

### **STATIC ANALYSIS**

Static analysis examines the malware before execution and involves reading outputs of software tools. If the sample is a Windows portable executable, this could consist of using tools like PEExplore, PEview or PeStudio. Observations on which resources it imports or whether the executable is packed so as to obfuscate information are part of this consideration. Other static options include disassemblers such as IDA, which summarize how an executable or DLL runs in machine code, or multipurpose and decompilation tools like Ghidra.



PeStudio is a multipurpose tool that detects functionality signatures for different types of malware and can detect abnormalities in imported files. Sometimes import purposes must be researched.

In this image, *PeStudio* is being used on our second malware sample to observe that that it is capable of making HTTP and FTP socket connections, likely to contact a hacker's server and send them private information about the host computer that will enable them easy access in the future: a backdoor.

ascii	9	0x000	-	-	-	-	anonymous
ascii	6	0x000		-	-	-	FTP://
ascii	6	0x000	-	-	-	-	ftp://
ascii	15	0x000	-	-	-	-	Content-Length:
ascii	10	0x000		-	-	-	HTTP/1.1 5
ascii	10	0x000	-	-	-	-	HTTP/1.1 3
ascii	10	0x000	-	-	-	-	HTTP/1.1 4
ascii	10	0x000	-	-	-	-	Expires: 0
ascii	40	0x000	-	-	-	-	Cache-Control: no-cache, must-revalidate
ascii	16	0x000	-	-	-	-	Pragma: no-cache
ascii	22	0x000	-	-	-	-	Connection: Keep-Alive
ascii	172	0x000	-	-	-	-	Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-
ascii	6	0x000	-	-	-	-	Host:
ascii	8	0x000	-	-	-	-	HTTP/1.1
ascii	7	0x000	-	-	-	-	HTTP://
ascii	7	0x000	-	-	-	-	http://
ascii	7	0x000	-	-	-	-	0.0.0.0
ascii	29	0x000	-	-	-	-	Microsoft TV/Video Connection
ascii	18	0x000	-	-	-	-	Fail To Send()\r\n
ascii	8	0x000	-	-	-	-	ftp:///
ascii	9	0x000		-	-	-	http:///



*IDA* is popular and powerful disassembler/debugger software that provides users with the capability to analyze and understand the low-level assembly code of executable files, allowing for reverse-engineering and inspection of the inner workings of malware.

In this image, *IDA* is being used on our second malware sample to observe its assembly code, names of internal functions, and import functions. With this information alone we were able to see that the malware installs and uninstalls services and makes changes to the registry.

🖹 IDA View-A	N Names window			🛱 Imports			×
	Name	Address P	<u>^</u>	Address Ordin	al Name	Library	^
🖽 N už	F PSLIST	10007025 P	-	10016000	LookupPrivilegeValueA	ADVAPI32	
	F nullsub_1	1000706F		10016004	OpenProcessToken	ADVAPI32	
; Attributes: library function	F nullsub_2	1000707C		🛱 10016008	RegCloseKey	ADVAPI32	
, neer ibaces r iibrary randeion	F StartEXS	10007ECB P		🛱 1001600C	RegQueryValueExA	ADVAPI32	
aulldiv proc near	F HandlerProc	1000C9DF		10016010	RegOpenKeyExA	ADVAPI32	
	F ServiceMain	1000CF30 P		10016014	CreateProcessAsUserA	ADVAPI32	
arg_0= dword ptr 0Ch	F DIMain(x,x,x)	1000D02E		10016018	RegSetValueExA	ADVAPI32	
arg_4= dword ptr 10h	F InstallRT	1000D847 P		🛱 1001601C	RegDeleteValueA	ADVAPI32	
arg_8= dword ptr 14h arg C= dword ptr 18h	F InstallSA	1000DEC1 P		10016020	RegEnumKeyA	ADVAPI32	
arg_C= dword ptr 18h	F InstallSB	1000E892 P		10016024	RegOpenKeyA	ADVAPI32	
push ebx	F UninstallSA	1000EA05 P		10016028	SetTokenInformation	ADVAPI32	
push esi	F UninstallSB	1000F138 P		🛱 1001602C	DuplicateTokenEx	ADVAPI32	
mov eax, [esp+arg_C]	F UninstallRT	1000F405 P		🛱 10016030	RegEnumValueA	ADVAPI32	
or eax, eax	F StartAddress	10010740		10016034	AdjustTokenPrivileges	ADVAPI32	
jnz short loc_10014FD2	F GetModuleFileNameExA	100111A6		10016038	RegCreateKeyA	ADVAPI32	



Ghidra is a multipurpose tool which can decompile a binary program and provide insight into the code pages, source code, and code flow for a malware sample. In this image, Ghidra is being used on our third malware sample to observe creation of a mutex in the assembly code. The mutex allows the malware to know if it already exists on the target system.

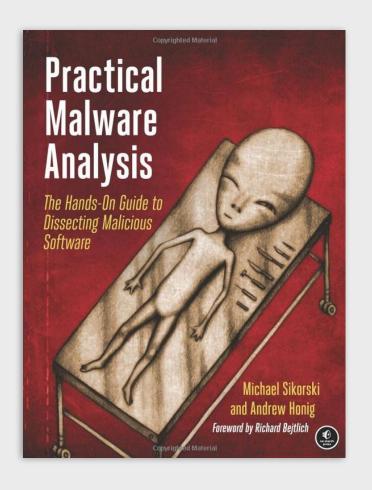
CodeBrowser: Lab07_02 Ghidra project:/Lab07_01.exe								
<u>F</u> ile <u>E</u> dit <u>A</u> nalysis <u>G</u> raph <u>N</u> avigation <u>S</u> earch Se <u>l</u> ect Tools <u>W</u> indow <u>H</u> elp								
🔚   (=   ]]) ]]) ]] ]] ]] ]] ]] [] [] [] [] [] [] [] []								
🚠 Symbol Tree 🕼 🍡 🗶 🖽 Listing: Lab07_01.exe								
<ul> <li>Imports</li> <li>ADVAPI32.DLL</li> <li>KERNEL32.DLL</li> <li>KERNEL32.DLL</li> <li>CreateMutexA</li> <li>f. CreateThread</li> <li>f. CreateWaitableTimerA</li> <li>f. ExitProcess</li> </ul>	<pre>************************************</pre>	FUN_00401						
<ul> <li>f. FreeEnvironmentStringsA</li> <li>f. FreeEnvironmentStringsW</li> </ul>	*****	*** <b>D</b> _						

# Malware Analysis

### **ABOUT THE PROJECT**

Malware analysis consists of two types of analysis-static and dynamic-each presenting its own challenges. The team constructed a specialized malware analysis lab environment using the VMware hypervisor for virtualization and integrated the Windows XP and REMnux operating systems to establish a secure and versatile lab setting. For an added layer of security, we engineered the environments around a host-only network configuration, simulating network connections via INetSim and FakeDNS. To build a comprehensive toolkit, we installed and deployed sophisticated malware analysis software tools including Windows Sysinternals Suite, IDA, PeStudio, Ghidra and OllyDbg.

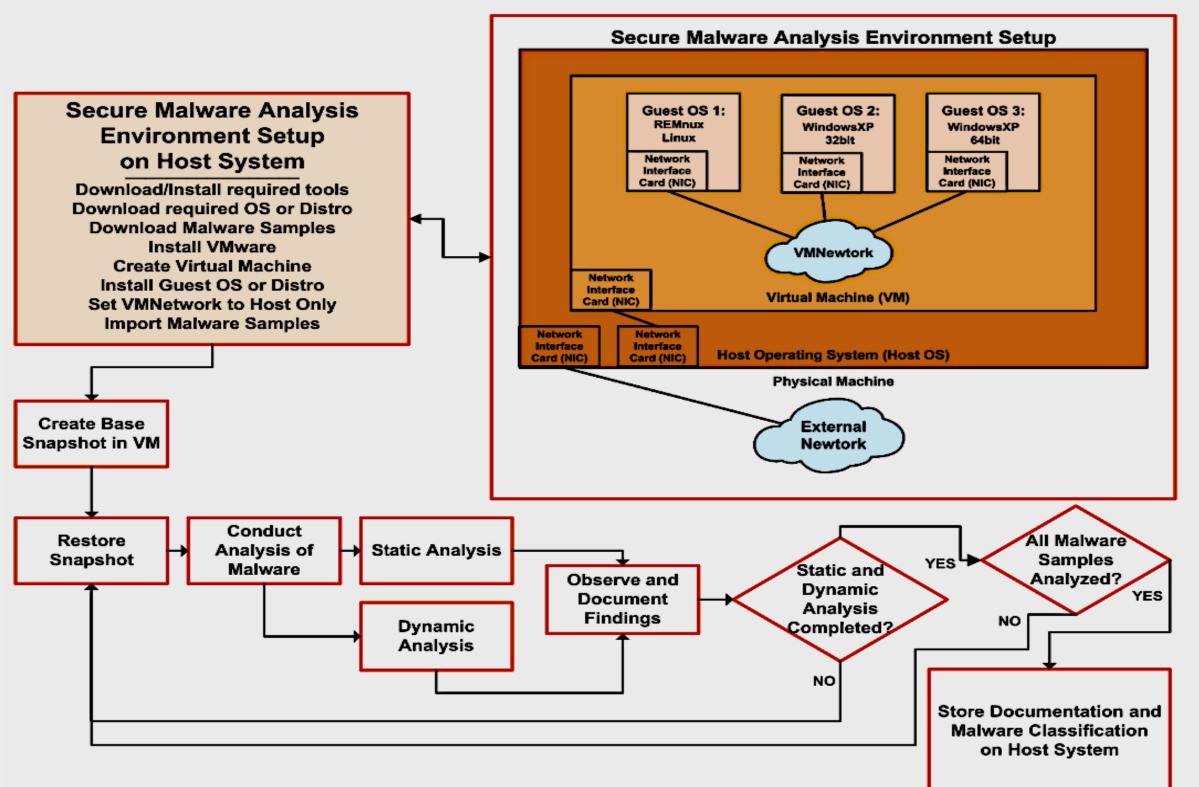
We tackled the analyses by dividing the team into two pairs, analyzing a set of malware samples sourced from the "Practical Malware Analysis" textbook. Each pair performed comprehensive investigations of three unique samples, systematically alternating between static and dynamic analyses and collecting our findings into documents to facilitate a broad understanding. The textbook, combined with our individual experiences, provided crucial insight into the correct usage, optimal setup, and appropriate application of malware analysis tools. During our bi-weekly virtual meetings, we engaged in thoughtful, detailed walkthroughs of malware analysis tools and shared our documented findings to further deepen our collective understanding.







### Malware Analysis Structural Diagram



Sponsor: Bill Pfeil Michael Banks Jennifer Bowers Steven Hunt Mark Kaiser

**m**ware<sup>®</sup>

## **DYNAMIC ANALYSIS**

Dynamic analysis involves executing malware, which requires containerization and knowledge of assembly code. Software from Windows Sysinternals such as Process Monitor are used to observe actions taken such as registry entries, file creations and network connections by the malware. Network analysis tools such as Wireshark or Netstat monitor network activity in great detail. Debuggers such as OllyDbg and WinDbg allow for modifying assembly code. With these tools put together malware activity can be closely monitored or forced into execution.



crocshock911@comcast.net

Process Monitor observes the files and registry entries as they are created, changed, or removed, and observes persistence mechanisms of malware such as service creation, scheduled tasks and auto runs.

In this image, Process Monitor is being used on our first malware sample to observe queries to create and modify a log file called practicalmalwareanalysis.log, which was evidence for the malware being a keylogger. We were able to seek out and find the log file using its path and observed that while the malware runs each key press by the current user was logged. In practice this file could contain passwords and be automatically sent over a network connection to an adversary.

p.00.11.100101000.010		o, international mayore is about national manuary in hayore sale to many concerned have set practice in the read
8:38:17.60 🚞 svchost.exe	1880 🛃 CreateFile	C:\MalwareAnalysis\Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_3L\practicalma
8:38:17.64 🚞 svchost.exe	1880 🛃 CreateFile	C:\MalwareAnalysis\Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_3L\practicalmai
8:38:17.67 🚞 svchost.exe	1880 🛃 CreateFile	C:\MalwareAnalysis\Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_3L\practicalma
3:38:17.70 🞆 svchost.exe	1880 🔜 CreateFile	C:\MalwareAnalysis\Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_3L\practicalma
8:38:17.74 🚞 svchost.exe	1880 🛃 CreateFile	C:\MalwareAnalysis\Labs\Practical Malware Analysis Labs\BinaryCollection\Chapter_3L\practicalma
R-39-17-77 Prochastieve	1890 🔍 CreateFile	C:\MalwareAnalusis\Labs\Practical Malware Analusis Labs\RinaruCollection\Chapter_3L\practicalma



*OllyDbg* is a disassembler/debugger designed for analysis of x86 Windows executables (.exe), dynamic-link libraries (.dll), and other binary files that supports both static and dynamic analysis, allowing for analysis of a program's behavior in real-time. In this image, *OllyDbg* is being used on our third malware sample to find assembly code that specifies a time that the malware will run. The year it was set to run was 834 in hexadecimal, which translates to the year 2100. By changing this value to 7E7 were were able to run the malware in the year 2023. Note also that the XOR operation is used to zero out the EDX register before being moved into other time variables on the stack like day, month, and minutes. The malware was set to detonate on January 1st at midnight.

68 30504000	PUSH Lab07_01.0040503C		Malservice"	
68 3C504000	PUSH Lab07_01.0040503C	ServiceName = "	bd - ' ••	
56 EE1E 00404000	PUSH ESI  CALL DWORD PTR DS:[<&ADVAPI32.CreateSer	hManager CrosteServiceO	Assemble at 004010DE	
33D2	XOR EDX.EDX	Createservicen		
8D4424 14	LEA EAX, DWORD PTR SS:[ESP+14]			
895424 04	MOV DWORD PTR SS:[ESP+4],EDX		MOV WORD PTR SS:[ESP+C],7E7	
8D4C24 04	LEA ECX, DWORD PTR SS: [ESP+4]		·	
895424 08 50	MOV DWORD PTR SS:[ESP+8],EDX	<b>r</b> pFileTime		
895424 10	MOV DWORD PTR SS:[ESP+10],EDX	priterine		
51	PUSH ECX	pSystemTime	Fill with NOP's	Assemb
895424 18	MOV DWORD PTR SS:[ESP+18],EDX			
66:C74424 0C	MOV WORD PTR SS:[ESP+C],834			
FF15_14404000	CALL DWORD PTR DS:[<&KERNEL32.SystemTim	SystemTimeToFil	eTime	



*Wireshark* aids in visibility of network activity to include the protocol, remote host, and possibly clear text communications between host and server. In this image, *Wireshark* is being used on our third malware sample to identify an HTTP GET request to www.practicalmalwareanalysis.com performed by the malware. These GET requests were being sent really fast on an infinite loop: a Denial of Service (DoS) attack.

-								
No.	Time	Source	Destination	Protocol	Total Length	Identification	Info	
	76 13.920698	3.33.152.147	172.16.144.134	НТТР	411	0x1185 (4485)	HTTP/1.1 301 Moved Perm	anently (t
	77 13.920898	172.16.144.134	3.33.152.147	HTTP	153	0x0651 (1617)	GET / HTTP/1.1	
	78 13.921006	3.33.152.147	172.16.144.134	ТСР	40	0x1186 (4486)	80 → 1104 [ACK] Seq=222	9 Ack=792 W
4	79 13.936789	15.197.142.173	172.16.144.134	HTTP	390	0x1187 (4487)	HTTP/1.1 301 Moved Perm	anently (t
-+>	80 13.937055	172.16.144.134	15.197.142.173	НТТР	182	0x0652 (1618)	GET / HTTP/1.1	
	81 13.937184	15.197.142.173	172.16.144.134	ТСР	40	0x1188 (4488)	80 → 1106 [ACK] Seq=105	0 Ack=569 W
	82 13.944605	3.33.152.147	172.16.144.134	HTTP	412	0x1189 (4489)	HTTP/1.1 301 Moved Perm	anently (t
	83 13.944662	15.197.142.173	172.16.144.134	HTTP	389	0x118a (4490)	HTTP/1.1 301 Moved Perm	anentlv (t
► Et ► In ► Tr ▼ Hy	hernet II, Src: Wi ternet Protocol Wi ansmission Contro pertext Transfer GET / HTTP/1.1\r\ User-Agent: Inter Connection: Keep- Cache-Control: no Host: www.practio \r\n	Mware_ab:9d:c5 (00:0 ersion 4, Src: 172.1 l Protocol, Src Port Protocol net Explorer 8.0\r\r -Alive\r\n calmalwareanalysis.co		VMware_f1:9 7.142.173 Seq: 427,	d:62 (00:5	0010         00         b6         06         52           0020         8e         ad         04         52           0030         f6         d7         d5         66           0040         2f         31         2e         31           0050         3a         20         49         66           0060         72         65         72         20           0070         69         6f         6e         3a           0080         0a         43         61         63           0090         6e         6f         2d         63           00a0         77         77         77         2e	9d       62       00       0c       29       ab       9d       c5       08         2       40       00       80       06       18       e7       ac       10       90         2       00       50       6d       7a       fe       75       a9       aa       6e         5       00       00       47       45       54       20       2f       20       48         6       00       0a       45       73       65       72       2d       41       67         6       74       65       72       6e       65       74       20       45       78         6       38       2e       30       0d       0a       43       6f       6e       6e         3       20       4b       65       65       70       2d       41       6c       69         3       8       65       2d       43       6f       6e       74       72       6f         6       61       63       64       65       0d       0a       48       6f       73         6       61       6c	0       86       0f       c5         e       6e       50       18         3       54       54       50         7       65       6e       74         3       70       6c       6f         e       65       63       74         o       76       65       0d         f       6c       3a       20         a       74       3a       20         c       6d       61       6c



alwareanalysis.log ilwareanalysis.log lwareanalysis.log alwareanalysis.log



t/html) =64240 Len=0 t/html)

PV··b·· )· R@ RPmzunnP • f • · GE T / HTTF 1.1 Us er-Agent Intern et Explo er 8.0 Connect ion: Kee p-Alive Cache-C ontrol: no-cache Host www.prac ticalma wareanal ysis.com