Global Information Assurance Certification Paper

GIAC Reverse Engineering Malware
GREM Practical Assignment
Version 1.0

Malware: A Look at Reverse Engineering MSRLL.EXE

Lorna J. Hutcheson
Orlando SANS 2004

TABLE OF CONTENTS

Abstract

This practical will cover the reverse engineering of a malicious piece of code given to us to analyze. It will use the procedures taught in class and will follow the outline of the Table of Contents listed above.

Part 1: Laboratory Setup

Introduction

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

Hardware Setup

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

Networking Setup

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

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

Software Resources

VMWare Workstation 4.5.1

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

Linux Redhat

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

Windows 98

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

Windows 2000

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

PEInfo

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

Ollydbg

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

OEP manually through Ollydbg will be described later in detail. This can be found at http://home.t-online.de/home/Ollydbg/.

Ethereal

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

SNORT

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

Filemon

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

RegMon

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

TDIMon

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

LordPE

LordPE is a tool that was used during the analysis without useful results. The tool is intended to allow a look at processes that are running and how it interacts. It also allows to you dump a process from memory. This was used after launching msrll.exe to attempt to get an unpacked version of the code. However, the code that was obtained was still not readable so another method was used which was ollydbg. It can be found at http://mitglied.lycos.de/yoda2k/LordPE/info.htm.

Notepad

Notepad is a built in text editor that comes with Windows systems. It is used as a method to safely view files without launching them. This was used in many different forms throughout the analysis to look at files and view output from tools.

Regshot

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

Netcat

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

Process Explorer

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

PSkill

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

MD5sum

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

Part 2: Properties of the Malware Specimon

Type of File

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

Size of the File

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

MD5 Hash of the File

The MD5 hash of the file in its packed state is 84acfe96a98590813413122c12c11aaa. This was determined by using a command line tool called md5sum.exe. This tool is used by issueing the following command at the commandline "md5sum.exe msrll.exe". I placed MD5sum.exe in the directory where the malware was located to avoid modifying the path. The MD5 hash of the file was also taken after it was launched with the following results:
\84acfe96a98590813413122c12c11aaa *C:\WINNT\system32\mfm\msrll.exe.
This was done to ensure no modification of the file occurred after it was launched.

Operating System it Runs on

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

Strings Embedded into it

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

!This program cannot be run in DOS mode	?msg	GetExitCodeProcess	StartServiceCtrlDispatcherA
.idata	?kb	GetFileSize	kernel32.dll
.aspack	?sklist	GetFullPathNameA	AddAtomA
.adata	?unset	GetLastError	CloseHandle
.newIID	?uattr	GetModuleFileNameA	CopyFileA
?insmod	?dccsk	GetModuleHandleA	CreateDirectoryA
?rmmod	?con	GetProcAddress	CreateFileA

