Reverse Engineering of msrll.exe

GIAC Reverse Engineering Malware (GREM)
Practical Assignment
Version 1.0

Submitted by: Erlend Garberg
02 December 2004

Abstract:
Behavioral analysis and code analysis are used to learn about the capabilities of the malware specimen msrll.exe. The specimen is using AsPack compression and MD5 passwords to make analysis harder.

Table of Contents

List of Figures

Introduction

This is the Practical Assignment for GIAC Reverse Engineering Malware. I thank Lenny Zeltser for an informative course.

Laboratory Setup

This section describes the laboratory setup used in this assignment.

Hardware

My host computer for the laboratory setup is an Intel Pentium IV 3GHZ with 1GB RAM running Windows XP SP2. Two virtual machines were set up with VmWare, one Red Hat Linux 9.0 and one Windows XP SP2.

Networking

For the networking part of the laboratory setup, I follow the recommendations from the course material and use VmWare host-only networking. This provides isolation; communication is only possible between virtual machines (VM) and host, virtual machines cannot reach machines not on the laboratory network. The network infrastructure is illustrated in Figure 1.

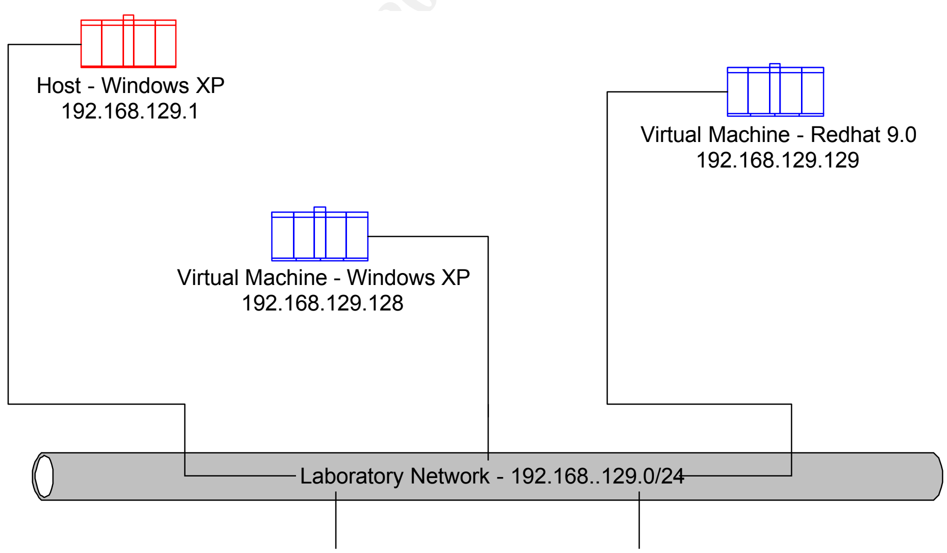

Figure 1 - Network infrastructure

The DHCP server in VmWare provides IP-addresses for the virtual machines.

Software resources

The following software is used in the analysis:

Name	Description	How/Where used
WinZip	Does File Extraction.	Extracting of malware.
NetCat	Network Swiss Army Knife.	Connecting to backdoor of malware and faking services for the malware to connect to.
VmWare	Emulator for Intel hardware. Makes it possible to run many virtual computers simultaneously on one workstation.	Running multiple machines in the lab and for enforcing system isolation.
MD5sum	Checksum application.	Creating checksum of malware specimen.
FileMon	Logs access to files.	Finding files accessed by the malware specimen.
RegMon	Logs access to registry.	Finding registry keys accessed by the malware specimen.
TDIMon	Logs network connections.	Finding network connections opened by the malware specimen.
RegShot	Snapshots file system and registry.	Finding differences in file system and registry before/after running the malware specimen.
BinText	Finds strings embedded in a binary file.	Finding strings in the malware specimen.
IDA Pro	Interactive Disassembler	Disassembly and debugging of the malware specimen.
PEInfo	PE file info	Finding type of file, size, OS etc of the malware specimen.
AsPackDie	Extracts executables packed with AsPack	Uncompressing the malware specimen.
Snort	Packet sniffer	Packet sniffing
ircd	Internet Relay Chat Server	Analyzing network connections to port 6667 from the malware specimen.
Process Explorer	Shows Process Detail	Getting summary of process resources.
passwd	Sets Linux MD5 password	Making MD5 password

Properties of the Malware Specimen

Using the shared folders of Vmware (Read-Only), I transfer the malware specimen (msrll.zip) to the Windows VM. On the VM, the specimen is unpacked to C:\malware\msrll.exe.

Type of file and size

To find the file type of msrll.exe, I open it in PEInfo and IDA Pro. As shown in Figure 2 the file is an executable file of size 41984 bytes. As shown in Figure 3, the message from IDA Pro at startup indicates that the executable is packed/compressed. This means that it will be harder to analyze, because it needs to be unpacked before Code Analysis can take place.

