Malware Reverse Engineering Report Practical 3

By: Gary Jones
Jonegn1@ufl.edu
CAP4136 Practical 3: Reverse Malware Engineering

Executive summary

Project overview

The goal of Practical 3 is to dissect the functionality of sample3.dll using static and dynamic analysis.

Summary of findings

Sample3.dll has proven to be a challenging piece of malware to analyze. It is packed and it uses anti-debugging techniques. This malware challenges the user at every turn. Through the use of various tools it was found that this malware bares hallmarks of the EnigmaProtector and Armadilo Packer. In addition the family of malware it is most closely related to is Emotet which means that it focuses on downloading other malware from the internet and discovering other systems on a users network. This malware was also found to target the WPAD registry. This means that it could be extracting passwords and might be compromising the system for a man in the middle attack. Adding on to this the malware was also found to contain strings that point to hooking so this is a possible operation performed by the program.

Technical report

Findings: Static Analysis

1. Identify the apparent compilation date of the program.

The compilation date of sample3.dll is 04MAR2022. This is in the past however it is incredibly recent and could be a suspicious indicator for malfeasance.

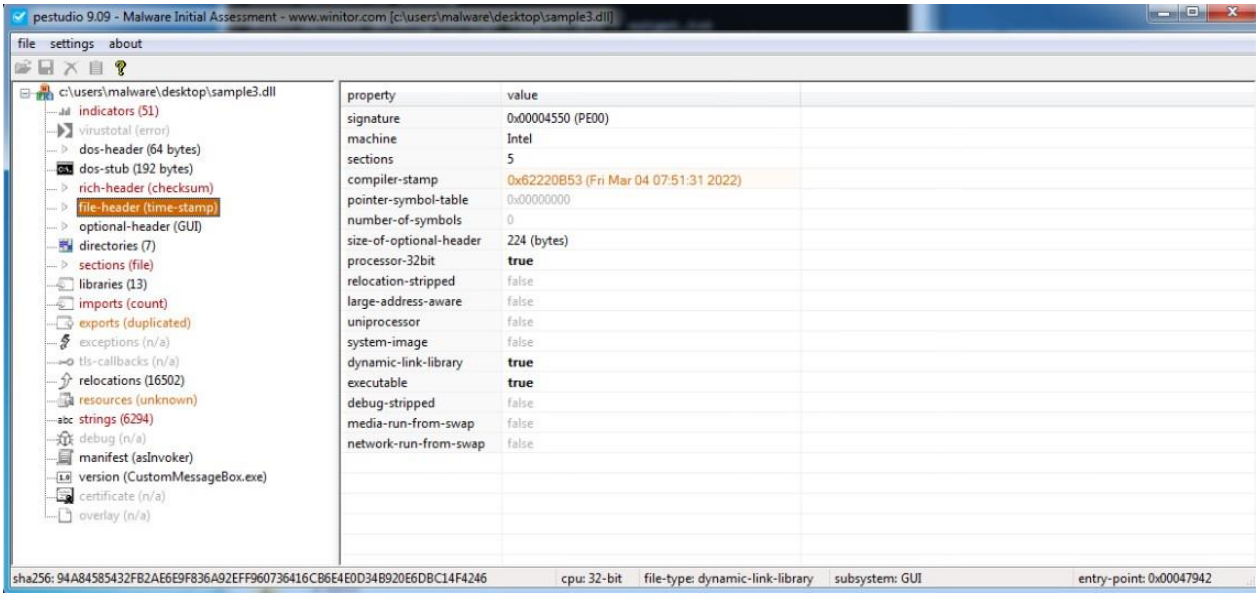


Figure 1: Compilation Date of Sample3.dll

2. Identify any suspicious properties of the program's Imports.

Sample3.dll has a significant amount of imports with a number of 534. Within this number there are 61 blacklisted and 2 imports without a listed group. There are 15 groups that the blacklisted functions fall within and two names that do not have a group attributed to them. These groups are as follows:

- Windowing
- System-information
- Storage
- Shell
- Resource
- Registry
- Memory
- Keyboard-and-mouse
- Hooking
- File
- Execution
- Exception-handling
- Dynamic-library
- Data-exchange
- Console

The most suspicious among these groups are as follows: keyboard-and-mouse, hooking, data-exchange, and registry.

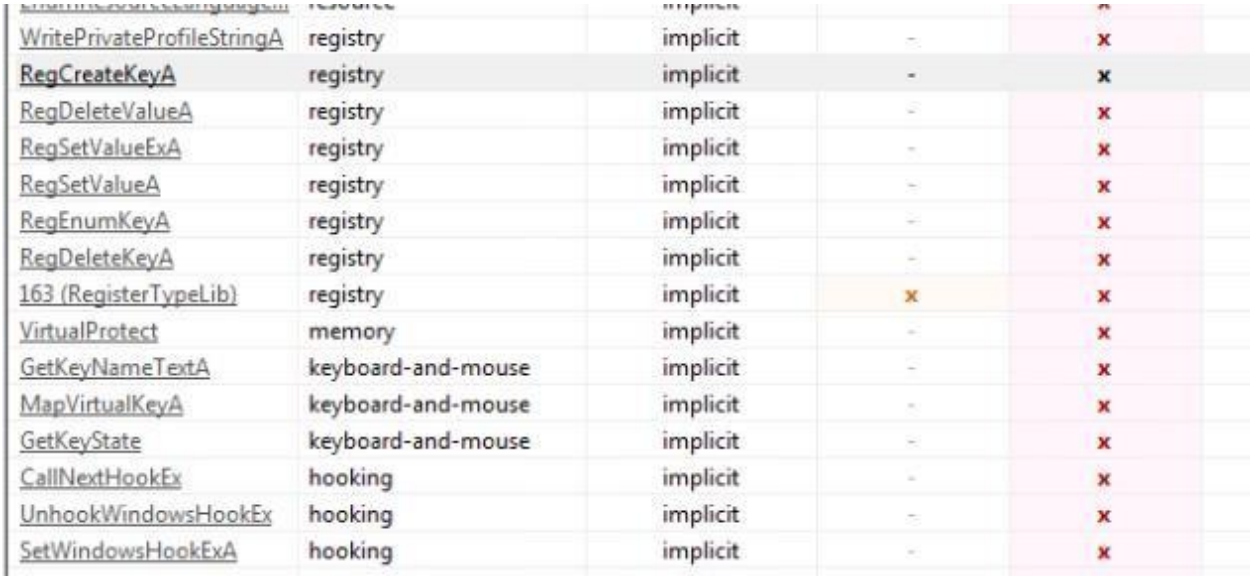
The imports associated with keyboard-and-mouse: GetKeyNameTextA, MapVirtualKeyA, and GetKeyState

The imports associated with Hooking: CallNextHookEx, UnhookWindowsHookEx, SetWindowsHookExA

The imports associated with Data-exchange: GetAtomNameA, GlobalGetAtomNameA, GlobalFindAtomA, GlobalAddAtomA, GlobalDeleteAtom, RegisterClipboardFormatA, OleSetClipboard, OleFlushClipboard

