GIAC
CERTIFICATIONS

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 permited without express written permission.

Interested in learning more?

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

Reverse Engineering Malicious Code

GIAC Reverse Engineering Malware

Practical Assignment

Version 1.0

Zekeria A. Sheikh
Las Vegas, October 2004

Submitted: December 31, 2004

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

Table of Contents

Abstract *3*

Laboratory Setup *3*

Hardware Resources 3

Networking Setup 4

Software Resources 5

Windows XP SP2 5
Windows 2000 5
Snort 5
Undernet-IRCU2 5
Ollydbg 5
Regmon 6
Filemon 6
IDAPro 6
TDIMon 6
LordPE 6
RegShot 6
MD5sum 6
PEInfo 7
BinText 7
ASpackdie 7

Properties of the Malware Specimen *7*

Type of File 7

Size of the File 7

MD5 Hash of the File 8

Operating System it runs on 8

Strings Embedded into it 8

Behavioral Analysis *14*

Monitoring file system access 14

Monitoring registry / configuration access 15

Monitoring / redirecting network connections 16

Monitoring Processes on the system 18

Code Analysis *18*

Unpacking the ASpacked executable 18

Finding Authentication Method 23

Analysis Wrap-UP *25*

References *33*

Software Resources *33*

© SANS Institute 2005 Author retains full rights.

Abstract

This paper will be discussing various methods and procedures used to analyze an unknown Malware specimen. The goal is to analyze the specimen, understand it, and finally control it. I will be using behavioral and code analysis to determine the characteristics of the malware specimen. If the specimen requires authentication to control or command it, I will attempt to extract the password during code analysis. If the password could not be extracted from the executable or if the password is encrypted, I will use the patching method to bypass authentication. During analysis I will be using many different freely available tools to identify and understand the unknown malware specimen. Tools such as BinText, Snort, OllyDbg, Regmon, Filemon, IDAPro, TDIMon, LordPE, RegShot, MD5sum, PEInfo, and ASPackDie will be used. Finally, after controlling the malware, the different commands seen from extracted strings will be tested and explained.

Laboratory Setup

Hardware Resources

Four desktop computers were used to setup my Laboratory instead of using VMWare. The choice to use actual computers instead of virtual machines is due To malware programmers checking for the use of virtual machine for analysis and making the malware behave differently. The hardware configuration of these computers is summarized in Table 1.1.

Computer Name	REM1	REM2	REM3	REM4
Processor	PIII 850 MHz	Celeron 500	Celeron 500	Celeron 500
Memory	128MB	128MB	128MB	128MB
O/S	Windows XP SP2	Windows 2000	Redhat 9	Redhat 9
IP Address	192.168.1.1	192.168.1.2	192.168.1.3	192.168.1.4
Network Card	10/100	10/100	10/100	10/100
Application	Ollydbg Regmon Filemon IDA Pro TDIMon, BinText LordPE, PEinfo RegShot MD5sum	None	Snort	IRC HTTPD FTPD IRC Client

Table 1.1 – Malware Lab Configuration

© SANS Institute 2005 Author retains full rights.

Each computer in my Reverse Engineering Malware (REM) lab was given a sequential name: REM1, REM2, REM3, and REM4. I used REM1(the Pentium III 850 MHz computer) for analysis. This is the computer that will be used to be infected by the malware specimen. This computer also will have all the necessary software utilities I need to do the analysis of the specimen. The second computer I am using will have only windows 2000 installed. This computer is going to be used to participate on a possible zombie computer. The third computer has Redhat 9 and Snort installed on it. This computer will be responsible only for gathering network traffic between the four computers in the lab. The fourth computer is used as a service server. It will be running IRC, Web and FTP servers. This computer will also run an IRC client.

Networking Setup

The lab network is configured as shown in Figure 1.1. It includes four Pentium computers and a 10base-T hub. A hub instead of a switch is used so that the network traffic is broadcasted to all the ports. REM3, the computer that is running snort, is configured with one 10/100 network card. This network card is running in promiscuous mode so it can capture all traffic on the wire. The computers are configured for TCP/IP. They are configured with static IP address as shown in table 1.1. These computers are entirely isolated from any other networks. In addition to the setting mentioned above REM1, the computer that will be infected is configured as follows:

IP Address:	192.168.1.1
Subnet Mask:	255.255.255.0
Default Gateway:	192.168.1.1
Preferred DNS Server:	192.168.1.1
Alternate DNS Server:	192.168.1.2

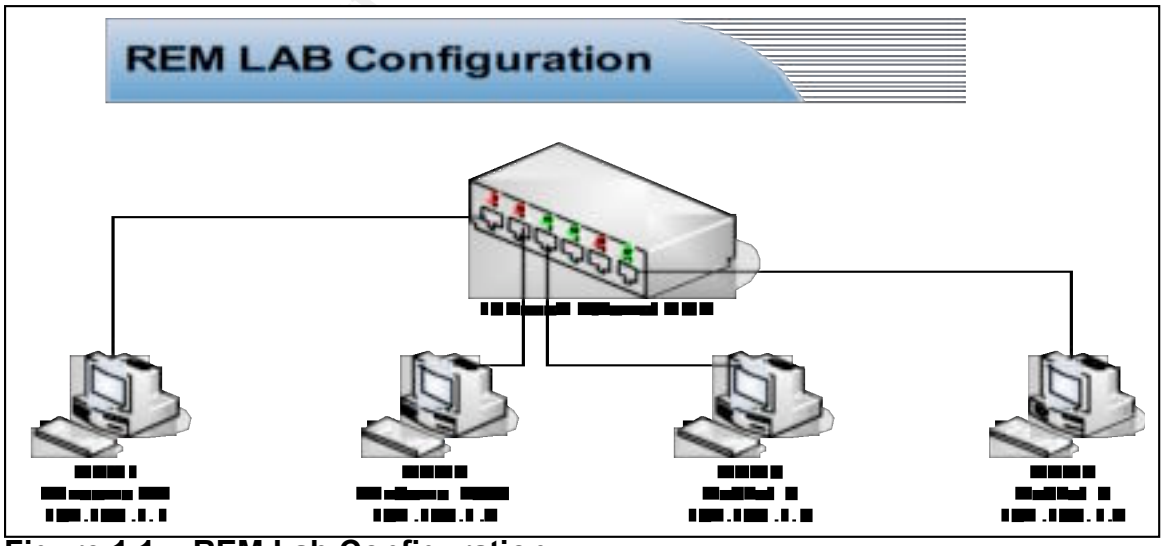

Figure 1.1 - REM Lab Configuration

© SANS Institute 2005 Author retains full rights.

Software Resources

Windows XP SP2

Windows XP is a 32 bit operating system developed by Microsoft Corporation. This was used in REM1 which is the analysis computer. This installation is patched with service pack 2. All the firewall functionality of the operating system is disabled. More information about this Operating system can be found at www.microsoft.com.

Windows 2000

Windows XP is a 32 bit operating system developed by Microsoft Corporation. This computer is used for participation in a DDOS and to demonstrate that multiple computers can be controlled by the specimen. This installation of the operating system is not patched. More information about this Operating system can be found at www.microsoft.com.

Snort

Snort is a light weight intrusion detection system developed by Marty Roesch. It is capable of sniffing and logging real-time network traffic. This software can be used in multiple different configurations. It has both a signature-based engine and anomaly detection engine. This lab utilizes snort as packet sniffer. I used snort to monitor the specimen's network activity. Snort can be downloaded both as source or binary at www.snort.org.

Undernet-IRCU2

I installed Undernet's IRC daemon on REM4 to satisfy the requirement of the specimen and to keep the lab isolated. This was done so that the specimen does not join a public IRC server. The IRC server is capable of listening on any port that the user defines. The configuration of the IRC daemon is performed through the ircd.conf file. This IRC daemon can be downloaded form http://prdownloads.sourceforge.net/undernet-ircu/ircu2.10.11.07.tar.gz?download.

Ollydbg

"OllyDbg is a 32-bit assembler level analyzing debugger for Microsoft Windows. Emphasis on binary code analysis makes it particularly useful in cases where source is unavailable. It predicts contents of registers, recognizes procedures, API calls, switches, tables, constants and strings, locates routines from object files and libraries, allows custom labels and comments in disassembled code, writes patches back to executable file and more. You can write your own plug-ins - dynamic link libraries that attach to OllyDbg and provide new functions. Plugins can insert entries into pop-up menus of OllyDbg windows, process keyboard shortcuts, save data to .ini and .udd files and call more than 170 functions exported by OllyDbg.[1]" This software is free and can be downloaded at

© SANS Institute 2005 Author retains full rights.

http://downloads-zdnet.com.com/OllyDbg/3000-2383_2-10242634.html?tag=lst-0-1

Regmon

This is a registry monitoring tool. Key creation, modification and deletion are captured by Regmon. I used this tool to monitor the modification the specimen made to the host computers registry. The utility works on multiple windows platforms. This utility is free and can be downloaded from Sysinternals at http://www.sysinternals.com/ntw2k/source/regmon.shtml.

Filemon

Filemon is a great utility that can show file activities. Filemon can monitor file copying, deletion, and creation. I used Filemon to track what files the specimen created or copied. This is a free utility and can be downloaded at http://www.sysinternals.com/ntw2k/source/filemon.shtml.

IDAPro

IDAPro is a Windows and Linux based disassembler. I used this utility to disassemble the specimen. It is very helpful when trying to find out the different subroutine jumps. It also has strings when clicked will jump to the code where the string is referenced. This is a commercial product, but a demo version is available at http://www.datarescue.be/downloaddemo.htm.

TDIMon

TDIMon is a utility that monitors TCP and UDP activity. I used this utility to determine if the infected machine is listening on a port. This is free software that can be downloaded at http://www.sysinternals.com/ntw2k/freeware/tdimon.shtml.

LordPE

This tool is utilized to edit PE headers. This utility can also be used to dump processes from memory to file. I utilized this software to find out the ImageBase for the packed executable to determine the OPE (Original Point of Entry). This utility can be downloaded from http://www.softpedia.com/progDownload/LordPE-Download-29.html.

RegShot

RegShot is a registry comparison utility. I was able to determine what registry modifications were made by the specimen. This allows you to take a snap shot of the registry before and after the specimen is executed and then compares the registry content and display the difference. This is a free utility that can be downloaded at http://regshot.ist.md/

MD5sum

MD5sum is a utility used to determine a message digest (Hash value) of the specimen before and after execution. This was done to verify that the specimen

© SANS Institute 2005 Author retains full rights.

did not change the executable after the execution of the file. MD5sum can be found at http://www.weihenstephan.de/~syring/win32/UnxUtils.html.