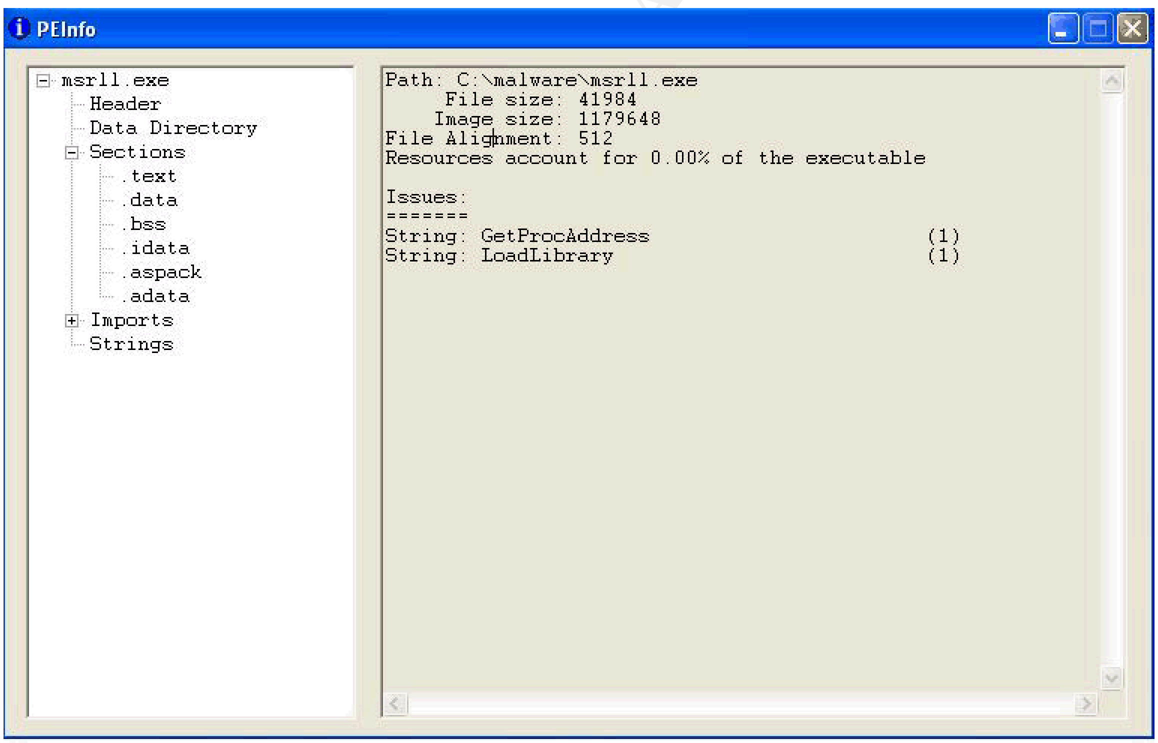

Figure 2 – PEInfo

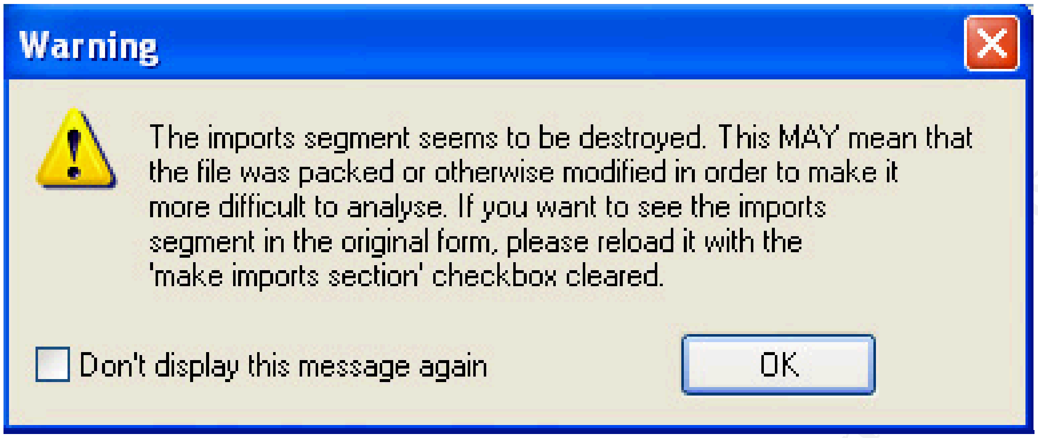

Figure 3 - IDA Pro

MD5 hash

To make an md5 hash I use the application md5sum. As shown in Figure 4 the file has the checksum 84acfe96a98590813413122c12c11aaa.

Figure 4 - md5sum

Operating systems

As shown in Figure 5, the OperatingSystemVersion field in the PE-header of the executable is set to 4.00, which corresponds to Windows NT 4.0. That means that the executable will run on Windows versions newer than or equal to Windows NT 4.0. The file is a Win32 executable.