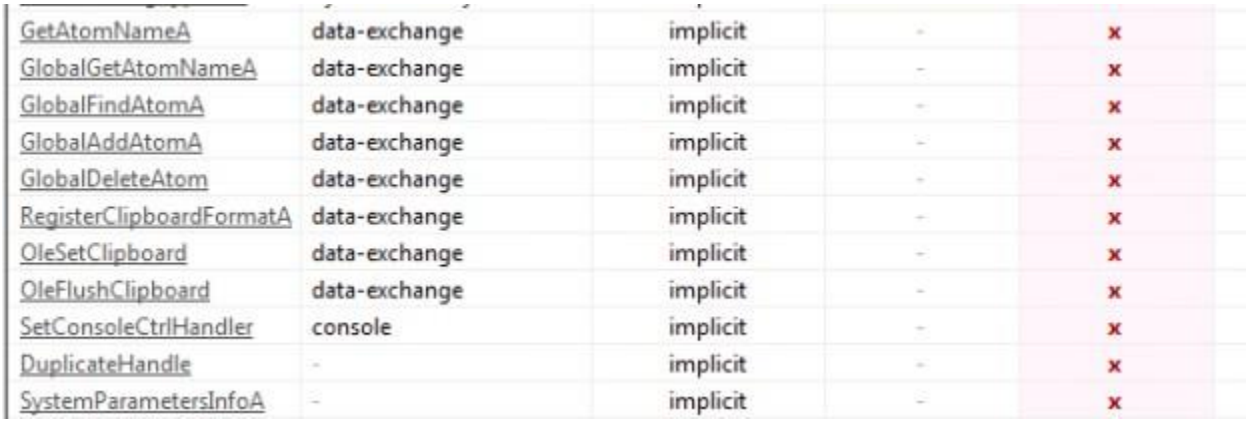
The imports associated with Registry: RegCreateKeyA, RegDeleteValueA, RegSetValueExA, RegSetValueA, RegEnumKeyA, RegDeleteKeyA, 163 (RegisterTypeLib)

The two imports not associated with any group but are still blacklisted are DuplicateHandle and SystemParametersInfoA



WritePrivateProfileStringA	registry	implicit	-	x
RegCreateKeyA	registry	implicit	-	x
RegDeleteValueA	registry	implicit	-	x
RegSetValueExA	registry	implicit	-	x
RegSetValueA	registry	implicit	-	x
RegEnumKeyA	registry	implicit	-	x
RegDeleteKeyA	registry	implicit	-	x
163 (RegisterTypeLib)	registry	implicit	x	x
VirtualProtect	memory	implicit	-	x
GetKeyNameTextA	keyboard-and-mouse	implicit	-	x
MapVirtualKeyA	keyboard-and-mouse	implicit	-	x
GetKeyState	keyboard-and-mouse	implicit	-	x
CallNextHookEx	hooking	implicit	-	x
UnhookWindowsHookEx	hooking	implicit	-	x
SetWindowsHookExA	hooking	implicit	-	x

Figure 2: Sample3.dll Imported Libraries Part 1



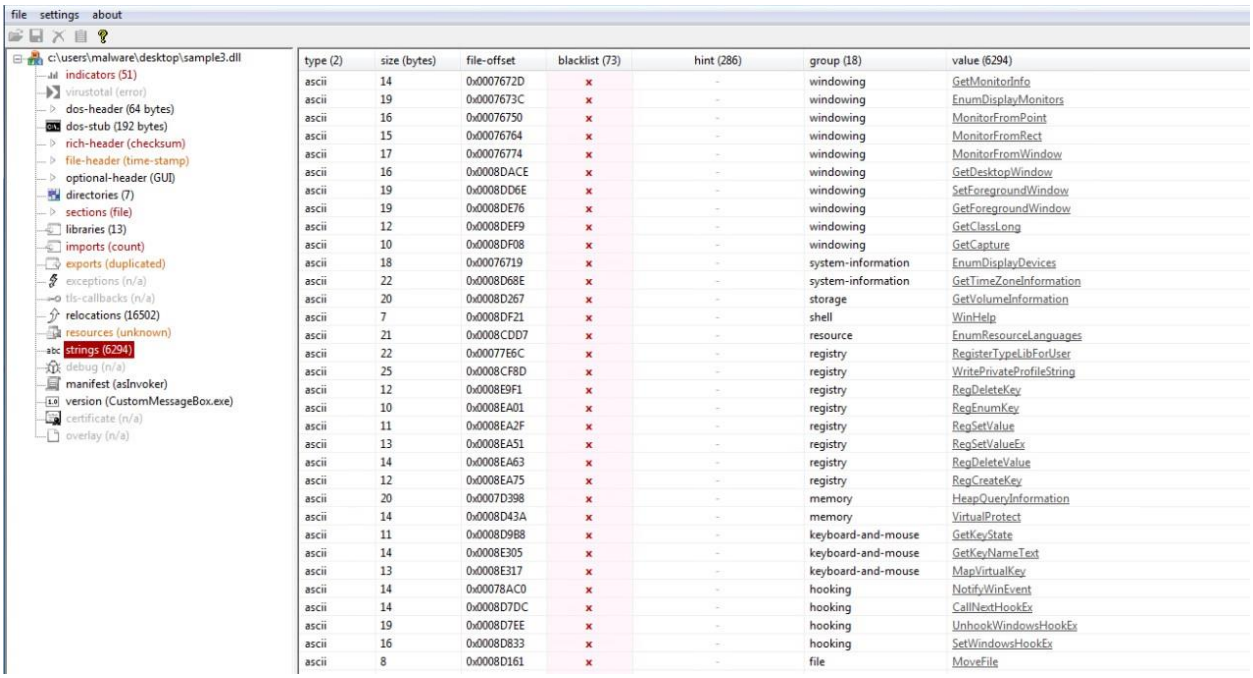
GetAtomNameA	data-exchange	implicit	-	x
GlobalGetAtomNameA	data-exchange	implicit	-	x
GlobalFindAtomA	data-exchange	implicit	-	x
GlobalAddAtomA	data-exchange	implicit	-	x
GlobalDeleteAtom	data-exchange	implicit	-	x
RegisterClipboardFormatA	data-exchange	implicit	-	x
OleSetClipboard	data-exchange	implicit	-	x
OleFlushClipboard	data-exchange	implicit	-	x
SetConsoleCtrlHandler	console	implicit	-	x
DuplicateHandle	-	implicit	-	x
SystemParametersInfoA	-	implicit	-	x

Figure 3: Sample3.dll Imported Libraries Part 2

3. Identify any suspicious or relevant strings (IP addresses, urls, process names, file names, etc.).

After looking at the strings I was not able to identify any IP addresses or URL names. However, there are 6294 strings with 73 blacklisted strings. An interesting observation is that a number of the blacklisted strings are also found in the import section and are like for like with the reported groups: keyboard-and-mouse, hooking, registry, and data-exchange.

With that said, other suspicious strings include activities with files such as MoveFile, DeleteFile, LockFile, UnlockFile, SetFileAttributes, PathRemoveFileSpec and more.



type (2)	size (bytes)	file-offset	blacklist (73)	hint (286)	group (18)	value (6294)
ascii	14	0x0007672D	x	-	windowing	GetMonitorInfo
ascii	19	0x0007673C	x	-	windowing	EnumDisplayMonitors
ascii	16	0x00076750	x	-	windowing	MonitorFromPoint
ascii	15	0x00076764	x	-	windowing	MonitorFromRect
ascii	17	0x00076774	x	-	windowing	MonitorFromWindow
ascii	16	0x0008DACE	x	-	windowing	GetDesktopWindow
ascii	19	0x0008DD6E	x	-	windowing	SetForegroundWindow
ascii	19	0x0008DE76	x	-	windowing	GetForegroundWindow
ascii	12	0x0008DEF9	x	-	windowing	GetClassLong
ascii	10	0x0008DF08	x	-	windowing	GetCapture
ascii	18	0x00076719	x	-	system-information	EnumDisplayDevices
ascii	22	0x0008D68E	x	-	system-information	GetTimeZoneInformation
ascii	20	0x0008D267	x	-	storage	GetVolumeInformation
ascii	7	0x0008DF21	x	-	shell	WinHelp
ascii	21	0x0008CDD7	x	-	resource	EnumResourceLanguages
ascii	22	0x00077E6C	x	-	registry	RegisterTypeLibForUser
ascii	25	0x0008CF8D	x	-	registry	WritePrivateProfileString
ascii	12	0x0008E9F1	x	-	registry	RegDeleteKey
ascii	10	0x0008EA01	x	-	registry	RegEnumKey
ascii	11	0x0008EA2F	x	-	registry	RegSetValue
ascii	13	0x0008EA51	x	-	registry	RegSetValueEx
ascii	14	0x0008EA63	x	-	registry	RegDeleteValue
ascii	12	0x0008EA75	x	-	registry	RegCreateKey
ascii	20	0x0007D398	x	-	memory	HeapQueryInformation
ascii	14	0x0008D43A	x	-	memory	VirtualProtect
ascii	11	0x0008D9B8	x	-	keyboard-and-mouse	GetKeyState
ascii	14	0x0008E305	x	-	keyboard-and-mouse	GetKeyNameText
ascii	13	0x0008E317	x	-	keyboard-and-mouse	MapVirtualKey
ascii	14	0x00078AC0	x	-	hooking	NotifyWinEvent
ascii	14	0x0008D7DC	x	-	hooking	CallNextHookEx
ascii	19	0x0008D7EE	x	-	hooking	UnhookWindowsHookEx
ascii	16	0x0008D833	x	-	hooking	SetWindowsHookEx
ascii	8	0x0008D161	x	-	file	MoveFile