PEInfo

PEInfo was developed by Tom Liston. This utility can be used to find out PE header info, file size of an executable, embedded strings and some more information. I also used this software to determine the size of the executable specimen.

BinText

This utility extracts strings from an executable program. I used it to find out the command the specimen used. This utility is free and can be downloaded from http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/freetools.htm.

ASpackdie

ASpackdie is a utility that enables you to unpack executables that have been packed using ASpack. I used this utility as a second alternative to unpacking the malicious executable. This is a very easy to use utility that can be downloaded at http://www.woodmann.com/crackz/Unpackers/Aspdie.zip.

Properties of the Malware Specimen

The Malware specimen has many properties that are of interest. Type of the malware file, size of the file, MD5 hash of the file, operating system it runs on, and strings embedded into it are some of the properties of the malware listed below.

Type of File

The malware specimen is Packed Executable. It is packed using software called aspack. This software is used to compress and protect executables. The method in which the malware specimen was packed was evident when the file is opened using PEInfo and BinText. The string "!This program cannot be run in DOS mode." This is evident that the malware specimen is an executable program.

Size of the File

The size of the malware specimen in a packed (compressed) state is 41984 bytes. This was determined by opening the file with PEInfo. See figure 2.1. The size of the file can also be determined by windows explorer and looking at the properties of the file.

© SANS Institute 2005 Author retains full rights.

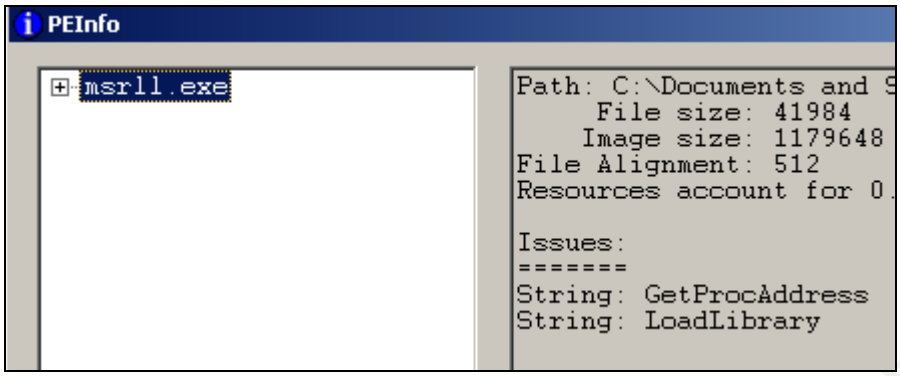

Figure 2.1 - PEInfo showing the size of packed executable

The size of the malware specimen after it has been unpacked using OllyDbg is 1182720 bytes as shown in figure 2.2.

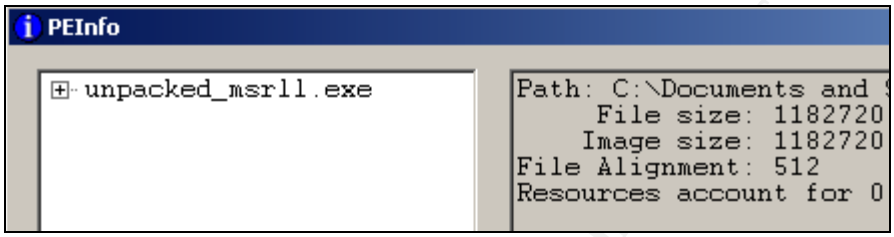

Figure 2.2 PEInfo showing the size of unpacked executable

MD5 Hash of the File

MD5 hash verifies the integrity of a file. The MD5 hash of the malware specimen was generated before and after execution to verify that the executable was not modified. The MD5 hash of msrll.exe, the specimen, before the file was executed is 84acfe96a98590813413122c12c11aaa. The MD5 hash of msrll.exe after the file was executed is 84acfe96a98590813413122c12c11aaa which shows that the file has not been modified by the malware specimen.

Operating System it runs on

This malware specimen runs on Microsoft Windows operating system. This can be found out by opening the executable with PEInfo and expanding the Imports tree. It shows that the executable references multiple dll files which are an evidence of Microsoft Operating System.

Strings Embedded into it

I was able to extract the string embedded into the executable after I unpacked it and opened it using BinText. Table 2.1 shows all the extracted strings.

!This program cannot be run in DOS mode.	__%s__	Sh$l@	7h*h@
.text	__%s___	PShZP@	jtram.conf
.data	NICK %s	mode %s +o %s	%s /t %s
.idata	%s %s	akick	jtr.home

© SANS Institute 2005 Author retains full rights.

.aspack	irc.chan	mode %s +b %s %s	%s\%s
.adata	%s %s	KICK %s %s	%s: possibly failed: code %u
.newIID	WHO %s	irc.pre	%s: possibly failed
?insmod	PPhV,@	Set an irc sock to preform %s command on	%s: exec of %s failed err: %u
?rmmod	USERHOST %s	Type	u.exf
?lsmod	logged into %s(%s) as %s	%csklist	Ph+j@
%s: <mod name>	<$hE:@	to view current sockets, then	Ph?j@
%s: mod list full	PhR:@	%cdccsk	jtr.id
%s: err: %u	nick.pre	<#>	%s: <url> <id>
mod_init	%s-%04u	%s: dll loaded	IREG
mod_free	irc.user	%s: %d	CLON
%s: cannot init %s	irc.usereal	RhHY@	ICON
%s: %s loaded (%u)	irc.real	RhHY@	WCON
%s: mod allready loaded	irc.pass	said %s to %s	#%u [fd:%u] %s:%u [%s%s] last:%u
%s:%s err %u	tsend(): connection to %s:%u failed	usage: %s <target> "text"	|\=> [n:%s fh:%s] (%s)
%s:%s not found	USER %s localhost 0 :%s	%s not on %s	|---[%s] (%u) %s
%s: unloading %s	NICK %s	usage: %s <nick> <chan>	| |-[%s%s] [%s]
[%u]: %s hinst:%x	Ph <@	%s logged in	|=> (%s) (%.8x)
unloading %s	PRIVMSG	Sh [@	B$PRhco@
%s: invalid_addr: %s	trecv(): Disconnected from %s err:%u	sys: %s bot: %s	%s <pass> <salt>
%s%s [port]	NOTICE	preformance counter not avail	%s <nick> <chan>
finished %s	%s %s :%s	usage: %s <cmd>	PING %s
%s <ip> <port> <t_time> <delay>	Ph}D@	%s free'd	mIRC v6.12 Khaled Mardam-Bey
sockopt: %u	MODE %s -o+b %s *@%s	unable to free %s	VERSION %s
sendto err: %u	C'PSWh	0h+\@	dcc.pass
sockraw: %u	Sh'G@	later!	temp add %s
syn: done	MODE %s -bo %s %s	unable to %s errno:%u	$h%u@
%s <ip> <duration> <delay>	Sh'G@	service:%c user:%s inet connection:%c contype:%s reboot privs:%c	%s%u-%s
sendto: %u	%s.key	Ph@]@	%s opened (%u)
jolt2: done	Ph'G@	%-5u %s	%u bytes from %s in %u seconds saved to %s
%s <ip> <p size> <duration> <delay>	sk#%u %s is dead!	%s: %s	(%s %s): incomplete! %u bytes
Err: %u	s_check: %s dead? pinging...	%s: somefile	couldnt open %s err:%u
smurf done	PING :ok	PhHY@	(%s) %s: %s
PhV#@	s_check: send error to %s disconnecting	host: %s ip: %s	(%s) urlopen failed

© SANS Institute 2005 Author retains full rights.

&err: %u	expect the worst	capGetDriverDescriptionA	(%s): inetopen failed
?ping	s_check: killing socket %s	cpus:%u	Whjv@
?smurf	irc.knick	WIN%s (u:%s)%s%s mem:(%u/%u) %u%% %s %s	Ph w@
?jolt	jtr.%u%s.iso	%s: %s (%u)	no file name in %s
PONG :%s	ison %s	%s %s	%s created
0h (@	servers	%s bad args	%s %s to %s Ok
%s!%s@%s	s_check: trying %s	3hTg@	3hI~@
%s!%s	Ph9K@	akick	%0.2u/%0.2u/%0.2u %0.2u:%0.2u %15s %s
SVh=+@	PhkK@	%s[%u] %s	%s (err: %u)
irc.nick	ShtK@	%s removed	ShHY@
NICK %s	uYVh|K@	couldnt find %s	err: %u
NETWORK=	%s.mode	%s added	%s %s :ok
irc.pre	MODE %s %s	%s allready in list	unable to %s %s (err: %u)
_%s__	ShRP@	usage: %s +/- <host>	ShHY@
%-16s %s	?unset	SSL_new	$5FWhy
%-16s (%u.%u.%u.%u)	?uattr	SSL_set_fd	#4EVgx
[%s][%s] %s	?dccsk	SSL_connect	$5FWhy
closing %u [%s:%u]	?killsk	SSL_write	gN]HU
unable to close socket %u	VERSION*	SSL_read	desired_keysize != NULL
using sock #%u %s:%u (%s)	IDENT	SSL_shutdown	ctr.c
Invalid sock	%ud %02uh %02um %02us	SSL_free	ctr != NULL
usage %s <socks #>	%02uh %02um %02us	SSL_CTX_free	key != NULL
leaves %s	%um %02us	kernel32.dll	count != NULL
:0 * * :%s	jtram.conf	QueryPerformanceCounter	ct != NULL
joins: %s	jtr.*	QueryPerformanceFrequency	pt != NULL
ACCEPT	DiCHFc2ioiVmb3cb4zZ7zWZH1oM=	RegisterServiceProcess	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
resume	conf_dump: wrote %u lines	jtram.conf	?456789:;<=
err: %u	get of %s incomplete at %u bytes	irc.user	!"#$%&'()*+,-./0123
DCC ACCEPT %s %s %s	get of %s completed (%u bytes), %u seconds %u cps	%s : USERID : UNIX : %s	base64.c
dcc_resume: cant find port %s	error while writing to %s (%u)	QUIT :FUCK %u	outlen != NULL
dcc.dir	chdir: %s -> %s (%u)	Killed!? Arrg! [%u]	out != NULL
%s\%s\%s\%s	dcc_wait: get of %s from %s timed out	QUIT :%s	in != NULL
unable to open (%s): %u	dcc_wait: closing [#%u] %s:%u (%s)	SeShutdownPrivilege	_ARGCHK '%s' failure on line %d of file %s

© SANS Institute 2005 Author retains full rights.

resuming dcc from %s to %s	%4s #%.2u %s %ucps %u%% [sk#%u] %s	%s\%s	crypt.c
DCC RESUME %s %s %u	%u Send(s) %u Get(s) (%u transfer(s) total) UP:%ucps DOWN:%ucps Total:%ucps	%s\%s\%s	name != NULL
?clone	PRQh0	Rll enhanced drive	cipher != NULL
?clones	send of %s incomplete at %u bytes	software\microsoft\windows\currentversion\run	hash != NULL
?login	send of %s completed (%u bytes), %u seconds %u cps	/d "%s"	prng != NULL
?uptime	cant open %s (err:%u) pwd:{%s}	< u&	LibTomCrypt 0.83
?reboot	DCC SEND %s %u %u %u	./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz	Endianess: little (32-bit words)
?status	%s %s	usage %s: server[:port] amount	Clean stack: disabled
?jump	%s exited with code %u	%s: %s	Ciphers built-in:
?nick	%s\%s	%s %s %s <PARAM>	Blowfish
?echo	%s: %s	%s: [NETWORK|all] %s <"parm"> ...	RC2
?hush	exec: Error:%u pwd:%s cmd:%s	USER %s localhost 0 :%s	RC5
?wget	dcc.pass	NICK %s	RC6
?join	bot.port	PSVh	Serpent
?akick	%s bad pass from "%s"@%s	md5.c	Safer+
?part	%s: connect from %s	md != NULL	Safer
?dump	jtr.bin	buf != NULL	Rijndael
?md5p	msrll.exe	hash != NULL	XTEA
?free	jtr.home	message digest	Twofish
?update	jtr.id	abcdefghijklmnopqrstuvwxyz	CAST5
?hostname	irc.quit	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789	Noekeon
?!fif	servers	1.23457E+79	Hashes built-in:
?play	collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.zxy0.com:8080	sprng	SHA-512
?copy	irc.chan	sprng.c	SHA-384
?move	#mils	buf != NULL	SHA-256
?sums	1KZLPLKDf$W8kl8Jr1X8DOHZsmIp9qq0	rc6.c	TIGER
?rmdir	1KZLPLKDf$55isA1ITvamR7bjAdBziX.	skey != NULL	SHA1
?mkdir	SSL_get_error	key != NULL	MD5
?exec	SSL_load_error_strings	ct != NULL	MD4

© SANS Institute 2005 Author retains full rights.

?kill	SSL_library_init	pt != NULL	MD2
?killall	SSLv3_client_method	#4EVgx	Block Chaining Modes:
?crash	SSL_set_connect_state	$5FWhy	CFB
?sklist	SSL_CTX_new	#4EVgx	OFB
CTR	<die|join|part|raw|msg>	GetTempPathA	memset
PRNG:	AdjustTokenPrivileges	GetTickCount	printf
Yarrow	CloseServiceHandle	GetVersionExA	raise
SPRNG	CreateServiceA	GlobalMemoryStatus	realloc
RC4	CryptAcquireContextA	InitializeCriticalSection	setvbuf
PK Algs:	CryptGenRandom	IsBadReadPtr	signal
RSA	CryptReleaseContext	LeaveCriticalSection	sprintf
DH	GetUserNameA	LoadLibraryA	srand
ECC	LookupPrivilegeValueA	MoveFileA	strcat
KR	OpenProcessToken	OpenProcess	strchr
Compiler:	OpenSCManagerA	PeekNamedPipe	strcmp
WIN32 platform detected.	RegCloseKey	Process32First	strcpy
GCC compiler detected.	RegCreateKeyExA	Process32Next	strerror
Various others: BASE64 MPI HMAC	RegSetValueExA	QueryPerformanceFrequency	strncat
/dev/random	RegisterServiceCtrlHandlerA	ReadFile	strncmp
Microsoft Base Cryptographic Provider v1.0	SetServiceStatus	ReleaseMutex	strncpy
bits.c	StartServiceCtrlDispatcherA	RemoveDirectoryA	strstr
buf != NULL	AddAtomA	SetConsoleCtrlHandler	toupper
t9VWS	CloseHandle	SetCurrentDirectoryA	ShellExecuteA
prng != NULL	CopyFileA	SetFilePointer	DispatchMessageA
<"tx< tf< t	CreateDirectoryA	SetUnhandledExceptionFilter	ExitWindowsEx
< tV< t	CreateFileA	Sleep	GetMessageA
< tJ< tF	CreateMutexA	TerminateProcess	PeekMessageA
#NAME?	CreatePipe	WaitForSingleObject	GetFileVersionInfoA
<ip> <total secs> <p size> <delay>	CreateProcessA	WriteFile	VerQueryValueA
modem	CreateToolhelp32Snapshot	_itoa	InternetCloseHandle
Lan	DeleteFileA	_stat	InternetGetConnectedState
Proxy	DuplicateHandle	_strdup	InternetOpenA
none	EnterCriticalSection	_stricmp	InternetOpenUrlA
m220 1.0 #2730 Mar 16 11:47:38 2004	ExitProcess	__getmainargs	InternetReadFile
unable to %s %s (err: %u)	ExitThread	__p__environ	WSAGetLastError
unable to kill %s (%u)	FileTimeToSystemTime	__p__fmode	WSASocketA
%s killed (pid:%u)	FindAtomA	__set_app_type	WSAStartup
AVICAP32.dll	FindClose	_beginthread	__WSAFDIsSet
unable to kill %u (%u)	FindFirstFileA	_cexit	accept

© SANS Institute 2005 Author retains full rights.

pid %u killed	FindNextFileA	_errno	closesocket
error!	FreeLibrary	_fileno	connect
ran ok	GetAtomNameA	_onexit	gethostbyaddr
MODE %s +o %s	GetCommandLineA	_setmode	gethostbyname
set %s %s	GetCurrentDirectoryA	_vsnprintf	gethostname
Mozilla/4.0	GetCurrentProcess	abort	getsockname
Accept: */*	GetCurrentThreadId	atexit	htonl
<DIR>	GetExitCodeProcess	clock	htons
Could not copy %s to %s	GetFileSize	fclose	inet_addr
%s copied to %s	GetFullPathNameA	fflush	inet_ntoa
0123456789abcdef	GetLastError	fgets	ioctlsocket
%s unset	GetModuleFileNameA	fopen	listen
unable to unset %s	GetModuleHandleA	fprintf	ntohl
(%s) %s	GetProcAddress	fread	select
%s %s	GetStartupInfoA	fwrite	sendto
libssl32.dll	GetSystemDirectoryA	malloc	setsockopt
libeay32.dll	GetSystemInfo	memcpy	shutdown
socket	CryptReleaseContext	IsBadReadPtr	realloc
ADVAPI32.DLL	GetUserNameA	LeaveCriticalSection	setvbuf
KERNEL32.dll	LookupPrivilegeValueA	LoadLibraryA	signal
msvcrt.dll	OpenProcessToken	MoveFileA	sprintf
msvcrt.dll	OpenSCManagerA	OpenProcess	srand
SHELL32.DLL	RegCloseKey	PeekNamedPipe	_mbscat
USER32.dll	RegCreateKeyExA	Process32First	strchr
VERSION.dll	RegSetValueExA	Process32Next	strcmp
WININET.DLL	RegisterServiceCtrlHandlerA	QueryPerformanceFrequency	_mbscpy
WS2_32.DLL	SetServiceStatus	ReadFile	strerror
VirtualAlloc	StartServiceCtrlDispatcherA	ReleaseMutex	strncat
VirtualFree	kernel32.dll	RemoveDirectoryA	strncmp
kernel32.dll	AddAtomA	SetConsoleCtrlHandler	strncpy
ExitProcess	CloseHandle	SetCurrentDirectoryA	strstr
user32.dll	CopyFileA	SetFilePointer	toupper
MessageBoxA	CreateDirectoryA	SetUnhandledExceptionFilter	shell32.dll
wsprintfA	CreateFileA	Sleep	ShellExecuteA
LOADER ERROR	CreateMutexA	TerminateProcess	USER32.dll
The procedure entry point %s could not be located in the dynamic link library %s	CreatePipe	WaitForSingleObject	DispatchMessageA
The ordinal %u could not be located in the dynamic link library %s	CreateProcessA	WriteFile	ExitWindowsEx
(08@P	CreateToolhelp32Snapshot	msvcrt.dll	GetMessageA

© SANS Institute 2005 Author retains full rights.

D4l|M	DeleteFileA	_itoa	PeekMessageA
;;F,s	DuplicateHandle	_stat	version.dll
,;F0s	EnterCriticalSection	_mbsdup	GetFileVersionInfoA
;F4s	ExitProcess	_strcmpi	VerQueryValueA
D$$W3	ExitThread	msvcrt.dll	wininet.dll
kernel32.dll	FileTimeToSystemTime	__getmainargs	InternetCloseHandle
GetProcAddress	FindAtomA	__p__environ	InternetGetConnectedState
GetModuleHandleA	FindClose	__p__fmode	InternetOpenA
LoadLibraryA	FindFirstFileA	__set_app_type	InternetOpenUrlA
advapi32.dll	FindNextFileA	_beginthread	InternetReadFile
msvcrt.dll	FreeLibrary	_cexit	ws2_32.dll
msvcrt.dll	GetAtomNameA	_errno	WSAGetLastError
shell32.dll	GetCommandLineA	_fileno	WSASocketA
user32.dll	GetCurrentDirectoryA	_onexit	WSAStartup
version.dll	GetCurrentProcess	_setmode	__WSAFDIsSet
wininet.dll	GetCurrentThreadId	_vsnprintf	accept
ws2_32.dll	GetExitCodeProcess	abort	closesocket
AdjustTokenPrivileges	GetFileSize	atexit	connect
_itoa	GetFullPathNameA	clock	gethostbyaddr
__getmainargs	GetLastError	fclose	gethostbyname
ShellExecuteA	GetModuleFileNameA	fflush	gethostname
DispatchMessageA	GetModuleHandleA	fgets	getsockname
GetFileVersionInfoA	GetProcAddress	fopen	htonl
InternetCloseHandle	GetStartupInfoA	fprintf	htons
WSAGetLastError	GetSystemDirectoryA	fread	inet_addr
advapi32.dll	GetSystemInfo	fwrite	inet_ntoa
AdjustTokenPrivileges	GetTempPathA	malloc	ioctlsocket
CloseServiceHandle	GetTickCount	memcpy	listen
CreateServiceA	GetVersionExA	memset	htonl
CryptAcquireContextA	GlobalMemoryStatus	printf	select
CryptGenRandom	InitializeCriticalSection	raise	sendto

Table 2.1 – Strings embedded into msrll.exe

Behavioral Analysis

Before I started behavioral analysis, first I made a backup of registry and system state files just incase the Malware specimen destroys the analysis workstation. I started my analysis by first running Regshot and making a comparison of the registry before and after the malware was run. I then ran Regmon, Filemon, and TDIMon to log some of the activities the Malware performed. The findings are explained in detail next.