?lsmod	?killsk	GetStartupInfoA	CreateMutexA
%s: <mod name>	VERSION*	GetSystemDirectoryA	CreatePipe
%s: mod list full	%ud %02uh %02um %02us	GetSystemInfo	CreateProcessA
%s: err: %u	%02uh %02um %02us	GetTempPathA	CreateToolhelp32Snapshot
mod_init	%um %02us	GetTickCount	DeleteFileA
mod_free	jtram.conf	GetVersionExA	DuplicateHandle
%s: cannot init %s	DiCHFc2ioiVmb3cb4zZ7zWZH1oM=	GlobalMemoryStatus	RtlEnterCriticalSection
%s: %s loaded (%u)	conf_dump: wrote %u lines	InitializeCriticalSection	ExitProcess
%s: mod allready loaded	get of %s incomplete at %u bytes	IsBadReadPtr	ExitThread
%s:%s err %u	dcc_wait: get of %s from %s timed out	LeaveCriticalSection	FileTimeToSystemTime
%s:%s not found	dcc_wait: closing [#%u] %s:%u (%s)	LoadLibraryA	FindAtomA
%s: unloading %s	%4s #%.2u %s %ucps %u%% [sk#%u] %s	MoveFileA	FindClose
[%u]: %s hinst:%x	%u Send(s) %u Get(s) (%u transfer(s) total) UP:%ucps DOWN:%ucps Total:%ucps	OpenProcess	FindFirstFileA
unloading %s	send of %s incomplete at %u bytes	PeekNamedPipe	FindNextFileA
%s: invalid_addr: %s	send of %s completed (%u bytes), %u seconds %u cps	Process32First	FreeLibrary
%s%s [port]	cant open %s (err:%u) pwd:{%s}	Process32Next	GetAtomNameA
finished %s	DCC SEND %s %u %u %u	QueryPerformanceFrequency	GetCommandLineA
%s <ip> <port> <t_time> <delay>	%s exited with code %u	ReadFile	GetCurrentDirectoryA
sockopt: %u	%s: %s	ReleaseMutex	GetCurrentProcess
sendto err: %u	exec: Error:%u pwd:%s cmd:%s	RemoveDirectoryA	GetCurrentThreadId
sockraw: %u	dcc.pass	SetConsoleCtrlHandler	GetExitCodeProcess
syn: done	bot.port	SetCurrentDirectoryA	GetFileSize
%s <ip> <duration> <delay>	%s bad pass from "%s"@%s	SetFilePointer	GetFullPathNameA
sendto: %u	%s: connect from %s	SetUnhandledExceptionFilter	GetLastError
jolt2: done	jtr.bin	TerminateProcess	GetModuleFileNameA
%s <ip> <p size> duration> <delay>	msrll.exe	WaitForSingleObject	GetModuleHandleA
Err: %u	jtr.home	WriteFile	The procedure entry point %s could not be located in the dynamic link library %s
smurf done	jtr.id	_strdup	The ordinal %u could not be located in the dynamic link library %s
&err: %u	irc.quit	_stricmp	(08@P`p
PONG :%s	Servers	__getmainargs	kernel32.dll
%s!%s@%s	collective7.zxy0.com,collective7.zxy0.com :9999!,collective7.zxy0.com:8080	__p__environ	GetProcAddress
SVh=+@	irc.chan	__p__fmode	GetModuleHandleA
irc.nick	1KZLPLKDf$W8kl8Jr1X8DOHZsmIp9qq0	__set_app_type	LoadLibraryA
NICK %s	1KZLPLKDf$55isA1ITvamR7bjAdBziX.	_beginthread	advapi32.dll
NETWORK=	SSL_get_error	_cexit	msvcrt.dll
irc.pre	SSL_load_error_strings	_errno	msvcrt.dll
__%s__	SSL_library_init	_fileno	shell32.dll
__%s___	SSLv3_client_method	_onexit	user32.dll
NICK %s	SSL_set_connect_state	_setmode	version.dll
irc.chan	SSL_CTX_new	_vsnprintf	wininet.dll
WHO %s	SSL_new	atexit	ws2_32.dll
PPhV,@	SSL_set_fd	fclose	AdjustTokenPrivileges
USERHOST %s	SSL_connect	fflush	__getmainargs
logged into %s(%s) as %s	SSL_write	fprintf	ShellExecuteA
<$hE:@	SSL_read	fwrite	DispatchMessageA
nick.pre	SSL_shutdown	malloc	GetFileVersionInfoA
%s-%04u	SSL_free	memcpy	InternetCloseHandle
irc.user	SSL_CTX_free	memset	WSAGetLastError
irc.usereal	kernel32.dll	printf	advapi32.dll
irc.real	QueryPerformanceCounter	realloc	AdjustTokenPrivileges
irc.pass	QueryPerformanceFrequency	setvbuf	CloseServiceHandle
tsend(): connection to %s:%u failed	RegisterServiceProcess	signal	CreateServiceA
USER %s localhost 0 :%s	jtram.conf	sprintf	CryptAcquireContextA

NICK %s	irc.user	stricmp	CryptGenRandom
PRIVMSG	%s : USERID : UNIX : %s	strcat	CryptReleaseContext
trecv(): Disconnected from %s err:%u	QUIT :FUCK %u	strchr	GetUserNameA
NOTICE	Killed!? Arrg! [%u]	strcmp	LookupPrivilegeValueA
%s %s :%s	QUIT :%s	strcpy	OpenProcessToken
MODE %s -o+b %s *@%s	SeShutdownPrivilege	strerror	OpenSCManagerA
C'PSWh	%s\%s\%s	strncat	RegCloseKey
MODE %s -bo %s %s	Rll enhanced drive	strncmp	RegCreateKeyExA
%s.key	software\microsoft\windows\currentversion\run	strncpy	RegSetValueExA
sk#%u %s is dead!	/d "%s"	strstr	RegisterServiceCtrlHandlerA
s_check: %s dead? pinging...	./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz	toupper	SetServiceStatus
PING :ok	usage %s: server[:port] amount	ShellExecuteA	GetProcAddress
s_check: send error to %s disconnecting	%s: %s	DispatchMessageA	GetStartupInfoA
expect the worst	%s %s %s <PARAM>	ExitWindowsEx	GetSystemDirectoryA
s_check: killing socket %s	%s: [NETWORK|all] %s <"parm"> ...	GetMessageA	GetSystemInfo
irc.knick	USER %s localhost 0 :%s	PeekMessageA	GetTempPathA
jtr.%u%s.iso	NICK %s	GetFileVersionInfoA	GetTickCount
ison %s	buf != NULL	VerQueryValueA	GetVersionExA
servers	hash != NULL	InternetCloseHandle	GlobalMemoryStatus
s_check: trying %s	message digest	InternetGetConnectedState	InitializeCriticalSection
uYVh|K@	abcdefghijklmnopqrstuvwxyz	InternetOpenA	IsBadReadPtr
%s.mode	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789	InternetOpenUrlA	RtlLeaveCriticalSection
MODE %s %s	1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890	InternetReadFile	LoadLibraryA
PShZP@	sprng.c	WSAGetLastError	MoveFileA
mode %s +o %s	buf != NULL	WSASocketA	OpenProcess
mode %s +b %s %s	skey != NULL	WSAStartup	PeekNamedPipe
KICK %s %s	key != NULL	__WSAFDIsSet	Process32First
irc.pre	ct != NULL	accept	Process32Next
Set an irc sock to preform %s command on Type	pt != NULL	closesocket	QueryPerformanceFrequency
%csklist	#4EVgx	connect	ReadFile
to view current sockets, then	$5FWhy	gethostbyaddr	ReleaseMutex
%cdccsk	#4EVgx	gethostbyname	RemoveDirectoryA
%s: dll loaded	$5FWhy	gethostname	SetConsoleCtrlHandler
%s: %d	#4EVgx	getsockname	SetCurrentDirectoryA
said %s to %s	$5FWhy	inet_addr	SetFilePointer
usage: %s <target> "text"	desired_keysize != NULL	inet_ntoa	SetUnhandledExceptionFilter
%s not on %s	ctr != NULL	ioctlsocket	TerminateProcess
usage: %s <nick> <chan>	key != NULL	listen	WaitForSingleObject
%s logged in	count != NULL	select	WriteFile
sys: %s bot: %s	ct != NULL	sendto	msvcrt.dll
preformance counter not avail	pt != NULL	setsockopt	_mbsdup
usage: %s <cmd>	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/	shutdown	_strcmpi
%s free'd	?456789:;<=	socket	msvcrt.dll
unable to free %s	!"#$%&'()*+,-./0123	ADVAPI32.DLL	__getmainargs
later!	base64.c	KERNEL32.dll	__p__environ
unable to %s errno:%u	outlen != NULL	msvcrt.dll	__p__fmode
service:%c user:%s inet connection:%c contype:%s reboot privs:%c	out != NULL	msvcrt.dll	__set_app_type
%-5u %s	in != NULL	SHELL32.DLL	_beginthread
%s: %s	_ARGCHK '%s' failure on line %d of file %s	USER32.dll	_cexit
%s: somefile	crypt.c	VERSION.dll	_errno
host: %s ip: %s	name != NULL	WININET.DLL	_fileno

capGetDriverDescriptionA	cipher != NULL	WS2_32.DLL	_onexit
cpus:%u	hash != NULL	VirtualAlloc	_setmode
WIN%s (u:%s)%s%s mem:(%u/%u) %u%% %s %s	prng != NULL	VirtualFree	_vsnprintf
%s: %s (%u)	LibTomCrypt 0.83	kernel32.dll	atexit
%s bad args	Endianess: little (32-bit words)	ExitProcess	fclose
%s[%u] %s	Clean stack: disabled	user32.dll	fflush
%s removed	Ciphers built-in:	MessageBoxA	fprintf
couldnt find %s	Blowfish	wsprintfA	fwrite
%s added	RC2	LOADER ERROR	malloc
%s allready in list	RC5	The procedure entry point %s could not be located in the dynamic link library %s	memcpy
usage: %s +/- <host>	RC6	The ordinal %u could not be located in the dynamic link library %s	memset
jtram.conf	Serpent	(08@P`p	printf
%s /t %s	Safer+	kernel32.dll	realloc
jtr.home	Safer	GetProcAddress	setvbuf
%s: possibly failed: code %u	Rijndael	GetModuleHandleA	signal
%s: possibly failed	XTEA	LoadLibraryA	sprintf
%s: exec of %s failed err: %u	Twofish	advapi32.dll	strcat
jtr.id	CAST5	msvcrt.dll	strchr
%s: <url> <id>	Noekeon	msvcrt.dll	strcmp
#%u [fd:%u] %s:%u [%s%s] last:%u	Hashes built-in:	shell32.dll	strcpy
|\=> [n:%s fh:%s] (%s)	SHA-512	user32.dll	strerror
|---[%s] (%u) %s	SHA-384	version.dll	strncat
| |-[%s%s] [%s]	SHA-256	wininet.dll	strncmp
|=> (%s) (%.8x)	TIGER	ws2_32.dll	strncpy
B$PRhco@	SHA1	AdjustTokenPrivileges	strstr
%s <pass> <salt>	MD5	__getmainargs	toupper
%s <nick> <chan>	MD4	ShellExecuteA	shell32.dll
PING %s	MD2	DispatchMessageA	ShellExecuteA
mIRC v6.12 Khaled Mardam-Bey	Block Chaining Modes:	GetFileVersionInfoA	USER32.dll
VERSION %s	CFB	InternetCloseHandle	DispatchMessageA
dcc.pass	OFB	WSAGetLastError	ExitWindowsEx
temp add %s	CTR	advapi32.dll	GetMessageA
%s%u-%s	Yarrow	AdjustTokenPrivileges	PeekMessageA
%s opened (%u)	SPRNG	CloseServiceHandle	version.dll
%u bytes from %s in %u seconds saved to %s	RC4	CreateServiceA	GetFileVersionInfoA
(%s %s): incomplete! %u bytes	PK Algs:	CryptAcquireContextA	VerQueryValueA
couldnt open %s err:%u	RSA	CryptGenRandom	wininet.dll
(%s) %s: %s	ECC	CryptReleaseContext	InternetCloseHandle
(%s) urlopen failed	Compiler:	GetUserNameA	InternetGetConnectedState
(%s): inetopen failed	WIN32 platform detected.	LookupPrivilegeValueA	InternetOpenA
no file name in %s	GCC compiler detected.	OpenProcessToken	InternetOpenUrlA
%s created	Various others: BASE64 MPI HMAC	OpenSCManagerA	InternetReadFile
%s %s to %s Ok	/dev/random	RegCloseKey	ws2_32.dll
%0.2u/%0.2u/%0.2u %0.2u:%0.2u %15s %s	Microsoft Base Cryptographic Provider v1.0	RegCreateKeyExA	WSAGetLastError
%s (err: %u)	XTEA	RegSetValueExA	WSASocketA
err: %u	bits.c	RegisterServiceCtrlHandlerA	WSAStartup
%s %s :ok	buf != NULL	SetServiceStatus	__WSAFDIsSet
unable to %s %s (err: %u)	prng != NULL	StartServiceCtrlDispatcherA	accept
%-16s %s	<"tx< tf<	kernel32.dll	closesocket
%-16s (%u.%u.%u.%u)	-LIBGCCW32-EH-SJLJ-GTHR-MINGW32	AddAtomA	connect
[%s][%s] %s	<ip> <total secs> <p size> <delay>	CloseHandle	gethostbyaddr
closing %u [%s:%u]	modem	CopyFileA	gethostbyname
unable to close socket %u	Proxy	CreateDirectoryA	gethostname
using sock #%u %s:%u (%s)	none	CreateFileA	getsockname
Invalid sock	m220 1.0 #2730 Mar 16 11:47:38 2004	CreateMutexA	inet_addr
usage %s <socks #>	unable to %s %s (err: %u)	CreatePipe	

leaves %s	unable to kill %s (%u)	CreateProcessA	inet_ntoa
:0 * * :%s	%s killed (pid:%u)	CreateToolhelp32Snapshot	ioctlsocket
joins: %s	AVICAP32.dll	DeleteFileA	listen
ACCEPT	unable to kill %u (%u)	DuplicateHandle	select
resume	pid %u killed	RtlEnterCriticalSection	sendto
err: %u	error!	ExitProcess	setsockopt
DCC ACCEPT %s %s %s	ran ok	ExitThread	shutdown
dcc_resume: cant find port %s	MODE %s +o %s	FileTimeToSystemTime	socket
dcc.dir	set %s %s	FindAtomA	
%s\%s\%s\%s	Mozilla/4.0	FindClose	
unable to open (%s): %u	Accept: */*	FindFirstFileA	
resuming dcc from %s to %s	<DIR>	FindNextFileA	
DCC RESUME %s %s %u	Could not copy %s to %s	FreeLibrary	
?si	%s copied to %s	GetAtomNameA	
?ssl	0123456789abcdef	GetCommandLineA	
?clone	%s unset	GetCurrentDirectoryA	
?clones	unable to unset %s	GetCurrentProcess	
?login	(%s) %s	GetCurrentThreadId	
?uptime	libssl32.dll	GetExitCodeProcess	
?reboot	libeay32.dll	GetFileSize	
?status	<die|join|part|raw|msg>	GetFullPathNameA	
?jump	AdjustTokenPrivileges	GetLastError	
?nick	CloseServiceHandle	GetModuleFileNameA	
?echo	CreateServiceA	GetModuleHandleA	
?hush	CryptAcquireContextA	The procedure entry point %s could not be located in the dynamic link library %s	
?wget	CryptGenRandom	The ordinal %u could not be located in the dynamic link library %s	
?join	CryptReleaseContext	(08@P`p	
?op	GetUserNameA	kernel32.dll	
?aop	LookupPrivilegeValueA	GetProcAddress	
?akick	OpenProcessToken	GetModuleHandleA	
?part	OpenSCManagerA	LoadLibraryA	
?dump	RegCloseKey	advapi32.dll	
?set	RegCreateKeyExA	msvcrt.dll	
?die	RegSetValueExA	msvcrt.dll	
?md5p	RegisterServiceCtrlHandlerA	shell32.dll	
?free	SetServiceStatus	user32.dll	
?raw	StartServiceCtrlDispatcherA	version.dll	
?update	AddAtomA	wininet.dll	
?hostname	CloseHandle	ws2_32.dll	
?fif	CopyFileA	AdjustTokenPrivileges	
?!fif	CreateDirectoryA	__getmainargs	
?del	CreateFileA	ShellExecuteA	
?pwd	CreateMutexA	DispatchMessageA	
?play	CreatePipe	GetFileVersionInfoA	
?copy	CreateProcessA	InternetCloseHandle	
?move	CreateToolhelp32Snapshot	WSAGetLastError	
?dir	DeleteFileA	advapi32.dll	
?sums	DuplicateHandle	AdjustTokenPrivileges	
?ls	EnterCriticalSection	CloseServiceHandle	
?cd	ExitProcess	CreateServiceA	
?rmdir	FileTimeToSystemTime	CryptAcquireContextA	
?mkdir	FindAtomA	CryptGenRandom	
?run	FindClose	CryptReleaseContext	
?exec	FindFirstFileA	GetUserNameA	
?ps	FindNextFileA	LookupPrivilegeValueA	
?kill	FreeLibrary	OpenProcessToken	
?killall	GetAtomNameA	OpenSCManagerA	
?crash	GetCommandLineA	RegCloseKey	
?dcc	GetCurrentDirectoryA	RegCreateKeyExA	
?get	GetCurrentProcess	RegSetValueExA	
?say	GetCurrentThreadId	RegisterServiceCtrlHandlerA	
		SetServiceStatus	

The first step in starting to analyze the malware was to look at the strings for any clues about the capabilities of the malware and what it might do. One of the first things that popped out were references to “IRC” in some of the strings. So my initial assumption was it was an IRC bot of some sort. In light of this, I looked for strings that might be used as IRC commands. I found several that were proceeded by a ? and looked like good candidates such as “?login”, “?uptime”, “?join”, “?die” etc. Other strings that caught my attention were things like “irc.pass” and “dcc.pass” as possible ways for IRC password validation. The string “bot.port” seemed to indicate either a port that might be listening on the infected machine or that you could use it to specifiy the port that bot was connecting on so it was something to look at later. I also noticed the strings “Ciphers built-in” and “Hashes built-in” so I figured I would have to look for their usage and might make password identification difficult. “Mozilla/4.0” and strings such as “InternetCloseHandle” and “InternetOpenUrlA” might mean web server of some sort was being used. Also found were references to one server (collective7.zxy0.com) but two different ports were specified: 9999 and 8080. The other didn’t specifiy a port, so it looked like it might be a webserver connection. There were two strings that seemed similar that I found that I thought might be useful “1KZLPLKDf$W8kl8Jr1X8DOHZsmIp9qq0” and “1KZLPLKDf$55isA1ITvamR7bjAdBziX”. I did not know what they were for, but I wanted to explore them further. Also mentioned were “Ping” “UDP” “Smurf”, “Jolt2”, and “SYN”. This was evident that it possessed to capability to do denial of service (DOS) attacks.

Part 3: Behavioral Analysis

Behavior Before Code Analysis

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

Monitoring of File System Access

Filemon showed some very key things that are listed and discussed below.

We see that the malware created a directory named mfm in the system32 folder
177 3:08:45 PM msrll.exe:252CREATE C:\WINNT\system32\mfm
SUCCESS Options: Create Directory Access: All

Next the malware copied itself to the directory, which was verified via an MD5 hash to be the same as the orginal file.
224 3:08:45 PM msrll.exe:252CREATE
C:\WINNT\system32\mfm\msrll.exe SUCCESS Options: OverwriteIf Sequential Access: AllMonitoring

Then we see the malware creates a file calld jtram.conf
869 3:09:04 PM msrll.exe:1044 CREATE
C:\WINNT\system32\mfm\jtram.conf SUCCESS Options: OverwriteIf Access: All

Monitoring registry/configuration Access

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

Regmon shows us that it is putting its self in to run as a service.
1462 101.79209788 SERVICES.EXE:212 CreateKey
HKLM\System\CurrentControlSet\Services\mfm SUCCESS Key: 0xE13704E0

Regmon also shows that several keys were created dealing with Cryptograph such as the one below. Probably to ensure that it had the crypto capabilities that it was wanting.

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

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

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

Monitoring/Redirecting Network Connections

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

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

Monitoring Processes on the System

I used Task Manager to check the processes running on the system and I found that msrll.exe was indeed running now as a process. I tried to kill it from the process list and found that I was denied. I then went to look at the services and

found I could not kill it from there either. I opened Sysinternals Process Explorer and found it there. I was able to kill it using Process Explorer or the command line tool PSkill also from Sysinternals.

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

Behavior After Code Analysis

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

COMMAND	RESULTS
?si	This command gives the system information about the computer that you connect to via telnet. "WIN2k (u:Administrator) mem☹176/255) 30% GenuineIntel Intel(R) Celeron(R) CPU 2.70 GHz"
?ssl	This returned "?ssl: -1" I believe that this tells you whether SSL is being used or not
?clone	It showed the following: "usage ?clone: server[:port] amount". I tested this by typing "?clone 192.168.6.129:2200 1kb and received ***bot.port: connect from 192.168.6.129"
?clones	This showed usage of "[NETWORK|all] <die|join|part|raw|msg> <"parm">…", however, I could not get it to do anything. I believe it allows you to send commands to multiple clones.
?login	?login requires username then <ENTER> and password then <ENTER>
?uptime	Shows how long the system has been up and how long the bot has been up in h/m/s
?reboot	This command reboots the system you are connected to and responds with "later!"
?status	Shows information about the computer the bot is on: "service:Y user:SYSTEM inet connection:Y contype:Lan reboot privs:Y"
?jump	I got no response from this command

?nick	This tells me to “Set an irc sock to perform ?nick command on Type .sklist to view current sockets, then .dccsk <#>” The .sklist shows me the IRC channel and everyone in it and then it shows me just the irc server I am on. If you type .dccsk and then the number of the irc socket you get “using sock #1 collective7zxy0.com:667 (XmCMYbzhM, which is the current nick of the bot that is logged in on the channel from the machine I am connected to port 2200 on.
?echo	In watching packet captures, this appears to echo whatever was typed to the bot
?hush	I got no response to hush
?wget	This command gets you a file from the system you are connected to
?join	?join functioned in the same way as ?nick above
?op	I got “bad args” with this one, but I don’t know what they are looking for here.
?aop	No response
?akick	No response
?part	No response
?dump	No response
?set	This command lets you set values of jtr.bin, jtr.home, bot.port, jtr.id, irc.quit, servers and their ports, irc.chan, pass (a hash, but not sure what the password is for) and dcc.pass which is the password of the login to the bot.port. The password can be changed using this command.
?die	Killed all windows that I had open on the Linux box that was acting as my IRC server
?md5p	Takes the parameters <pass> <salt> and returns 1SALT$Password Hash
?free	Takes the value <?cmd> and releases it from use. Tested with passing it ?pwd and ?pwd no longer returned results even after quiting the telnet session and relogging in. I had to terminate the msrll.exe and restart it to get the functionality back.
?raw	?raw functioned in the same way as ?nick above
?update	Takes the parameters <url> <id>, but I couldn’t find how to use it.
?hostname	Returns the name of the computer and the IP address
?fif	No response
?!fif	No response
?del	Deletes the name of a file and you can specify the path
?pwd	Tells you the current directory that you are in
?play	Shows you the contents of the file specified
?copy	Copies the file specified to another location and to the name specified
?move	Moves the file specified to the location of choice and to the name specified
?dir	Shows the directories and files of the directory that is your present working directory
?sums	Gives you the md5 hash of the files and their versions if known
?ls	Same as ?dir, Unix based command
?cd	Changes directories and uses 8.3 filename convention.
?rmdir	Deletes the directory specified and the path can be included
?mkdir	Creates a directory where specified
?run	Runs the executable specified, but you have to give the path to it. It shows in the processes as running, but not on the screen. Indicates OK for success
?exec	Same as run above, but does not indicate whether it successfully completed as ?run does
?ps	Lists all the processes running and their Process ID
?kill	Kills the specified process by ?kill <PID> Reports success or failure
?msg	?msg functioned in the same way as ?nick above
?kb	?kb functioned in the same way as ?nick above
?sklist	Shows the information about the current sockets
?unset	Stopped any commands from displaying information, although Snort showed that the correct information was being returned for the commands run such as ?set.
?uattr	?uattr functioned in the same way as ?nick above
?dccsk	Used to connect to a socket that is specified by the socket number that can be found with ?sklist

?con	Unsure of its use, but when used with a file namespecified such as notepad I received the following: ***chdir: C:\winnt\system32\mfm -> C:\winnt\system32\mfm (0)
?killsk	Said it couldn't kill the socket and specified a socket number
?insmod	Installs loadable modules
?rmmod	Removes Loadable modules
?lsmod	Lists loadable modules

The "Bot Army"

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

Part 4: Code Analysis

Unpacking/Unencrypting

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

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

wrote down of the image base. To find the memory location of the code, you use this formula: OEP – Base Image. In our case its 1240 so now we select plugins from the top of the Ollydbg main window and then choose Ollydump and Dump debugged Process. You put 1240 in the modify box, and then select the "rebuild import" box and dump the code. Now you will have a dumped code that is unpacked and readable for later use.

Program Code Disassembly

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

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

Debugging

To find where the password was being used, I set breakpoints on all commands referencing anything looking like it was associated with a password such as "dcc.pass" and "irc.pass". I then attempted to launch commands from the IRC window and started by stepping through the code line by line from the breakpoint and watching what was happening. However, after hours of looking at the code, I realized that I was not seeing a password and could not figure out how irc.pass was working, which appeared to control the password to the IRC channel. I then attempted to login via the telnet session using the commands "?login testlogin" and "?pwd testpwd" (it was later determined that ?pwd was not for the password,

but rather for the “present working directory”) which kept triggering the dcc.pass and then I would end up hitting these two lines of:
0040BC6A PUSH msrll.0040BB49 ASCII "bot.port"
0040BC6F PUSH msrll.0040BB52 ASCII "%s bad pass from "%s"@%s"

I knew I had found the login code for port 2200. During the process, I found one of the hashes appear that were mentioned in the strings earlier and being passed as a value: “1KZLPLKDf$55isA1ITvamR7bjAdBziX”. In watching the code be parsed, I saw the value passed and it was broke down by the $. 1 was always a constant value passed, then the KZLPLKDf and finally the last letters. These were concatenated together. Later I realized that the ?login required a username (appears to be ignored) to be entered and then a password to be entered. The above string is the hash of the password that expected. I tested three passwords and set a breakpoint at Address 0040D655 which is the final compare of the password inputted and the actual password. To do this I used different usernames with each of the passwords. Every password had its own unique hash regardless of the username that I used. Since I was only seeing the hash, which was determined to be stored encrypted in used I decided to see if I could bypass the authentication by modifiying the registers. I set a breakpoint at 0040BBD9 PUSH msrll.0040BB40 ASCII “dcc.pass" Here are the key lines of code that will allow you to bypass the authentication.

```
0040BBD9   PUSH msrll.0040BB40  Arg2 = 0040BB40 ASCII "dcc.pass"
0040BBDE   PUSH EDX             Arg1
0040BBDF   CALL msrll.00405872  msrll.00405872
0040BBE4   ADD ESP,10
0040BBE7   TEST EAX,EAX
0040BBE9   JE SHORT msrll.0040BC5A
```

If you read what this is doing, you see its passing “dcc.pass” as a variable to the Call to msrll.405872 procedure. Then it adjusts the stack size with the “ADD ESP,10” and finally the key to the puzzle is its checking EAX. “TEST EAX, EAX” tell you to check the value of EAX and see if its 0 or 1 (False or True). If its True, it takes the jump and you end up with a bad password, but if its false you get authenticated. So, I forced it to bypass the jump by modifiying the value of EAX to be 1 (right click and then Increment). After pressing F9, I returned to my telnet session and sure enough I was still running. To see if I was authenticated, I typed ?si and got the information of the system. I was finally in. Once authenticated I tested the ?md5p and got the following results “?md5p <pass> <salt>”. So I put in a password of malware and a salt of test and got the following results: “1test$02Ctxuuyv0OHiS01Hx6hS1”. I know now that KZLPLKDf is the salt being used, but I still had no way of determining the password that was in use.

Another important file was created and used by the malware was jtram.conf. To find out what this file was used for I set breakpoints in the code that referenced the file. This file was written to on occasion, six lines at a time, usually after

several login and log outs by the bot due to me pausing the code and stepping through it. However, it was written to after using the ?set command for the passwords. At these times it appears that the passwords generated for the bot was written to it. However, the hashes do not match those passed as arguments so my guess is that they are the password hashes that have been encrypted and written to the file. From all indications, this file is read at initial startup and then not again. I believe this to be the case because I never saw it read during any login attempts.

Part 5: Analysis Wrap-Up

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

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

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

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

Citation of Sources

Liston, Tom. Conversation via Jabber on finding the OEP. September 2004

Smit, Ferdi. "Assembly Tutorial." 1996. URL: http://www.xs4all.nl/~smit/asm01001.htm (30 August 2004).

Vonck, Tjerk. "IRC FAQ: Introduction to IRC for people using Windows." URL: http://www.mirc.com/ircintro.html (1 September 2004).

Sites for Tools

VMWare Workstation 4.5.1: http://www.vmware.com/
Linux Redhat: http://www.linux.org/.
Ollydbg: http://home.t-online.de/home/Ollydbg/.
Ethereal: http://www.ethereal.com/.
SNORT: http://www.snort.org/.
Filemon: http://www.sysinternals.com/ntw2k/source/filemon.shtml.
RegMon: http://www.sysinternals.com/ntw2k/source/regmon.shtml.
TDIMon: http://www.sysinternals.com/ntw2k/freeware/tdimon.shtml.
LordPE: http://mitglied.lycos.de/yoda2k/LordPE/info.htm.
Regshot: http://regshot.ist.md/ however it was unavailable the last time I checked. It can be found easy by a simple Google query.
Netcat: http://netcat.sourceforge.net.
Process Explorer: http://www.sysinternals.com/ntw2k/freeware/procexp.shtml.
PSkill: http://www.sysinternals.com/ntw2k/freeware/pskill.shtml.
MD5sum: http://www.gnu.org/software/textutils/textutils.html.

Last Updated: July 26th, 2017

Upcoming Training

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 Fall 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 Boston FOR610	Boston, MA	Oct 09, 2017 - Oct 14, 2017	Community SANS
SANS Tokyo Autumn 2017	Tokyo, Japan	Oct 16, 2017 - Oct 28, 2017	Live Event
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
Cyber Threat Intelligence Summit & Training 2018	Bethesda, MD	Jan 29, 2018 - Feb 05, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced