Regshot is another registry monitoring utility that takes a pre-malware snapshot of the registry and then after executing the suspected malware compares the state of the registry with the pre-execution state and reports any differences. This utility was used during the behavioral analysis to see the changes made to the registry.

**Netcat [\[12\]](#)**

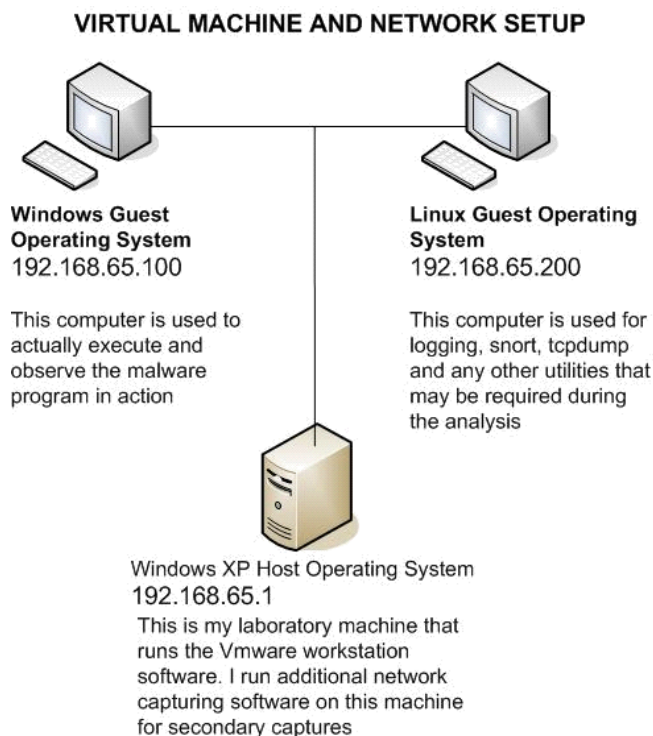
Netcat is an open source tool originally written by Hobbit, which allows you to establish raw TCP and UDP connections and send data from one host to another. It can be used as a makeshift listener to capture data being sent to a particular port. I used netcat during the behavioral stage to transfer files from one guest machine to the other.

**VMware Workstation [\[13\]](#)**

In order to safely and thoroughly examine a malware specimen it was necessary to observe the malware in action and actually infect a computer's operating system. In order to do this in the safest most controlled environment possible, I used the VMware Workstation Software for the behavioral analysis. I used the Microsoft Windows XP, Service Pack 2 Operating System on my laboratory computer and then

utilized the VMware software to run a second version of Microsoft Windows XP (guest) operating system. The Microsoft Windows XP installation used as the guest “victim” machine was an unpatched version of XP. In addition, a second VMware virtual machine containing RedHat v. 9 Linux was initially used as a network monitoring tool and later as part of the behavioral analysis. This Linux guest machine utilized the 2.4.20-8 Linux kernel.

The following diagram is an example of the analysis setup that I used in the laboratory environment.



**Figure 2 - Virtual machine and network setup**

© SANS

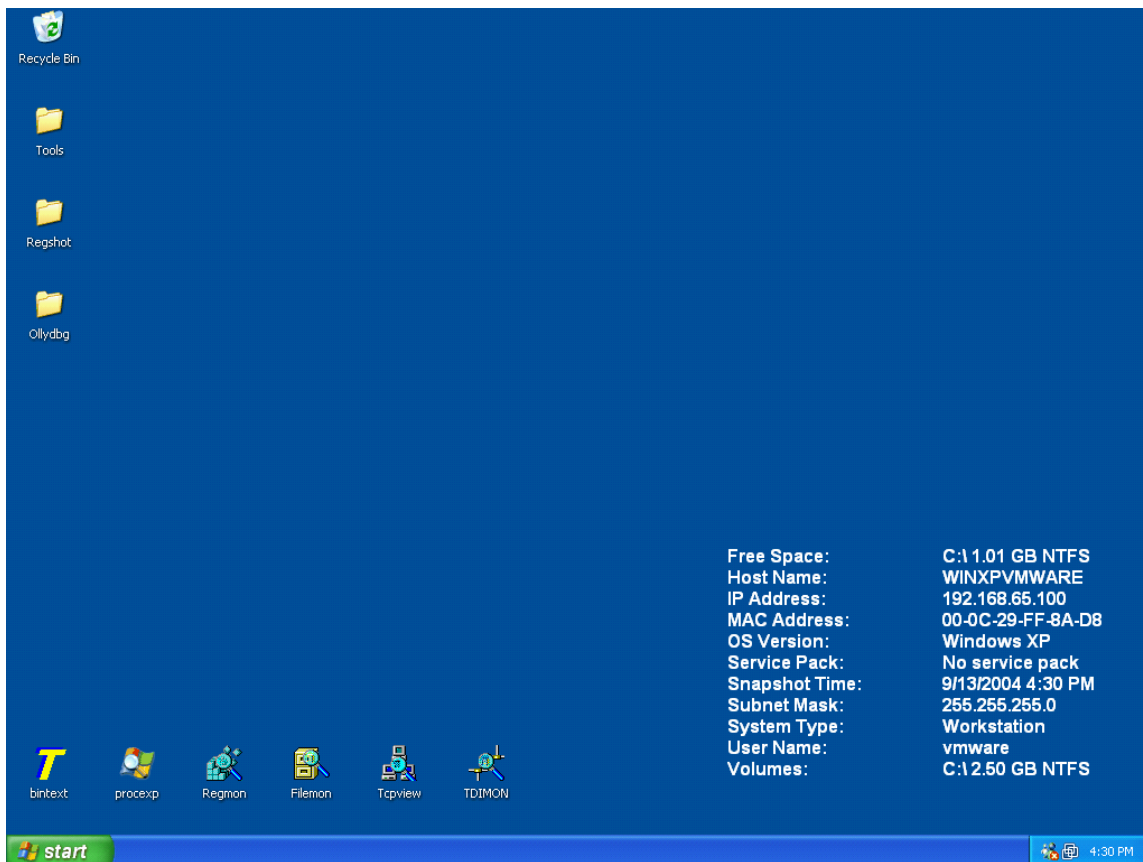
Host-only networking was the used. Host-only mode isolated the virtual machine (s) and limited the data communication on the virtual network to the virtual machines and directly to the host. The host did not operate as a gateway and forward the communication beyond the virtual Ethernet adapter. This configuration ensured that any network communication created by the malware was confined on the virtual network and was not allowed to travel outside of the virtual network. The following network parameters were used:

Host - XP Operating System	IP Address = 192.168.65.1
Guest – XP Operating System	IP Address = 192.168.65.100
Guest – RedHat Linux	IP address = 192.168.65.200

Additionally, the host machine was completely disconnected from any additional network connections to ensure that all communications remained on the virtual network. The built-in Microsoft Windows Firewall was also activated and configured to deny all network communication from any host, including those on the same subnet. This ensured that whatever propagation vector was being used by the malware, it could not communicate with my Host operating system.

After installing the various VMware virtual machines and the associated tools discussed above, I created several “snapshots” to save their current state. VMware Workstation allows you to create “snapshots” of your virtual Operating systems and then go back to that state at anytime. The “snapshot” feature allowed me to begin the behavior analysis of a malware specimen and then go back to a clean state and do it over and over again without having to spend time reinstalling all the necessary tools or the operating system.

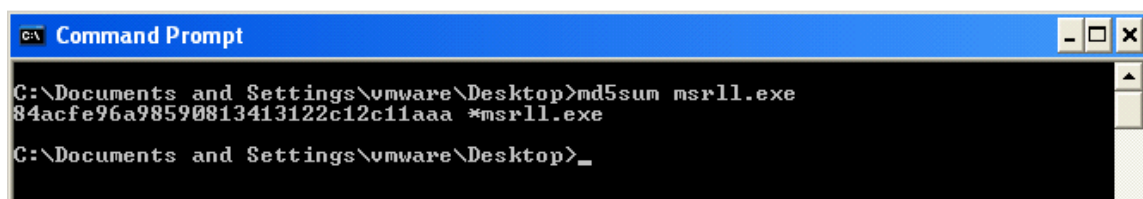
© SANS Institute 20



**Figure 3 - Desktop screenshot of Windows virtual machine with tools preloaded and ready for analysis**

## Properties of the Malware Specimen

For this analysis, I have obtained a malware specimen which is named “msrll.exe”. I have transferred a copy of the malware specimen to my Guest Windows XP virtual machine that I eventually infected with this malware. Using the md5sum.exe tool I mentioned above, I calculated the MD5 hash value of this executable.



**Figure 4 - computing the MD5 hash value of the malware specimen using md5sum.exe**

Additionally, I have noted the following file properties:

Name: msrll.exe  
Logical Size: 41,984 bytes  
Executable type: Windows PE executable (Windows 95, 98, 2000 & XP)  
MD5 hash: 84ACFE96A98590813413122C12C11AAA

I examined the executable using the Winhex hex editor and the observed the executable header, noting that it was a Windows PE executable:

```
MZ|.....yy..  
.....@..  
.....  
.....!  
..@..'.f!..L!Th  
is program canno  
t be run in DOS  
mode....$.....  
PE..L...5.y@....
```

Figure 5 - PE File Header

A closer examination of the header revealed that the executable was packed using a real-time compressor named Aspack [\[14\]](#). Executable compressors (also referred to as packers) are commonly used by malware authors to assist in avoiding detection by anti-virus software as well as making it more difficult to reverse engineer. Some compressors actually make the executable larger even though they are considered compressors. Aspack is a commonly used executable compressor and can be easily detected by observing the text “.aspack” in the Sections portion of the file’s header:

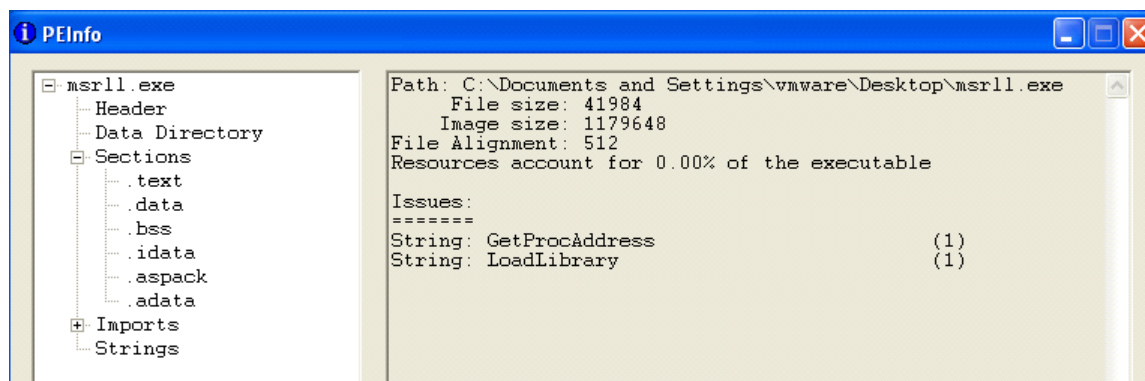
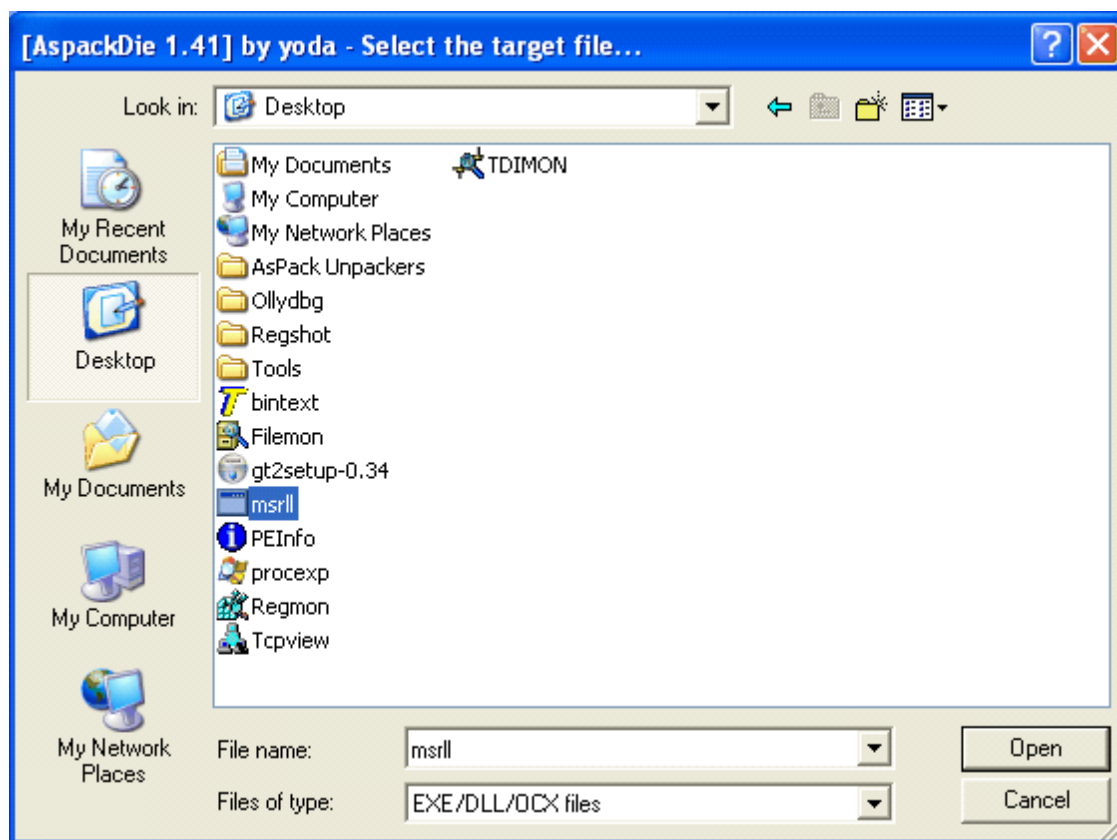


Figure 6 - Using PEInfo to examine the file properties

In order to properly analyze this executable during the code analysis later in this document, the file needed to be uncompressed before it could be correctly read and analyzed by a disassembler. Since the file was in a compressed state, the text fragments that are normally visible when using a utility such as BinText were not visible in plain text. The file needed to be uncompressed in order to examine any embedded readable strings. Since the malware had been identified as being compressed with Aspack, an uncompressor was needed to properly extract the malware back to its original state. A program named AspackDie [\[15\]](#)

written by yoda was used to unpack the program compressed with a versions of Aspack.



**Figure 7 - Select the file you would like to unpack**

Once the file to be unpacked is selected this utility will attempt to correctly unpack the original malware and save it to the file named “unpacked.exe” in the same directory as where the original is located. I renamed the unpacked version to “msrll.exe – unpacked.exe” and noted the following file attributes:

Name:	msrll - unpacked.exe
Logical Size:	1,175,552 bytes
Executable type:	Windows PE executable
MD5 hash:	2c39d6d0c7bd4da3e5b40948f7a9a0df

A tool named GT2 (GetType) by PHaX [\[16\]](#) was also used to analyze and possibly determine the type of compressor or packer used on the executable. It should be noted that this utility does not detect every type of compressor, but it is a good starting point to help identify which compressor was used.

```
C:\Documents and Settings\vmware\Desktop>gt2 msrll.exe
gt2 0.34 (c) 1999-2004 by PHaX (coding@helger.com)

- msrll.exe (41984 bytes) - binary

Is a DOS executable
Size of header:      00000040h/64 bytes
File size in header: 00000490h/1168 bytes
Entrypoint:         00000040h/64
Overlay size:       00007F70h/40816 bytes
No relocation entries

PE EXE at offset 00000080h/128
Entrypoint:         00009201h / 37377
Entrypoint RVA:     0011D001h
Entrypoint section: '.aspack'
Calculated PE EXE size: 0000A400h / 41984 bytes
Image base:        00400000h
Required CPU type:  80386
Required OS:       4.00 - Win 95 or NT 4
Subsystem:         Windows GUI
Linker version:    2.56
Stack reserve:     00200000h / 2097152
Stack commit:      00001000h / 4096
Heap reserve:      00100000h / 1048576
Heap commit:       00001000h / 4096
Flags:
  Relocation info stripped from file
  File is executable
  Line numbers stripped from file
  Local symbols stripped from file
  Debugging info stripped from file in .DBG file

Processed with:
  Found packer 'ASPack 2.12'
```

Figure 8 - GT2.EXE used to help determine which packer was used

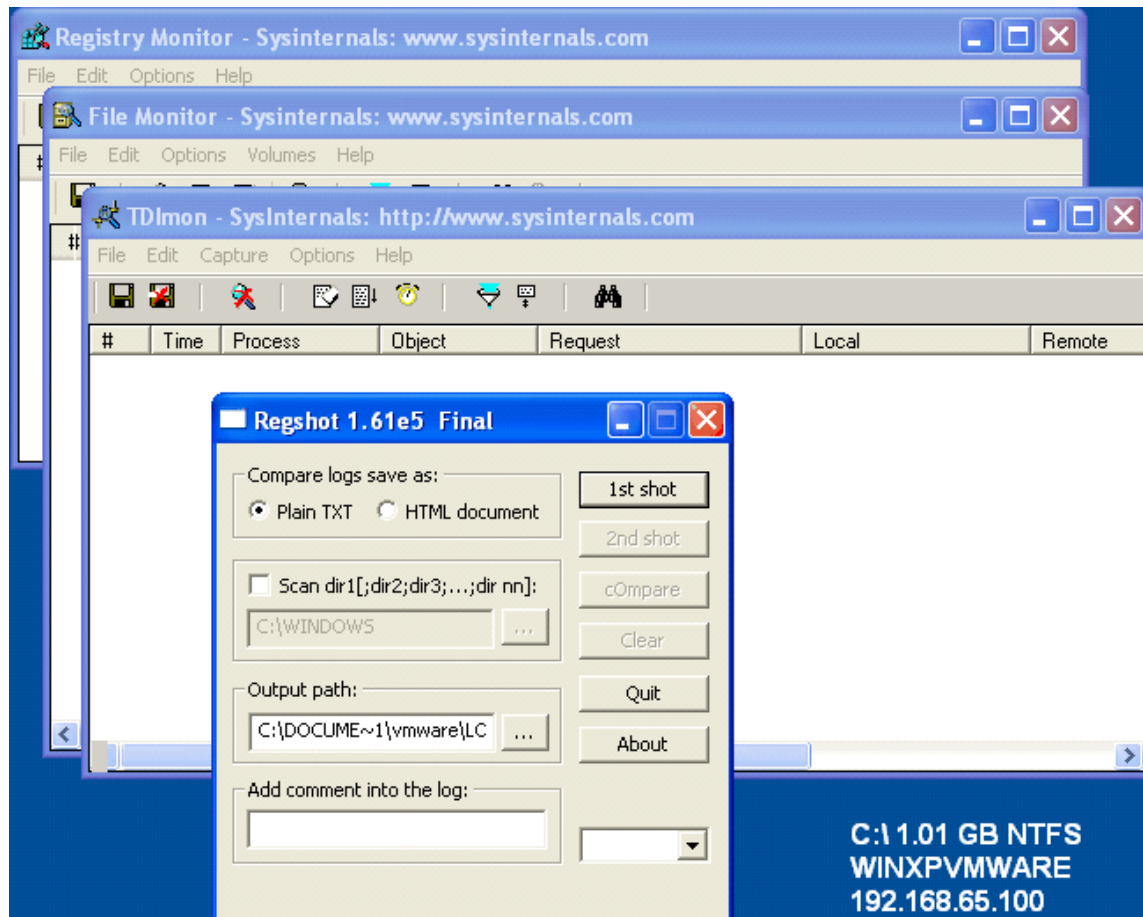
[illegible]

Numerous recognizable and important text fragments were located inside this malware specimen. Many references to IRC and IRC commands, as well as a registry key were located in the uncompressed executable.



## Behavioral Analysis

To begin the behavioral analysis, I setup some monitoring utilities to begin monitoring the file system, network and registry. I started Filemon, Regmon and then paused them until just prior to the malware being executed. Once the system monitoring tools were started, a registry snapshot was obtained using Regshot so it can later be compared with the post malware registry to look for changes.



**Figure 10 - Virtual machine setup ready for malware infection**

Before I infected the virtual machine with the malware, I started the network monitoring tool on the second virtual machine so they could begin capturing all network traffic.

Using snort, an open source Intrusion Detection System (IDS) written by Martin Roesch [\[5\]](#), I began capturing all network traffic and saving the captured packets to a text file for later review.

I started Snort in packet capture mode using the following command:

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

This command puts the snort IDS system into packet dump mode (acts like a simple packet capturing tool) and causes it to capture all packets at the application level and saves the data into the sniffer.log file as well as displaying it to standard out (console).

```
[root@localhost root]# snort -vd | tee /tmp/sniffer.log
Running in packet dump mode
Log directory = /var/log/snort

Initializing Network Interface eth0

    === Initializing Snort ===
Initializing Output Plugins!
Decoding Ethernet on interface eth0

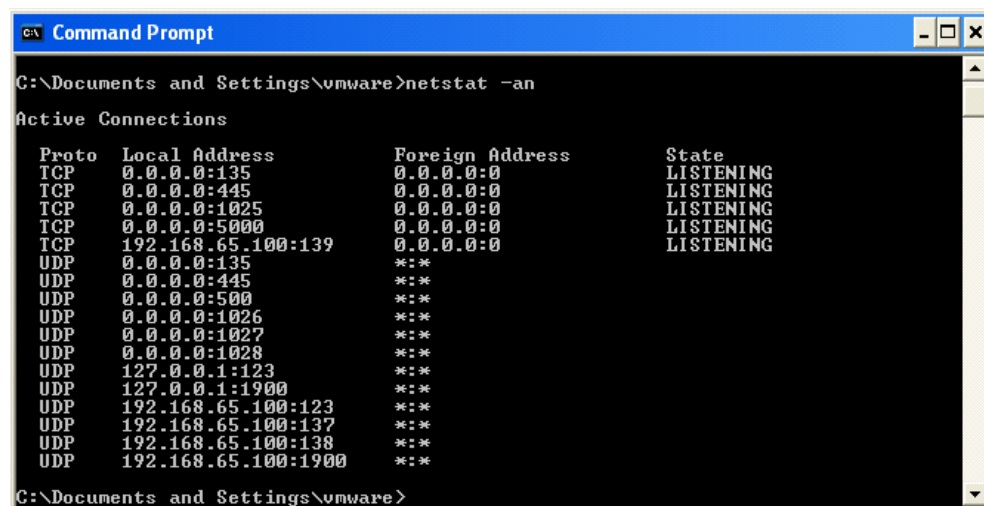
    === Initialization Complete ===

-*> Snort! <*-
Version 2.0.4 (Build 96)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
```

Figure 11 - Starting the snort packet capturing software on the second virtual machine

Before I started the monitoring tools and launched the malware executable, I used the netstat command to get a list of listening ports and established communications so I could use them to compare after infection. I saved this information by piping it to a text file using the command:

```
"netstat -an > before-ports.txt"
```



```
C:\Documents and Settings\vmware>netstat -an

Active Connections

Proto Local Address           Foreign Address         State
TCP   0.0.0.0:135              0.0.0.0:0               LISTENING
TCP   0.0.0.0:445              0.0.0.0:0               LISTENING
TCP   0.0.0.0:1025             0.0.0.0:0               LISTENING
TCP   0.0.0.0:5000             0.0.0.0:0               LISTENING
TCP   192.168.65.100:139       0.0.0.0:0               LISTENING
UDP   0.0.0.0:135              *:*                     *:*
UDP   0.0.0.0:445              *:*                     *:*
UDP   0.0.0.0:500              *:*                     *:*
UDP   0.0.0.0:1026             *:*                     *:*
UDP   0.0.0.0:1027             *:*                     *:*
UDP   0.0.0.0:1028             *:*                     *:*
UDP   127.0.0.1:123            *:*                     *:*
UDP   127.0.0.1:1900           *:*                     *:*
UDP   192.168.65.100:123       *:*                     *:*
UDP   192.168.65.100:137       *:*                     *:*
UDP   192.168.65.100:138       *:*                     *:*
UDP   192.168.65.100:1900      *:*                     *:*

C:\Documents and Settings\vmware>
```

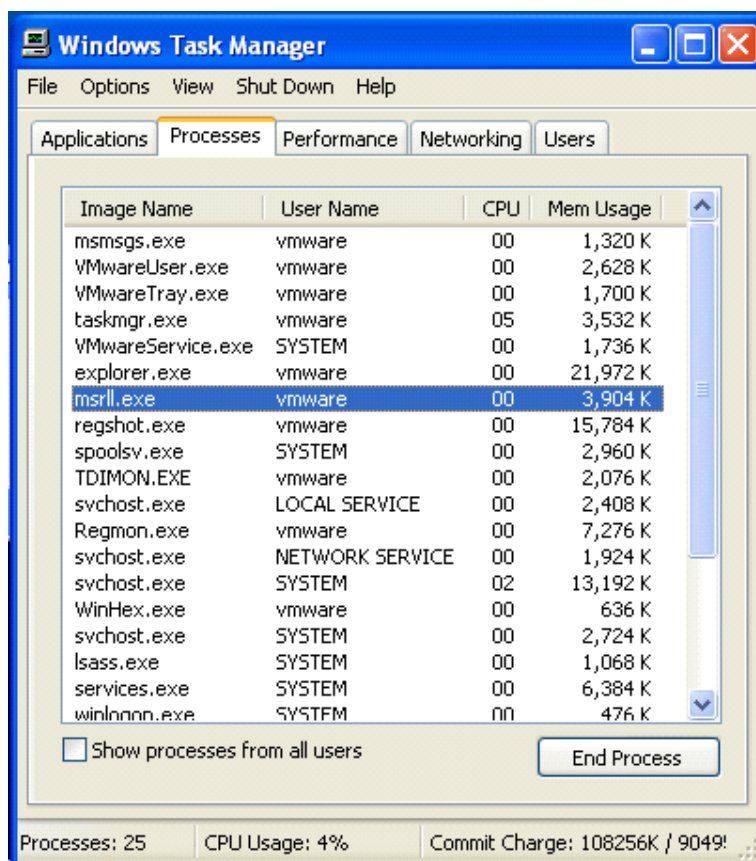
**Figure 12 – “netstat –an” command to capture all open ports**

As a good precaution, before I launched the malware I took the original netstat.exe executable that is located in the c:\windows\system32 folder and made a copy of it and named it netstat.backup.exe. Since I will want to use this tool again after the malware has been executed I needed to ensure the malware specimen did not alter or replace it with a modified version. I further safeguarded this by using the md5sum utility and generated a MD5 hash of the original netstat.exe and then compared the hash with the netstat.exe file whenever I wanted to use it post infection.

```
md5sum c:\windows\system32\netstat.exe > C:\saved_md5\netstat.txt
```

## Executing the malware

Once I collected this preliminary information, I started the monitoring tools and executed the malware. Upon executing the malware, I observed the mouse pointer turn to an hourglass as if the system was busy. Approximately a second later, the file I had originally clicked on (msrll.exe), disappeared off the virtual machine desktop as if it had been deleted. After letting it execute for approximately 30 seconds, I viewed the task manager and observed the malware executable still running.