Monitoring file system access

Examining Filemon reveled the malware specimen did the following:

© SANS Institute 2005 Author retains full rights.

1. The specimen created a directory C:\windows\system32\mfm. Figure 3.1 shows excerpt from Filemon log.

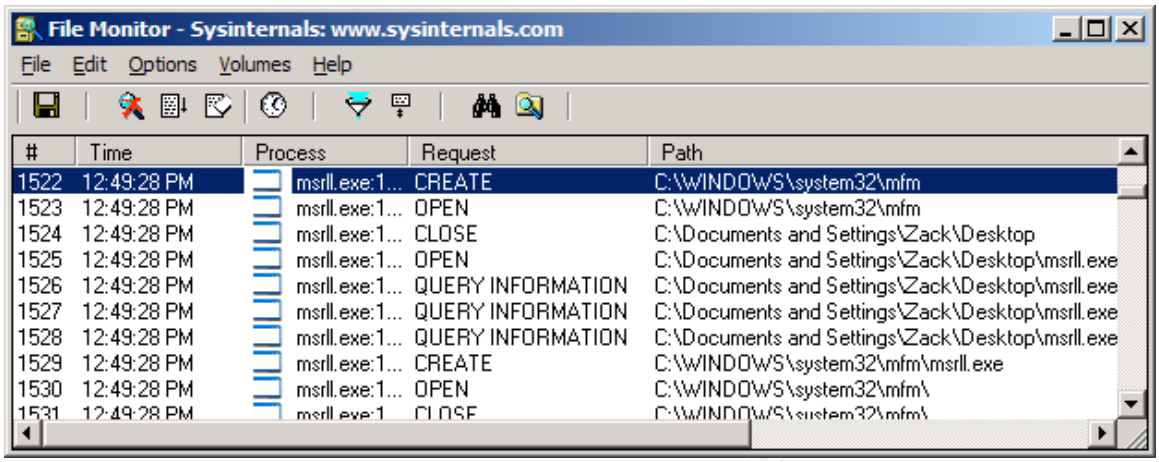

Figure 3.1 – The specimen created C:\windows\system32\mfm directory

2. The specimen copied it self from desktop, where it was executed, to C:\windows\system32\mfm directory.
3. The specimen deleted the copy of itself (msrll.exe) from the desktop. See figure 3.2.

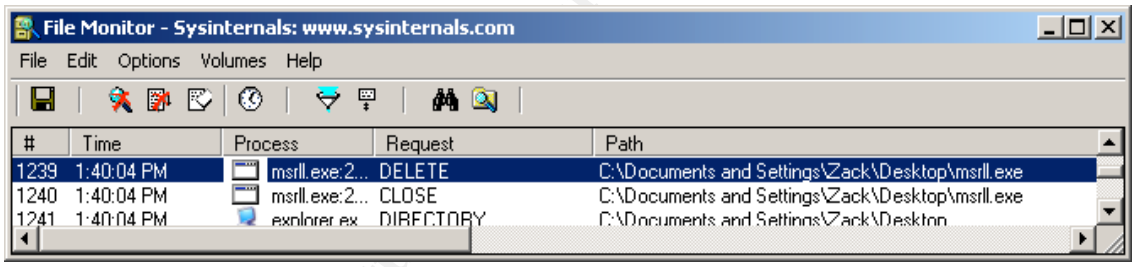

Figure 3.2 – The malware specimen deleted msrll.exe form desktop

4. The specimen opened and read a file jtram.conf often.

Monitoring registry / configuration access

To analyze what registry changes are made by the malware specimen, I first used registry and file comparison tool called regshot. I ran regshot and took the snapshot of the registry before the malware was run. I ran the malware and took a second shot of the registry. The comparison of the two snapshots indicated that the malware added 5 keys, added 22 values and modified 8 values. One interesting key that was added was HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mfm. What made this interesting is that the specimen ran as a service instead of a program that ran when the operating system started. This service was named RII enhanced drive and was set to start automatically. I was able to verify this by looking at the services as shown in Figure 3.3.

© SANS Institute 2005 Author retains full rights.

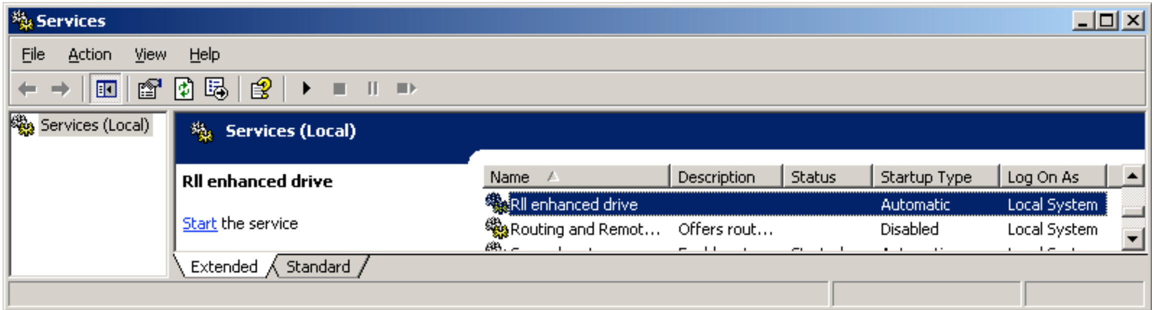

Figure 3.3 – msrll.exe running as service

It seems that the programmer of this specimen is trying to hide the purpose of this program by giving it a name that is associated with hard drive technology. There were other keys that were added or modified that dealt with cryptography.

Monitoring / redirecting network connections

To monitor the network connections I used Snort with the following command line statement: Snort –vd | tee /tmp/grem.log. I then ran the malware specimen and observed the snort log. I was able to find that the infected host attempted multiple DNS host name resolution with out any success. It was attempting to resolve Collective7.zxy0.com. See Figure 3.4 below.

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

12/17-10:22:06.939599 192.168.1.1:1091 -> 192.168.1.2:53
UDP TTL:128 TOS:0x0 ID:51413 IpLen:20 DgmLen:66
Len: 38
9C 80 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.....

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

12/17-10:22:06.940792 192.168.1.2 -> 192.168.1.1
ICMP TTL:128 TOS:0x0 ID:52135 IpLen:20 DgmLen:56
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
192.168.1.1:1091 -> 192.168.1.2:53
UDP TTL:128 TOS:0x0 ID:51413 IpLen:20 DgmLen:66
Len: 38
** END OF DUMP
00 00 00 00 45 00 00 42 C8 D5 00 00 80 11 EE 81  ....E..B........
C0 A8 01 01 C0 A8 01 02 04 43 00 35 00 2E 43 85  .........C.5..C.

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

Figure 3.4 – Snort log showing the specimen attempting to connect to IRC server

© SANS Institute 2005 Author retains full rights.

My previous investigation of the Malware's strings using BinText had identified this domain of "Collective7.zxy0.com" (see Table 2.1). I added Collective7.zxy0.com to the hosts file of the infected computer and resolved it to 192.168.1.4, which is my REM4 server that is running FTP, HTTP, and IRCD services. After resolving the domain name to an IP address, the specimen attempted to connect to the server using ports 8080, 9999, and 6667. In all cases the server responded with ACK/RST as shown in figure 3.5.

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

12/17-10:25:13.654471 192.168.1.1:3987 -> 192.168.1.4:6667
TCP TTL:128 TOS:0x0 ID:51424 IpLen:20 DgmLen:48 DF
******S* Seq: 0x73EC71F8  Ack: 0x0  Win: 0xFFFF  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

12/17-10:25:13.654590 192.168.1.4:6667 -> 192.168.1.1:3987
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x0  Ack: 0x73EC71F9  Win: 0x0  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Figure 3.5 – The specimen is connecting to an IRC server on port 6667

I configured the web server to listen on port 8080, since port 8080 is usually used by web proxy servers. When the infected host attempted to connect to the server using port 8080, it responded with an ACK/FIN. This indicated that the infected host is not trying to connect to a web server via port 8080. I proceeded to modify my IRC server configuration so that it listens on port 8080. The infected machine was able to connect to the IRC server using port 8080. The snort logs also show that the infected machine joined the #mils channel with a nick name of FniigYEru. See figure 3.6 below.

The nick name is a randomly generated string which is typical for IRC bots. At this time I came to a conclusion that the malware specimen is some type of an IRC Bot. The Bot was also trying to connect to ports 9999 and 6777; therefore, I configured the IRC server to listen on port 6667 to see if the Bot would behave differently. I was able to connect to the IRC server and joined the same channel, #mils. Once the Bot joined the IRC channel, I also joined the #mils channel hopping to be able to control the Bot. I tried some of the commands that are listed in table 2.1. I am assuming that the strings that start with "?" are the Bot command. I tried most of these command, but the Bot did not respond. Investigating the strings output, it was evident that the Bot might be using some type of authentication. Some of the strings that clued me are: irc.pass, dcc.pass, and "%s bad pass from "%s"@%s".

© SANS Institute 2005 Author retains full rights.

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
12/17-10:27:16.332789 192.168.1.1:3990 -> 192.168.1.4:6667
TCP TTL:128 TOS:0x0 ID:51432 IpLen:20 DgmLen:48 DF
******S* Seq: 0x9403E463  Ack: 0x0  Win: 0xFFFF  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

12/17-10:27:16.332988 192.168.1.4:6667 -> 192.168.1.1:3990
TCP TTL:64 TOS:0x8 ID:0 IpLen:20 DgmLen:48 DF
***A**S* Seq: 0x800A7FCB  Ack: 0x9403E464  Win: 0xB68  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

