Reverse Engineering

COURSE WORK ONE

COURSE ID: KH6052CEM

INSTRUCTOR: DR. AHMED SELIM

NAME: AHMED FAROUK MAHMOUD

STUDENT ID: CU2000512

Contents

List of Figures

List of Tables

Section 1: Executive Summary

SHA-256 hash	2FAA2637FEDC5788A9598691581000BA637DB86C8505FDE1DC9D7DBCA8CC41CD

The sample PE is a keylogger malware sample first identified on Dec 14th, 2022. It is written in C, compiled with GCC and runs on the x86 Windows operating system. It records all keystrokes and implements a substitution cipher on the keystrokes them before sending them to a remote server at 192.168.100.156 on port 333 over multiple TCP connections.

YARA signature rules are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

Basic Information	
File Name	Sample.exe
Md5 hash	ebc19c36143ba91a6668a369e26d3c67
Sha-256	2FAA2637FEDC5788A9598691581000BA637DB86C8505FDE1DC9D7DBCA8CC41CD
File type	Win32 EXE
Target Machine	Windows x64
Is packed	False

Table 1 Basic Information

Section 2: Basic Static Analysis

Extracted Strings:

Some of the interesting strings found in the portable executable where					
[DOWN]	[UP]	[HOME]	[ESCAPE]	[CONTROL]	[BACKSPACE]
Socket	192.168.100.156	Send	GetAsyncKeyState		Connect
0123456789ELTFYBCDMSUXNWGVORHIAQPJZK			cos(7493) / cos(241); : %d		

Table 2 Interesting Strings

File Header Analysis:

Properties	Values
File Type	Portable executable (PE)
File can be executed	True
Compiler stamp	Tue Dec 13 16:56:30 2022 | UTC
Signature	PE00
Sections	13
Virtual and raw size	32E4, 3400 [hex] values are very similar (probably not packed)

Table 3 File Header Analysis

Imported Libraries:

KERNEL32.dll	msvcrt.dll	msvcrt.dll	USER32.dll	WS2_32.dll

Table 4 Imported Libraries

Used functions:

Some of the intersecting functions used in the portable executable				
IsDebuggerPresent	connect	htons	send	socket
GetAsyncKeyState	ExitProcess	Sleep	atexit	Cos & Sin

Table 5 Intersecting Functions

Summary of findings (Basic Static Analysis)

After analyzing the strings, file headers, imported libraries, and the used functions, I can **assume** that

1. The PE uses the GetAsyncKeyState function which allow the PE to know what keys where pressed.
2. The PE sends some data to a remote server the reason being the ws2_32 (sockets) library was imported and functions like socket, connect, and send indicate that sockets are being used. Also, I was able to extract this IP from the strings (192.168.100.156) further emphasizing my claim.
3. The PE uses the IsDebuggerPresent function which means that the program can I identify if there is a debugger running it.
4. The PE does some trigonometric equations using the sin and the cos functions, also I was able to extract “cos(7493) / cos(241)” from the strings further emphasizing my claim.
5. The file uses the sleep function witch according to malapi.io is commonly used for time-based evasion by adding delays in the code.

Basic Static Analysis screenshots

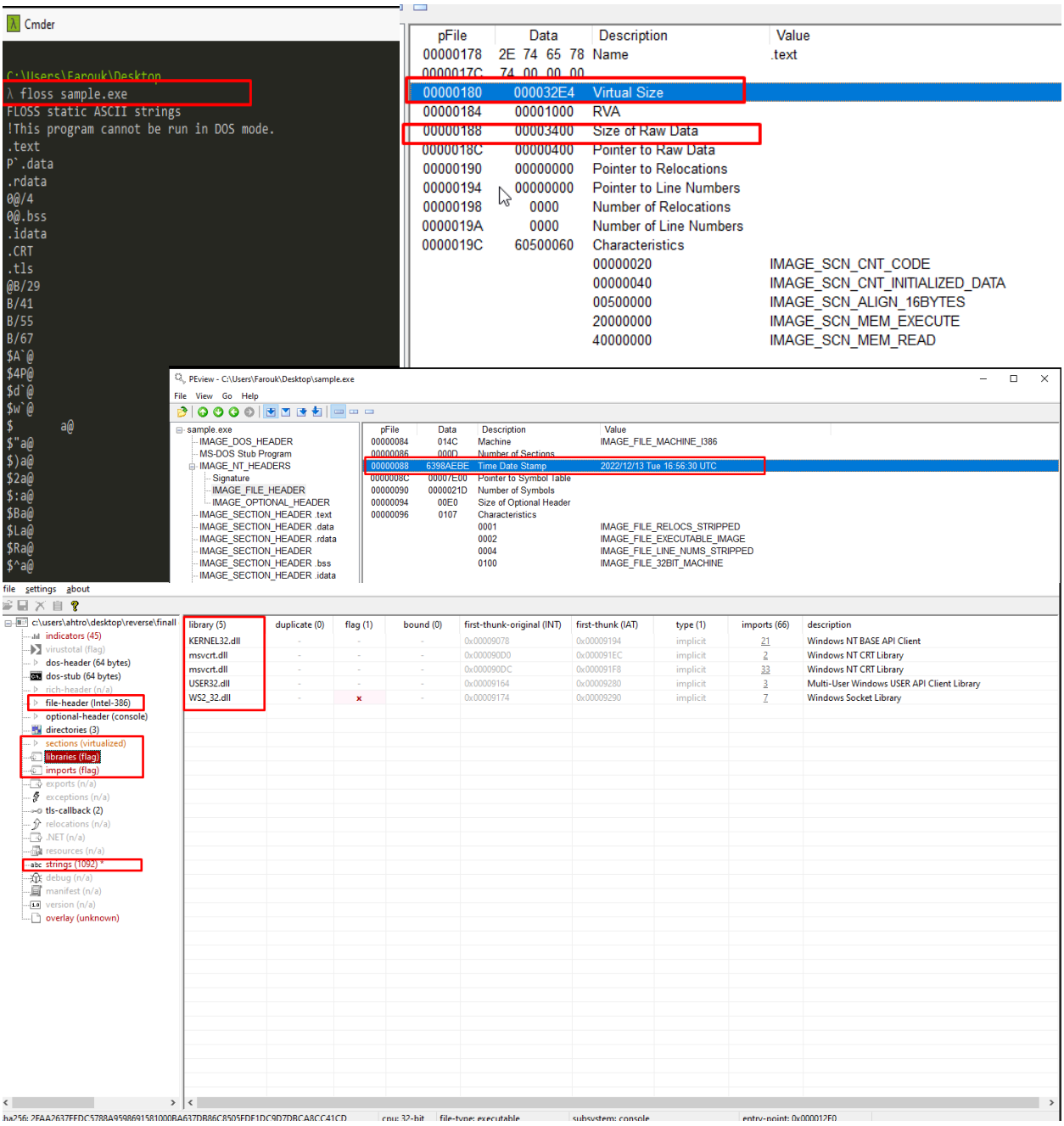


Figure 1 Basic Static Analysis Screenshots

Section 3: Basic Dynamic Analysis

Process Monitoring:

After running Procmon to monitor the processes, I was able to understand that the sample PE **does not** start other processes. Meaning that any action done by the PE is only done through the main process.

File Monitoring:

After deep analysis of the file system, I was able to see that the PE **does not** create, delete, or modify any files in the file system. Figure 2 below shows a screen shot of Procmon after applying a createfile filter on the PE process. Although there are many matches all of them have to do with the dll imported and **no new files were created**.

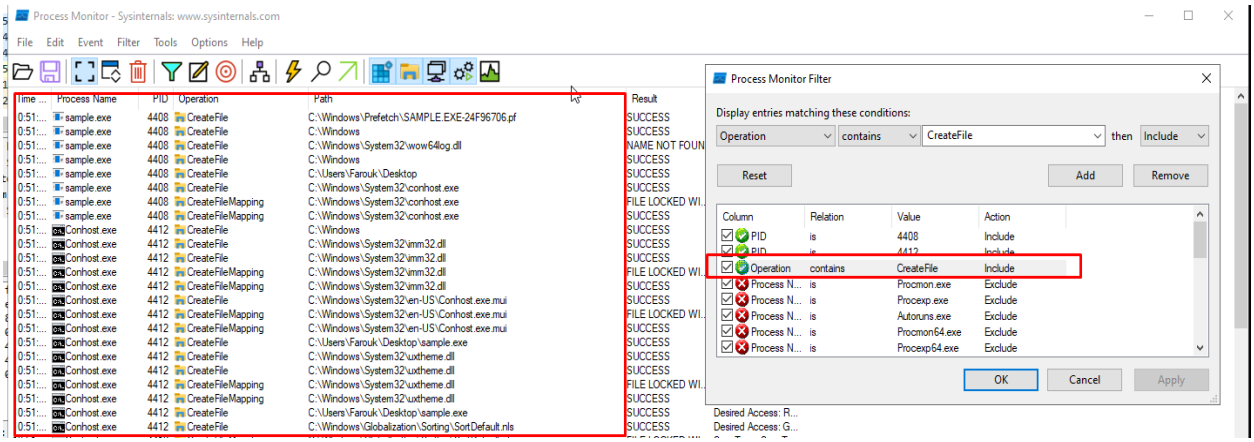


Figure 2 Basic Dynamic Analysis (File Monitoring)

Network traffic monitoring:

For the network monitoring section, I filtered the actions in Procmon to only display network related activities. As shown In Figure 3 below, The PE connects multiple times in the same minute to 192.168.100.156 on port 333. To further investigate I opened Wireshark and started analyzing packets sent and received from 192.168.100.156. I was able to see that every few seconds a new TCP connection is created, and the PE sends some bits of data when decoded in ascii it was sometimes [Down],[Control],[.], a random letter, or even a random number.

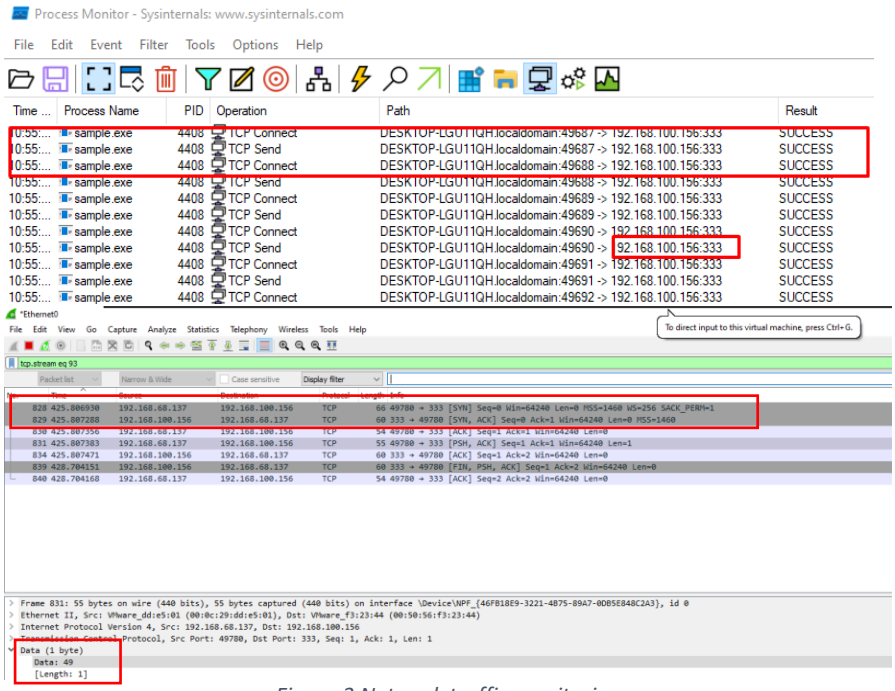


Figure 3 Network traffic monitoring

Summary of findings (Basic Dynamic Analysis)

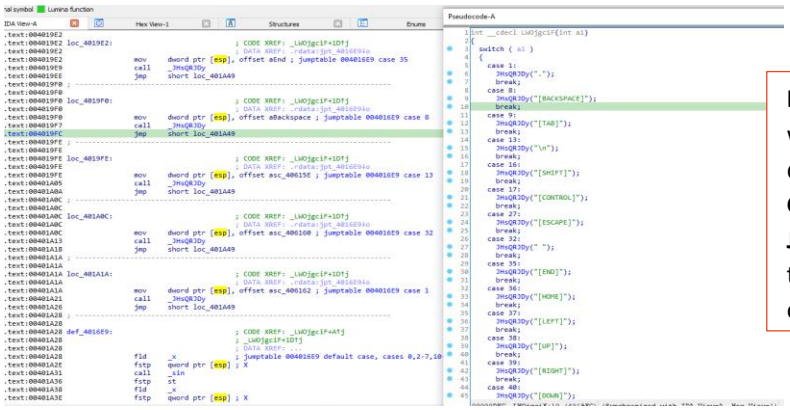
I was able to know from the basic static analysis done earlier that the PE has the ability of recording keystrokes using the function GetAsyncKeyState. And since their there were a lot of TCP connections every connection **holding a random letter or number**, I can assume that the sample PE is a keylogger that sends keystrokes in real time to 192.168.100.156:333. I was also able to prove my hypothesis by deeply analyzing the traffic with Wireshark and noticing that the more keys I pressed the more traffic generated; similarly, if no keys where pressed no traffic was generated. I can also assume that the keylogger does not store the keystrokes in a file since their where no file changes.

Section 4: Advanced Static Analysis

Main Function Analysis:

Figure 4 Main Function Analysis

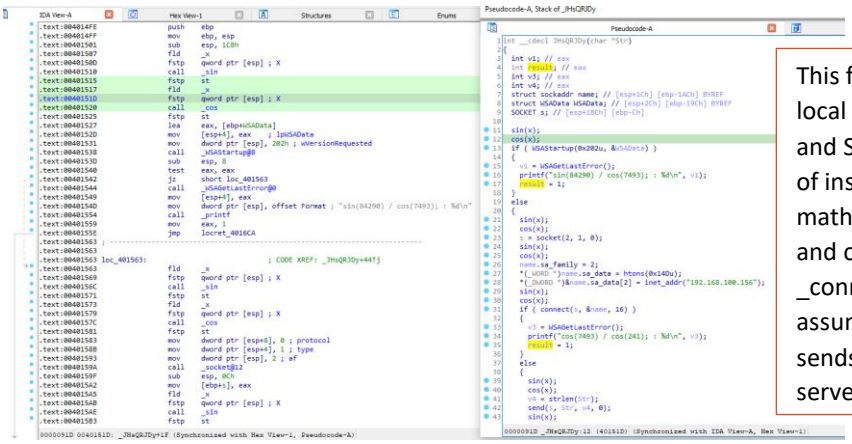
LWOjgciF Function Analysis:



LWOjgciF function contains a long switch with all the letters and numbers and depending on the return value of the **GetAsyncKeyState** this function calls the **JHsQRJDy function** passing **a static value** to it this value can be a **number**, a **letter**, or a different value like **[Shift].**

Figure 5 LWOjgciF Function Analysis

JHsQRJDy Function Analysis:



This fragment appears to define several local variables, including WSAData, s, and Str, and then contains a sequence of instructions that perform various math operations like by calling the sin and cos functions. It then calls the _connect and the _send functions. I can assume that this function connects and sends the keystrokes to the remote server.

Figure 6 JHsQRJDy Function Analysis

Summary of findings (Advanced Static Analysis)

After analyzing the **LWOjgciF**, **JHsQRJDy**, and the **main** function I can **predict** that the main function

- First, looks for a debugger and if there is a debugger the program will close. If there is no debugger the program will continue executing.
- Second, the program calls the **GetAsyncKeyState** function and passes the result to **LWOjgciF**
- Third, the **LWOjgciF** functions maps the input to a static number or letter or special characters and calls the **JHsQRJDy** function **for example the letter A is statically mapped to E**
- Finally, the **JHsQRJDy** function connects and sends to the server at IP 192.168.100.156 the newly mapped static number, letter, or special characters.

It is also very clear from the function names that the developer is trying to obfuscate the code by naming the functions wired names. It also appears that the trigonometric equations preformed do not severe a real purpose and may be a form of logical obfuscation.

Section 5: Advanced Dynamic Analysis

For this section I used x32 debugger to debug the sample PE and try to prove the 3 points claimed above in the advanced static analysis findings section

Claim One: (The main function checks if there is a debugger)

I was able to prove that this is true because the program closes itself while running it in a debugger. to overcome this part, I added a break point just before the function IsDebuggerPresent at address (00401A89) and stepped over 2 steps to the *test eax,eax* and edit the EAX registry from 1 to 0 bypassing the next jump command.

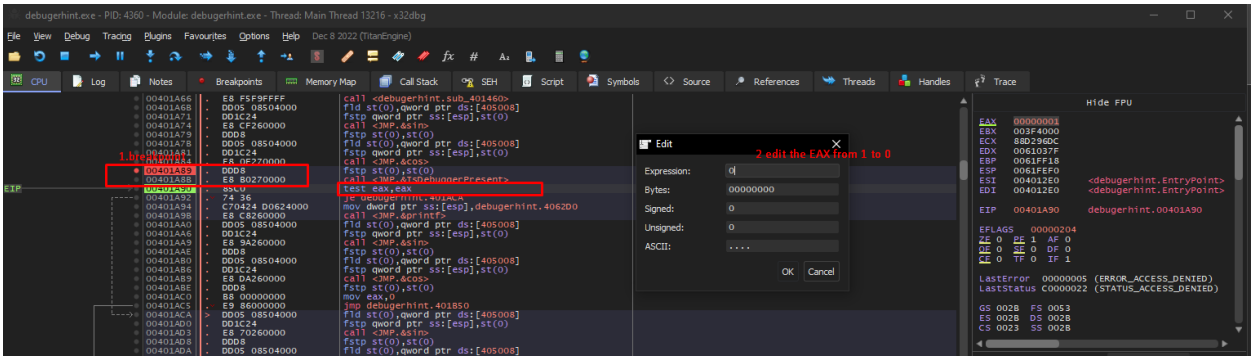
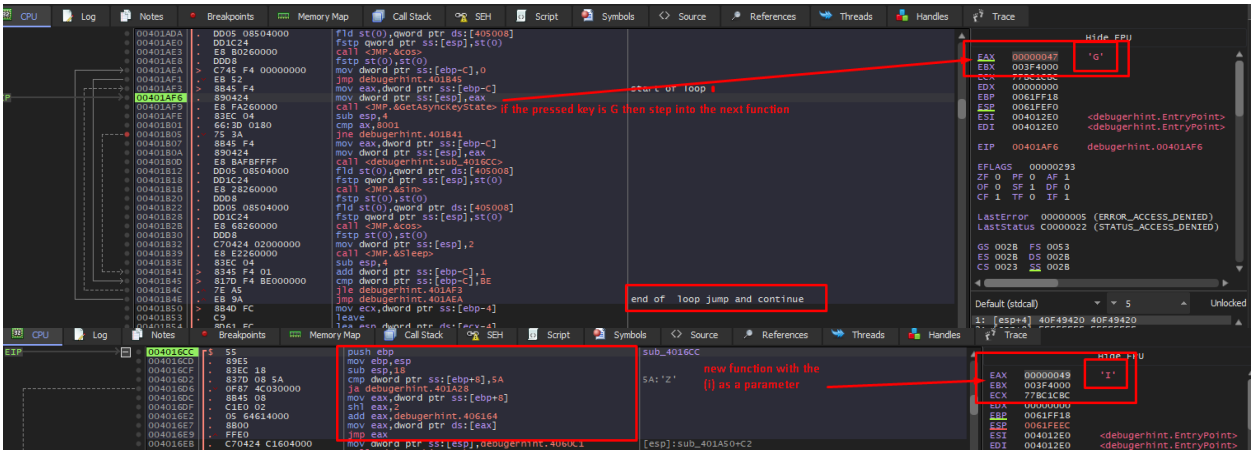


Figure 7 Claim One (Debugger)

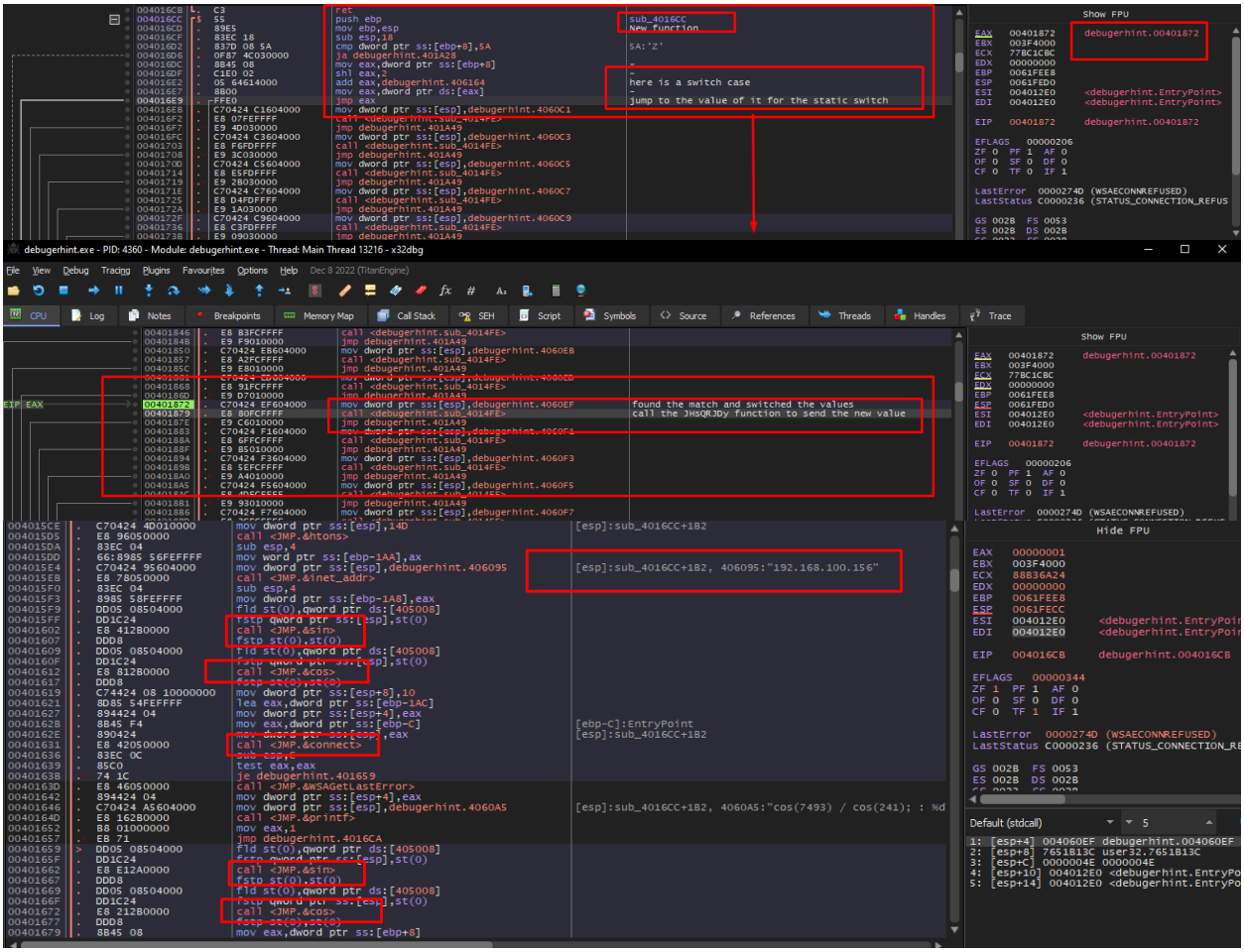
Claim Two: (The program calls the GetAsyncKeyState function and passes the result to LWOjgciF)

For this claim I was not completely correct as it appears form debugging the main function that there is a loop looping over all the ascii characters and if the key pressed the same as the ascii character then the program then steps into the new function **LWOjgciF** passing the pressed key.

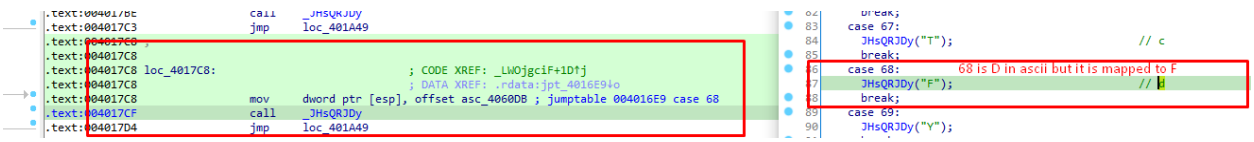


Claim Three: (LWOjgciF functions maps the input to a static number or letter or special characters and calls the JHsQRJDy function that is used to send to the server)

For this claim I was correct, it is seen in the EAX registry that the input key gets changed. Before sending it to the **JHsQRJDy** function which calls the connect and send function to send the keystroke to the server. It is also visible that the sin and cos functions get called multiple time without doing anything with the output confirming that the trigonometric functions are used for only for obfuscation.



After proving that the **LWOjgciF** functions preforms substitution cypher I returned to Ida and to decrypt the substitution cipher. Bellow is a sample of how I decrypted the cypher. The full table of the decrypted cypher can be found in the Appendices section.



Appendices

A. Decrypted Cypher

Keystroke	Mapped to	Keystroke	Mapped to
A	E	**N**	W
B	L	**O**	G
C	T	**P**	V
D	F	**Q**	O
E	Y	**R**	R
F	B	**S**	H
G	C	**T**	I
H	D	**U**	A
I	M	**V**	Q
J	S	**W**	P
K	U	**X**	J
L	X	**Y**	Z
M	N	**Z**	K

B. Yara Rules for detecting this malware

```
rule CW_keylogger{

    meta:

        last_updated = "2022-12-15"

        author = "Farouk"

        description = "A keylogger that infects windows operating systems"

    strings:

        $string1 = " 0123456789ELTFYBCDMSUXNWGVORHIAQPJZK" ascii

        $string2 = " 192.168.100.156" ascii

        $PE_magic_byte = "MZ"

        $suspicious_hex_string = {4465 6275 6767 6572}

    condition:

        $PE_magic_byte at 0 and

        ($string1 and $string2) or

        $ suspicious_hex_string

}
```