**Figure 13 - Task Manager on virtual machine showing malware still running**

## Port Information

Before I terminated the running malware, I used the “netstat -an” command to again collect open ports and save it to a text file named after-ports.txt. Examining the ports that were captured before the malware execution and the post infection list, I could clearly see several ports that have been opened by the malware.

before-ports.txt - Notepad				after-ports.txt - Notepad			
File Edit Format View Help				File Edit Format View Help			
Active Connections				Active Connections			
Proto	Local Address	Foreign Address		Proto	Local Address	Foreign Address	
TCP	0.0.0.0:135	0.0.0.0:0		TCP	0.0.0.0:113	0.0.0.0:0	
TCP	0.0.0.0:445	0.0.0.0:0		TCP	0.0.0.0:135	0.0.0.0:0	
TCP	0.0.0.0:1025	0.0.0.0:0		TCP	0.0.0.0:445	0.0.0.0:0	
TCP	0.0.0.0:5000	0.0.0.0:0		TCP	0.0.0.0:1025	0.0.0.0:0	
TCP	192.168.65.100:139	0.0.0.0:0		TCP	0.0.0.0:2200	0.0.0.0:0	
UDP	0.0.0.0:135	*:*		TCP	0.0.0.0:5000	0.0.0.0:0	
UDP	0.0.0.0:445	*:*		TCP	192.168.65.100:139	0.0.0.0:0	
UDP	0.0.0.0:500	*:*		UDP	0.0.0.0:135	*:*	
UDP	0.0.0.0:1026	*:*		UDP	0.0.0.0:445	*:*	
UDP	0.0.0.0:1027	*:*		UDP	0.0.0.0:500	*:*	
UDP	0.0.0.0:1028	*:*		UDP	0.0.0.0:1026	*:*	
UDP	127.0.0.1:123	*:*		UDP	0.0.0.0:1027	*:*	
UDP	127.0.0.1:1900	*:*		UDP	0.0.0.0:1028	*:*	
UDP	192.168.65.100:123	*:*		UDP	127.0.0.1:123	*:*	
UDP	192.168.65.100:137	*:*		UDP	127.0.0.1:1900	*:*	
UDP	192.168.65.100:138	*:*		UDP	192.168.65.100:123	*:*	
UDP	192.168.65.100:1900	*:*		UDP	192.168.65.100:137	*:*	
				UDP	192.168.65.100:138	*:*	
				UDP	192.168.65.100:1900	*:*	

**Figure 14 - pre-infection port states .vs post-infection port states**

I then used the TCPView tool by Sysinternals, which showed me the open ports and the processes responsible for each port.

© SANS Inst

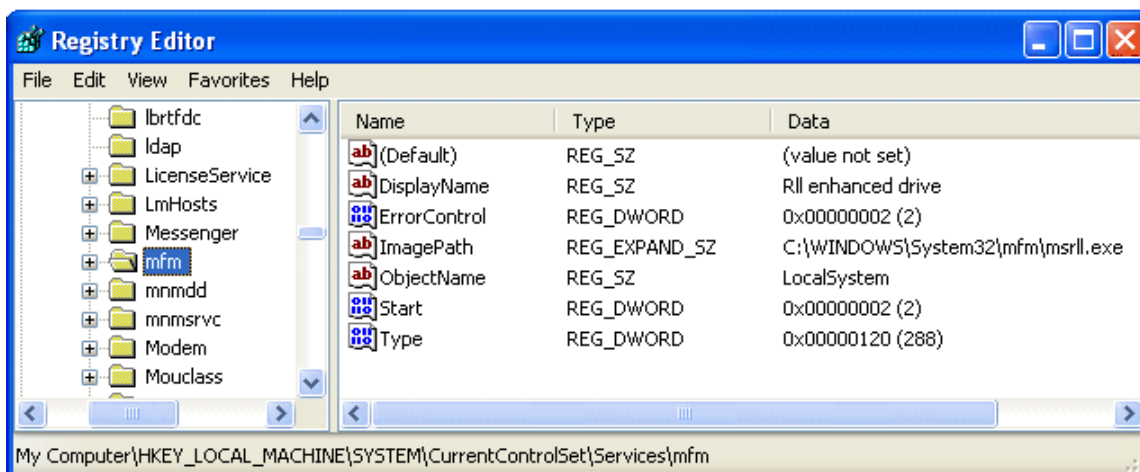
Proc...	Protocol	Local Address	Remote Address	Stat
lsass.exe:696	UDP	0.0.0.0:500	*,*	
msrll.exe:1440	TCP	0.0.0.0:113	0.0.0.0:0	LISTEN
msrll.exe:1440	TCP	0.0.0.0:2200	0.0.0.0:0	LISTEN
svchost.exe:1124	UDP	0.0.0.0:1026	*,*	
svchost.exe:1140	TCP	0.0.0.0:5000	0.0.0.0:0	LISTEN
svchost.exe:1140	UDP	127.0.0.1:1900	*,*	
svchost.exe:1140	UDP	192.168.65.100:1900	*,*	
svchost.exe:864	TCP	0.0.0.0:135	0.0.0.0:0	LISTEN
svchost.exe:864	UDP	0.0.0.0:135	*,*	
svchost.exe:956	TCP	0.0.0.0:1025	0.0.0.0:0	LISTEN
svchost.exe:956	UDP	0.0.0.0:1027	*,*	
svchost.exe:956	UDP	0.0.0.0:1028	*,*	
svchost.exe:956	UDP	127.0.0.1:123	*,*	
svchost.exe:956	UDP	192.168.65.100:123	*,*	
System:4	TCP	0.0.0.0:445	0.0.0.0:0	LISTEN
System:4	TCP	192.168.65.100:139	0.0.0.0:0	LISTEN
System:4	UDP	0.0.0.0:445	*,*	
System:4	UDP	192.168.65.100:137	*,*	
System:4	UDP	192.168.65.100:138	*,*	

Figure 15 - Fport.exe output

I observed two additional TCP ports appeared after I executed the malware. TCP Port 113 & port 2200 were being help open by the “msrll.exe” program and this information corroborated the information I previously collected with the netstat command. I then terminated the malware process by killing it via the task manager and paused all the monitoring tools

## Registry Information

Using the Regshot tool, a second registry shapshot was taken and saved. Examining the Regshot comparison revealed several added registry keys that pertained to the execution of the malware. Specifically the registry key **HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Services\mfm** was added. Under this key several values were added. Looking at the live registry I saw that a new service has been added under the “mfm” key.



**Figure 16 - Newly installed service caused by the malware**

The service named “Rll enhanced drive” was added which will execute the program named c:\windows\system32\mfm\msrll.exe upon system startup.