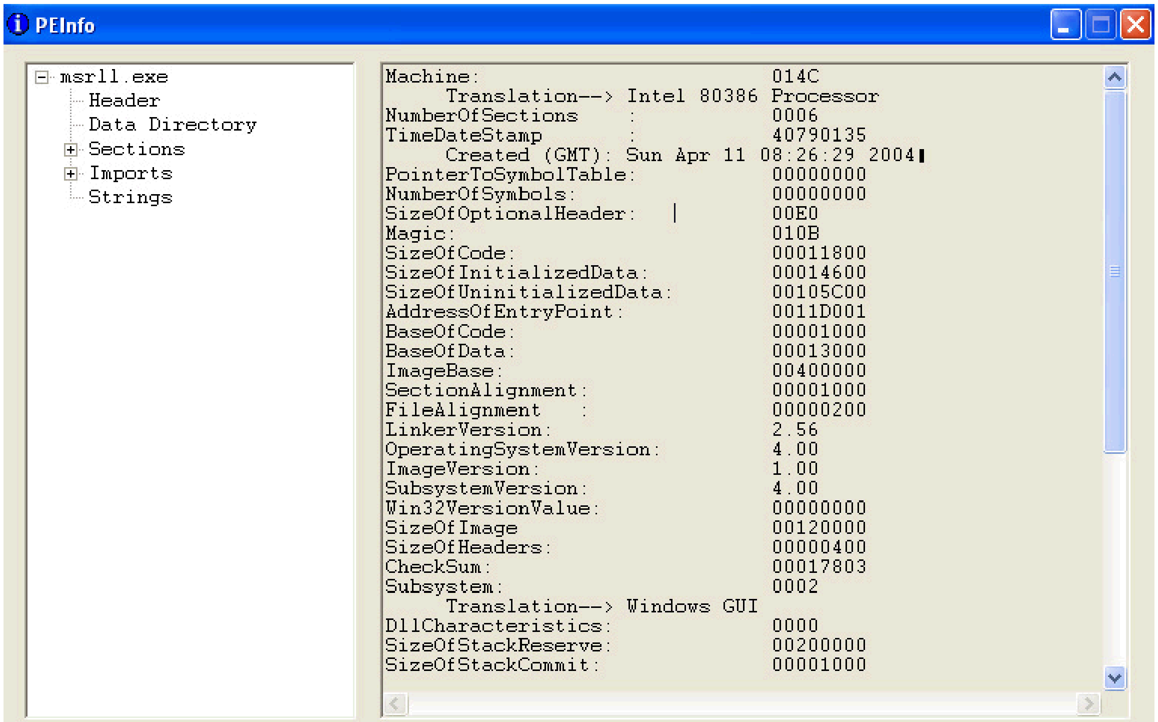

Figure 5 - Operating System version from PEInfo

Embedded strings

I use BinText to extract strings embedded into the malware specimen. This is shown in Figure 6. The strings give no info about the executable since it is compressed. An exception is the PE section names, but those can also be found with PEInfo.

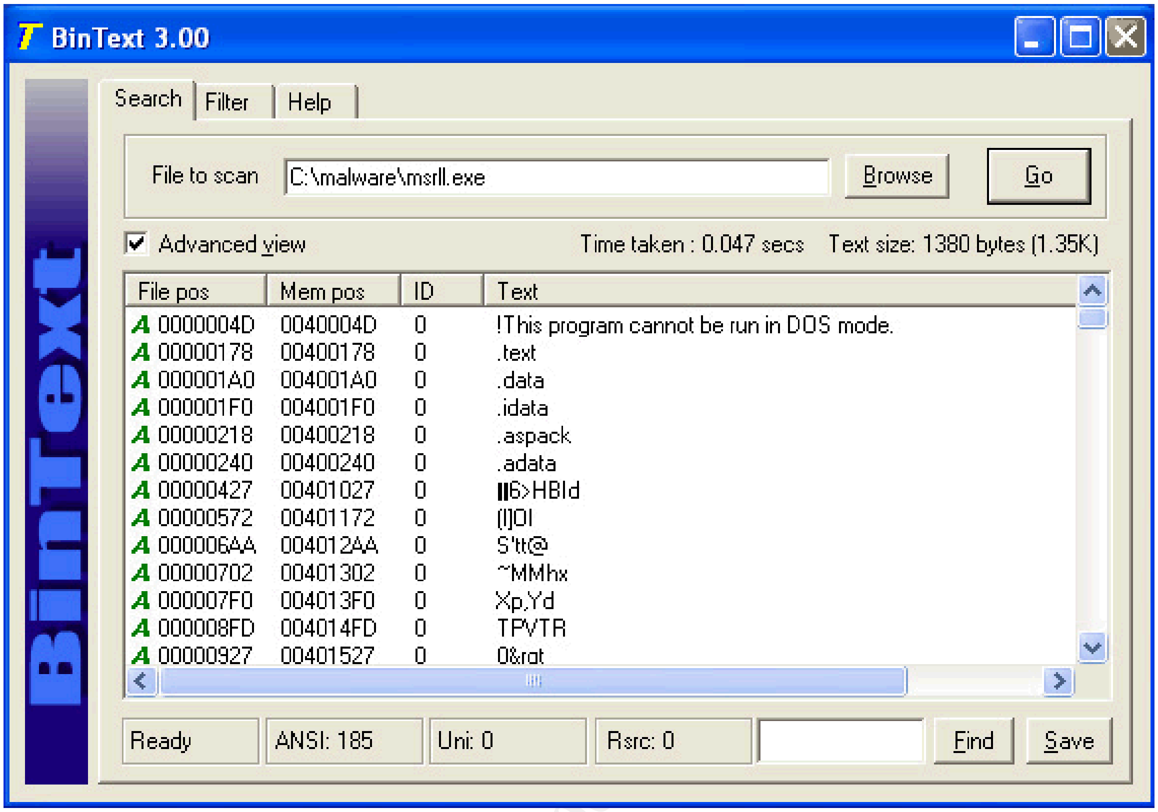

Figure 6 - BinText

Behavioral Analysis

I begin the behavioral analysis with starting monitoring tools:

- I start RegMon, FileMon and TDIMon
- I take a snapshot of the system with RegShot

I then launch msrll.exe and let it run for about 30 seconds. Afterwards I kill it with the task manager. Finally I pause the monitoring tools.

Findings

I notice the following events after disregarding changes to files and registry keys that are not related to the malware specimen:

The following files are added:
C:\WINDOWS\system32\mfm\jtram.conf
C:\WINDOWS\system32\mfm\msrll.exe

The following files are deleted:
C:\malware\msrll.exe

The following registry keys are added:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm\Security

The following registry values are added:
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\Security\Security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 02 00 1C 00 01 00 00 00 02 80 14 00 FF 01 0F 00 01 01 00 00 00 00 00 01 00 00 00 00 02 00 60 00 04 00 00 00 00 00 14 00 FD 01 02 00 01 01 00 00 00 00 00 05 12 00 00 00 00 00 18 00 FF 01 0F 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 00 00 14 00 8D 01 02 00 01 01 00 00 00 00 00 05 0B 00 00 00 00 00 18 00 FD 01 02 00 01 02 00 00 00 00 00 05 20 00 00 00 23 02 00 00 01 01 00 00 00 00 00 05 12 00 00 00 01 01 00 00 00 00 00 05 12 00 00 00
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\Type: 0x00000120
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\Start: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ErrorControl: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ImagePath: "C:\WINDOWS\system32\mfm\msrll.exe"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\DisplayName: "Rll enhanced drive"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\mfm\ObjectName: "LocalSystem"

In other words, msrll.exe copies itself to C:\windows\system32\mfm\, deletes itself from the former location (C:\malware), and creates a new Windows Service for the executable in C:\windows\system32\mfm. As shown in Figure 7, the new service is set to start automatically at boot, but is not started yet.

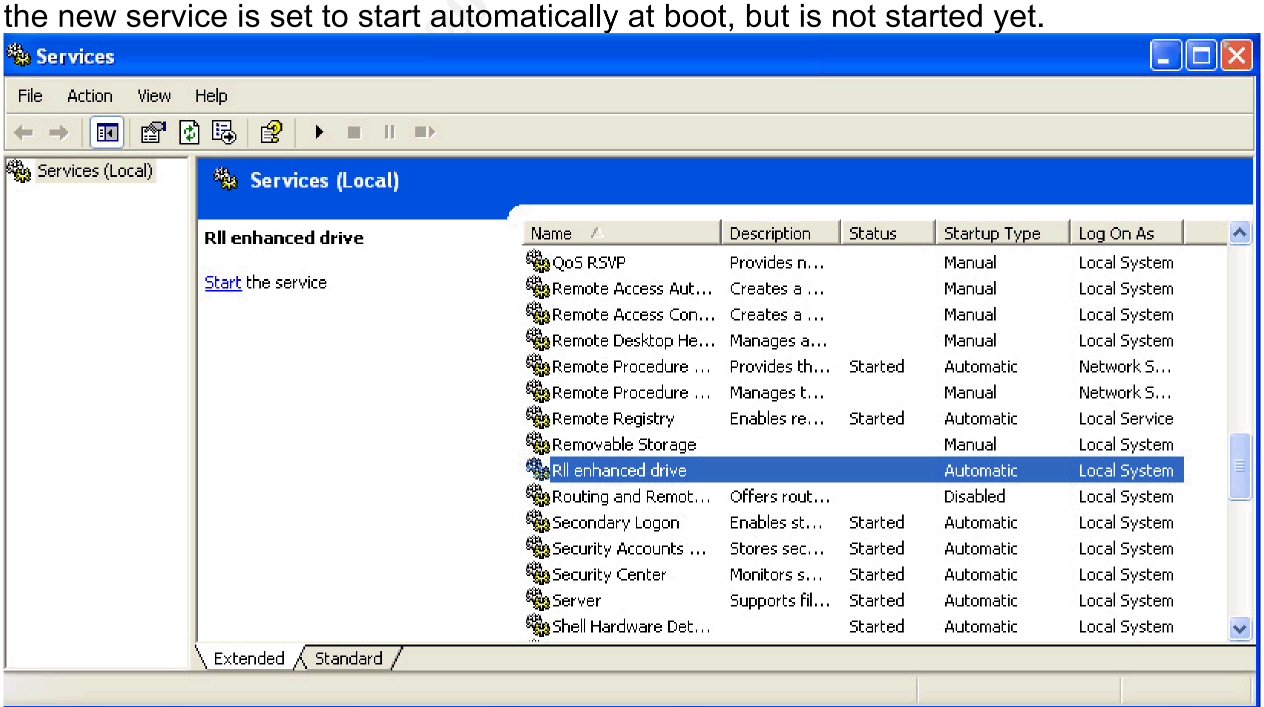

Figure 7 - Service added

The checksum for the copied file is:
84acfe96a98590813413122c12c11aaa *msrll.exe
This is the same as the original C:\malware\msrll.exe had. This shows that the copy is identical to the original file.

From the filename, C:\WINDOWS\system32\mfm\jtram.conf seems to be a configuration file for the malware specimen. The file seems to be encrypted, so no information can be gained from it. (See Figure 8)