12/17-10:27:16.333118 192.168.1.1:3990 -> 192.168.1.4:6667
TCP TTL:128 TOS:0x0 ID:51433 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x9403E464  Ack: 0x800A7FCC  Win: 0xFFFF  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
.
.Snip
.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
12/17-10:27:48.476224 192.168.1.1:3990 -> 192.168.1.4:6667
TCP TTL:128 TOS:0x0 ID:51446 IpLen:20 DgmLen:53 DF
***AP*** Seq: 0x9403E4C1  Ack: 0x800A84BD  Win: 0xFB0E  TcpLen: 20
4A 4F 49 4E 20 23 6D 69 6C 73 20 3A 0A           JOIN #mils :.
```

Figure 3.6 – The infected host joins an IRC channel

Monitoring Processes on the system

Examining the TDIMon logs, I discovered that the infected host is now listening on ports 2200 and 113. Port 113 is used for Ident, but I was not sure why the host was listening on port 2200. I confirmed this information by using "netstat -an" command. I used telnet to find out if I could connect to the listing port. I used the command "telnet 192.168.1.1 2200", and I got a "#:" prompt. I tried some of the commands that start with "?", but I received no response.

Code Analysis

Unpacking the ASpacked executable

ASpack is a utility programmers use to compress executables. The presence of ".aspack" in the BinText strings and output from PEinfo suggests that this malware specimen was packed using ASpack. There are few methods available to unpack an ASpacked executable. I used two techniques to verify that the unpacking was successful. The first technique I used was the "ASpackDie " software program. This is a very easy to use utility that can be downloaded at http://www.woodmann.com/crackz/Unpackers/Aspdie.zip. The second technique I used utilized a debugger called OllyDbg to unpack the Aspacked executable. I was able to find a good tutorial on how to unpack Aspacked

© SANS Institute 2005 Author retains full rights.

executables at http://biw.rult.at/tuts/mupaspack.rar submitted by an individual with the alias Mr-Geek.[2] I opened OllyDbg and proceeded to open msrll.exe. Immediately, an entry point alert message appeared, and I proceeded by clicking OK. Assembly code of msrll.exe is displayed in Figure 4.1.

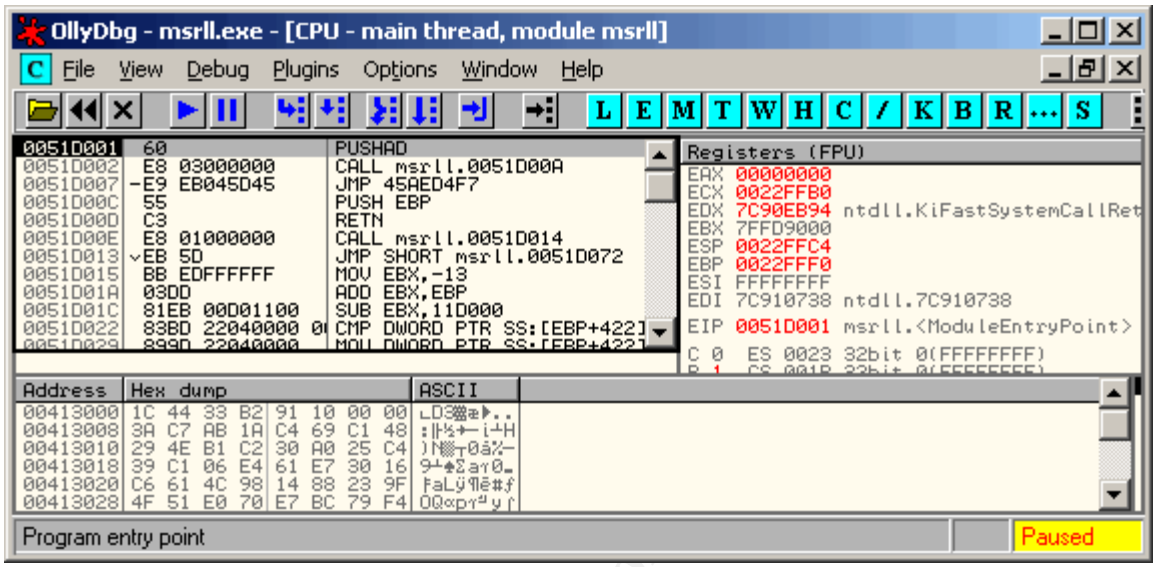

Figure 4.1 - msrll.exe as it appears in OllyDbg

Once msrll.exe is opened in OllyDbg, it is at the Entry Point of our packed executable. As you can see in Figure 4.1, the entry point is at memory address 0051D001. The objective here is to find the original entry point of msrll.exe prior to unpacking. When the executable was packed a code that unpacks it when it is executed is appended to the beginning of the executable (msrll.exe). The current entry point at memory address 0051D001 is where the unpacking routine begins. Next, the breakpoint should be set for OllyDbg to stop executing the program before it executes to the original code. To find the original entry point I pressed F8, the step over function, which executed the current code and stepped to the next instruction CALL msrll.0051D00A. At this point I noted the values of the ESP register and EDI register located in the right-hand "Registors (FPU)" window. The 7C910738, the EDI register value, is where the next breakpoint should be set. Now, at the Registers pane, right-click on the value of the ESP, 0022FFC4, and click on Follow in Dump. See Figure 4.2.

© SANS Institute 2005 Author retains full rights.

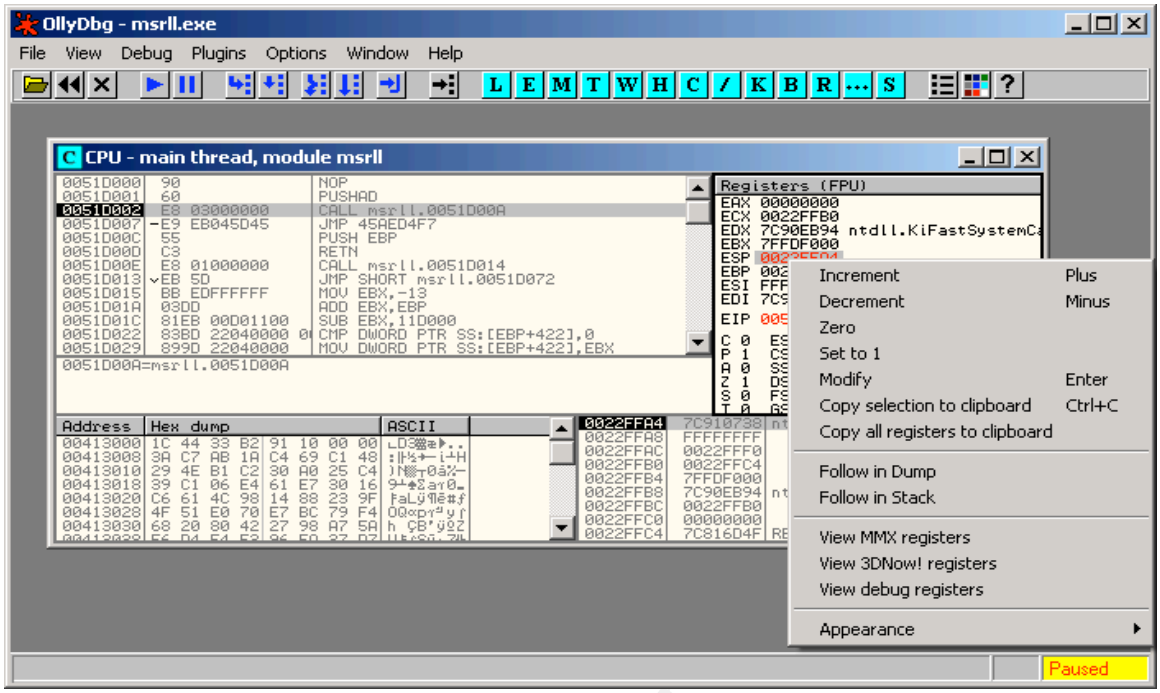

Figure 4.2 – Follow in dump.

Go to the dump pane, located at the left-bottom corner of OllyDbg, and highlight the first four bytes of HEX dump (38 07 91 7C) as shown in Figure 4.3. These four bytes are the EDI value reading it from right to left. Right click on the highlighted HEX code and click on "Breakpoint" → "Hardware, on access" → "Dword".

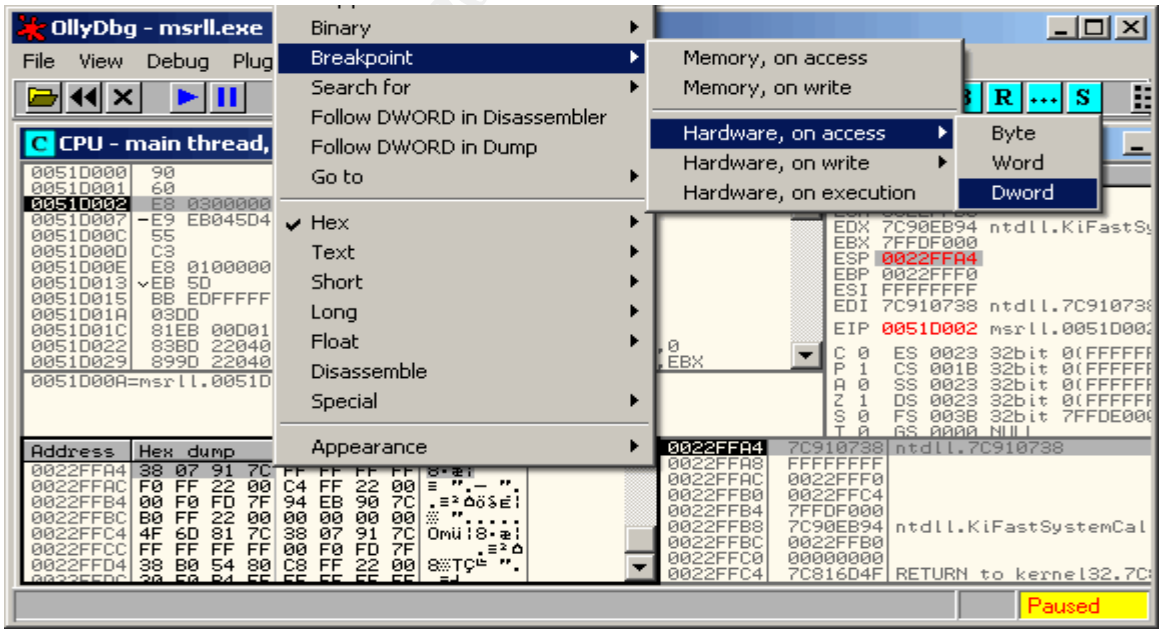

Figure 4.3 – Set Breakpoint

© SANS Institute 2005 Author retains full rights.

Now that the breakpoint is set, all the code from the entry point up to the breakpoint can be executed by pressing F9 or by clicking on debug and then Run. I ran this segment of the malware's code. The CPU pane should look like figure 4.4. At this point we are nearing the OPE (Original Point of Entry). OPE is the starting address of the malware before it was packed. Next, all the commands up to the memory address 0051D3BF should be run one line at a time by pressing F8. At the RETN instruction set, press F7 to trace into it.

```
0051D3B0  v75 08          JNZ SHORT msrll.0051D3BA
0051D3B2  B8 01000000     MOV EAX,1
0051D3B7  C2 0C00         RETN 0C
0051D3BA  68 40124000     PUSH msrll.00401240
0051D3BF  C3              RETN
0051D3C0  8B85 26040000   MOV EAX,DWORD PTR SS:[EBP+426]
0051D3C6  8D8D 3B040000   LEA ECX,DWORD PTR SS:[EBP+43B]
0051D3CC  51              PUSH ECX
0051D3CD  50              PUSH EAX
```

Figure 4.4 – Getting close to OPE

Now we are at the OPE. See figure 4.5. The memory address 00401240 is the OPE. The code shown in machine code represented in HEX.

```
00401240   55   DB 55   CHAR
00401241   89   DB 89
00401242   E5   DB E5
00401243   83   DB 83
00401244   EC   DB EC
00401245   08   DB 08
00401246   C7   DB C7
00401247   04   DB 04
00401248   24   DB 24   CHAR
00401249   01   DB 01
0040124A   00   DB 00
0040124B   00   DB 00
0040124C   00   DB 00
0040124D   FF   DB FF
0040124E   15   DB 15
0040124F   08   DB 08
00401250   B5   DB B5
00401251   51   DB 51   CHAR
00401252   00   DB 00
```

Figure 4.5 – OPE Found

The HEX code can be converted to assembly language by right clicking on the body of the HEX code and press Analysis → Analyze Code. This will convert the HEX code to Assembly code as shown in Figure 4.6.

```
00401240  . 55            PUSH EBP
00401241  . 89E5          MOV EBP,ESP
00401243  . 83EC 08       SUB ESP,8
00401246  . C70424 010000 MOV DWORD PTR SS:[ESP],1
0040124D  . FF15 08B55100 CALL DWORD PTR DS:[51B508]   msvc
00401253  . E8 88FFFFFF   CALL msrll.004011E0
00401258  . 89EC          MOV ESP,EBP
0040125A  . 31C0          XOR EAX,EAX
0040125C  . 5D            POP EBP
0040125D  . C3            RETN
0040125E    89F6          MOV ESI,ESI
00401260  . 55            PUSH EBP
00401261  . 89E5          MOV EBP,ESP
00401263  . 83EC 08       SUB ESP,8
00401266  . C70424 020000 MOV DWORD PTR SS:[ESP],2
0040126D  . FF15 08B55100 CALL DWORD PTR DS:[51B508]   msvc
00401273  . E8 68FFFFFF   CALL msrll.004011E0
00401278  . 89EC          MOV ESP,EBP
0040127A  . 31C0          XOR EAX,EAX
```

Figure 4.6 – Assembly Code

The next step is to dump the memory, which contains the unpacked code, to a file using the OllyDump plug-in. Before we can do that we need to figure out the

© SANS Institute 2005 Author retains full rights.

offset. The formula to find the offset is: Offset = OPE – ImageBase. We already have the OPE value (Figure 4.5). Now we need to find out the ImageBase. Using LordPE, the ImageBase is identified as seen in figure 4.7. In this case the ImageBase is 0040000; therefore, the offset is 00401240 – 0040000 = 1240.

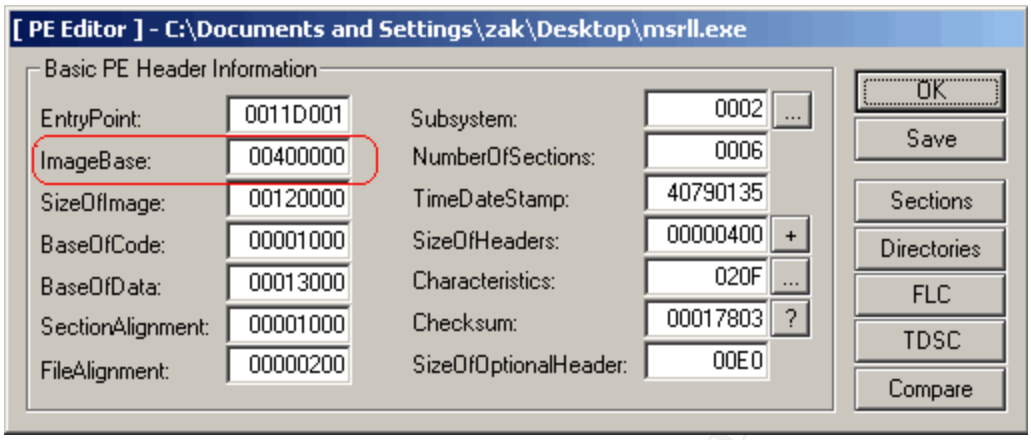

Figure 4.7 – Using LordPE to find ImageBase

To open OllyDump, click on plug-in and then OllyDump → Dump Debugged Process. This should look like figure 4.8. In most cases OllyDump will have the correct offset value(specified in the Modify field). If not the value can be entered where –>modify form entry is located. Make sure that the Rebuild Import check box is selected, and click on "Dump" to dump the memory content to file.

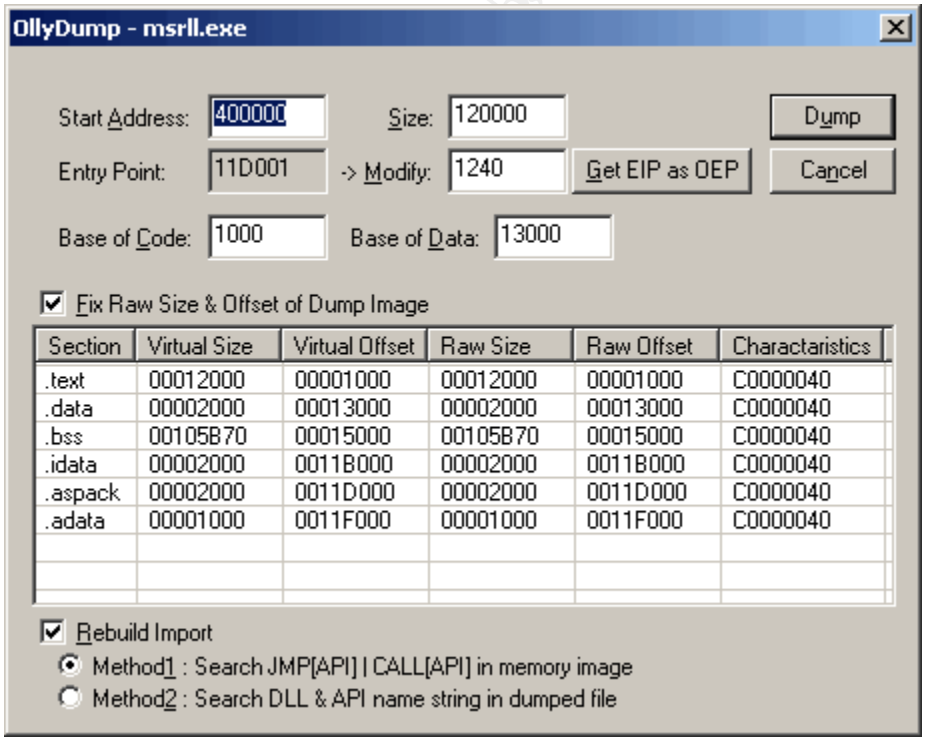

Figure 4.8 OllyDump

© SANS Institute 2005 Author retains full rights.

Finding Authentication Method

The first step to finding the authentication method of the IRC Bot was loading the unpacked version of msrll.exe in to a disassembler called IDApro. Using the strings pane, I located the %s bad pass from \"%s\" @ %s string. It appears that this is the error message that will be displayed by the Bot if a wrong password was entered during authentication. I trace backwards to find out what instruction set calls the subroutine that displayed the error for bad password. I found out that the command "jz short loc_40BC5A" at memory location 0040BBE9 is responsible for calling the subroutine that is called when wrong password is entered. Figure 4.9 shows the assembly code responsible for authentication. Figure 4.9 also shows that the subroutine at memory location 0040BBDF, call Sub_405872, is responsible for comparing the user entered password to the hard coded password. This is a good place to set a breakpoint in OllyDbg.

```
IDA View-A
.text:0040BBC9 loc_40BBC9:                                  ; CODE XREF: sub_40BB6B+26↑j
.text:0040BBC9                 test    byte ptr [ebx+205Ch], 40h
.text:0040BBD0                 jz      loc_40BCA6
.text:0040BBD6                 sub     esp, 8
.text:0040BBD9                 push    offset dword_40BB40
.text:0040BBDE                 push    edx
.text:0040BBDF                 call    sub_405872
.text:0040BBE4                 add     esp, 10h
.text:0040BBE7                 test    eax, eax
.text:0040BBE9                 jz      short loc_40BC5A
.text:0040BBEB                 sub     esp, 0Ch
.text:0040BBEE                 push    33Ch
.text:0040BBF3                 call    malloc
.text:0040BBF8                 mov     [ebp+var_C], eax
.text:0040BBFB                 cld
```

Figure 4.9 - Subroutine that executes when bad password is entered

Next, I opened msrll.exe, the version that was copied to c:\windows\system32\mfm directory using OlldDbg. Located the memory location 0040BBDF and set the breakpoint as shown in figure 4.10. I ran the program by pressing F9 (Run) and waited until the Bot connected to the IRC server and joined the #mils channel. I then joined the #mils channel with nick of zack. I attempted to authenticate to the Bot by entering ?login badpass, but the Bot did not respond. Next I tried ?login zack badpass, still no response from the Bot.

© SANS Institute 2005 Author retains full rights.

```
0040BBDF  . E8 8E9CFFFF   CALL msrll.00405872                  msrll.00405872
0040BBE4  . 83C4 10       ADD ESP,10
0040BBE7  . 85C0          TEST EAX,EAX
0040BBE9  .v74 6F         JE 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
0040BBFB  . FC            CLD
0040BBFC  . B9 CF000000   MOV ECX,0CF
0040BC01  . B8 00000000   MOV EAX,0
0040BC06  . 8B7D F4       MOV EDI,DWORD PTR SS:[EBP-C]
0040BC09  . F3:AB         REP STOS DWORD PTR ES:[EDI]
0040BC0B  . 83C4 08       ADD ESP,8
0040BC0E  . FFB3 64200000 PUSH DWORD PTR DS:[EBX+2064]         src
0040BC14  . FF75 F4       PUSH DWORD PTR SS:[EBP-C]            dest
0040BC17  . E8 94650000   CALL <JMP.&msvcrt._mbscpy>           strcpy
0040BC1C  . 83C4 04       ADD ESP,4
0040BC1F  . FFB3 64200000 PUSH DWORD PTR DS:[EBX+2064]         block
0040BC25  . E8 B6640000   CALL <JMP.&msvcrt.free>              free
0040BC2A  . C783 64200000 MOV DWORD PTR DS:[EBX+2064],0
0040BC34  . 8B45 F4       MOV EAX,DWORD PTR SS:[EBP-C]
0040BC37  . C740 28 21000 MOV DWORD PTR DS:[EAX+28],21
0040BC3E  . 8983 60200000 MOV DWORD PTR DS:[EBX+2060],EAX
0040BC44  . C783 70200000 MOV DWORD PTR DS:[EBX+2070],msrll.00408
0040BC4E  . C783 5C200000 MOV DWORD PTR DS:[EBX+205C],2
0040BC58  .vEB 65         JMP SHORT msrll.0040BCBF
0040BC5A  > 83EC 08       SUB ESP,8
0040BC5D  . FFB3 64200000 PUSH DWORD PTR DS:[EBX+2064]
0040BC63  . 8D83 04200000 LEA EAX,DWORD PTR DS:[EBX+2004]
0040BC69  . 50            PUSH EAX                             Arg5
0040BC6A  . 68 49BB4000   PUSH msrll.0040BB49                  Arg4 = 0040BB49 ASCII "bot.port"
0040BC6F  . 68 52BB4000   PUSH msrll.0040BB52                  Arg3 = 0040BB52 ASCII "%s bad pass from "%s"@%s"
0040BC74  . 6A 00         PUSH 0                               Arg2 = 00000000
0040BC76  . 6A 20         PUSH 20                              Arg1 = 00000020
```

Figure 4.10 – Setting the Breakpoint

In frustration, I went back to the strings to see if I can find any more clues. I noticed that there were two strings that refer to pass. The first one was irc.pass and the send one was dcc.pass. The breakpoint I set dealt with dcc.pass and not the irc.pass. I remembered that during behavioral analysis TDIMon reported that the infected computer was listening on port 2200. I used telnet and tried to connect to the infected machine on port 2200. I was able to receive a command prompt "#:". I used all the authentication methods I mentioned above. Each time I try to authenticate, the code is doing comparison of my password to the hard coded one. Since I set the breakpoint at the compare subroutine, the program would pause while performing the compare. The register pane of OllyDbg would show the two passwords being compared. It seems like I found the authentication method for port 2200. See Figure 4.11. The user entered password is hashed and compared to the hard coded password which is also hashed. The person who wrote this Bot is trying very hard to keep the control of the Bot. While running "John the Ripper" trying to crack the Bot password, I decided to use the method of patching to get control of the IRC Bot.

```
Registers (FPU)
EAX 00000000
ECX 004152EC ASCII ".ZM/Z0hHUSuCAGxXVHD3n."
EDX 0022CD2C ASCII "55isA1ITvamR7bjAdBziX."
EBX 004189A4 msrll.004189A4
ESP 0022CDC0
EBP 0022CDE8
ESI 00000001
EDI 0041A1BC msrll.0041A1BC

EIP 0040BBE4 msrll.0040BBE4

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
```

Figure 4.11 – User entered password compared to hard coded password

The best way to bypass authentication if time is critical is to patch the program.[3] Once the compare subroutine returns the results, it is checked to see if it is a match. If it is not a match, then jump to a subroutine that would display "bad

© SANS Institute 2005 Author retains full rights.

pass". If the compare results in a match, then execute the next instruction which in turn authenticate the user. What can we do to bypass the "JE SHORT msrll.0040BC5A", no matter what the password? A good way of doing that is replacing the "JE SHORT msrll.0040BC5A" instruction with "NOP" (no operation) which does nothing. To replace the instruction with NOP, highlight the instruction and press spacebar. A dialog box will appear with form field. Enter NOP in the form field and press assemble. Once the above change is made, the user can gain control of the Bot whether or not a good password is used. See figure 4.12. After modifying the code I ran the program and connected to the infected host using telnet on port 2200. At the prompt I typed "?id", and I got a response from the infected host with the computer information of the infected host. I now had control over the Bot.

```
0040BBB7  . 83C4 10          ADD ESP,10
0040BBBA  > C783 64200000    MOV DWORD PTR DS:[EBX+2064],0
0040BBC4  .vE9 F9000000      JMP msrll.0040BCC2
0040BBC9  > F683 5C200000    TEST BYTE PTR DS:[EBX+205C],40
0040BBD0  .v0F84 D0000000    JE msrll.0040BCA6
0040BBD6  . 83EC 08          SUB ESP,8
0040BBD9  . 68 40BB4000      PUSH msrll.0040BB40                    2 = 0040BB40 ASCII "dcc.pass"
0040BBDE  . 52               PUSH EDX                               1
0040BBDF  . E8 8E9CFFFF      CALL msrll.00405872                    ll.00405872
0040BBE4  . 83C4 10          ADD ESP,10
0040BBE7  . 85C0             TEST EAX,EAX
0040BBE9  . v74 6F           JE SHORT msrll.0040BC5A
0040BBEB  . 83EC 0C          SUB ESP,0C
0040BBEE  . 68 3C030000      PUSH 33C                               e = 33C (828.)
0040BBF3  . E8 58650000      CALL <JMP.&msvcrt.malloc>              loc
0040BBF8  . 8945 F4          MOV DWORD PTR SS:[EBP-C],EAX
0040BBFB  . FC               CLD
0040BBFC  . B9 CF000000      MOV ECX,0CF
0040BC01  . B8 00000000      MOV EAX,0
0040BC06  . 8B7D F4          MOV EDI,DWORD PTR SS:[EBP-C]
0040BC09  . F3:AB            REP STOS DWORD PTR ES:[EDI]
0040BC0B  . 83C4 08          ADD ESP,8
0040BC0E  . FFB3 64200000    PUSH DWORD PTR DS:[EBX+2064]           src
0040BC14  . FF75 F4          PUSH DWORD PTR SS:[EBP-C]              dest
0040BC17  . E8 94650000      CALL <JMP.&msvcrt._mbscpy>             strcpy
0040BC1C  . 83C4 04          ADD ESP,4
0040BC1F  . FFB3 64200000    PUSH DWORD PTR DS:[EBX+2064]           block
0040BC25  . E8 B6640000      CALL <JMP.&msvcrt.free>                free
0040BC2A  . C783 64200000    MOV DWORD PTR DS:[EBX+2064],0
0040BC34  . 8B45 F4          MOV EAX,DWORD PTR SS:[EBP-C]
0040BC37  . C740 28 21000    MOV DWORD PTR DS:[EAX+28],21
0040BC3E  . 8983 60200000    MOV DWORD PTR DS:[EBX+2060],EAX
0040BC44  . C783 70200000    MOV DWORD PTR DS:[EBX+2070],msrll.00408
0040BC4E  . C783 5C200000    MOV DWORD PTR DS:[EBX+205C],2
0040BC58  .vEB 65            JMP SHORT msrll.0040BCBF
0040BC5A  > 83EC 08          SUB ESP,8
0040BC5D  . FFB3 64200000    PUSH DWORD PTR DS:[EBX+2064]
0040BC63  . 8D83 04200000    LEA EAX,DWORD PTR DS:[EBX+2004]
0040BC69  . 50               PUSH EAX                               Arg5
0040BC6A  . 68 49BB4000      PUSH msrll.0040BB49                    Arg4 = 0040BB49 ASCII "bot.port"
0040BC6F  . 68 52BB4000      PUSH msrll.0040BB52                    Arg3 = 0040BB52 ASCII "%s bad pass from "%s"@%s"
0040BC74  . 6A 00            PUSH 0                                 Arg2 = 00000000
0040BC76  . 6A 20            PUSH 20                                Arg1 = 00000020
0040BC78  . E8 0CE9FFFF      CALL msrll.0040A589                    msrll.0040A589
```

Figure 4.12 - Use NOP to bypass authentication

Analysis Wrap-UP

In the previous part control of the malicious specimen was achieved. The specimen was identified as an IRC Bot. The owner of the Bot designed multiple way of controlling the Bot. The two methods that seemed very clear are via telnet and IRC client. I spent three weeks analyzing the assembly code to find a way to patch and get authenticated via the IRC channel with no success. I also tried to crack the hashed password using John the Ripper without any success. As a last resort I downloaded Reverse-compiling tool called REC to reverse the unpacked version of the executable to C code. The REC tool can be downloaded at www.backerstreet.com/rec/rec.htm. Although the code was

© SANS Institute 2005 Author retains full rights.

human understandable, the authentication method used by the IRC channel was complicated. I often broke the code trying to patch it and gain access. The only way I could successfully control the Bot was through telnet to port 2200.

The infected host listens on port 2200 as discussed in the behavioral analysis section. The Bot seems to have multiple powerful commands that the Bot manager could use. I picked some of the command to discuss in detail. The ?clone command seems to duplicate itself. After using the ?clone command as shown in figure 5.1, I executed netstat –an on the infected host and I was able to see additional host listing on port 2200. I tried the ?clone command with different IP address and port, but it did not work. The Bot just ignored the command.

```
?clone
usage ?clone: server[:port] amount
?clone 192.168.1.1:2200 1
*** bot.port: connect from 192.168.1.1
?clone 192.168.1.1:2200 5
*** bot.port: connect from 192.168.1.1
*** bot.port: connect from 192.168.1.1
*** bot.port: connect from 192.168.1.1
*** bot.port: connect from 192.168.1.1
*** bot.port: connect from 192.168.1.1
```

Figure 5.1 – ?clone command

The ?set command is very useful command. It allows the Bot manager to change which binary to run, the directory in which the executable should run from, the Bot port, IRC servers to connect to, the IRC channel to connect to, and passwords. I was able to change the password of one of the authentication methods by issuing “set pass zack” command to zack as seen in figure 5.2.

```
?set pass zack
set
(?login zack) set
?set
set jtr.bin msrll.exe
set jtr.home mfm
set bot.port 2200
set jtr.id run5
set irc.quit
set servers
collective7.zxy0.com,collective7.zxy0.com:9999!,collective7.zxy0.com:
8080
set irc.chan #mils
set pass zack
set dcc.pass $1$KZLPLKDf$55isA1ITvamR7bjAdBziX.
```

Figure 5.2 - ?set command

The ?copy command works just like the DOS copy command and Linux cp command. By issuing the command “?copy Source Destination” one can copy a

© SANS Institute 2005 Author retains full rights.

file from source to destination. A transcript of the ?copy command is listed in figure 5.3. The ?move command works the same way as the ?copy command.

```
?copy
?copy jtram.conf jtram2.conf
jtram.conf copied to jtram2.conf
?dir
12/21/2004  05:30   <DIR>           .
12/21/2004  05:30   <DIR>           ..
12/21/2004  05:26             1060  jtram.conf
12/21/2004  05:26             1060  jtram2.conf
11/20/2004  17:58          1182720  msrll.exe
```

Figure 5.3 - ?copy command

The ?ps command works just as Linux ps command. It lists all the currently running processes with their process ids. One can issue the ?kill <pid> to kill any process identified by pid. Figure 5.4 shows example of the ?ps and ?kill commands.

```
?ps
 0     [System Process]
 4     System
 536   smss.exe
 600   csrss.exe
 624   winlogon.exe
 668   services.exe
 680   lsass.exe
 832   svchost.exe
 928   svchost.exe
 1024  svchost.exe

.
Snip
.

 2748  msrll.exe
 3964  cmd.exe
 3160  notepad.exe
 1460  NOTEPAD.EXE
 3520  sol.exe
 2484  mspaint.exe
?kill 3520
pid 3520 killed
sol.exe exited with code 0
```

Figure 5.4 – ?ps and ?kill commands

The ?jolt command and ?smurf commands are the purpose of the Bot. I believe that the developer's purpose for the Bot was to perform DDOS to targeted host. I tried the ?jolt command to see if in fact I cause a DOS to one of my lab computers (REM2). I used the command "?jolt 192.168.1.2 10 1" while using snort to log the traffic, and I was able to see a 200,000 bytes of data logged in

© SANS Institute 2005 Author retains full rights.

ten seconds. Figure 5.5 shows the ?jolt command I used and Figure 5.6 shows an excerpt of the snort log.

```
?jolt
?jolt <ip> <duration> <delay>
?jolt 192.168.1.2 10 1
jolt2: done
?jolt 192.168.1.2 60 1
?jolt2: done

?smurf
?smurf <ip> <p size> <duration> <delay>
?smurf 192.168.1.2 20 10 1
smurf done
```

Figure 5.5 - ?jolt and ?smurf commands

```
12/20-21:35:06.168094 192.168.1.4:2096 -> 192.168.1.1:2200
TCP TTL:64 TOS:0x10 ID:45556 IpLen:20 DgmLen:76 DF
***AP*** Seq: 0x2B25B249  Ack: 0xA4A8AC5D  Win: 0x25B0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 176010291 17599800
3F 6A 6F 6C 74 20 31 39 32 2E 31 36 38 2E 31 2E  ?jolt 192.168.1.
32 20 31 30 20 32 0D 0A                          2 10 2..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

12/20-21:35:06.170851 192.168.1.1 -> 192.168.1.2
ICMP TTL:255 TOS:0x0 ID:256 IpLen:20 DgmLen:28
Frag Offset: 0x1FFE   Frag Size: 0x0008
08 00 F7 FF 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

12/20-21:35:06.174246 192.168.1.1 -> 192.168.1.2
ICMP TTL:255 TOS:0x0 ID:256 IpLen:20 DgmLen:28
Frag Offset: 0x1FFE   Frag Size: 0x0008
08 00 F7 FF 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

12/20-21:35:06.184242 192.168.1.1 -> 192.168.1.2
ICMP TTL:255 TOS:0x0 ID:256 IpLen:20 DgmLen:28
Frag Offset: 0x1FFE   Frag Size: 0x0008
08 00 F7 FF 00 00 00 00                          ........

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

12/20-21:35:06.194273 192.168.1.1 -> 192.168.1.2
ICMP TTL:255 TOS:0x0 ID:256 IpLen:20 DgmLen:28
Frag Offset: 0x1FFE   Frag Size: 0x0008
08 00 F7 FF 00 00 00 00                          ........
```

Figure 5.6 – Snort log generated by the ?jolt command

© SANS Institute 2005 Author retains full rights.

The commands ?run and ?exec ran any given executable as a process. See Figure 5.7. Although, the windows task manager shows that the programs are running, there was no GUI available for the notepad and solitaire.

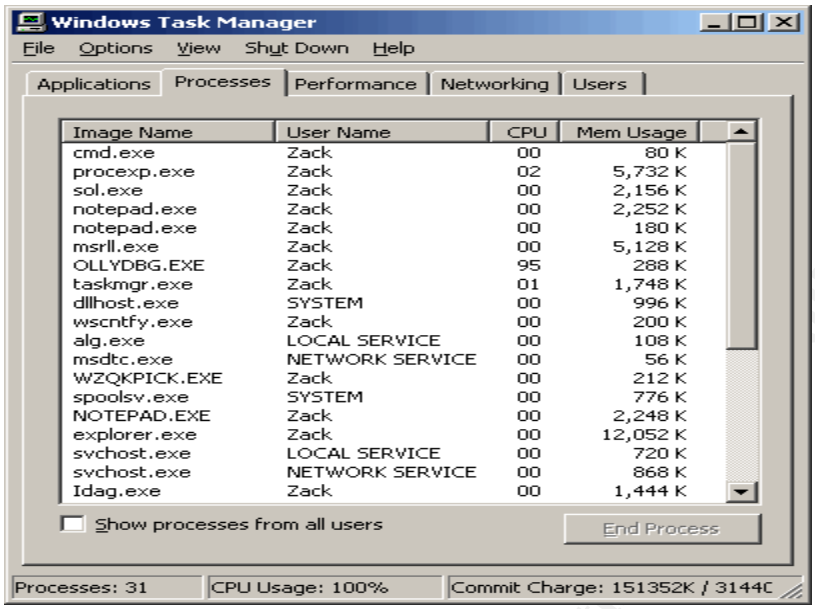

Figure 5.7 – Windows Task manager

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

```
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
#:?login zack
zack

?uptime
sys: 20d 07h 53m 04s bot: 32m 13s

?insmod
?insmod: <mod name>

?rmmod
?rmmod: <mod name>

?ismod
(?login zack) ?ismod

?clones
?clones: [NETWORK|all] <die|join|part|raw|msg> <"parm"> ...

?status
service:N user:Zack inet connection:Y contype: Lan  reboot privs:Y

?jump

?nick
_Set an irc sock to preform ?nick command on_
  Type _.sklist_ to view current sockets, then _.dccsk_ <#>

?echo
(null)

?hush
_Set an irc sock to preform ?hush command on_
  Type _.sklist_ to view current sockets, then _.dccsk_ <#>

?wget
?wget jtram.conf
no file name in jtram.conf

?join
_Set an irc sock to preform ?join command on_
  Type _.sklist_ to view current sockets, then _.dccsk_ <#>

?akick

?part
_Set an irc sock to preform ?part command on_
  Type _.sklist_ to view current sockets, then _.dccsk_ <#>

?dump

?md5p
?md5p <pass> <salt>
```

Figure 5.8 - More Bot commands

© SANS Institute 2005 Author retains full rights.

```
?free

?sums
jtram.conf       05400996e509679a5575e4175140e569
jtram2.conf      05400996e509679a5575e4175140e569
msrll.exe        298d1fbc2781b288913f8bf5a43f88f7

?mkdir zack
zack created

?move jtram2.conf zack\jtram2.conf
?move jtram2.conf to zack\jtram2.conf Ok
?dir
12/21/2004  05:34   <DIR>           .
12/21/2004  05:34   <DIR>           ..
12/21/2004  05:26             1060  jtram.conf
11/20/2004  17:58          1182720  msrll.exe
12/21/2004  05:34   <DIR>           zack
?cd zack
C:\WINDOWS\system32\mfm\zack

?cd ..
C:\WINDOWS\system32\mfm

?cd zack
C:\WINDOWS\system32\mfm\zack

?del jtram2.conf
jtram2.conf removed

?cd ..
C:\WINDOWS\system32\mfm
?rmdir zack
?rmdir zack :ok

?exec
?exec notepad
?exec c:\windows\notepad.exe
?exec sol.exe

?sklist
  #1 [fd:356] collective7.zxy0.com:6667 [IRC _IATH_ IREG ICON RNL ]
last:14
   |\=> [n:MBUGOrUfBSrQ fh:MBUGOrUfBSrQ!HTIETObi@192.168.1.1]
(UnderNet)
   |
   |---[#mils] (2) +
   |     |-[MBUGOrUfBSrQ] [192.168.1.1]
   |     |-[@zack] [192.168.1.4]
  #2 [fd:404] 192.168.1.4:0 [DCC ICON RNL ] last:0
   |=> (?login zack) (00000021)
  #3 [fd:1396] 192.168.1.1:2200 [IRC CLON ICON RNL ] last:267
  #4 [fd:1384] 192.168.1.1:0 [DCC ICON RNL ] last:2516
   |=> (USER titBw localhost 0 :TKvyM) (00000021)
```

Figure 5.9 – More Bot Commands

© SANS Institute 2005 Author retains full rights.

The ?crash command crashed the infected machine, and I lost the telnet connection to the Bot. I had to restart the infected host to gain control again. The ?reboot command rebooted the infected host and displayed the later! And Connection closed by foreign host messages as seen in figure 5.10.

```
?unset

?uattr
_Set an irc sock to preform ?uattr command on_
  Type _.sklist_ to view current sockets, then _.dccsk_ <#>
?dccsk
usage ?dccsk <socks #>

?killsk
unable to close socket 4018072

?ping 192.168.1.2
?ping <ip> <total secs> <p size> <delay> [port]
?ping 192.168.1.2 5 10 2
finished 192.168.1.2

?crash
?uptime

?reboot
later!
Connection closed by foreign host.
[root@REM4 root]#
```

Figure 5.10 – More Bot Commands

© SANS Institute 2005

© SANS Institute 2005 Author retains full rights.

References

[1] ZDNet Downloads. OllyDbg 1.09d. CNET Networks, Inc. 2004. URL: http://downloads-zdnet.com.com/OllyDbg/3000-2383_2-10242634.html?tag=lst-0-1

[2] Mr-Geek, How to unpack Aspack using Ollydbg. February 2004, http://biw.rult.at/tuts/mupaspack.rar

[3] Sans Institute and Lenny Zeltser, Reverse-Engineering Malware: Tools and Techniqes. 2004

Software Resources

Windows XP SP2
www.microsoft.com.

Windows 2000
www.microsoft.com.

Snort
www.snort.org.

Undernet-IRCU2
http://prdownloads.sourceforge.net/undernet-ircu/ircu2.10.11.07.tar.gz?download.

Ollydbg
"http://www.downloads-zdnet.com.com/3001-2383_2-10242634.html.

Regmon
http://www.sysinternals.com/ntw2k/source/regmon.shtml.

Filemon
http://www.sysinternals.com/ntw2k/source/filemon.shtml.

IDAPro
http://www.datarescue.be/downloaddemo.htm.

TDIMon
http://www.sysinternals.com/ntw2k/freeware/tdimon.shtml.

LordPE
http://www.softpedia.com/progDownload/LordPE-Download-29.html.

© SANS Institute 2005 Author retains full rights.

RegShot
http://regshot.ist.md/

MD5sum
http://www.weihenstephan.de/~syring/win32/UnxUtils.html.

PEInfo
Could not find any reliable source other than the CD supplied in class.

BinText
http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/freetools.htm

ASpackdie
http://www.woodmann.com/crackz/Unpackers/Aspdie.zip.

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

Last Updated: May 1st, 2017

Upcoming Training

Click Here to {Get CERTIFIED!}

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
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 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
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 OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced