Project

Justin Langevin

8648380

Conestoga College

SECU74000: Root-Kits & Hacking

April 12, 2024

Project.doc

Executive Summary

This piece of malware was found on a classmate's computer, the classmate states they were sent the word document over email. Once opened, this Word document creates a scheduled task on the computer that starts at 8:45 AM every morning. This task has the functionality to contact a server to receive an executable payload at the assigned time. It then runs this payload, which from there captures 200 keyboard inputs by the user as well as the name of the computer. It then proceeds to sends this information back to the server. This allows the hacker to look for any useful information inputs captured. It is evident that this malware has keylogger capabilities and all the computers at the school should be examined for this malware.

Detailed Analysis

Program Details

The analysis of the Word document called Project.doc started with a hash value check on the internet to identify if this malware was detected anywhere else before, the hash values for Project.doc and q.exe can be found in Table 1. This piece of malware had not been detected at this current point in time. The second step was to extract all the strings from the document and analyse them. The strings indicated potential base64 encoded strings and the use of macros. The Strings can be found in Appendix A Figure 1.

Filename	Hash values
Project.doc	Md5 92a943c0531baa35a6764a208c820420 Sha1 7f812d23068c7d6633eab6a72413bcecd7350370
q.exe	Md5 357e7d833cf500432aa809fec4b4a7b6 Sha1 a832b7c7f37b14965bc2180aa9338804483492fc

Table 1 file hash values

Static Analysis

To further the investigation, the next step was to look for proof of the potential macros seen in the strings called **khhzrz**. To do this, I used a PDF parser tool check for the existence of macros in the file. This returned that there were macros found in the Word document. From there, I pulled all the visual basic application (VBA) code from the macro and started to take a deeper look. This can be found in Appendix A Figure 2 & 3. The VBA code that was extracted from the document was obfuscated so to gain a deeper understanding of its purpose I needed to first deobfuscate the code. After deobfuscating the VBA code, I've found that it reads base64 text that has been changed to the colour white so that it is hidden, this can be seen in Appendix A Figure 6 and Figure 7. After decoding the string, it was clear that this was a malicious script

that uses the forfiles command, this command is used to search for files within the C:\\WINDOWS\\system32 directory. For each file that this command finds it executes a command. This command includes a PowerShell script that downloads a file from a remote server and saves it as q.exe in the user's home directory, and from there It then runs the q.exe executable. The script then establishes persistence by creating a new registry item. This script can be seen in Appendix A Figure 8.

The second piece of functionality I found after deobfuscation is a second harmful command that preforms several actions. The first is it pings local host 30 times and clears the screen, this acts as a 30 second timer to create a delay. The second action it does is it creates a task called **MicrosoftWin32** that executes another PowerShell command **daily at 8:45AM**. This command seems to execute a command that retrieves a registry value. Once it has received this value it is then decoded from base64. This all can be seen in Appendix A Figure 4 and Figure 5.

The next step was for me was to analyse the capabilities of the q.exe. to analyse the executable. I first looked at the strings as seen in Appendix A Figure 9. These strings displayed the use of .Net Framework and HttpWebRequests which indicates it was written using C#. Because of this, the next step for me was to open the executable using dnspy which would allow me to look at the q.exe code in plane text, this can be seen in Appendix A Figure 10. This revealed to me that the plain text code was very obfuscated so to further do my analysis I first needed to deobfuscate it. Once deobfuscated the capabilities of this payload became clear to me. These capabilities resembled those of a keylogger. To explain the functionality, this malicious executable first hides the console windows on the system. From there, it sets up a keyboard hook and then enters a never ending loop using Application.Run() to ensure the program keeps running. This keyboard hook is then bound with the callback method of SendToServer as seen in Appendix A Figure 11 and Figure 12. The SendToServer method is where we can see the keylogging functionality. This Method check if a key press event is intercepted if so, it then adds the keys pressed to a string called capturedKeystrokes when the length of this string reaches 200 characters it is then sent to the URL of **xttp://telemetry.securityresearch.ca/t** along with the name of the machine and the keystrokes captured. From there, it is then formatted and sent to the server using a POST request. This can be seen in Appendix A Figure 13 and 14.

Dynamic Analysis

Once I had concluded my static analysis and had a good understanding of the capabilities of the malware, it was then time to get proof of its capabilities. To start the dynamic analysis, I first checked to see if the endpoint that the malware was sending back keystrokes to was still up and running. After making a fake post request, I received a 201 request back from the server which told me that this request was successful and new resources had potentially been created

on the endpoint. I also recreated the same thing using Postman which gave me the type of server being used to receive information. The Server being used was a Nginx Server running version 1.20.2. This can be seen in Appendix A Figure 15 and Figure 16. From there, I opened up Procmon to investigate the capabilities of the Project.doc file. In Appendix A Figure 17, we can see the process tree. In this process tree, we can see the use of CMD.exe, PING.exe, and Powershell.exe. From there, I included all these processes into the process monitor, this displayed to me the ping timer that lasted for 30 seconds as seen in Appendix A Figure 18. After the ping timer, you can see Powershell.exe script as well as cmd.exe create the scheduled task called MicrosoftWin32 as seen in Appendix A Figure 5 and Figure 19. The next step was for me to investigate if I missed any functionality of the Project.doc file. I did this by opening regshot and took my first shot before I ran the malicious document. From there, I compared the outputs. This displayed to me that there were 787 changes. To dive deeper into the findings, I looked through the text file to find proof of the scheduled task that was used established persistence as seen in Appendix A Figure 21, 22, 23, 24. From there, I went to go look for the MicrosoftWin32 scheduled task, this can be seen in Appendix A Figure 25. As we dive deeper into the action of this scheduled task, we can see that its is run at 8:45 AM as seen in the static analysis. This ensures persistence of the q.exe keylogger and is run in the morning to give it the best chance at logging credentials on the machine. The next step was for me to check the Fakenet logs these logs allowed me to see the network activity attempts the **xttp://telemetry.securityresearch.ca/t** domain as seen in Appendix A Figure 26.

Conclusions

To conclude this investigation of the malware based on the hash values displayed in Table 1, it is evident that this malware creates a scheduled task called MicrosoftWin32 to establish persistence that then runs a PowerShell script at 8:45 AM to receives an executable called q.exe from the domain of **xttp://telemetry.securityresearch.ca/t**. This payload has the capabilities of a keylogger, it first collects 200-characters worth of keystrokes as well as the name of the system. This information is then sent back with a post request to the server. This payload can be found within the home directory on the user's computer. Therefore, it is evident that we should take precautions to further investigate the computers at the school to ensure that this malware has not been installed on any other systems.

Appendix A – Collective Evidence

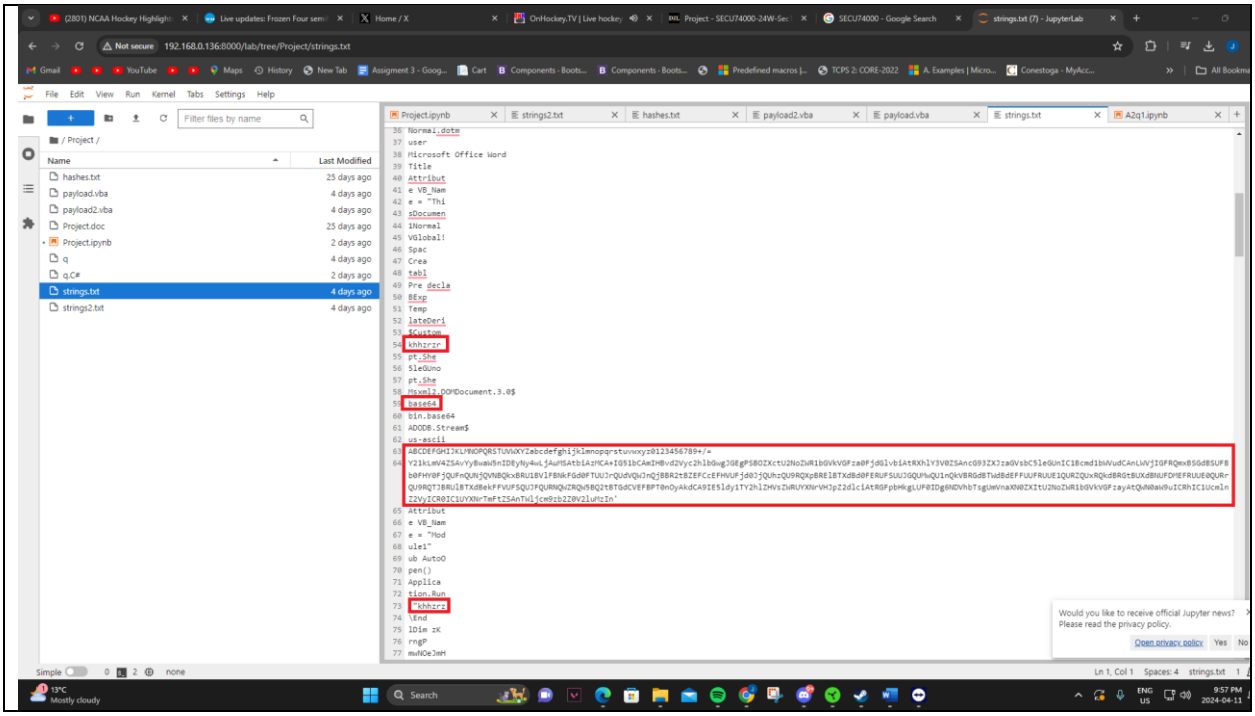

Figure 1 base64 string khhzrzr macro name used in another assignment.

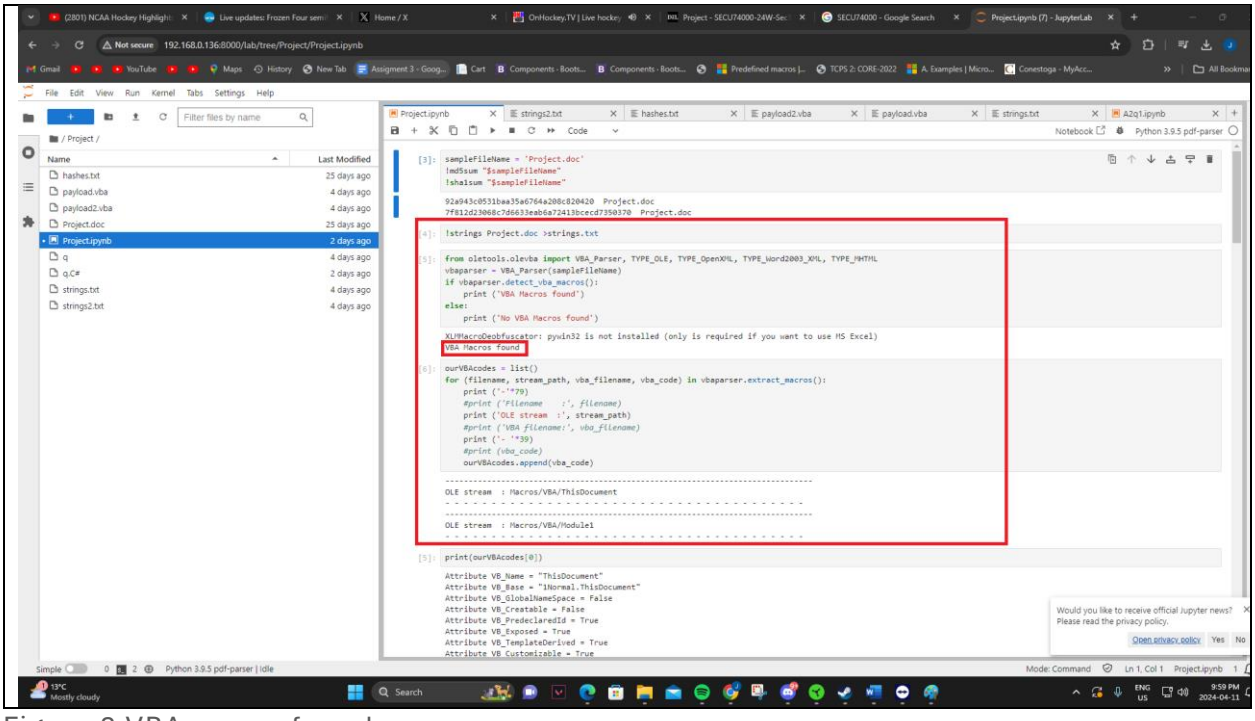

Figure 2 VBA macro found.

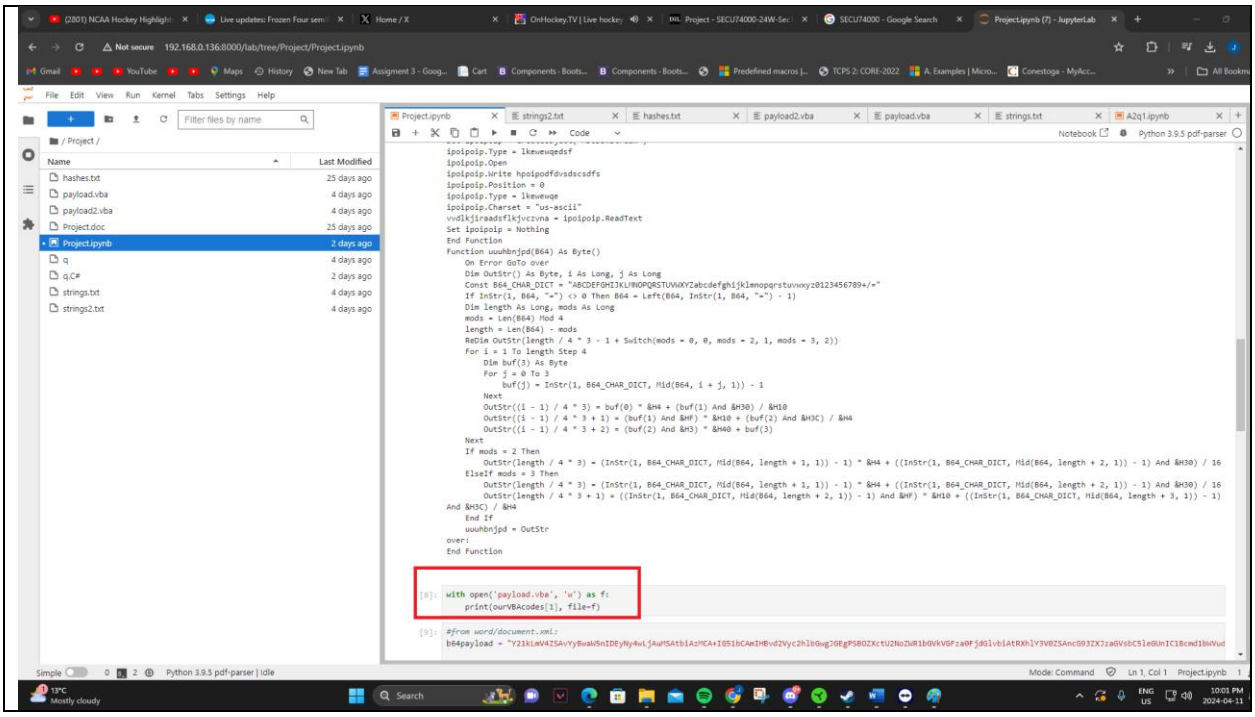

Figure 3 Extracting of macros in Project.doc.

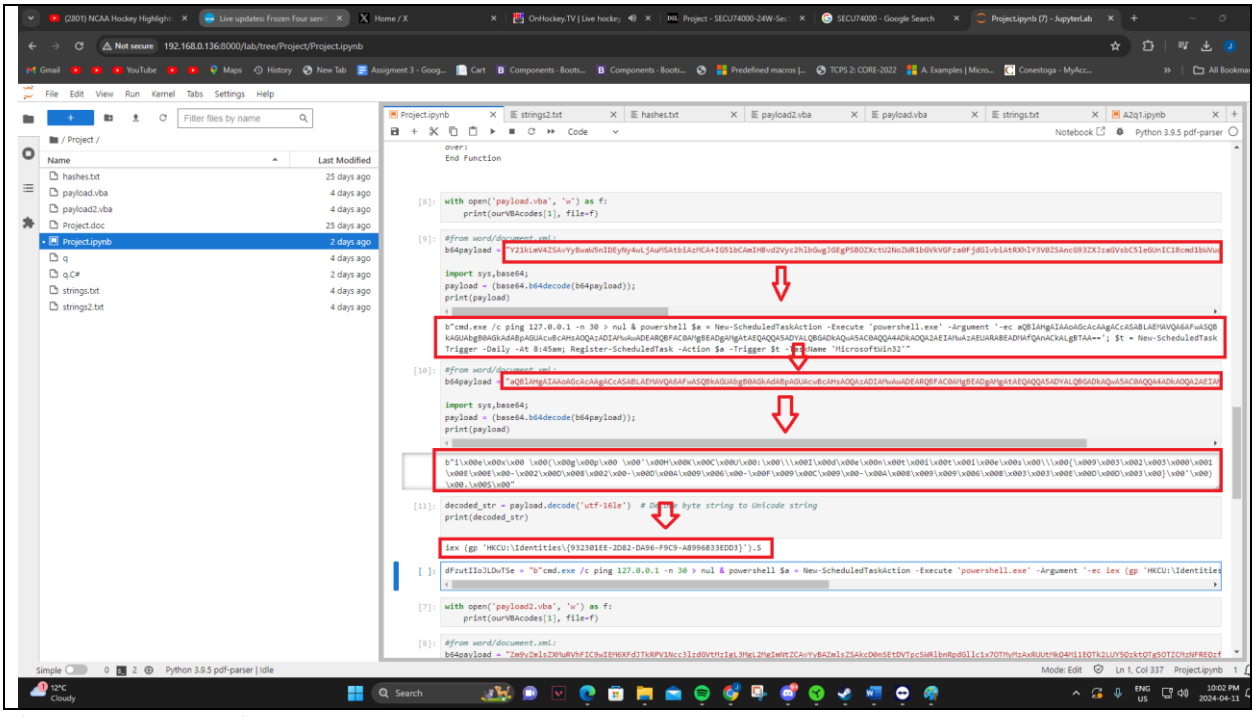

Figure 4 conversion of base64 and utf-16le.

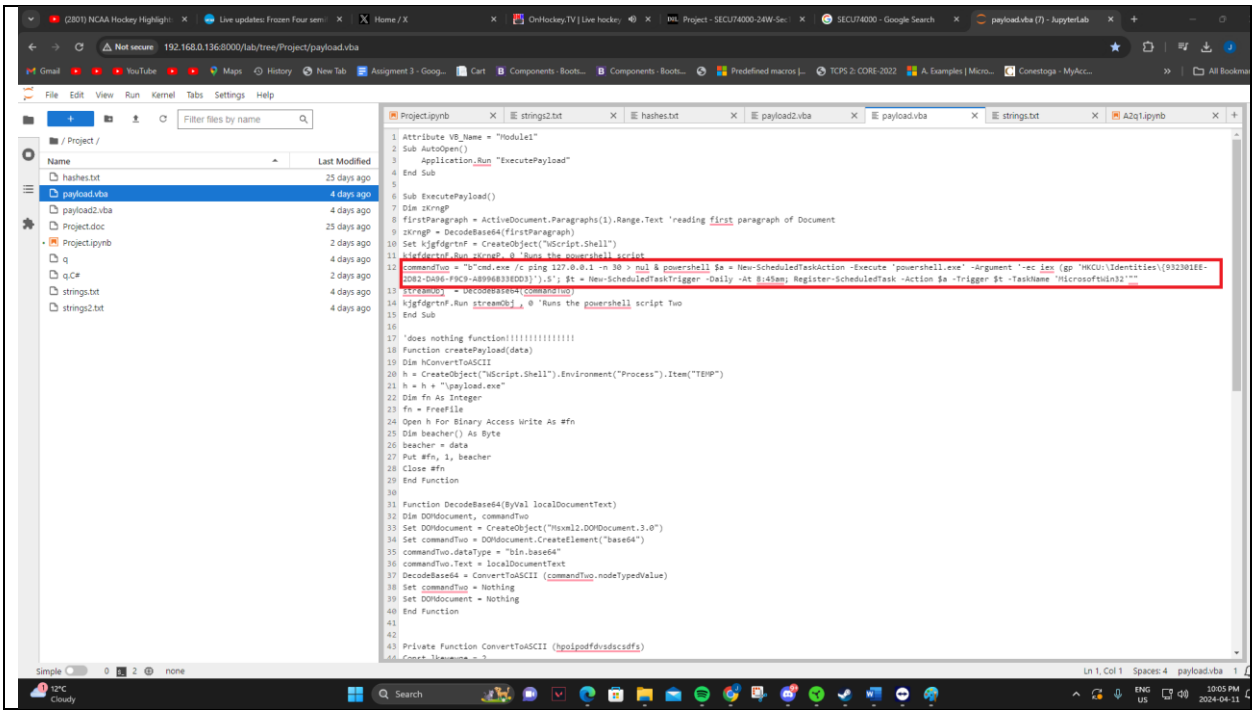

Figure 5 the converted string.

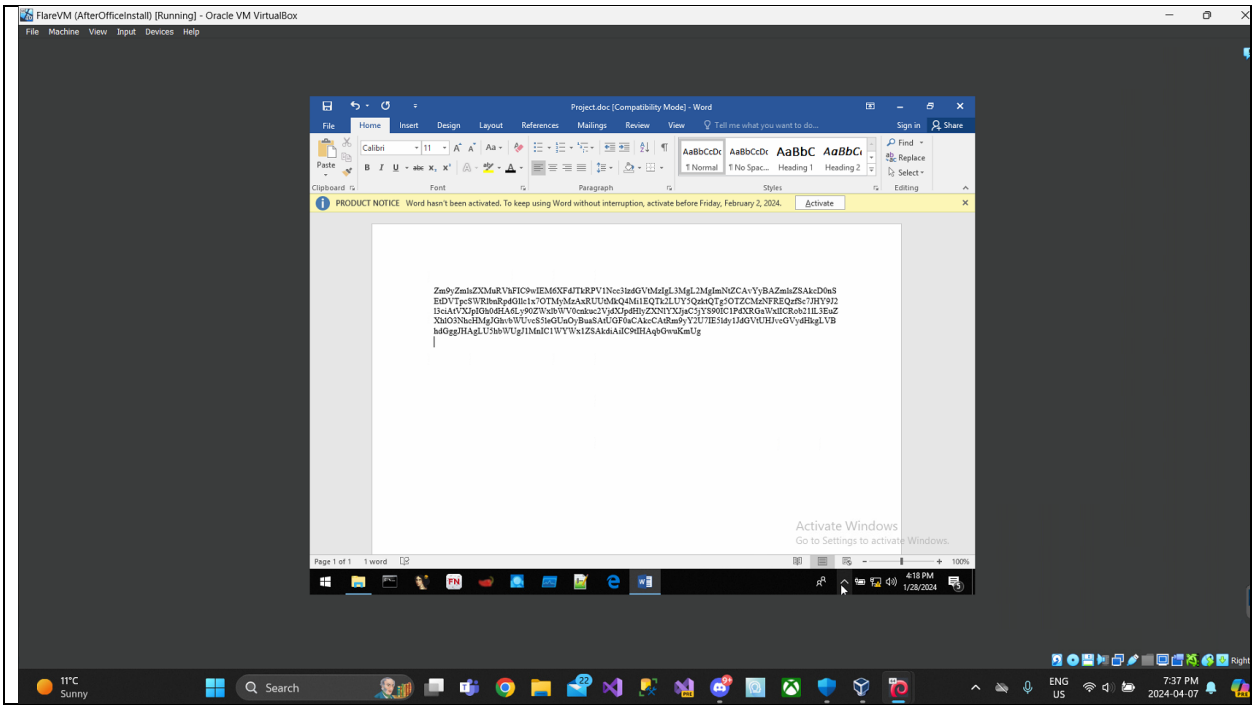

Figure 6 the Word document contents base64 hidden string.

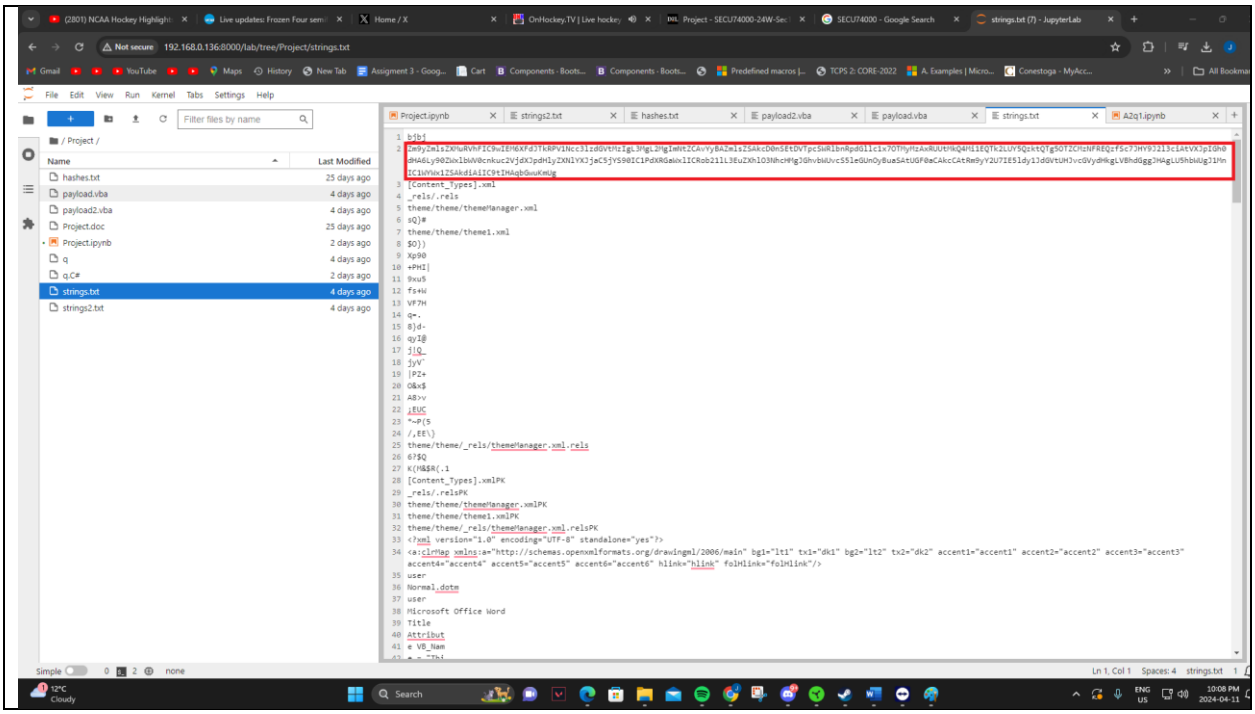

Figure 7 the string found within the word document in the strings.txt file.

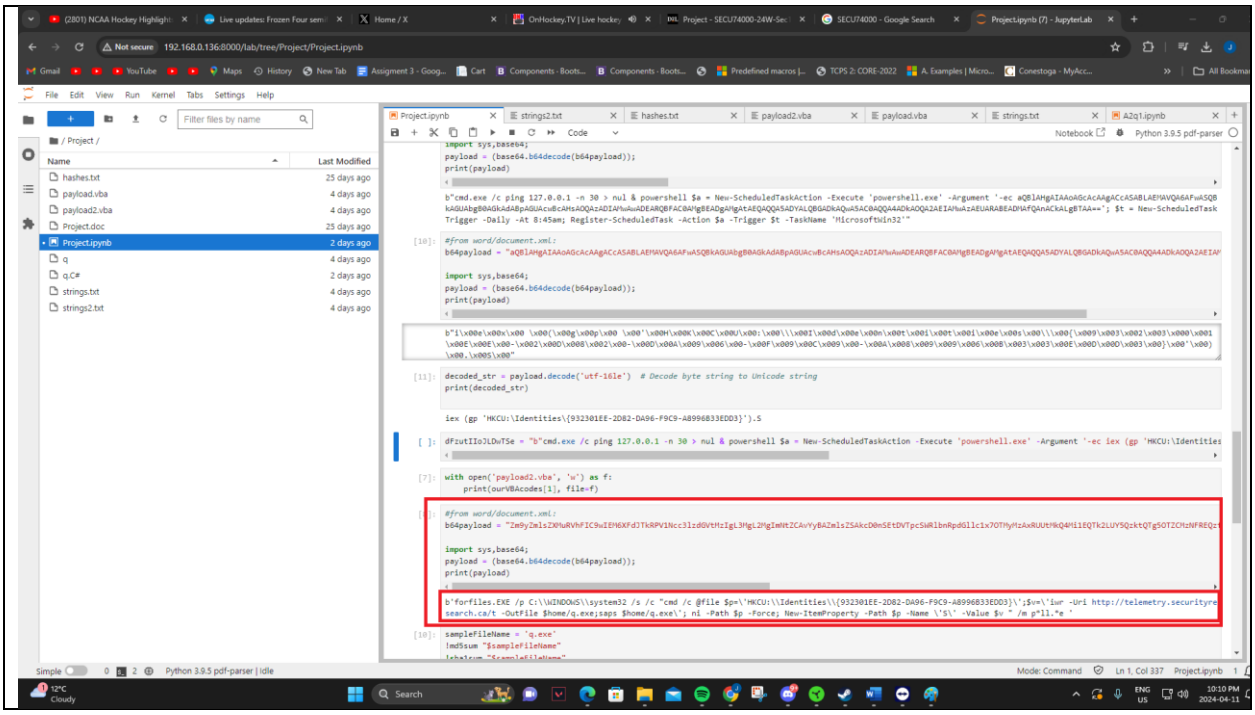

Figure 8 the decoded base64 hidden string in the word document.

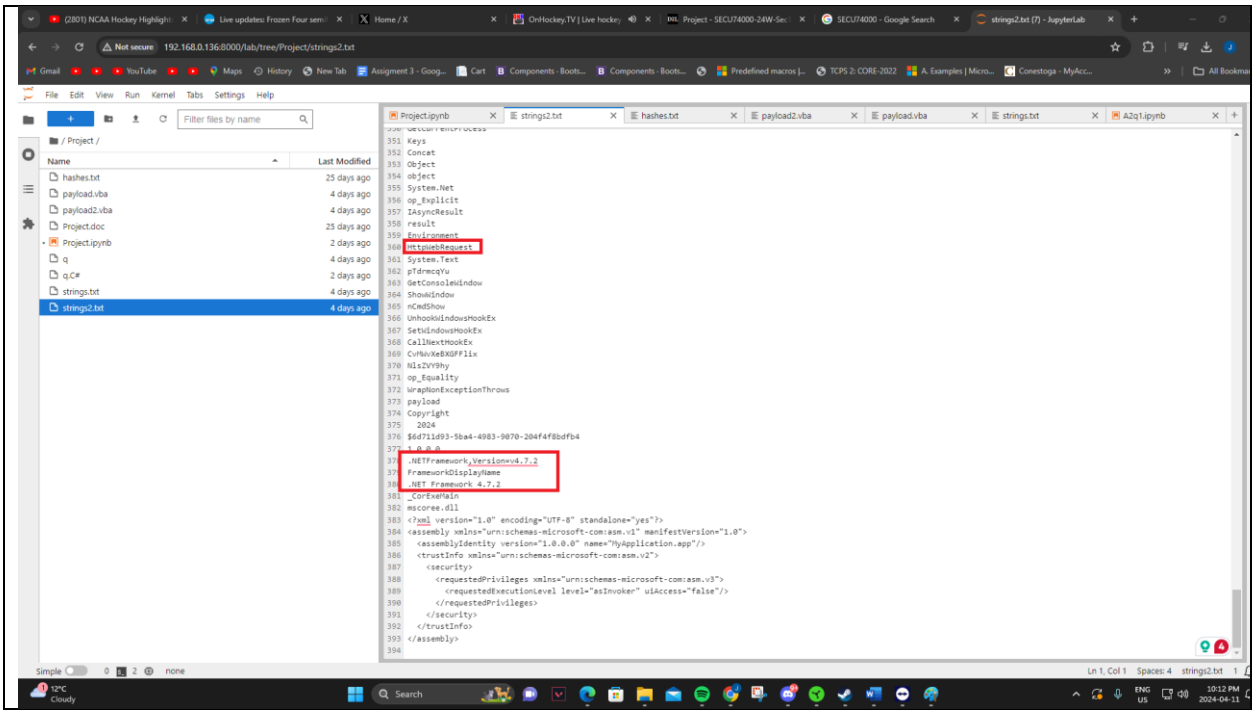

Figure 9 the q.exe strings and .net/network indication.

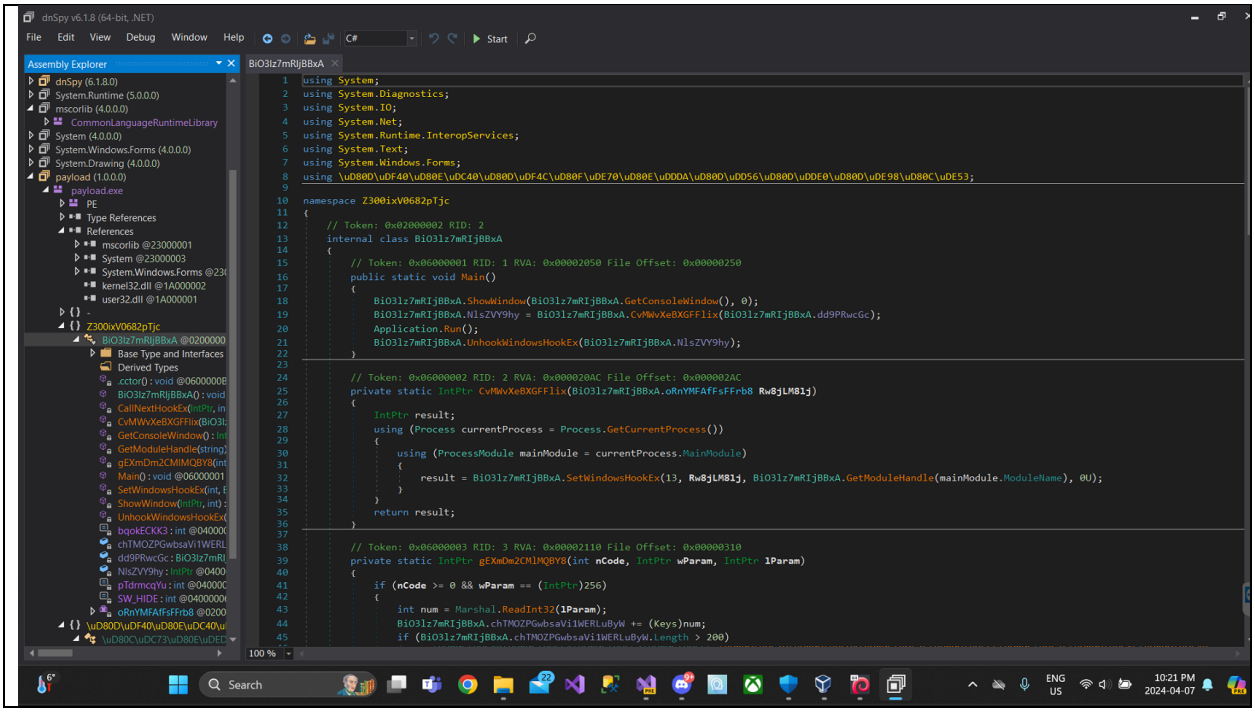

Figure 10 q.exe open in dnspy.

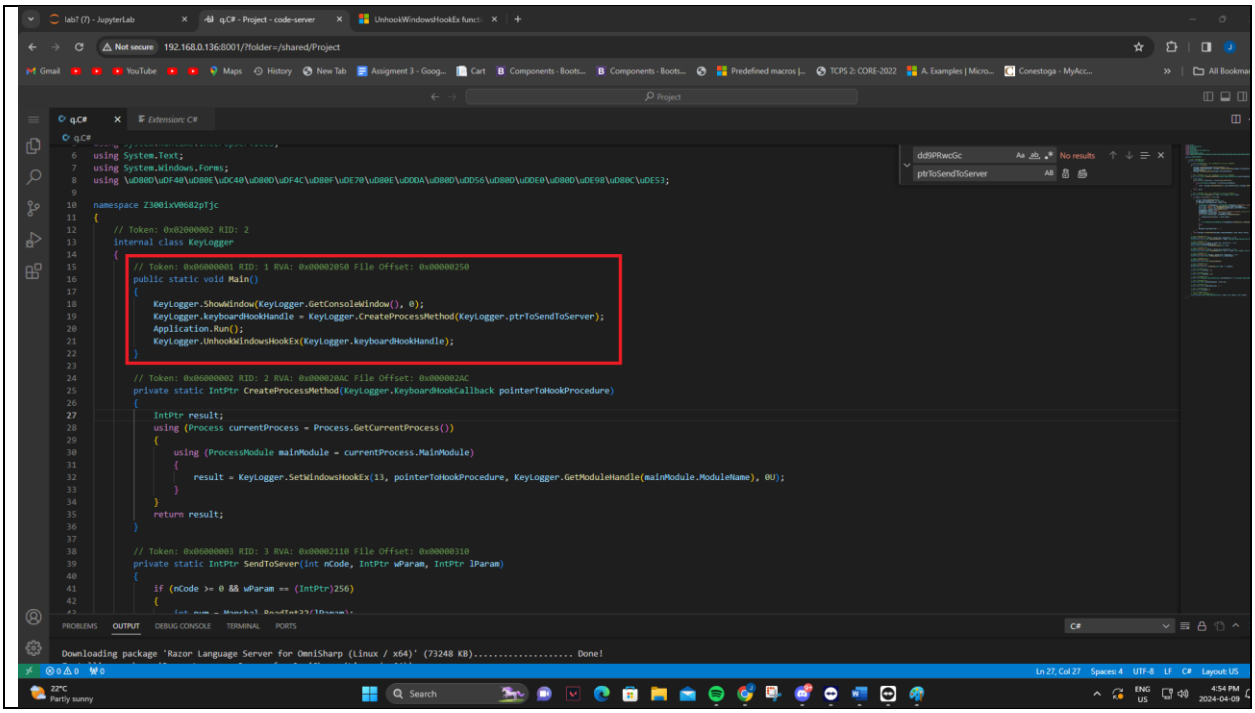

Figure 11 the q.exe main function deobfuscated.

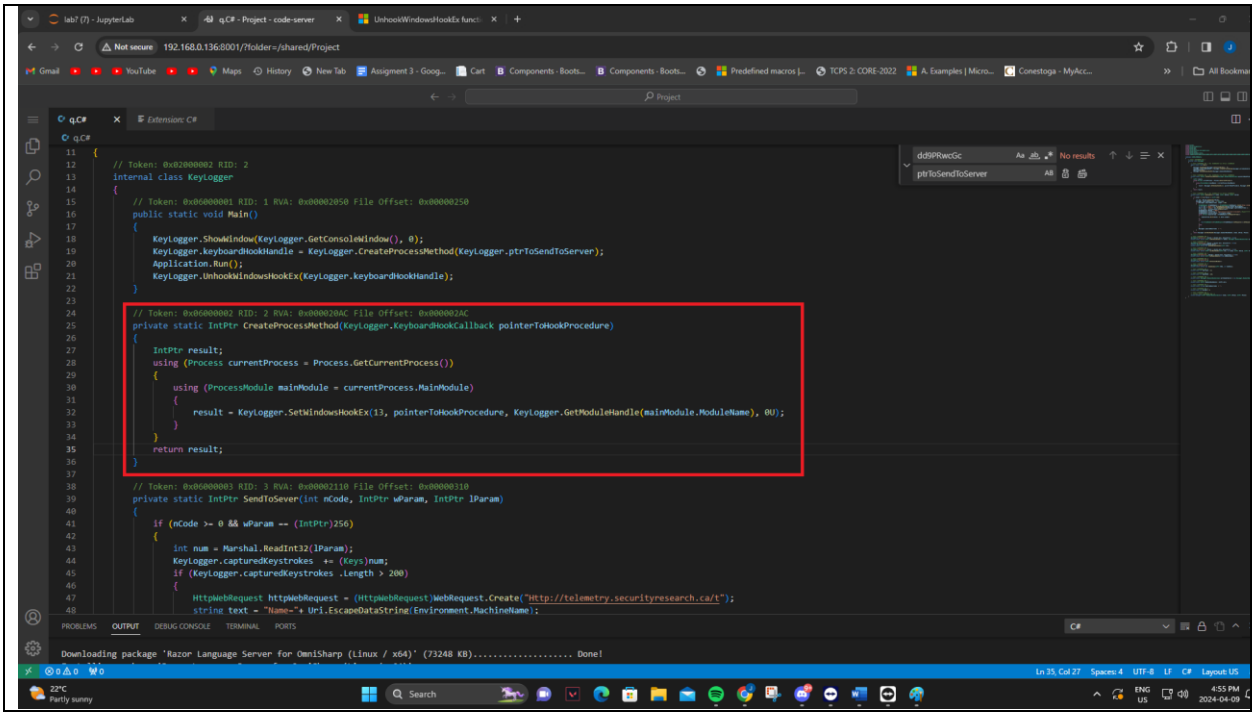

Figure 12 The q.exe process keyboard hook.

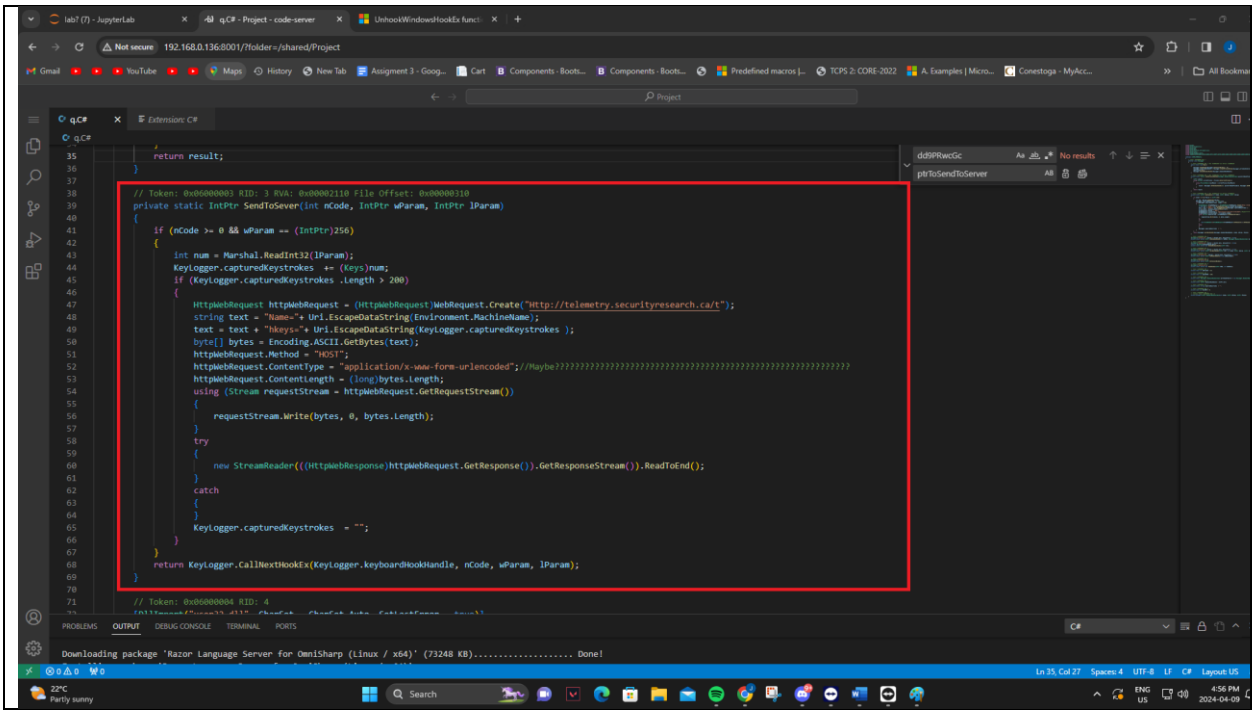

Figure 13 the q.exe HttpWebRequest function deobfuscated.

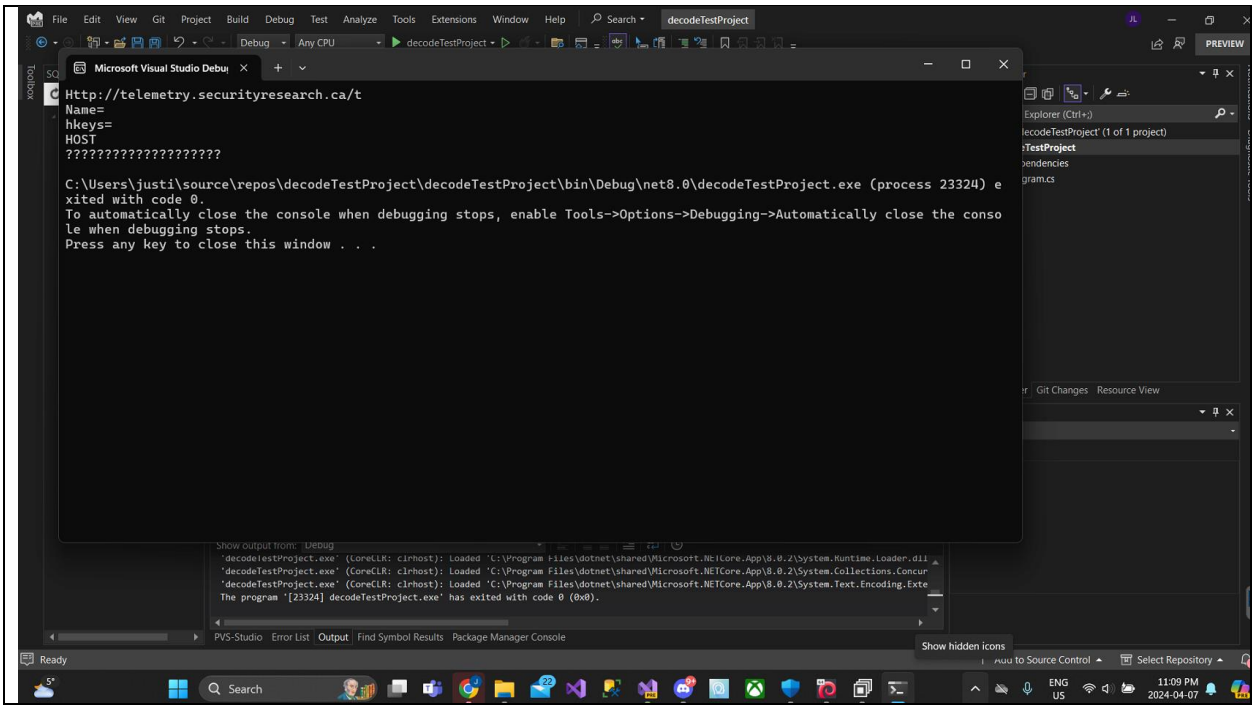

Figure 14 decoded the Unicode strings from q.exe.

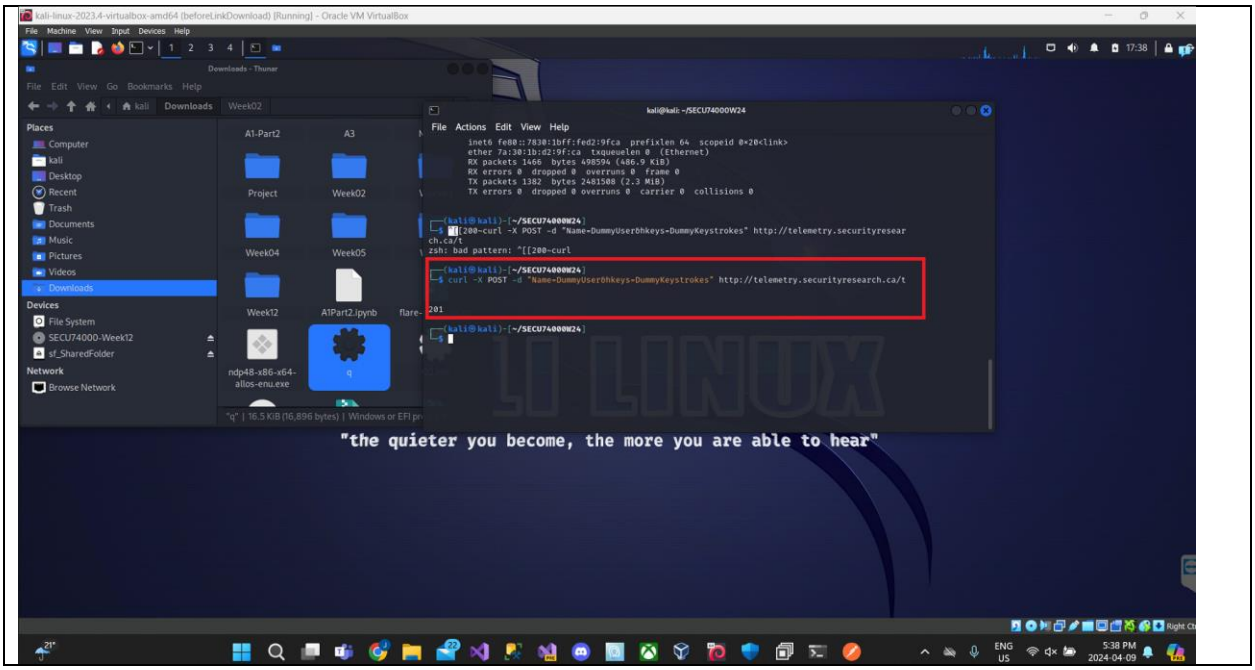

Figure 15 post request 201 to keylogging server.

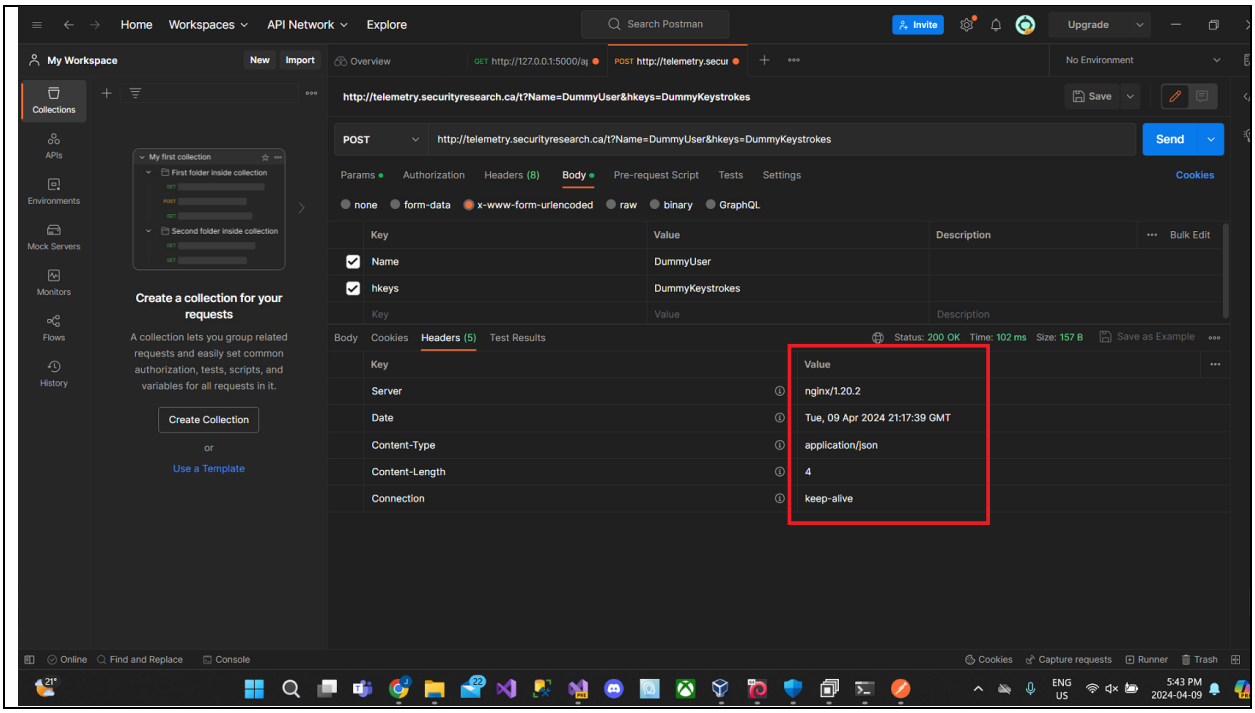

Figure 16 Postman information about the server.

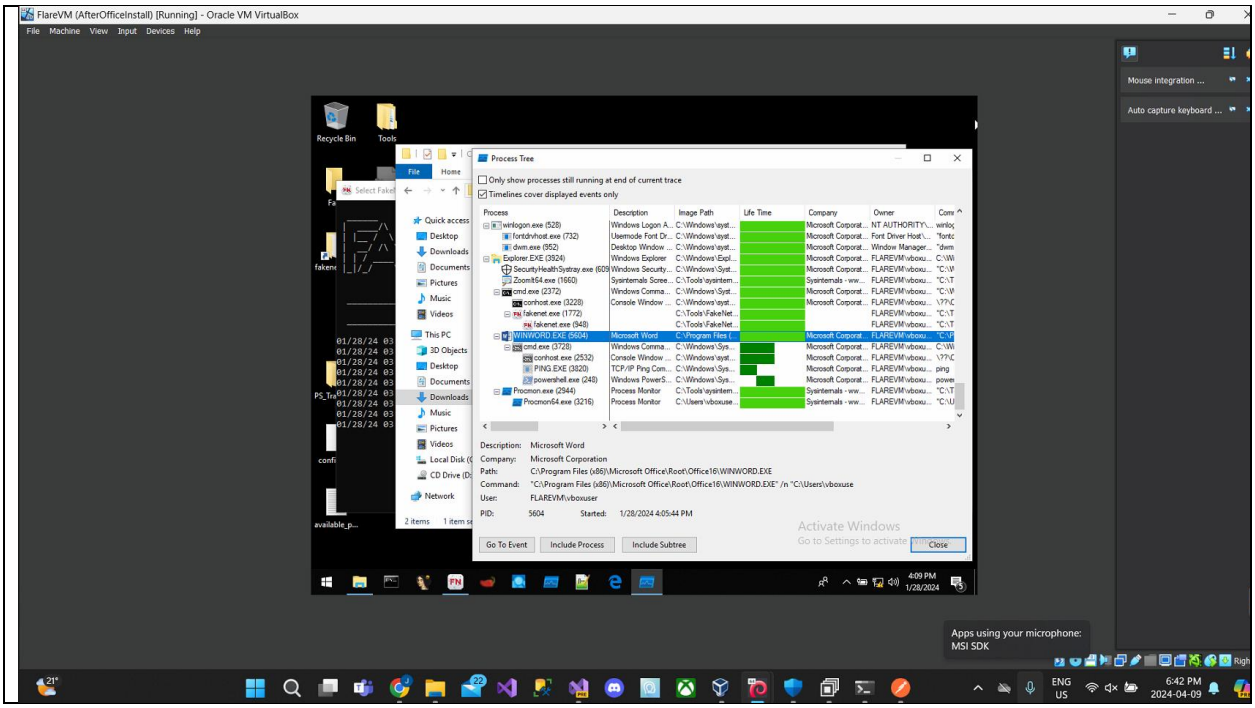

Figure 17 Procmon Project.doc process tree.

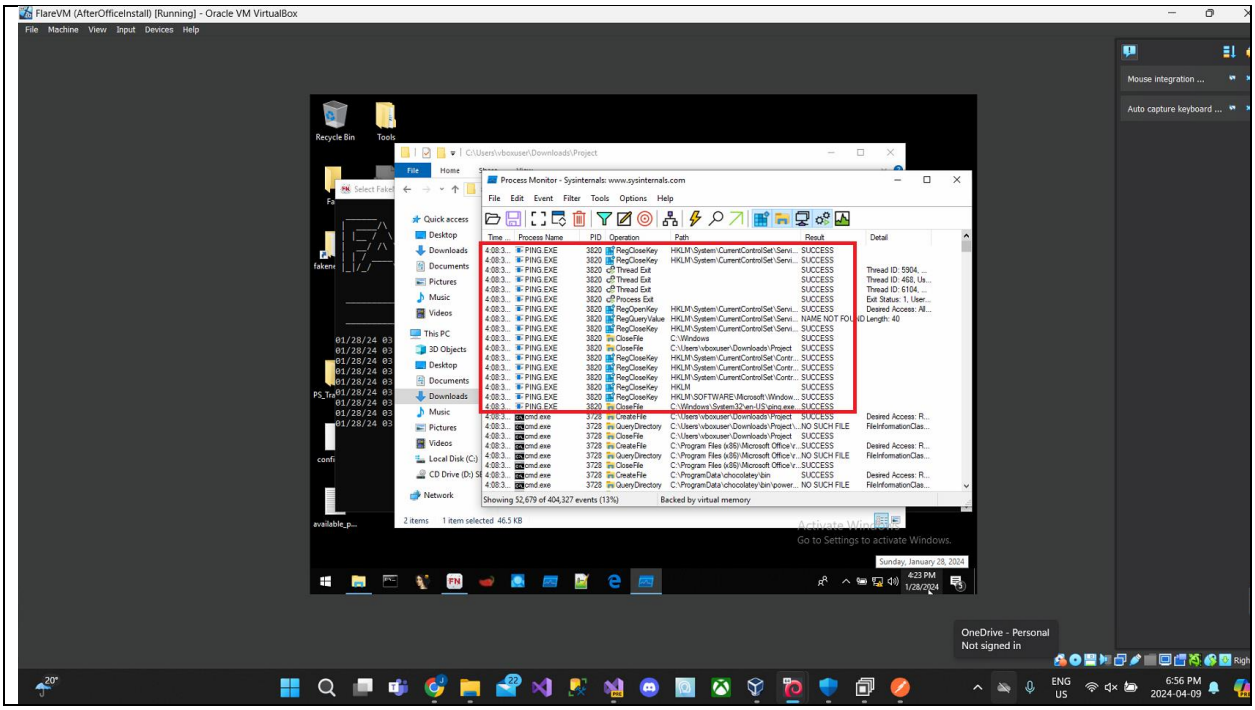

Figure 18 Procmon showing pings timer.

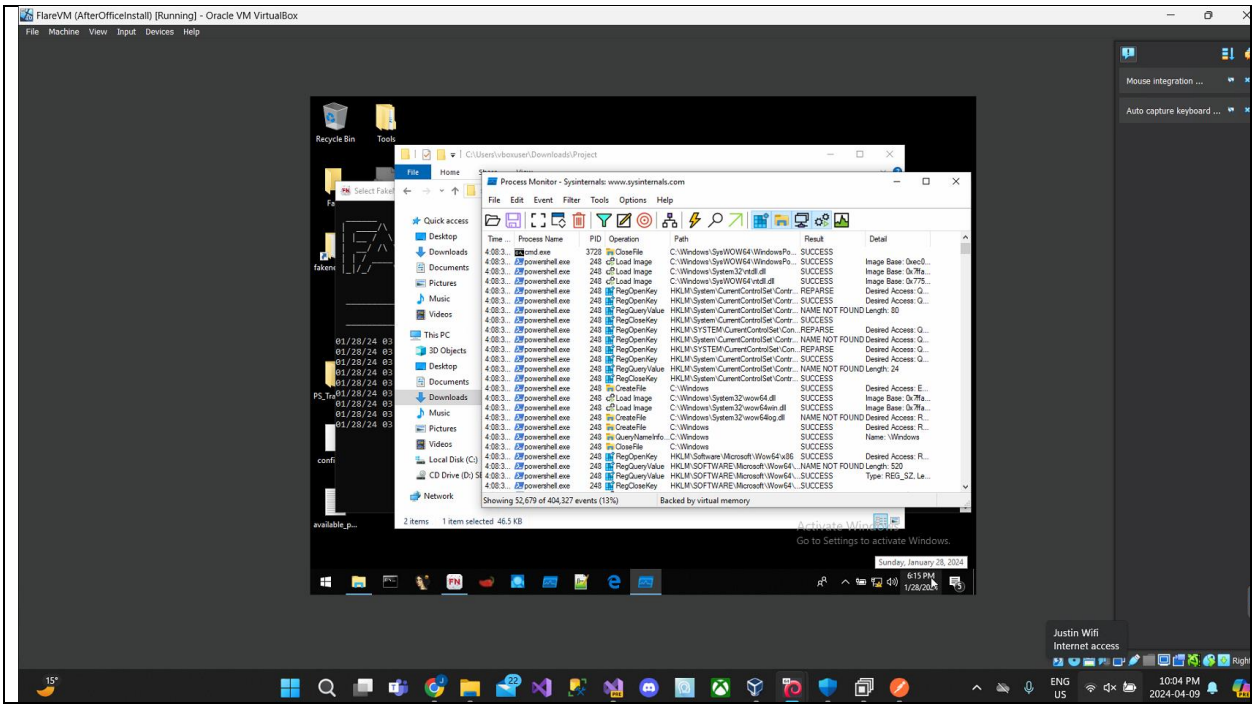

Figure 19 PowerShell Procmon output.

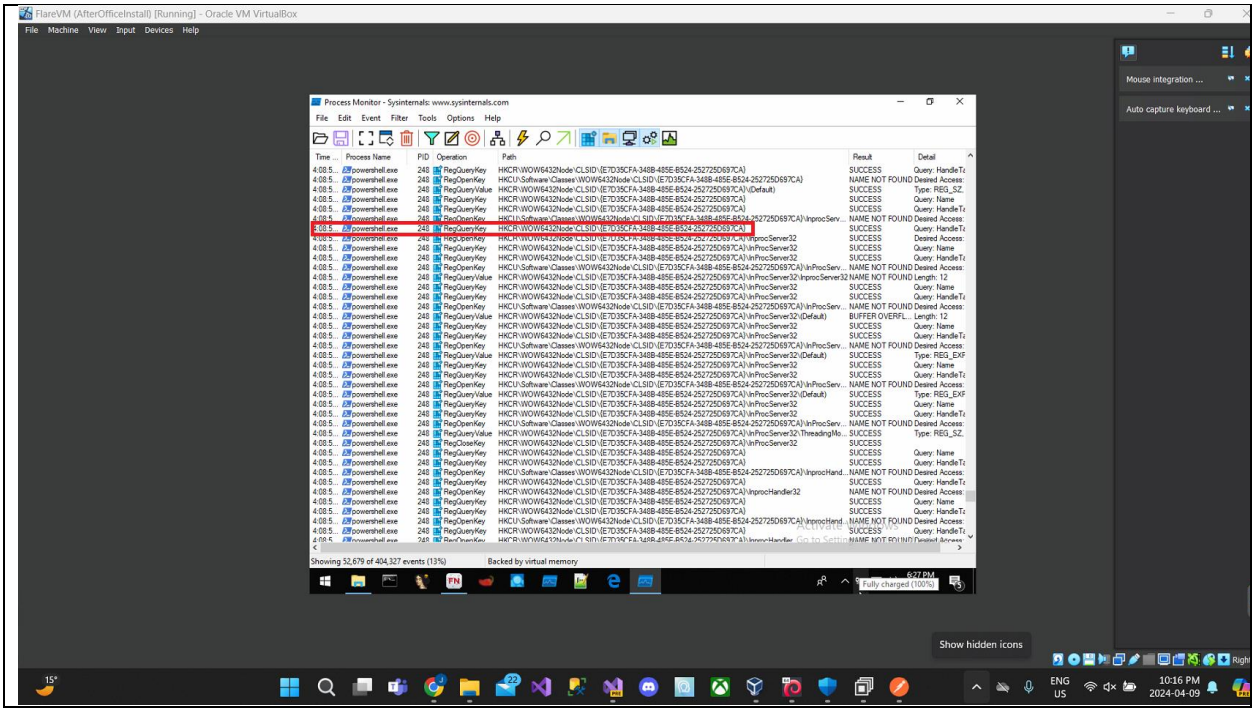

Figure 20 PowerShell registry query.

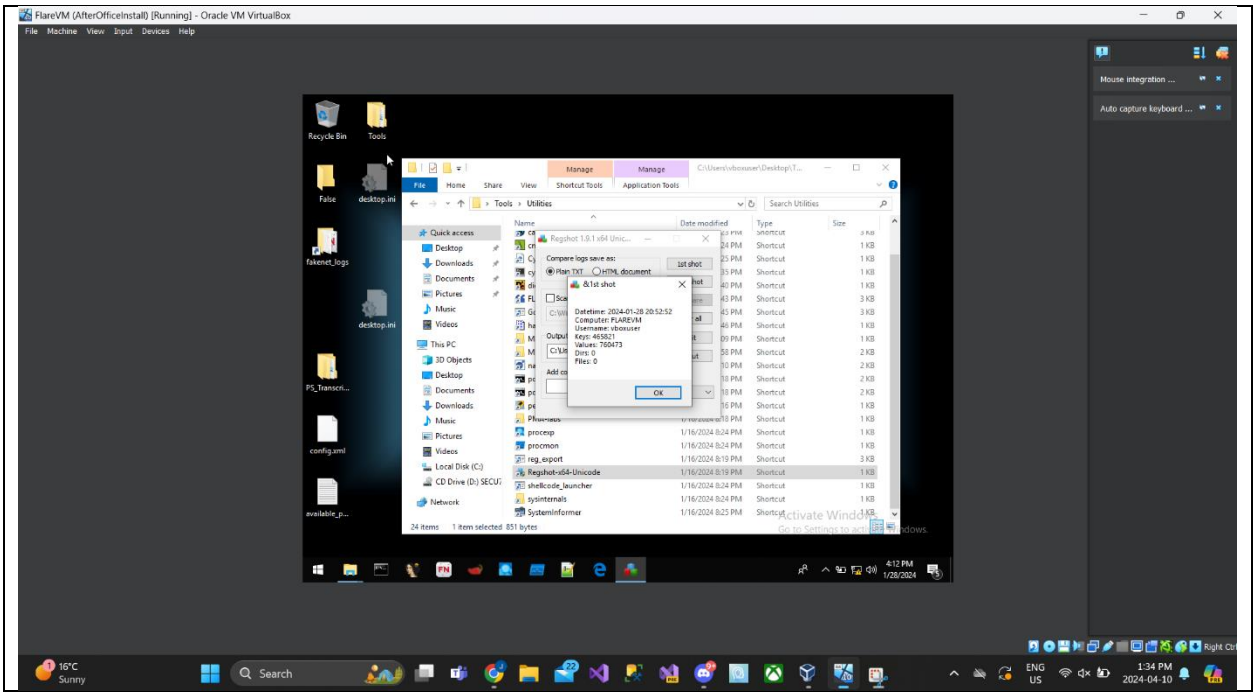

Figure 21 reg shot 1st shot.