Figure 4: Blacklisted strings part 1

type (2)	size (bytes)	file-offset	blacklist (73)	hint (286)	group (18)	value (6294)
ascii	8	0x0008D161	x	-	file	MoveFile
ascii	10	0x0008D16D	x	-	file	DeleteFile
ascii	8	0x0008D1EA	x	-	file	LockFile
ascii	10	0x0008D1F6	x	-	file	UnlockFile
ascii	13	0x0008D255	x	-	file	FindFirstFile
ascii	17	0x0008D373	x	-	file	SetFileAttributes
ascii	13	0x0008EAA3	x	-	file	SHGetFileInfo
ascii	17	0x0008EAEF	x	-	file	PathFindExtension
ascii	16	0x0008EB1D	x	-	file	PathFindFileName
ascii	18	0x0008EB53	x	-	file	PathRemoveFileSpec
ascii	23	0x0007D364	x	-	execution	SetThreadStackGuarantee
ascii	18	0x0008CCF6	x	-	execution	GetCurrentThreadId
ascii	16	0x0008CE08	x	-	execution	GetCurrentThread
ascii	13	0x0008CE84	x	-	execution	SuspendThread
ascii	19	0x0008CFC6	x	-	execution	GetCurrentProcessId
ascii	15	0x0008D18E	x	-	execution	GetThreadLocale
ascii	16	0x0008D496	x	-	execution	TerminateProcess
ascii	21	0x0008D58E	x	-	execution	GetEnvironmentStrings
ascii	21	0x0008D5C1	x	-	execution	GetEnvironmentStrings
ascii	22	0x0008D7B5	x	-	execution	SetEnvironmentVariable
ascii	24	0x0008DB9E	x	-	execution	GetWindowThreadProcessId
ascii	17	0x0008E341	x	-	execution	PostThreadMessage
ascii	14	0x0008D428	x	-	exception-handling	RaiseException
ascii	17	0x0008CD0D	x	-	dynamic-library	GetModuleFileName
ascii	17	0x0008D021	x	-	dynamic-library	GetModuleFileName
ascii	23	0x0007E80C	x	-	desktop	GetProcessWindowStation
ascii	24	0x0007E825	x	-	desktop	GetUserObjectInformation
ascii	16	0x0008CE1C	x	-	data-exchange	GlobalDeleteAtom
ascii	13	0x0008CEA5	x	-	data-exchange	GlobalAddAtom
ascii	14	0x0008CFF9	x	-	data-exchange	GlobalFindAtom
ascii	17	0x0008D00B	x	-	data-exchange	GlobalGetAtomName
ascii	11	0x0008D2C9	x	-	data-exchange	GetAtomName
ascii	23	0x0008E2D1	x	-	data-exchange	RegisterClipboardFormat

Figure 5: Blacklisted strings part 2

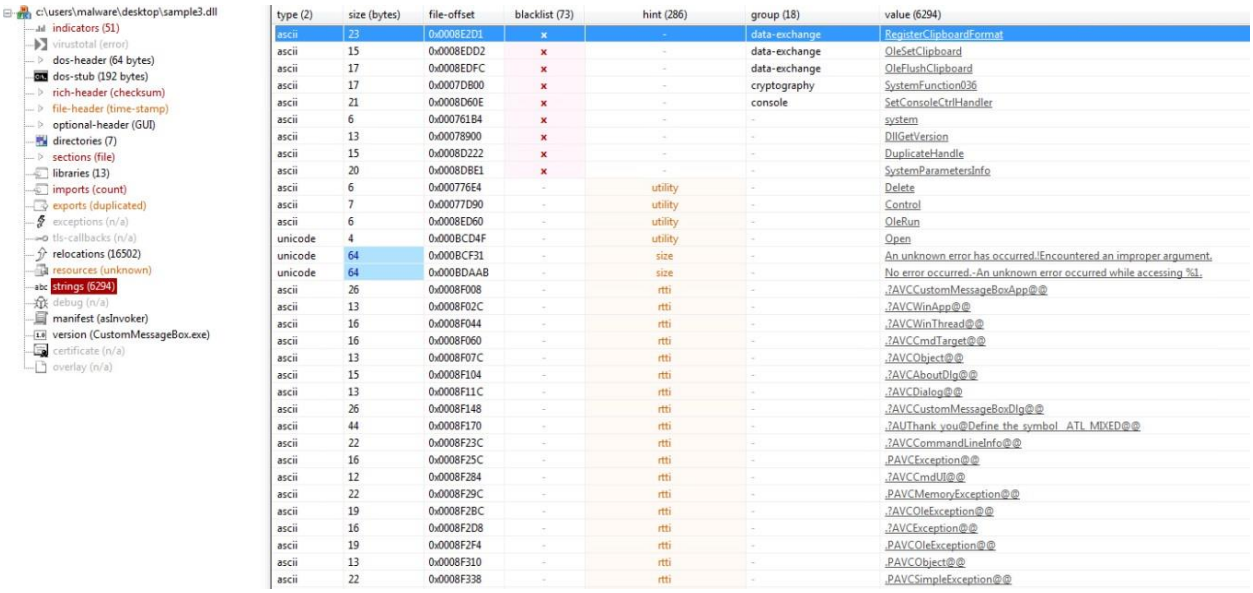


Figure 6:Blacklisted strings part 3

After the initial look at the program with pestudio the static analysis was extended to include Virus Total and Intezer Analyze. From these tools several other strings were identified to be indicative of malware. These strings include the following:

- 7 7$707@7L7\7h7x7 - associated with Generic Malware
- 4W5]5s5~5 – associated with poison Malware
- >5>E>X>p> - associated with RedLeaves Malware
- SVWjkXjef – associated with Turla Malware
- Y;D$8t – associated with Emotet Malware
- 1 272E2N2 – associated with RootKit, RatankbaPOS Malware
- GBBFRF – associated with Emotet Malware

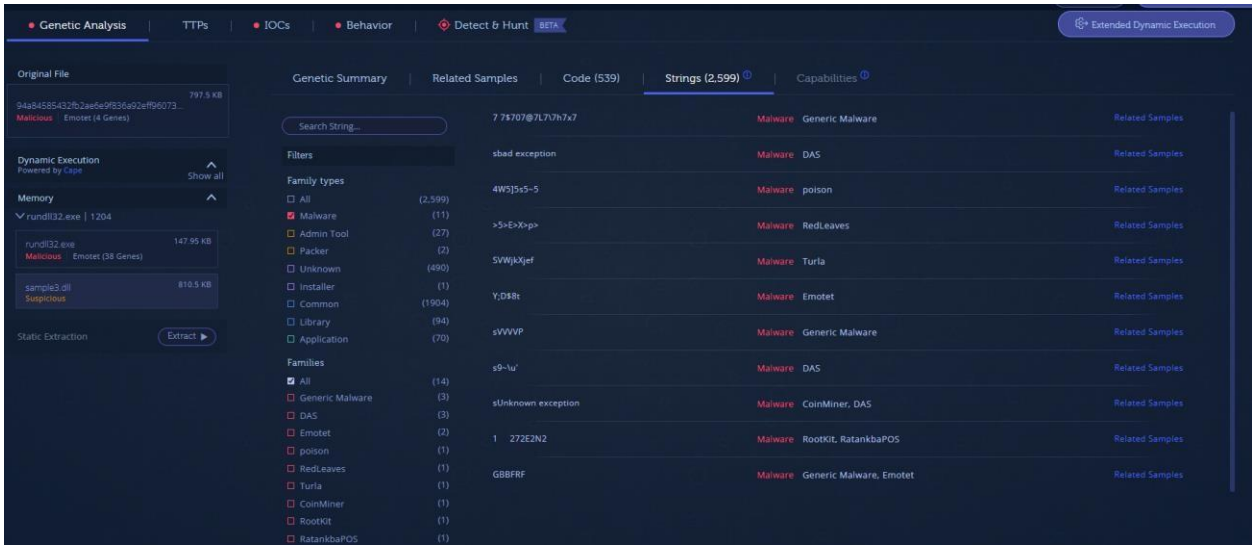


Figure 7: String Association with Malware

In addition to the malware family identifiers, Intezer Analyze also identified strings associated with different packers which include the following:

- 6:6>6B6F6J6N6R6V6Z6^6b6f6j6n6r6v6z6~6 – associated with EnigmaProtector Packer
- :6;<;L; - associated with Armadillo Packer

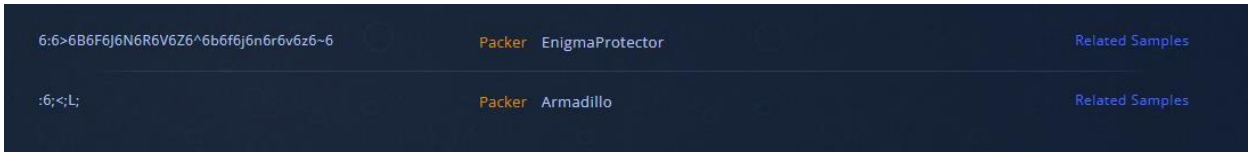


Figure 8: String Association with Packers

4. If you find anti-disassembly techniques, report them.

Through analysis with the debugger on the Windows 7 virtual machine anti-debugging properties were identified. As shown in Figure 9 we can see eax+2. This line is identifying whether or not debugger is being used. When this results in 1 it indicates that a debugger has been detected. For further verification we can also see the anti-debugging property in Ghidra as shown in FUN_10047030 (Figure 10).

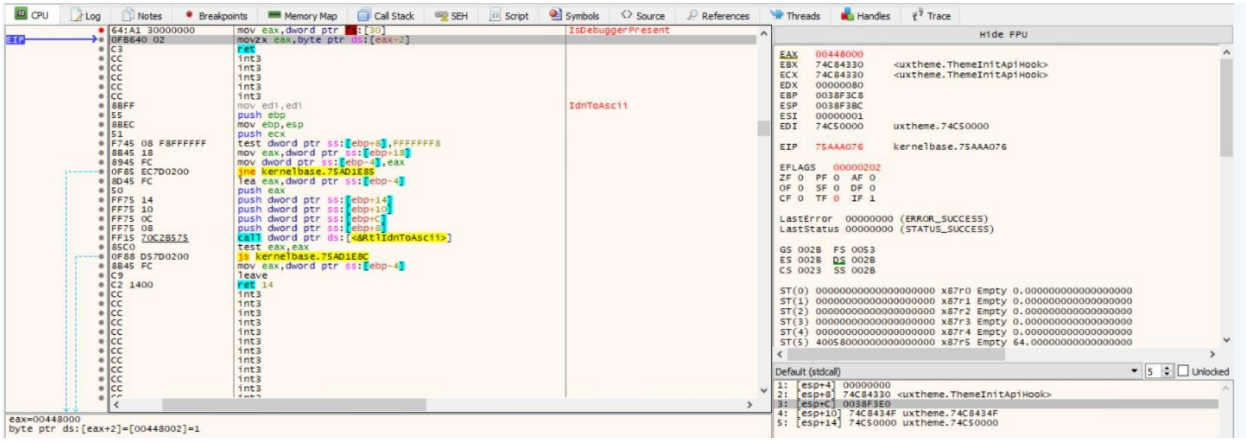


Figure 9: Anti-debugging property x32dbg

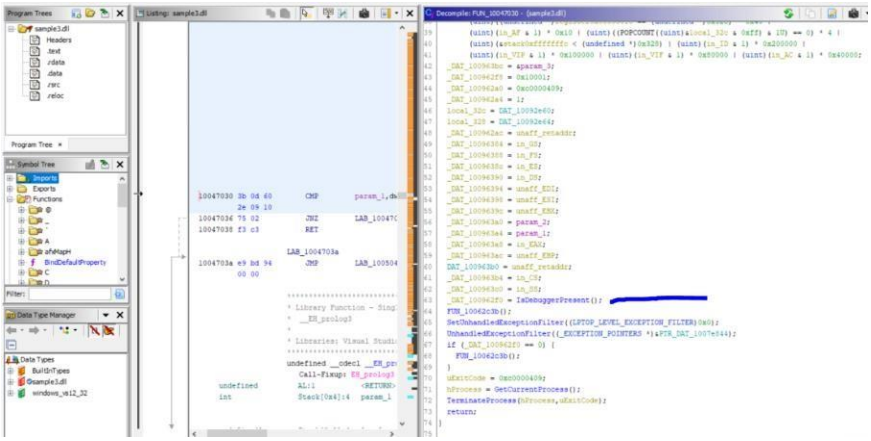


Figure 10: Ghidra anti-debugging

5. Describe the obfuscation methods you find. You will surely be impacted by them, but identifying any interesting patterns might be helpful in developing a tool to combat them.

With the use of JOESandbox Cloud the analysis of malware was analyzed and from the results some suspicious functions were identified that were identified as being related to encryption. Looking into it we find a high degree of relation to FUN_10001985 (see Figures 11 through 13) and FUN_10005367 (see Figures 14 through 16) with the latter function calling the former function numerous times – in a very suspicious manner that indicates that encryption is likely occurring.

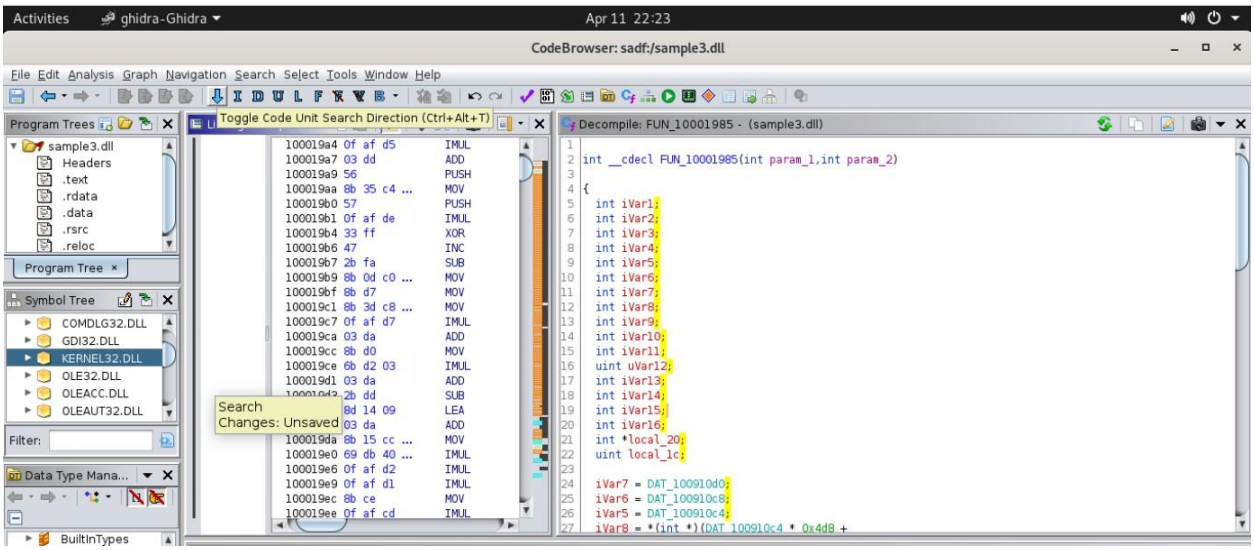


Figure 11: FUN_0001985 Part 1

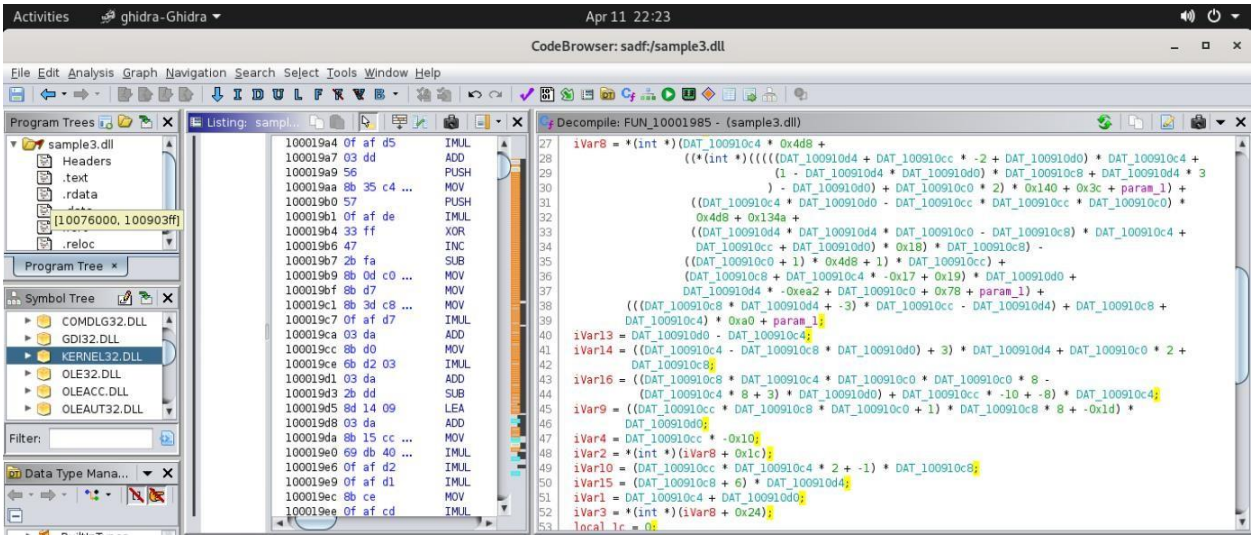


Figure 12: FUN_0001985 Part 2

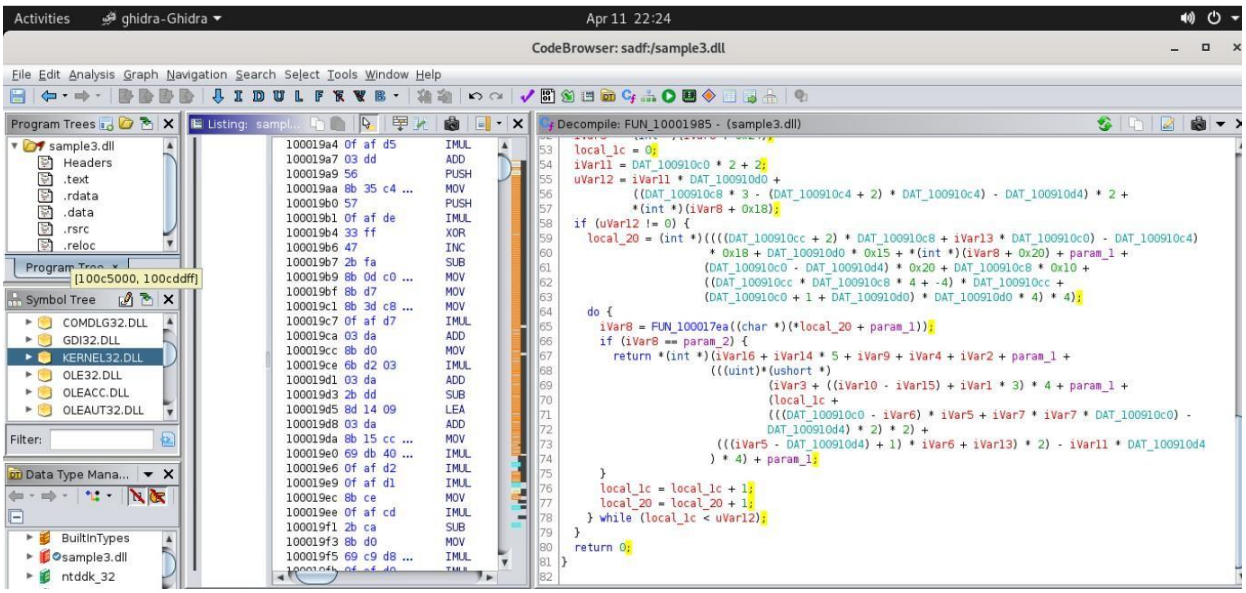


Figure 13: FUN_0001985 Part 3

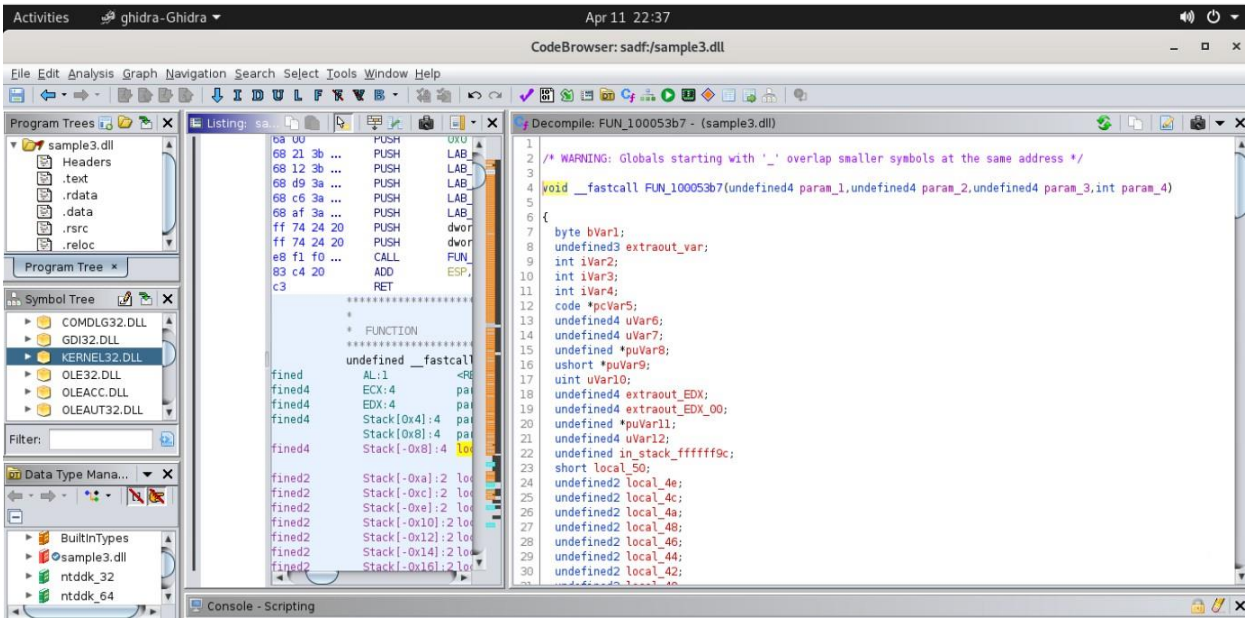


Figure 14: FUN_10005367 Part 1

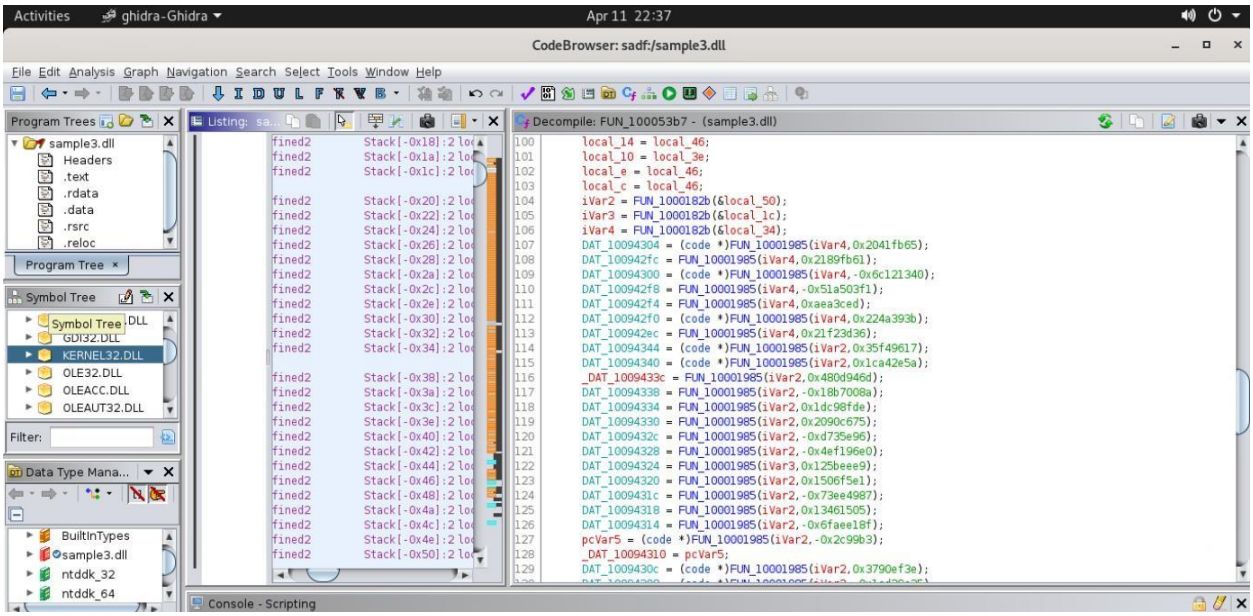


Figure 15: FUN_10005367 Part 2

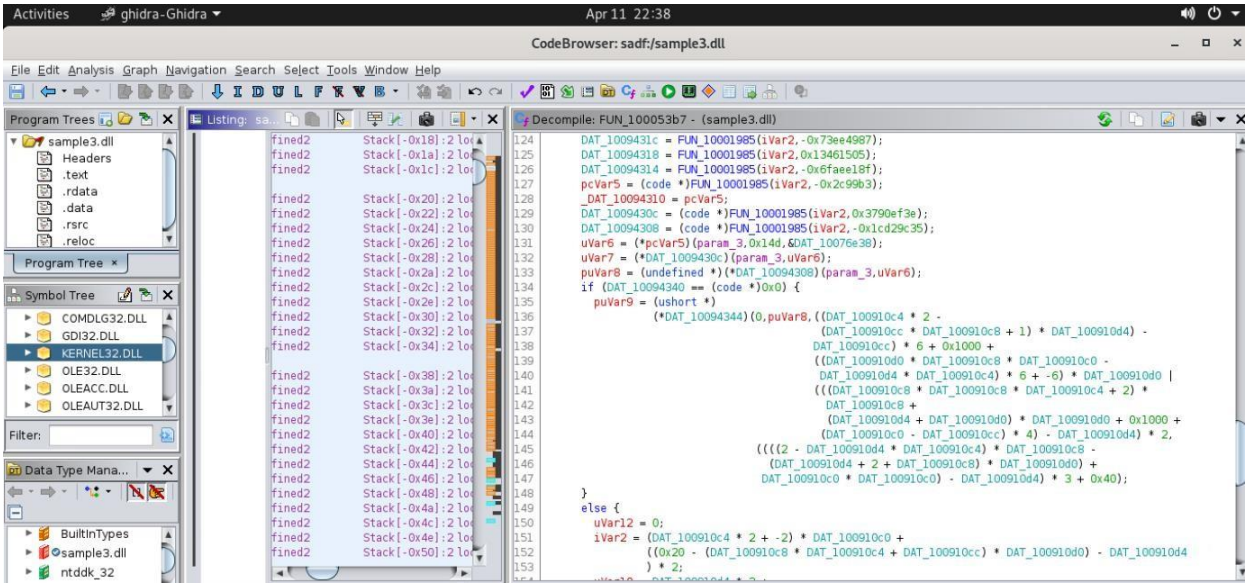


Figure 16: FUN_10005367 Part 3

Findings: Dynamic Analysis

1. Interesting behaviors that occur after the malware has executed.

The most notable behavior exhibited by the malware is its interaction with WPAD. From experience with penetration testing the modification of this software can be used to further compromise a system by stealing listed passwords and setting the environment for a man in the middle attack. However, no other notable behaviors were observed and there were no persistence mechanisms identified.

2. Identify whether or not there is any networking behavior exhibited by the program and, if so, record it.

By utilizing fakedns the network activity being utilized by the sample3.dll was analyzed. From this activity the following network activity was identified:

- dns.msftncsi.com
- fs.microsoft.com
- 255.245.168.192.in-addr.arpa
- 133.245.168.192.in-addr.arpa
- 104.195.71.45.in-addr.arpa
- 167.106.37.54.in-addr.arpa
- 2.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa
- 138.130.168.185.in-addr.arpa
- 177.244.44.37.in-addr.arpa
- 78.25.184.185.in-addr.arpa
- 15.168.148.185.in-addr.arpa
- 135.192.199.128.in-addr.arpa
- 141.209.59.37.in-addr.arpa
- 169.204.41.103.in-addr.arpa
- 120.58.42.103.in-addr.arpa
- 220.168.148.185.in-addr.arpa

As shown in Figures 17 and 18 there are more addresses attempting contact.

```
fakedns[INFO]: Response: teredo.ipv6.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: dns.msftncsi.com -> 192.168.245.133
fakedns[INFO]: Response: dns.msftncsi.com -> 192.168.245.133
fakedns[INFO]: Response: teredo.ipv6.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: fs.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: dns.msftncsi.com -> 192.168.245.133
fakedns[INFO]: Response: dns.msftncsi.com -> 192.168.245.133
fakedns[INFO]: Response: teredo.ipv6.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: dns.msftncsi.com -> 192.168.245.133
fakedns[INFO]: Response: settings-win.data.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: 255.245.168.192.in-addr.arpa -> 192.168.245.133
fakedns[INFO]: Response: 133.245.168.192.in-addr.arpa -> 192.168.245.133
fakedns[INFO]: Response: teredo.ipv6.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: tlu.dl.delivery.mp.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: tlu.dl.delivery.mp.microsoft.com -> 192.168.245.133
fakedns[INFO]: Response: 104.195.71.45.in-addr.arpa -> 192.168.245.133
fakedns[INFO]: Response: 167.106.37.54.in-addr.arpa -> 192.168.245.133
fakedns[INFO]: Response: 2.0.0.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa ->
 192.168.245.133
```

Figure 17: Network Activity Part 1

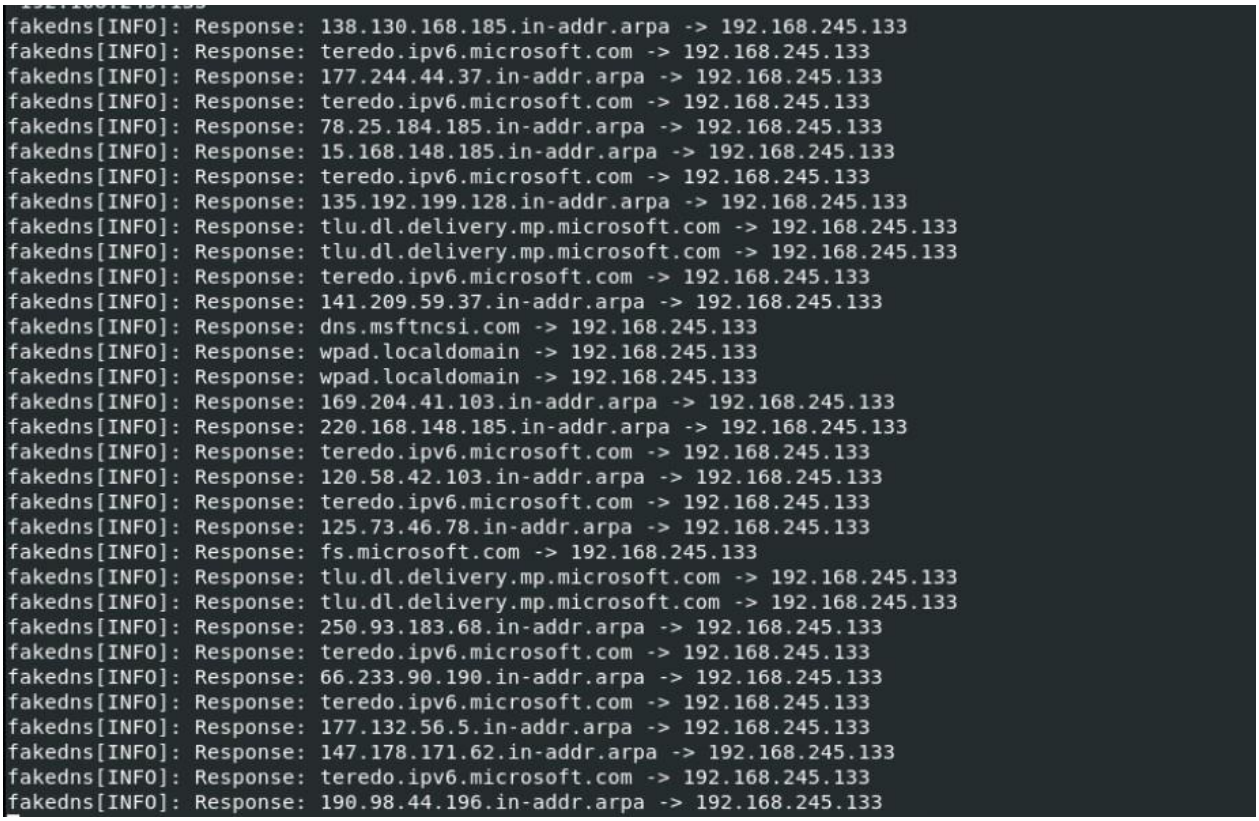


Figure 18: Network Activity Part 2

3. Identify any registry keys created/modified by the malware.

In order to identify any registry keys being created or modified procmon was used. After initiating the program and letting it run the file was saved and analyzed. By filtering for the Process Name: regsvr32.exe and the Operation: RegCreateKey I was able to identify the created keys of sample3.dll as shown in Figure 19. I did not notice the modification of any registry keys. From Figure 19 we can see extensive work being performed on WPAD, connections with the internet, and Tcpip parameters. The interaction with WPAD is especially concerning as it can be exploited to glisten passwords and perform a man in the middle attack.

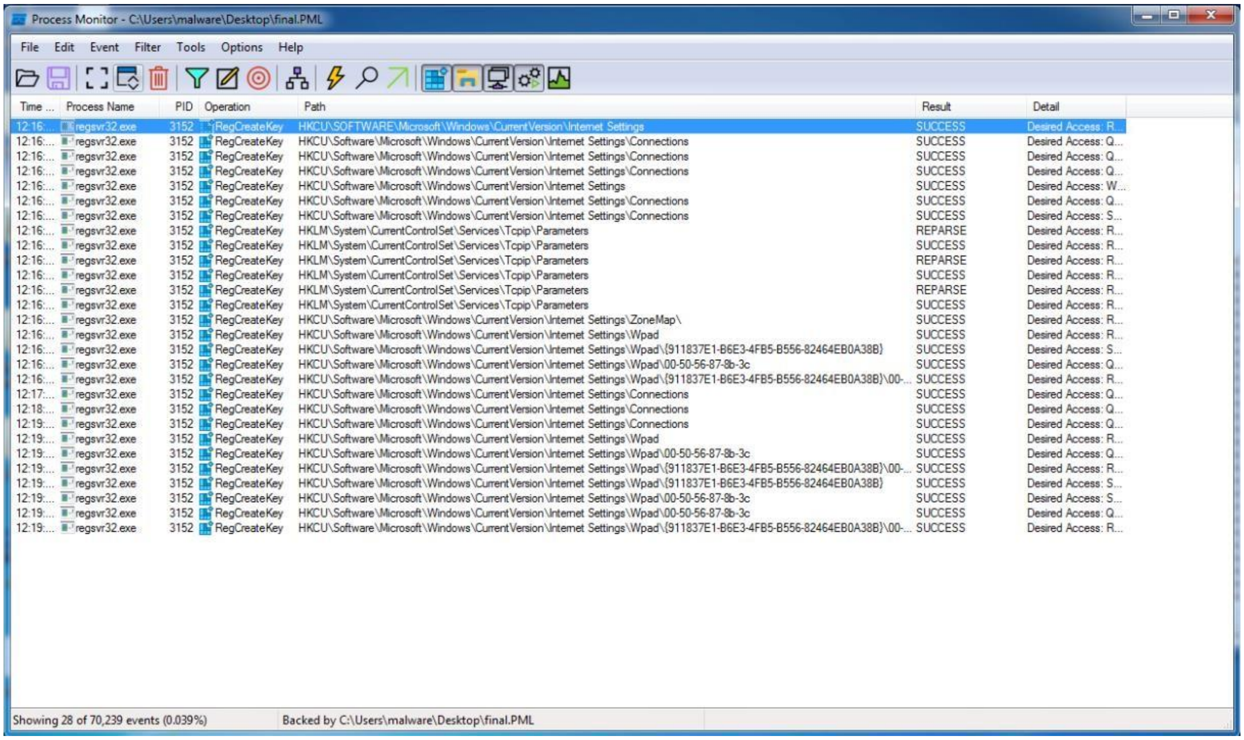


Figure 19: Created Registry Keys

4. Identify any files created/modified by the malware.

In order to identify the files created and modified procmon was used. As seen below in Figure 20 a significant amount of modifications was made in association with WPAD which as described earlier is a concerning indicator for the system being compromised.

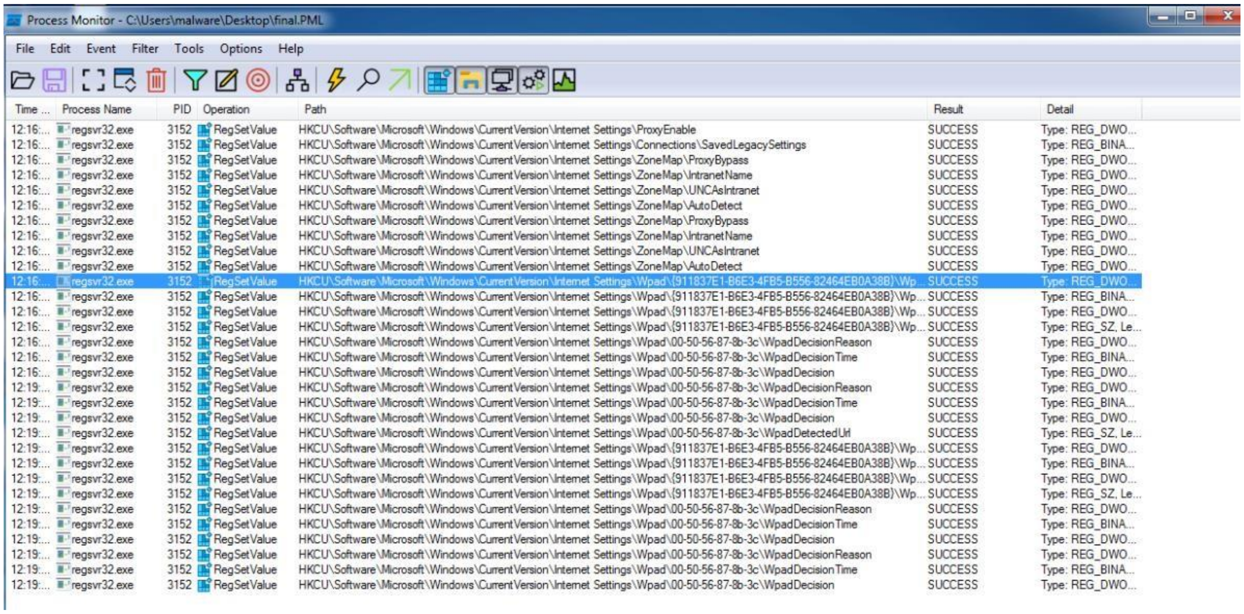


Figure 20: Modified Files

5. Identify any processes started by the malware.

By utilizing process explorer we can observe that there were no processes started by the sample3.dll file when run.

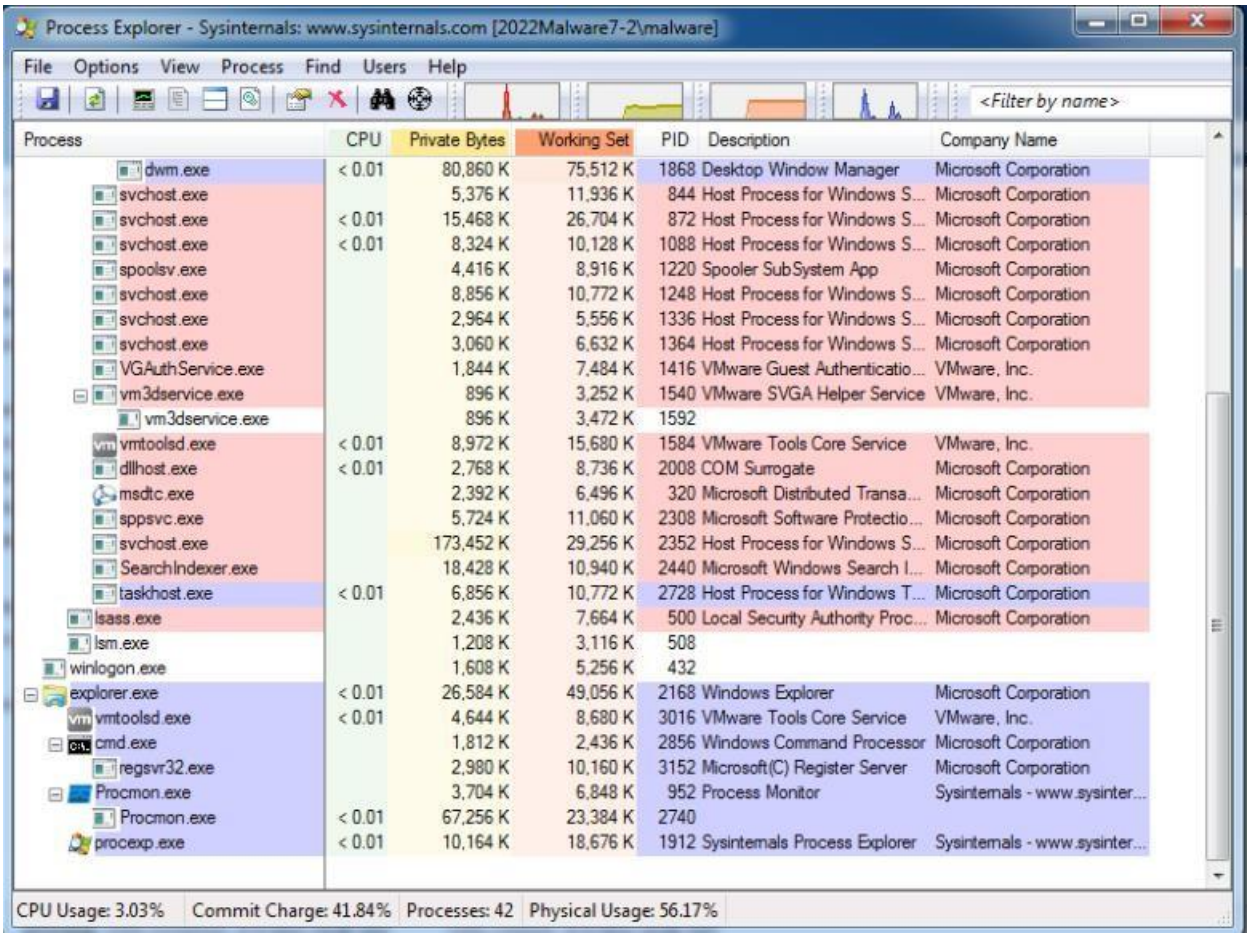


Figure 21: process explorer

6. Identify any persistence mechanisms employed by the malware.

After analyzing sample3.dll there was no persistence mechanisms observed with exception of the mutants stored in the Windows and System32 folders.

7. Describe what you did to overcome the obfuscation methods the program uses.

Directly overcoming the obfuscation methods issued by this malware was beyond my ability. To help alleviate overcome this obstacle and still analyze the malware virus total, JOESandbox Cloud, and Intezer Analyze were leveraged to help guide the direction taken. From there I used the tools available to the best of my ability to confirm sections of compromise.

Indicators of Compromise

There are several indicators of compromise for this malware sample including the obfuscation methods as well as anti-debugging techniques deployed. In addition there is observable interactions with WPAD, however; even more blatant is that several minutes after activation there is a significant uptick in network activity. These behaviors are the observed indicators of compromised of this malware.

To identify this malware a yara rule was created and tested for correct use. See below for the established rule:

```
rule creds_ru

{

meta:

        description = “simple YARA rule”

strings:

        $a = “Y;D$8t”

        $b = “GBBFRF”


condition:

        ($a and $b)

}
```