#### Windows Service Parameters [\[17\]](#)

DisplayName=Rll enhanced drive	Name displayed in Services MMC
ErrorControl=2	Setting in case service fails to start
ImagePath=c:\windows\system32\mfm\msrll.exe	Program to execute when service starts
ObjectName=LocalSystem	Userspace this service will operate in. This service will execute with SYSTEM privileges.
Start=2	Execute at system startup
Type=120 (288 decimal)	This is a system setting designating what kind of process this is

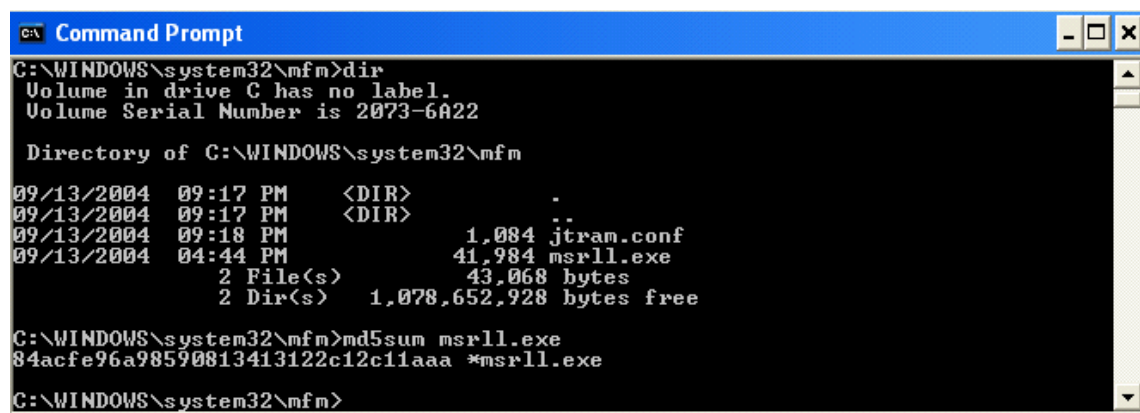
Due to the fact that the malware was installed as a service, with a Start parameter of “2”, this malware will be started each time the system is restarted regardless of whether anyone logs into the system or not. The malware is installed as a system service that starts automatically upon the operating system starting.

## File System Information

The Regshot analysis also identified two files that were added into a folder named C:\Windows\system32\mfm. Looking in that folder I located two new files:

```
C:\WINDOWS\system32\mfm\jtram.conf
C:\WINDOWS\system32\mfm\msrll.exe
```

Examining the msrll.exe file, I observed it to be the same exact size as the original malware I executed from the desktop. I then used the md5sum utility to check the hash value for this executable.



```
C:\WINDOWS\system32\mfms>dir
Volume in drive C has no label.
Volume Serial Number is 2073-6A22

Directory of C:\WINDOWS\system32\mfms

09/13/2004  09:17 PM    <DIR>          .
09/13/2004  09:17 PM    <DIR>          ..
09/13/2004  09:18 PM                1,084 jtrams.conf
09/13/2004  04:44 PM               41,984 msrll.exe
                2 File(s)                43,068 bytes
                2 Dir(s)          1,078,652,928 bytes free

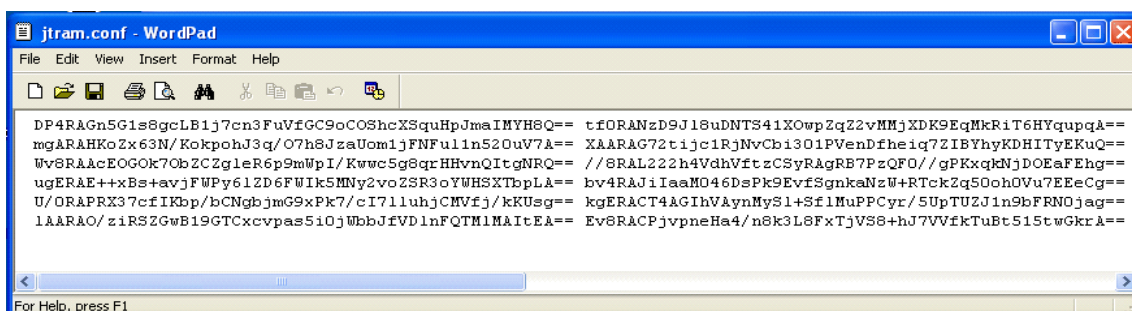
C:\WINDOWS\system32\mfms>md5sum msrll.exe
84acfe96a98590813413122c12c11aaa *msrll.exe

C:\WINDOWS\system32\mfms>
```

Figure 17 - Newly created folder named "MFM"

The MD5 hash matched the hash value of the original malware specimen I had examined before I executed it. This verified that the malware copies itself into this new folder under the system32 directory and is executed as a service each time the system starts up.

I examined the second file created in the new folder named jtrams.conf and found it to be a simple text file with what appears to be obfuscated text strings. Further code analysis will be needed to try and reveal these text strings.



```
jtrams.conf - WordPad
File Edit View Insert Format Help

DP4RAGn5G1s8gcLB1j7cn3FuVfGC9oCOShcXSquHpJmaIMYH8Q== tf0RANzD9J18uDNTS41XOwpZqZ2vMMjXDK9EqMkRiT6HYqupqA==
mgARAHKoZx63N/KokpohJ3q/O7h8JzaUomljFNFu1ln52OuV7A== XAARAG72tjic1RjNvCb13O1PVenDfheiq7ZIBYhyKDHITyEKuQ==
Uv8RAAcEOGOk7ObZCZgleR6p9mWpI/Kwuc5g8qrHHvMQItgNRQ== //8RAL222h4VdhVftzCSyRAgRB7PzQFO//gPKxqkNjDOEaFEhg==
ugERAEE++xBs+avjFWPy612D6FWIk5MNY2voZSR3oYWHSTbplA== bv4RAJiIaaM046DsPk9EvfSgnkaNzW+RTckZq50ohOVu7EEeCg==
U/ORAPRX37cfIKbp/bCNgbjmG9xPk7/cI711uhjCHVfj/kKUsq== kgERACT4AGIhVaynMyS1+Sf1MuPPCyr/5UpTUZJ1n9bFRN0jag==
1AARA0/z1RSZGwB19GTCxcvpaS5i0jWbbJfVDlnFQTM1MAItEA== Ev8RACpjvpneHa4/n8k3L8FXTjV8S+hJ7VVfkTbT515twGkrA==

For Help, press F1
```

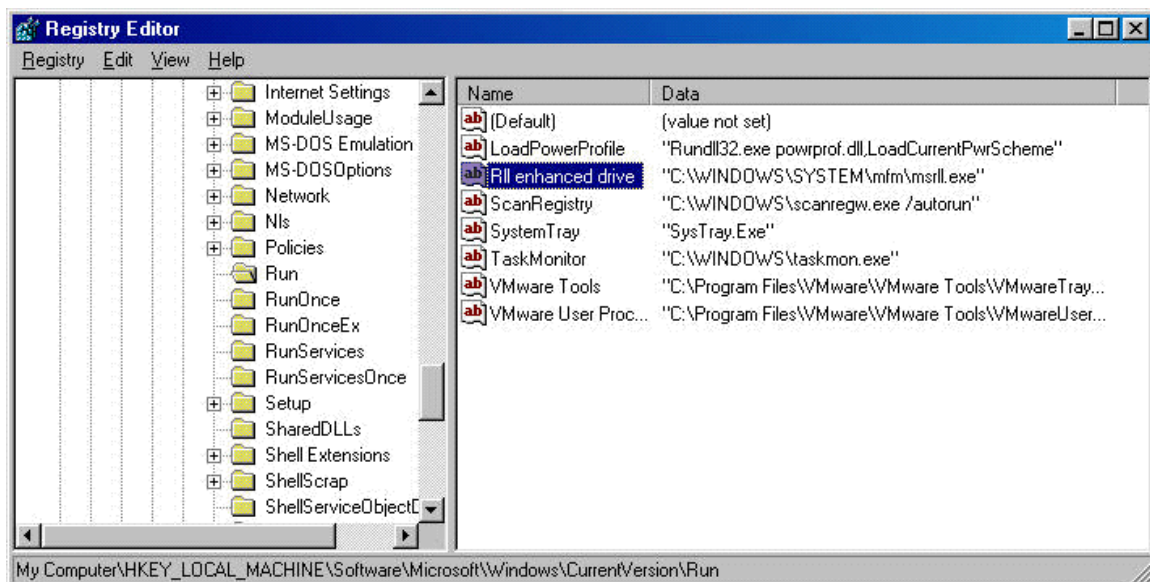
Figure 18 - Newly created text file

## Operating Systems

This malware was tested and successfully ran in Windows 95, 98, 2000 & XP. Different behaviors were displayed in different operating systems and are noted in the paragraphs below.

In Windows 2000 and Windows XP, the malware installs itself as a service as described above. In Windows 95 & Windows 98, the malware installs itself as a process which is invoked upon system startup by the "RUN" registry key. The malware is installed into the c:\windows\system\mfms directory.





**Figure 19 - Malware installed on Windows 98 machine in registry**

## Network Communications

A review of the captured network traffic revealed the malware specimen attempting to communicate via the network. Several DNS requests were attempted to a domain name of "collective7.zxy0.com". This domain name currently resolves to an IP address of: 193.136.99.11.

```
09/16-09:24:48.405816 192.168.65.100:1026 -> 4.2.2.2:53
UDP TTL:128 TOS:0x0 ID:596 IpLen:20 DgmLen:66
Len: 38
00 07 01 00 00 01 00 00 00 00 00 00 0B 63 6F 6C .....col
6C 65 63 74 69 76 65 37 04 7A 78 79 30 03 63 6F lective7.zxy0.co
6D 00 00 01 00 01 m.....
```

**Figure 20 - Capture of DNS request**

I then edited the "hosts" file on the infected Windows machine to include the hostname of collective7.zxy0.com with an IP address of my virtual linux machine located at 192.168.65.200. Upon rebooting the infected machine so the new "hosts" file would take effect, I observed the malware attempt to connect to TCP port 6667, which is commonly used for IRC traffic. The malware attempted three consecutive connection attempts to destination port 6667, but since the linux machine that the domain name of "collective7.zxy0.com" was pointed to did not have port 667 open, ACK-RST's were sent back to the malware (victim machine), resetting the connection.



```

=====
10/22-06:09:06.000802 192.168.65.100:1041 -> 192.168.65.200:6667
TCP TTL:128 TOS:0x0 ID:108 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xBD20C697 Ack: 0x0 Win: 0x4000 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=====
10/22-06:09:06.000842 192.168.65.200:6667 -> 192.168.65.100:1041
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xBD20C698 Win: 0x0 TcpLen: 20
=====

```

Figure 21 - attempted communication to port 6667 & reset

© SANS Institute 2000 - 2005, Author retains full rights.

A few seconds later, the malware then tried to establish a connection to the destination port of 9999 with the same result.

```
=====
10/22-06:09:35.981490 192.168.65.100:1042 -> 192.168.65.200:9999
TCP TTL:128 TOS:0x0 ID:109 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xBD8E01C2 Ack: 0x0 Win: 0x4000 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
=====

10/22-06:09:35.981563 192.168.65.200:9999 -> 192.168.65.100:1042
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xBD8E01C3 Win: 0x0 TcpLen: 28
=====
```

Figure 22 - attempted communication to port 9999 & reset

The malware then attempted to connect to destination port 8080, again with the same result.

```
=====
10/22-06:08:03.273039 192.168.65.100:1040 -> 192.168.65.200:8080
TCP TTL:128 TOS:0x0 ID:103 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xBC47E617 Ack: 0x0 Win: 0x4000 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
=====

10/22-06:08:03.273083 192.168.65.200:8080 -> 192.168.65.100:1040
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0 Ack: 0xBC47E618 Win: 0x0 TcpLen: 28
=====
```

Figure 23 - attempted communication to port 8080 & reset

Since my Linux VMware machine was not yet configured to listen on port 6667, 9999 or 8080, there were just sequences of TCP SYN being sent to the Linux machine which responded with TCP ACK RST packets because the port was not open. To explore this communication a little further, I setup a Netcat listener on the Linux machine (192.168.65.200) to listen on port 6667 and redirect all input received into a text file. Using the command "nc -l -p 6667 > nc-out.txt", I was able to capture some data the malware attempted to send.

```
[root@localhost tmp]# nc -l -p 6667 > nc-out.txt
[root@localhost tmp]# cat nc-out.txt
USER 0xUXPdulgFYEl localhost 0 :gXfQnfzBirOLLBMBtCncwXmWsYU
NICK hTQleFSHeASt
```

Figure 24 - Netcat listener on port 6667

I did the same for port 9999 with the following results:

```
[root@localhost tmp]# nc -l -p 9999 > nc-out.txt
[root@localhost tmp]# cat nc-out.txt
USER tJdfT localhost 0 :EWyaPSkAAEYCUaRfPoyCpVOExPHUTrgYxgqLpnnMEK
NICK kIMkEYLnvfCh
```

Figure 25 - Netcat listener on port 9999

And for port 8080 the following results:

```
[root@localhost tmp]# nc -l -p 8080 > nc-out.txt
[root@localhost tmp]# cat nc-out.txt
USER AkcdL localhost 0 :DWvufuOLDKhkrUEyeWiGPgaUudfEGxIYIGxawqTl
NICK PUUkpSLgZ
```

Figure 26 - Netcat listener on port 8080

Using the NetCat listener, I was able to capture what appears to be a username, machine name and a nickname, although the values appear to be obfuscated using some type of encryption or character replacement. The format of all three data communication attempts is consistent with a login to an IRC server.

I then installed and started an IRC server on the virtual Linux machine in hopes of discovering if in fact the malware would attempt to connect and join a particular IRC chat room. Upon installing and starting the IRC server on port 6667 of the virtual Linux machine, the malware quickly connected and joined the IRC server and went into a chat room named “#mils” and issued two commands: MODE #mils & WHO #mils.

The malware had what appeared to be a random or obfuscated nickname similar to the ones I had previously captured using the netcat listener.

Malware



Figure 27 - IRC channel "#MILS" that malware joined

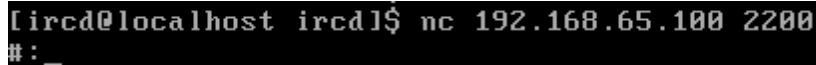
I attempted to interact with the malware using several different commands, some of which I found during the examination of the embedded text inside the executable (listed below in Code Analysis), but could not get the malware to respond to any of my commands. I attempted to establish a direct connect chat (DCC) with the malware, but that also went unacknowledged.

An analysis of the network traffic when the malware connects to the IRC server revealed that TCP port 113 on the victim machine is being used for AUTH/IDENT purposes. The following data was captured upon the initial connection of the malware to the IRCD server:

```
1099 , 6667 <- IRCD Server sent this data
1099 , 6667 : USERID : UNIX : RL <- Malware sent this data
```

Once the malware connected to the IRCD server, TCP port 113 closed and remained closed until the malware disconnected from the IRCD. Once I interrupted the connection to the IRCD, port 113 opened back up and remained open until the malware was able to connect to the IRCD again.

I then began examining the other port on the infected machine which the malware opened up, TCP port 2200 . Using the virtual Linux machine I used netcat to connect to the infected XP machine on port 2200 and was able to successfully connect. Upon connection, a limited prompt was displayed.



```
[ircd@localhost ircd]$ nc 192.168.65.100 2200
# : _
```

**Figure 28 - Connection to port 2200 of infected machine**

I entered several commands at the limited prompt but received no feedback or response. This appears to be a BNC bounce or also known as an IRC proxy which would allow a person to use this infected host as a proxy to connect to an IRC server. Most BNC proxies are protected by passwords to protect them from unauthorized connections.

## Code Analysis

As mentioned in the section above during the preliminary examination of the malware specimen, I was able to identify that the executable had been compressed using AsPack. An analysis of the embedded strings, using BinText, revealed numerous text fragments which appeared to be IRC commands.

?insmod	?dump	?mkdir
?rmmod	?md5p	?exec
?lsmod	?free	?kill
?ping	?update	?killall
?smurf	?hostname	?crash
?jolt	?!fif	?sklist
?clone	?play	?unset
?clones	?copy	?uattr
?login	?move	?dccsk
?uptime	?sums	?killsk
?reboot	?rmdir	?echo
?status	?wget	?hush
?jump	?join	?part
?nick	?akick	

During the code analysis, I attempted to research and answer three specific questions which arose from the behavior analysis:

1. What was contained in the text file named "jtram.conf"?
2. What purpose was TCP port 2200?
3. Could the malware be communicated with via the IRCD server

### JTRAM.CONF

Using OllyDbg, I attached to the already running msrll.exe service and had it disassemble the running code. I did a text search for the "jtram.conf" text and located several location where this text appeared. I was able to locate the portion of the code which read the configuration file into memory upon starting. Using breakpoints at this area, I was able to read the obfuscated text as it was read from the "jtram.conf" text file and then unencrypted into memory.

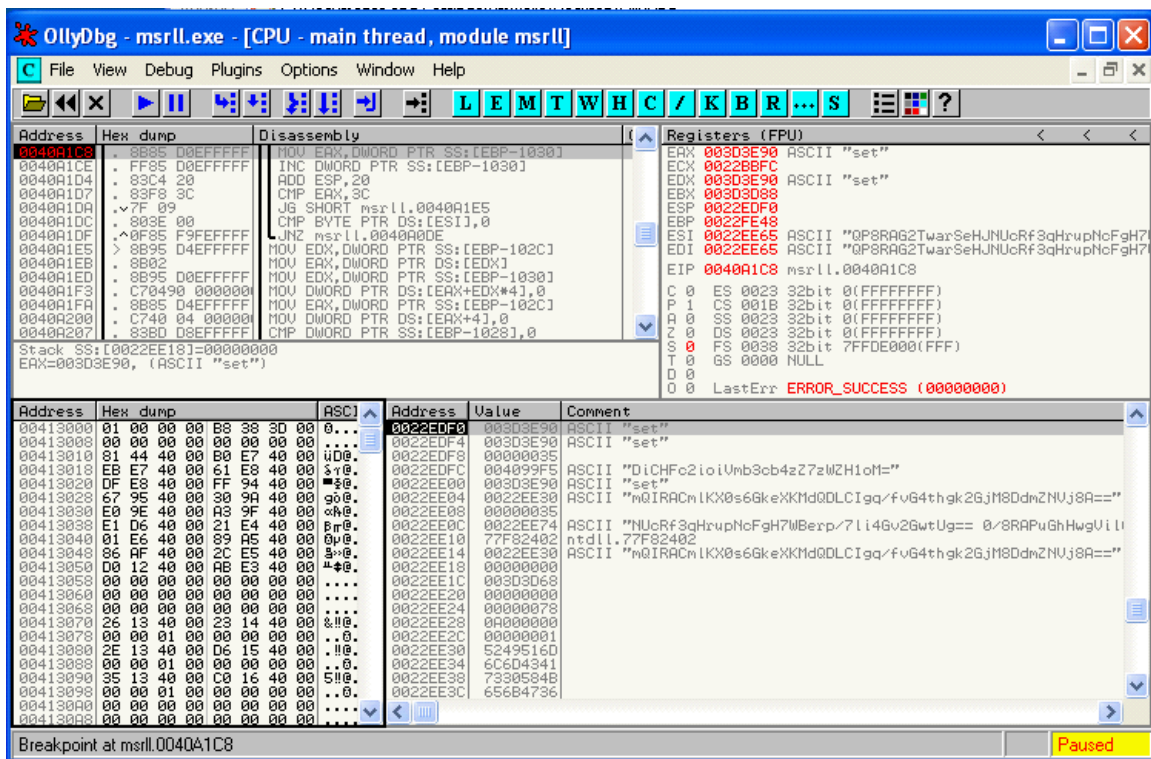


Figure 29 - OllyDbg windows showing unencrypted configuration string

In the above screen capture, you can see the obfuscated strings being read into the Stack window and the unencrypted text also being displayed.

In the next screenshot, you can see the first line of text in the “jtrm.conf” file matches the first portion of the text being read into the read/unencrypt function of the malware.

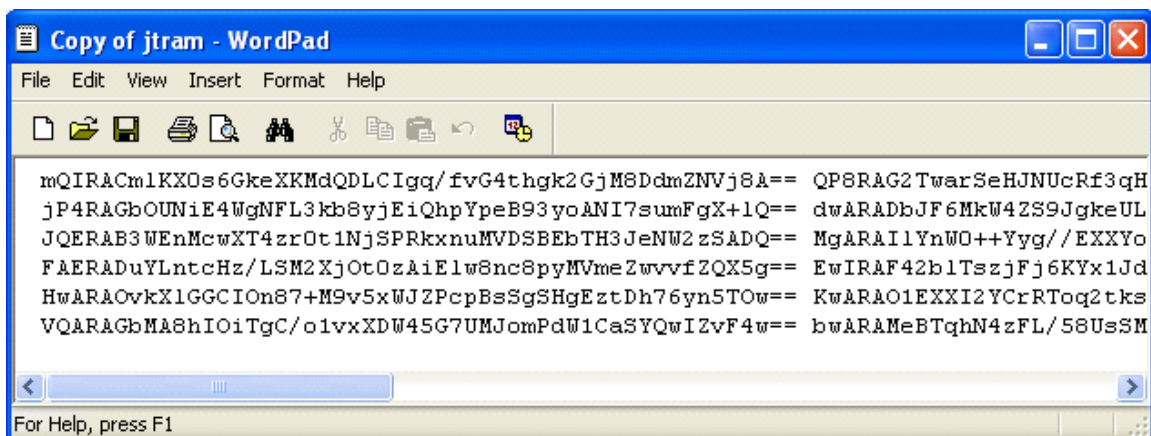


Figure 30 - First line "jtrm.conf" file corresponds to text being unencrypted

After observing and documenting this function as it reads through the “jtram.conf” file, the following text was found:

```
set    bot.port      2200
set    irc.quit
set    servers       collective7.zxy0.com,collective7.zxy0.com:9999!,
                        collective7.zxy0.com:8080"
set    irc.chan       #mils
set    pass           $1$KZLPLKdF$W8kl8Jr1X8DOHZsmlp9qq0
set    dcc.pass       $1$KZLPLKdF$55isA1ITvamR7bjAdBziX.
```

The file “jtram.conf” contains 6 lines of text, with three obfuscated text fragments per line. This corresponds to the six configuration settings observed above as well as the three parameters contained on each line; i.e. set, variable, value.

The last two parameters both appear to be MD5 hashes of passwords. The password format matches those used by Unix/Linux, where the first portion is the magic number, “\$1\$”, the second portion is the salt, “KZLPLKdF”, and the third is the actual hash value of the password that was generated using the salt: “55isA1ITvamR7bjAdBziX.”

## Port 2200

In attempt to discover the features and purpose of TCP port 2200, I searched for all the “strcmp” commands which could likely be used to compare two strings together in order to recognize a command or password. At offset 40D5AB, I located two parameters that were being compared and appeared to control the initial login function on port 2200.

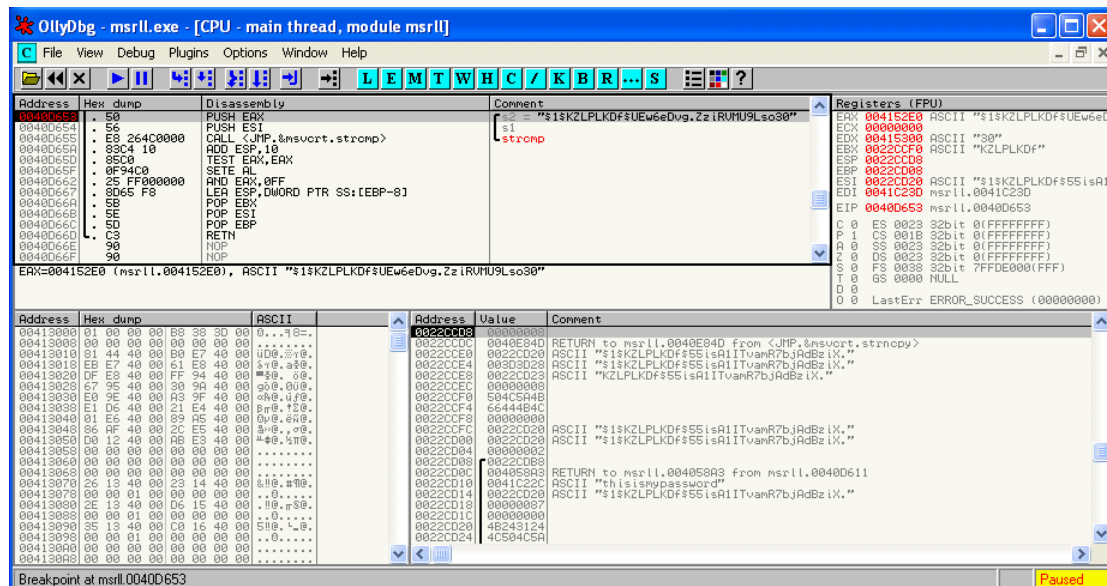
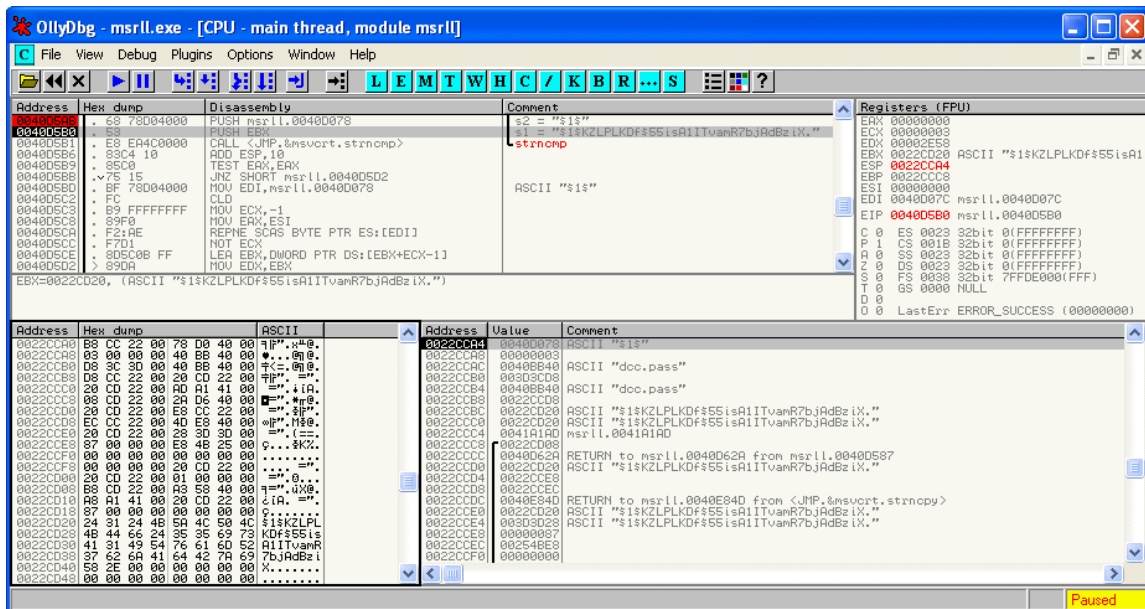


Figure 31 - strcmp function comparing parameters sent to port 2200

Through trial and error, I was able to determine the malware was expecting two parameters to be sent upon connecting to TCP port 2200. The first was used as a username and the second is a password. The password that is expected is the password that is stored in the "jtram.conf" file under the "set dcc.pass" parameter.