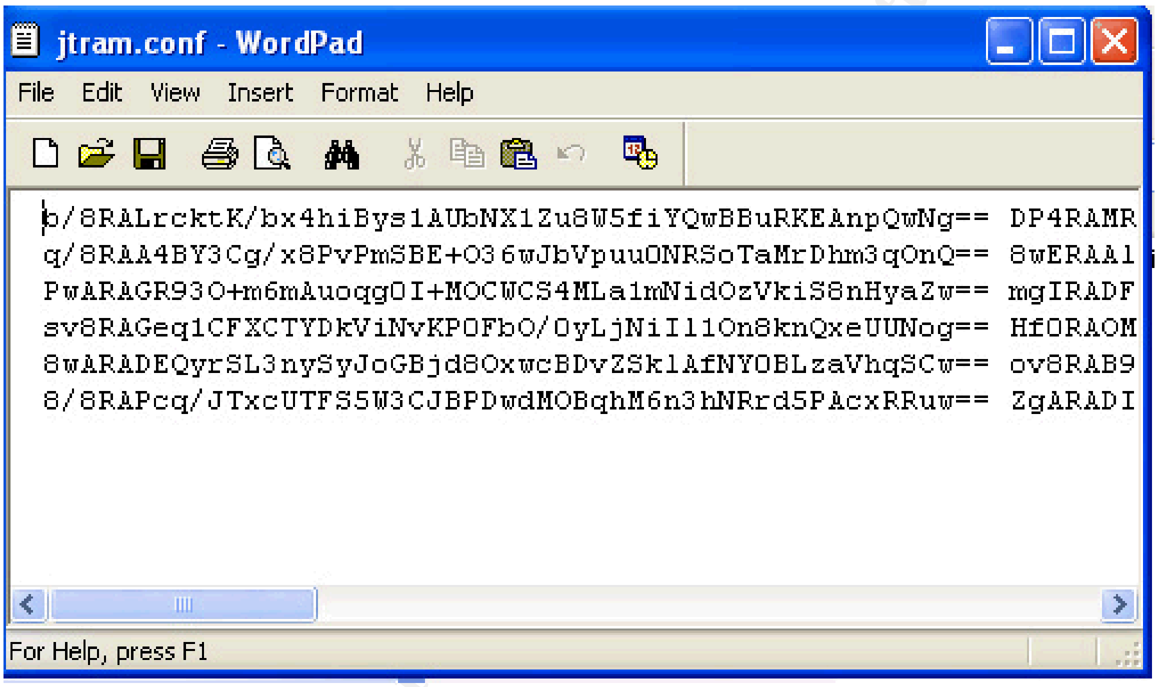

Figure 8 - jtram.conf

The following interesting information shows up in TDIMon:
15.81876495 msrll.exe:1032 819C8480 IRP_MJ_CREATE TCP:0.0.0.0:2200
SUCCESS Address Open
22.08134972 svchost.exe:1036 819C9A38 TDI_SEND_DATAGRAM
UDP:0.0.0.0:1025 192.168.129.1:53 SUCCESS Length:38
22.09424913 msrll.exe:1032 81AACEA0 IRP_MJ_CREATE TCP:0.0.0.0:113
SUCCESS Address Open

Msrll.exe listens on TCP-port 2200 and 113. It also connects to 192.168.129.1 on UDP-port 53.

In this stage of the analysis I assume that port 2200 is a backdoor and that port 113 is used for an ident daemon. The use of an ident daemon indicates that msrll.exe wants to connect to IRC; because many IRC servers require that the clients run identd to be allowed to connect.

The connection to port 192.168.129.1 on UDP-port 53 is probably an attempt to resolve a domain name, since port 53 belongs to DNS and 192.168.129.1 is set as DNS server on the Windows VM. To find which domain name that is attempted resolved, I launch snort on the Linux VM with the following command line and relaunch msrll.exe.

snort –vd –l /root/log

With snort I discover that msrll.exe attempts to resolve collective7.zxy0.com. (See Figure 9)

```
[root@localhost /]# cat /root/log/192.168.129.128/UDP\:1025-53
10/19-10:15:04.434895 192.168.129.128:1025 -> 192.168.129.1:53
UDP TTL:128 TOS:0x0 ID:212 IpLen:20 DgmLen:66
Len: 38
B8 F3 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 9 - snort dns

I telnet to port 2200 and 113 on the Windows VM to gain more information about the services running there.

```
[root@localhost /]# telnet 192.168.129.128 113
Trying 192.168.129.128...
Connected to 192.168.129.128.
Escape character is '^]'.
adf
adf : USERID : UNIX : YdGbQoJPc
Connection closed by foreign host.
[root@localhost /]# _
```

Figure 10 - identd is running on port 113

```
[root@localhost /]# telnet 192.168.129.128 2200
Trying 192.168.129.128...
Connected to 192.168.129.128.
Escape character is '^]'.
#:auth
a
Connection closed by foreign host.
```

Figure 11 - backdoor?

Figure 10 confirms that identd is running.
Figure 11 shows that some kind of backdoor is running on port 2200, but doesn't give any more information.

Molding the laboratory environment

DNS

To advance the analysis process it's now needed to change the laboratory environment. I begin with redirecting traffic for collective7.zxy0.com to the Linux VM. Entering 192.168.129.129 as the address for collective7.zxy0.com in C:\Windows\system32\drivers\etc\hosts does this.

```
7.21099330   msrll.exe:1632 818F6678   TDI_CONNECT   TCP:0.0.0.0:1091
        192.168.129.128:6667      CONNECTION_REFUSED-150           .
44.58069392  msrll.exe:372  818D00B0   TDI_CONNECT   TCP:0.0.0.0:1102
        192.168.129.129:9999      CONNECTION_REFUSED
74.79998100  msrll.exe:372  818DB1C0   TDI_CONNECT   TCP:0.0.0.0:1103
        192.168.129.129:8080      CANCELLED
```

The redirection of network traffic to the Linux VM shows that msrll.exe tries to connect to port 6667, 9999 and 8080 on collective7.zxy0.com. Port 6667 indicates an IRC connection.

IRC Port 6667

To continue the analysis, I launch an IRC server on the Linux VM.

```
[root@localhost /]# su - ircd
[ircd@localhost ircd]$ ./ircd
[ircd@localhost ircd]$ ps -u ircd
  PID TTY          TIME CMD
 2270 tty1     00:00:00 bash
 2311 ?        00:00:00 ircd
```

Figure 12 - Starting ircd

I then restart msrll.exe. Process Explorer shows that msrll.exe has established a connection with port 6667 on the linux VM.

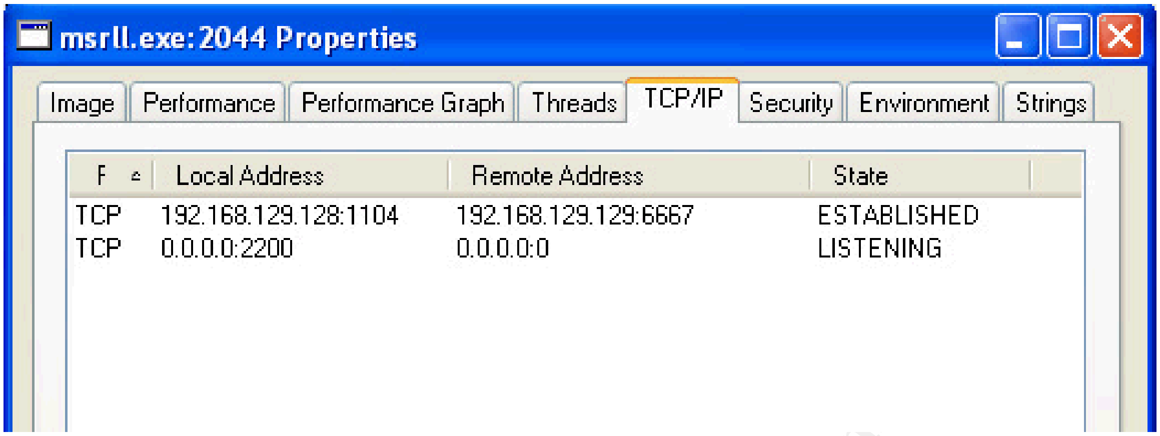

Figure 13 - Process Explorer

I launch an IRC client on the Linux VM and lists all created channels with the /list command. A channel #mils has been created. I join this channel and list all clients there with the command /who #mils. This is shown in Figure 14.

```
*** Your host is localhost.localdomain[localhost.localdomain/6667], running
+version 2.8/hybrid-6.3.1
*** This server was created Tue Jun 4 2002 at 16: 59:45 EDT
*** umodes available oOiwszcrkfydnxb, channel modes available biklmnopstve
*** WALLCHOPS PREFIX=(ov)@+ CHANTYPES=#& MAXCHANNELS=20 MAXBANS=25 NICKLEN=9
+TOPICLEN=120 KICKLEN=90 NETWORK=EFnet CHANMODES=b,k,l,imnpst MODES=4 are
+supported by this server
*** There are 0 users and 2 invisible on 1 servers
*** 1 channels have been formed
*** This server has 2 clients and 0 servers connected
*** Current local  users:  2  Max: 2
*** Current global users:  2  Max: 2
*** Highest connection count:  2 (2 clients) (5 since server was (re)started)
*** - localhost.localdomain Message of the Day -
*** - This is an IRC server. Authorized users only.
*** Mode change "+i" for user root by root
*** Channel     Users   Topic
*** #mils       1
*** root (~root@127.0.0.1) has joined channel #mils
*** #mils 1098199350
*** No argument specified
#mils       root      H   ~root@127.0.0.1 (root)
#mils       tIrKlMLgH H   ksjDsPyBxK@192.168.129.128 (uYhNZs)
[1] 01:22 root (+i) on #mils (+nt) * type /help for help
_
```

Figure 14 - irc

The malware specimen is joined as tlrKIMLgH on the channel. The nickname seems to be randomly generated, and repeated connections show that the nickname changes each time. I try to talk to the process to find commands, but to no avail.

Port 9999 and 8080

To find out what msrll.exe expects on port 9999 and 8080, I launch NetCat on the Linux VM with the command “nc –l –p 8080” and “nc –l –p 9999”. Then I restart msrll.exe. The ircd is stopped. As shown in Figure 15, msrll.exe expects an IRC server on port 9999 and 8080.

```
[root@localhost /]# nc -l -p 9999
^[      USER yCUWBjDPts localhost 0 :YDUiLDuZNwWQJJfJygnlKKVNujRKZgtmxSpE
NICK byDYFaJGRle
 punt!
[root@localhost /]#
[root@localhost /]# nc -l -p 8080
USER iQmPIJEc localhost 0 :XPipxGGO
NICK hnVbFWTxu
```

Figure 15 - Port 9999 and 8080

At this stage in the reverse engineering process behavioral analysis doesn’t seem to give any more information about the malware specimen. I therefore proceed with code analysis.

Code Analysis

Before I can disassemble and debug the binary, I need to unpack it.

Unpacking

Earlier in the analysis I established that the malware specimen was encrypted or compressed. Before code analysis can take place the malware specimen need to be unpacked/decrypted.

The aspack segment in the file indicates that the executable was packed with AsPack. Because of this, I try to extract the executable with the application AsPackDie, which was downloaded from
http://scifi.pages.at/yoda9k/files/AspackDie141.zip.
AsPackDie was able to extract the executable successfully, as shown in Figure 16.

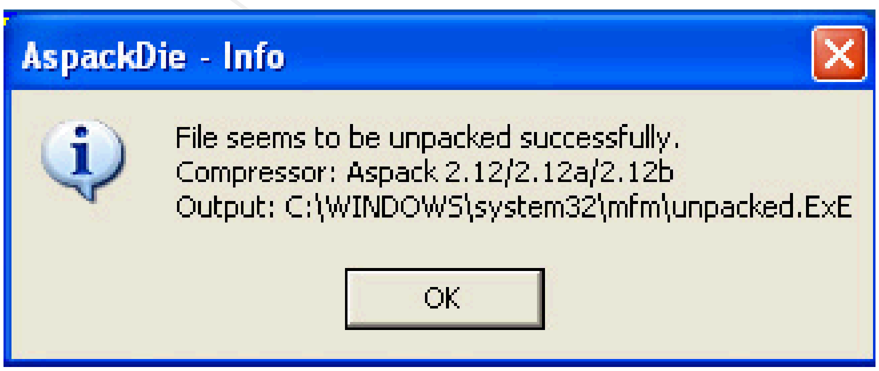

Figure 16 - Extracting msrll.exe with AspackDie

Running the new unpacked executable shows that the unpacking worked; the malware specimen is acting exactly like before.
I can then proceed with disassembly and debugging.

First I check if there are any interesting strings in the executable with BinText now that it is unpacked. The following strings seems to be potential commands to control the malware:

0000934E	0040934E	0	?clone
00009355	00409355	0	?clones
0000935D	0040935D	0	?login
00009364	00409364	0	?uptime
0000936C	0040936C	0	?reboot
00009374	00409374	0	?status
0000937C	0040937C	0	?jump
00009382	00409382	0	?nick
00009388	00409388	0	?echo
0000938E	0040938E	0	?hush
00009394	00409394	0	?wget
0000939A	0040939A	0	?join
000093A9	004093A9	0	?akick
000093B0	004093B0	0	?part
000093B6	004093B6	0	?dump
000093C6	004093C6	0	?md5p
000093CC	004093CC	0	?free
000093D7	004093D7	0	?update
000093DF	004093DF	0	?hostname
000093EE	004093EE	0	?!fif
000093FE	004093FE	0	?play
00009404	00409404	0	?copy
0000940A	0040940A	0	?move
00009415	00409415	0	?sums
00009423	00409423	0	?rmdir
0000942A	0040942A	0	?mkdir
00009436	00409436	0	?exec
00009440	00409440	0	?kill
00009446	00409446	0	?killall
0000944F	0040944F	0	?crash
0000946E	0040946E	0	?sklist
00009476	00409476	0	?unset
0000947D	0040947D	0	?uattr
00009484	00409484	0	?dccsk
00009490	00409490	0	?killsk

I try to control the bot with the strings that BinText gave, but there is still no response.

Disassembly

I proceed with disassembly in IDA Pro.

Address 40BDE0 seems to contain a MD5 hashed password. (String begins with 1) This can be seen in Figure 17.

```
.text:0040BD4E ; char ServiceName[]
.text:0040BD4E ServiceName     db 'mfm',0            ; DATA XREF: sub_40BE48:loc_40BF1B↓o
.text:0040BD4E                                       ; sub_40BFEF+B↓o ...
.text:0040BD52 a2200           db '2200',0           ; DATA XREF: .data:00413B64↓o
.text:0040BD57 aJtr_id         db 'jtr.id',0         ; DATA XREF: sub_40BE48+89↓o
.text:0040BD57                                       ; .data:00413B68↓o
.text:0040BD5E aRun5           db 'run5',0           ; DATA XREF: .data:00413B6C↓o
.text:0040BD63 aIrc_quit       db 'irc.quit',0       ; DATA XREF: .text:0040C788↓o
.text:0040BD63                                       ; .data:00413B70↓o
.text:0040BD6C asc_40BD6C      db ' ',0              ; DATA XREF: .data:00413B74↓o
.text:0040BD6E aServers_0      db 'servers',0        ; DATA XREF: .data:00413B78↓o
.text:0040BD76                 align 10h
.text:0040BD80 aCollective7_zx db 'collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.'
.text:0040BD80                                       ; DATA XREF: .data:00413B7C↓o
.text:0040BD80                 db 'zxy0.com:8080',0
.text:0040BDCA aIrc_chan       db 'irc.chan',0       ; DATA XREF: .data:00413B80↓o
.text:0040BDD3 aMils           db '#mils',0          ; DATA XREF: .data:00413B84↓o
.text:0040BDD9 aPass_0         db 'pass',0           ; DATA XREF: .data:00413B88↓o
.text:0040BDDE                 align 10h
.text:0040BDE0 a1Kzlplkdfw8kl8 db '$1$KZLPLKDf$W8kl8Jr1X8DOHZsmIp9qq0',0
.text:0040BDE0                                       ; DATA XREF: .data:00413B8C↓o
```

Figure 17 - configuration

A different password is located at address 40BE20. Since the passwords are MD5, the passwords to be used while authenticating won't be found in the binary file. I then have several options, I can find the authentication routine and patch it to always return true, or I can generate my own MD5 password and replace the original ones. I choose to replace the passwords.

Patching to change MD5 passwords

I open msrll.exe in a hex editor and locate the addresses 40BDE0 and 40BE20. I then replace the original MD5 strings with the string "1Ec0wBmCq$1P9cBkJQWQqpsiQNeuqGT.", which I generated with 'passwd' on a linux machine. The corresponding password is "!Nanoics".

The assembly snippet in Figure 18 is probably part of the authentication procedure. From the "%s logged in" part, I deduct that the authentication process uses a username in addition to a password.

```
.text:00405B2D                push    ebp
.text:00405B2E                mov     ebp, esp
.text:00405B30                push    esi
.text:00405B31                push    ebx
.text:00405B32                mov     edx, [ebp+8]
.text:00405B35                mov     esi, [ebp+0Ch]
.text:00405B38                mov     ebx, [ebp+14h]
.text:00405B3B                mov     eax, [esi+205Ch]
.text:00405B41                test    eax, 2
.text:00405B46                jnz     short loc_405B9B
.text:00405B48                cmp     dword ptr [edx+4], 0
.text:00405B4C                jz      short loc_405B9B
.text:00405B4E                test    eax, 10h
.text:00405B53                jz      short loc_405B9B
.text:00405B55                sub     esp, 8
.text:00405B58                push    offset aPass    ; "PASS"
.text:00405B5D                push    dword ptr [edx+4]
.text:00405B60                call    sub_405872
.text:00405B65                add     esp, 10h
.text:00405B68                test    eax, eax
.text:00405B6A                jz      short loc_405B9B
.text:00405B6C                mov     eax, [ebx+0FCh]
.text:00405B72                test    eax, 10000h
.text:00405B77                jnz     short loc_405B9B
.text:00405B79                or      eax, 10000h
.text:00405B7E                mov     [ebx+0FCh], eax
.text:00405B84                sub     esp, 0Ch
.text:00405B87                push    ebx
.text:00405B88                push    offset aSLoggedIn ; "%s logged in"
.text:00405B8D                push    esi
.text:00405B8E                push    dword ptr [ebp+10h]
```

Figure 18 - Authentication routine?

After changing the password I proceed with trying to login to the backdoor. I use NetCat to connect to the Windows VM on port 2200. Then I try to authenticate with an arbitrary username and the password "!Nanoics". The login is successful, the malware responds to the command "?hostname" and "?exec". This is shown in Figure 19.

```
[root@localhost tmp]# nc 192.168.129.128 2200
#:Erlend
!Nanoics
?hostname
host: reverser.localdomain ip: 192.168.129.128
?exec C:\windows\system32\cmd.exe /c dir
C:\windows\system32\cmd.exe exited with code 0
```

Figure 19 – login

Finding Capabilites

To get an overview of the bots capabilities, I tested all the potential commands found earlier. To save space, I will not use screenshots in this part. The results are presented in the following table:

Command	**Action**
?clone	Make clones on ircserver
?clones	Control clones (say/join/part)
?uptime	show uptime of system and bot
?reboot	Reboot the computer
?status	show status information about the bot
?jump	Probably change to next ircserver
?nick	Change nickname on irc
?echo	print argument
?hush	unknown
?wget	get file from ftp/http
?join	join channel on irc
?akick	kick host from irc?
?part	part channel on irc
?dump	unknown
?md5p	compute md5 password
?free	unknown
?update	update Trojan from URL ?
?hostname	Print hostname
?play	play audio file on infected host?
?copy	Copy file
?move	Move file
?sums	Show checksums for msrll.exe and config file
?rmdir	Delete directory
?mkdir	Make directory
?exec	Execute program
?kill	Kill process
?killall	Kill all processes?
?crash	Crash computer?
?sklist	List active network sockets
?unset	unknown
?killsk	Kill socket?
?ping	Pingflood target
?smurf	Smurf-attack target
?jolt	Unknown attack on target

With that I conclude the code analysis.

Analysis Wrap-Up

In this section I will summarize the findings in my analysis.

Capabilities

The malware specimen is capable of installing itself to a system directory, adding itself as a legal-looking service and connecting to IRC to wait for instructions from an attacker. It looks like it is intended to be part of a botnet belonging to the attacker. Based on the built in commands for attack, an attacker can use such a botnet for distributed denial of service attacks targeting sites on the Internet. In addition, the malware specimen can be controlled via a backdoor on port 2000. The attacker can easily update the Trojan software with the built in "?update" command.

Potential Users

Potential users for this program could be script kiddies wanting to build a botnet for DDOS attacks.

Defence

To eliminate current infections of msrll.exe, it would be enough to kill the msrll.exe process, delete C:\windows\system32\mfm\msrll.exe and remove the NT service. To prevent future infections, it could be possible to build a signature from the malware specimen which can be added to antivirus scanners. It could also be possible to use a firewall that could filter away IRC traffic based on layer 7 (application data) instead of fixed service ports.

References

Zeltser, Lenny. Reverse-Engineering Malware. Volume 1-4. SANS Press, Jun 04, 2004.

Last Updated: May 25th, 2017

Upcoming Training

Community SANS Madrid FOR610 (in Spanish)	**Madrid, Spain**	**Jun 12, 2017 - Jun 17, 2017**	**Community SANS**
Community SANS Columbia FOR610	**Columbia, MD**	**Jun 19, 2017 - Jun 24, 2017**	**Community SANS**
DFIR Summit & Training 2017	**Austin, TX**	**Jun 22, 2017 - Jun 29, 2017**	**Live Event**
SANSFIRE 2017	**Washington, DC**	**Jul 22, 2017 - Jul 29, 2017**	**Live Event**
SANS New York City 2017	**New York City, NY**	**Aug 14, 2017 - Aug 19, 2017**	**Live Event**
SANS Virginia Beach 2017	**Virginia Beach, VA**	**Aug 21, 2017 - Sep 01, 2017**	**Live Event**
SANS Network Security 2017	**Las Vegas, NV**	**Sep 10, 2017 - Sep 17, 2017**	**Live Event**
Community SANS Ottawa FOR610	**Ottawa, ON**	**Sep 18, 2017 - Sep 23, 2017**	**Community SANS**
Baltimore September 2017 - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	**Baltimore, MD**	**Sep 25, 2017 - Sep 30, 2017**	**vLive**
SANS Copenhagen 2017	**Copenhagen, Denmark**	**Sep 25, 2017 - Sep 30, 2017**	**Live Event**
SANS Baltimore Fall 2017	**Baltimore, MD**	**Sep 25, 2017 - Sep 30, 2017**	**Live Event**
SANS DFIR Prague 2017	**Prague, Czech Republic**	**Oct 02, 2017 - Oct 08, 2017**	**Live Event**
Community SANS New York FOR610	**New York, NY**	**Oct 16, 2017 - Oct 21, 2017**	**Community SANS**
SANS Tokyo Autumn 2017	**Tokyo, Japan**	**Oct 16, 2017 - Oct 28, 2017**	**Live Event**
SANS Sydney 2017	**Sydney, Australia**	**Nov 13, 2017 - Nov 25, 2017**	**Live Event**
SANS Cyber Defense Initiative 2017	**Washington, DC**	**Dec 12, 2017 - Dec 19, 2017**	**Live Event**
SANS OnDemand	**Online**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s Only**	**Anytime**	**Self Paced**