By setting a breakpoint at offset 0040D5B0, I was able to pause program execution and substitute a know password hash into the memory area, thus having the malware authenticate against my password and recognize me as a valid user.



Once authenticated, any text I typed I would get an echo reply similar to an IRC channel:





0040BBDE	52	PUSH EDX	Arg1
0040BBDF	E8 8E9CFFFF	CALL msrll.00405872	msrll.00405872
0040BBE4	83C4 10	ADD ESP,10	
0040BBE7	85C0	TEST EAX,EAX	
0040BBE9	75 6F	JNZ SHORT msrll.0040BC5A	
0040BBEB	83EC 0C	SUB ESP,0C	
0040BBEE	68 3C030000	PUSH 33C	size = 33C (828.)
0040BBF3	E8 58650000	CALL <JMP.<msvcrt.malloc>	malloc
0040BBF8	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	

Once I successfully bypassed this authentication step, I located the code statement at offset 0040BBE9 which was responsible for testing whether a correct password had been entered. I used a hex-editor to edit this offset and change the Op code from a 74 (JE) to a 75 (JNZ) which allowed me to bypass the password authentication each subsequent time I ran and connected to port 2200.

Once authenticated, I tried several different commands in this mode, but I was unable to get the malware to display any different behavior other than echoing the text back to me.

## IRCD CHANNEL #MILS

Once the malware connected to the IRCD server and joined the #mils channel, it remained dormant. I attempted to interact and provoke some type of response, but no additional behavior was displayed by the malware. I also attempted to locate the code section that received the text typed in the IRC channel to try and learn some commands or login procedure, but none could be located.

Several references to various encryption algorithms were found embedded in the executable. Along with the references to MD5, Blowfish, RSA, RC4, was a reference to the Microsoft Crypto library.

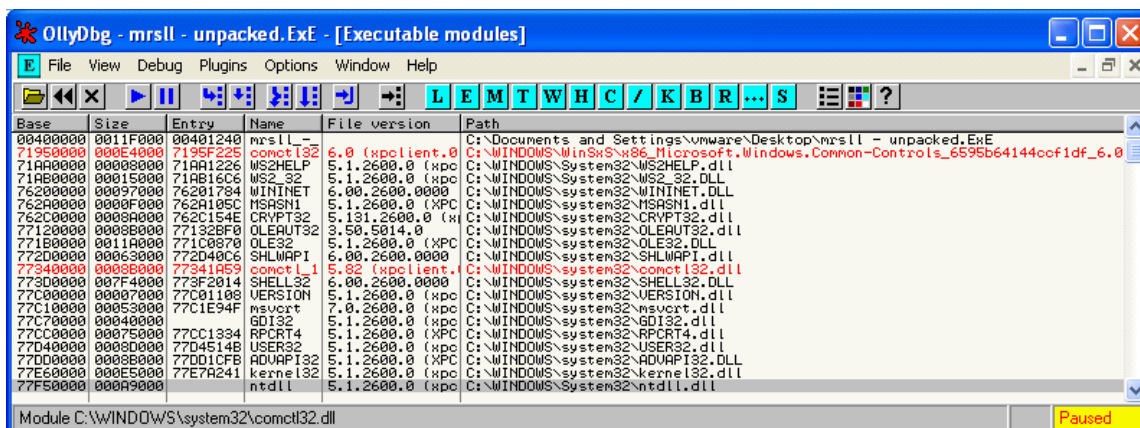


Figure 32 - DLL Modules linked to malware

The above image shows the listed Executable Modules (DLLs) that the malware depends on to execute. One of the listed DLLs is CRYPT32.DLL, which is the Windows Crypto API. The text file named "jtram.conf" contained data which appeared to be encrypted and obfuscated which is consistent with the

encryption references found in the malware.

## Conclusion

This malware specimen acts as an IRC bot which infects a victim computer and installs in the C:\%WINDIR%\SYSTEM32\MFM folder. Once installed the malware installs itself as a service which will cause the malware to execute each time the system is started. If the malware is run by a user with administrative privileges, the Malware runs in the local SYSTEM user space which gives it full access to everything in the operating system.

Once installed, the malware attempts one of three connections to the domain name of "collective7.zxy0.com". If the host is available, the malware will attempt to connect to an IRC server located at that address on one of three different ports; 6667, 8080 or 9999. Once connected the malware assumes a NICK which appears to be randomly generated or obfuscated in some manner. The malware then joins the IRC channel of #MILS with no private key (password). It is assumed that the malware remains in the IRC channel in a dormant state until the correct authentication technique is used. If the malware becomes disconnected, it will automatically reconnect when the network path becomes available again.

As a final step to try and identify this malware specimen, I used Symantec Anti-Virus to scan the packed version of this specimen and it was immediately detected as "Backdoor.IRC.Bot". Up-to-date virus signatures are required to detect ever changing malware specimens such as this one. Network Associates Inc. (NAI) Identifies this malware as "BackDoor-CGM Trojan", and a reference can be found here:

[http://vil.nai.com/vil/content/v\\_126653.htm](http://vil.nai.com/vil/content/v_126653.htm)

Additional precautions that can be taken in order to minimize the impact of malware such as this would involve installing personal firewall software which would limit the incoming network traffic. In addition, if the computer did become infected, the personal firewall software would limit the ability of the malware to communicate outbound and establish a connection to the IRCD server which acts as a control mechanism.

## Removal

To uninstall this malware the following steps could be taken:

### Windows 2000 & Windows XP

1. Click on Start->run and enter "services.msc", then hit enter
2. Select the service named "Rll enhanced drive" and double click
3. Change the start-up option to "Manual"
4. Reboot the computer and upon rebooting navigate to c:\%WINDIR%\system32\mfmm\
5. Delete the files in this directory

### Windows 95 & 98

1. Click on start -> run and enter regedit
2. Navigate to:  
HKEY\_LOCALMACHINE\Software\Microsoft\Windows\CurrentVersion\Run
3. Locate the key named "Rll enhanced drive", and delete this key
4. Reboot the computer and navigate to c:\windows\system\mfmm
5. Delete the files in this directory

It is apparent that this malware has many more capabilities and authentication techniques which prevent persons other than the author from accessing its full capabilities.

© SANS Institute

## References

- [1] - SANS Institute, Survival Time History, September 2004,  
<http://isc.sans.org/survivalhistory.php>
- [2] – Webopedia, Online Dictionary, September 2004,  
<http://www.webopedia.com>
- [3] – WinHex Editor, <http://www.winhex.com>
- [4] – Md5sum, National Software Reference Library (NSRL),  
<http://www.nsrl.nist.gov/ftp/code/hash>
- [5]- Snort, Roesch, Martin, <http://www.snort.org>
- [6] – Netstat, Microsoft Corporation, <http://www.microsoft.com>
- [7] - PEInfo-, SK, [http://www.geocities.com/s\\_k\\_s\\_k\\_s\\_kru/util.html#peinfo](http://www.geocities.com/s_k_s_k_s_kru/util.html#peinfo)
- [8] – OllyDbg, Oleh Yuschuk, <http://home.t-online.de/home/Ollydbg>
- [9] – Bintext & Fport, Foundstone Inc., <http://www.foundstone.com>
- [10] – TCPView, Filemon, Regmon & TDIMon, SysInternals,  
<http://www.sysinternals.com>
- [11] – Regshot, TiANWEi , <http://regshot.yeah.net/>
- [12] – Netcat, GNU Netcat, <http://netcat.sourceforge.net/>
- [13] –VMware Workstation, VMware, <http://www.vmware.com>
- [14] – AsPack, AsPack Software, <http://www.aspack.com/>
- [15] – AsPackDie, Yoda, <http://mitglied.lycos.de/yoda2k/>
- [16] – GT2, PHaX, [http://philip.helger.com/gt/p\\_gt2.htm](http://philip.helger.com/gt/p_gt2.htm)
- [17] – Windows Service Parameters, Windows IT PRO, September, 2004  
[http://www.win2000mag.com/Files/8723/table\\_01.html](http://www.win2000mag.com/Files/8723/table_01.html)