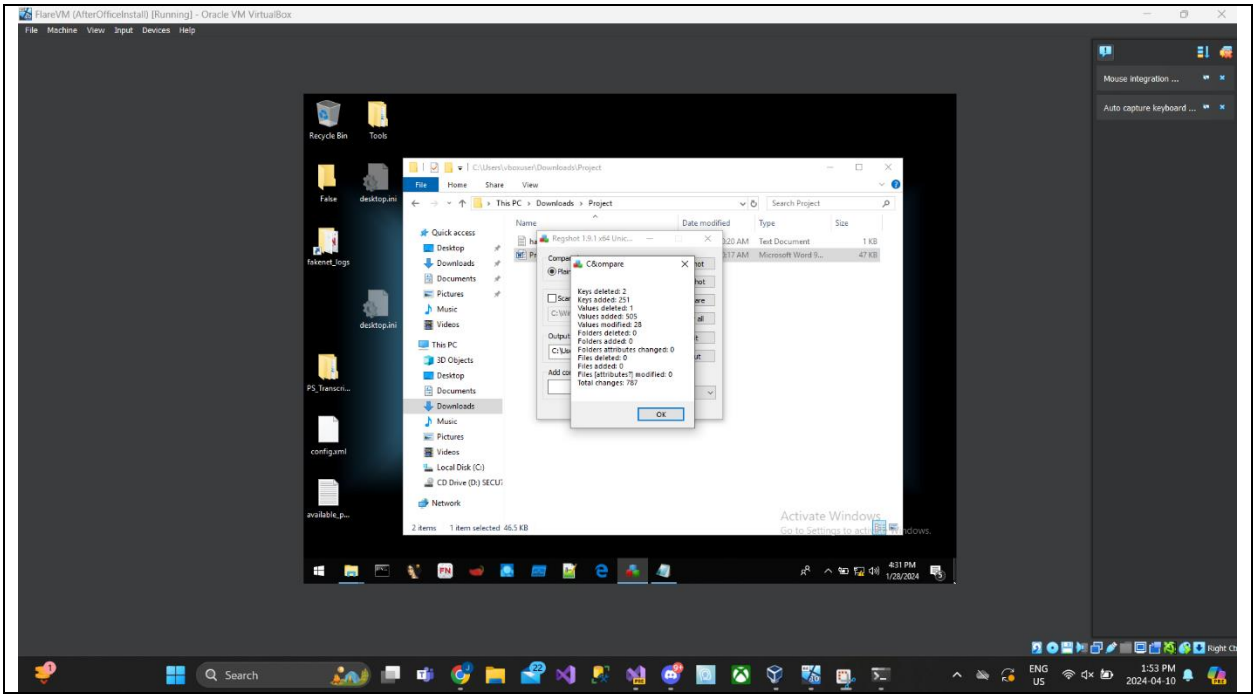

Figure 22 reg shot compare.

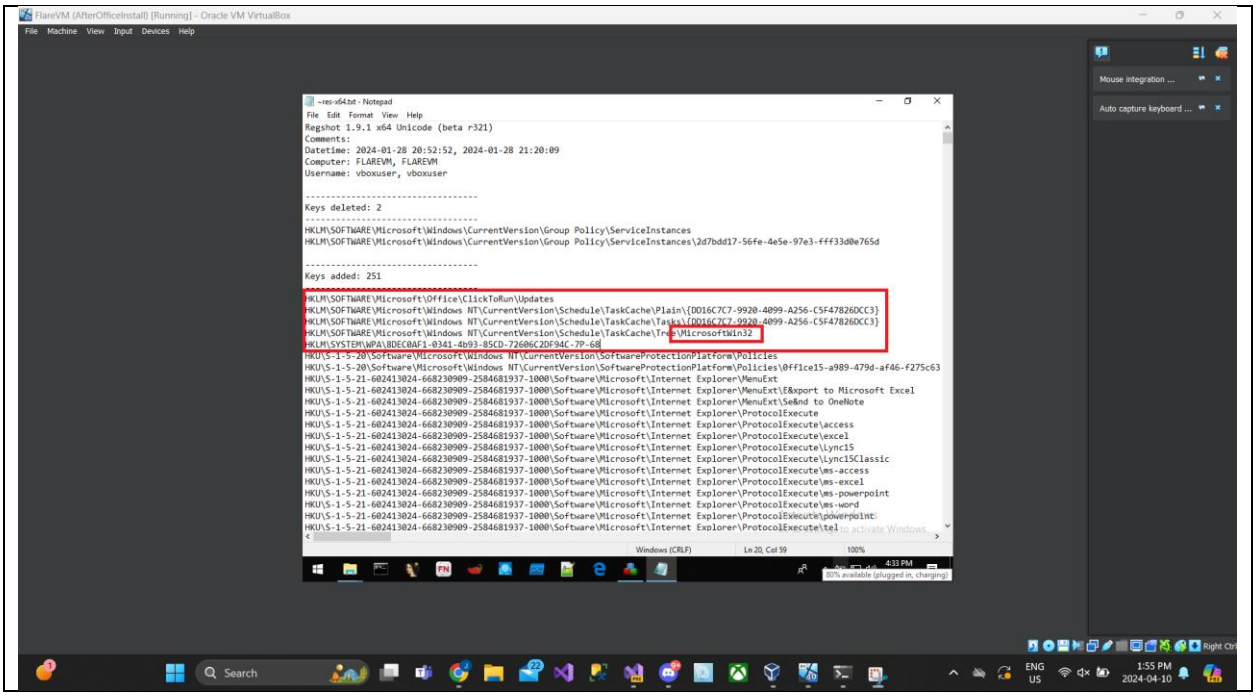

Figure 23 MicrosoftWin32 scheduled task.

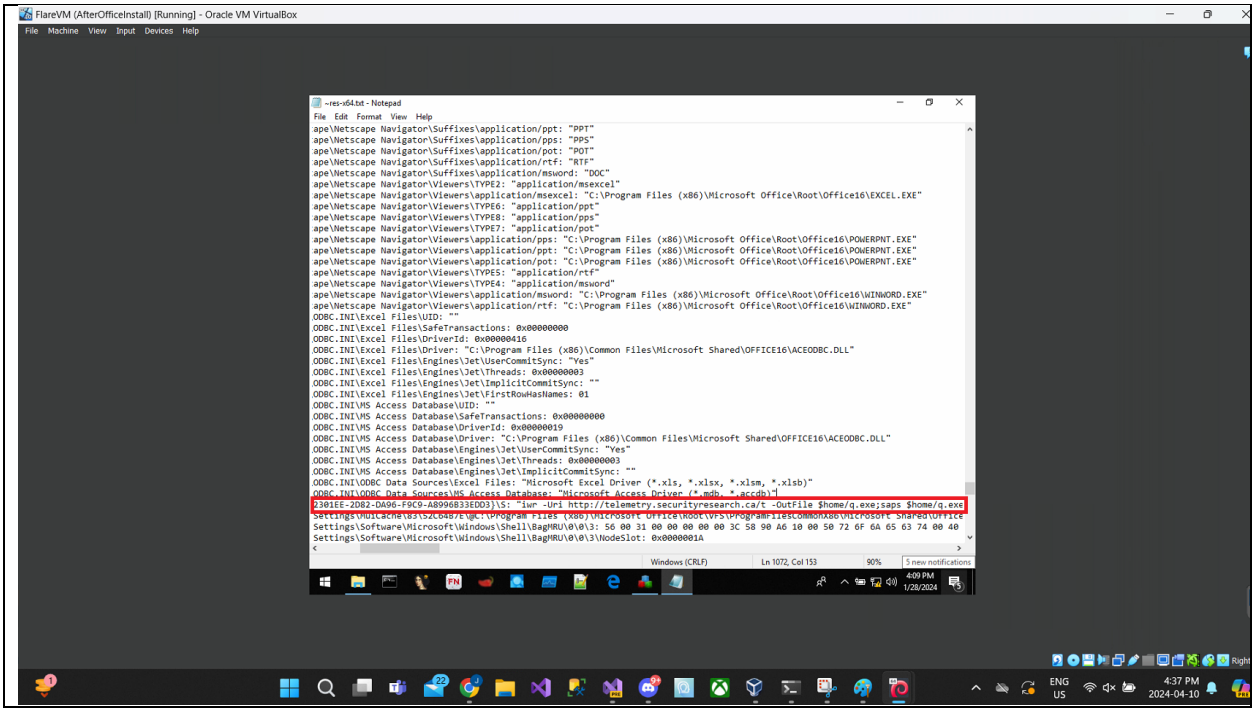

Figure 24 reg shot output for {932301EE-2D82-DA96-F9C9-A8996B33EDD3}

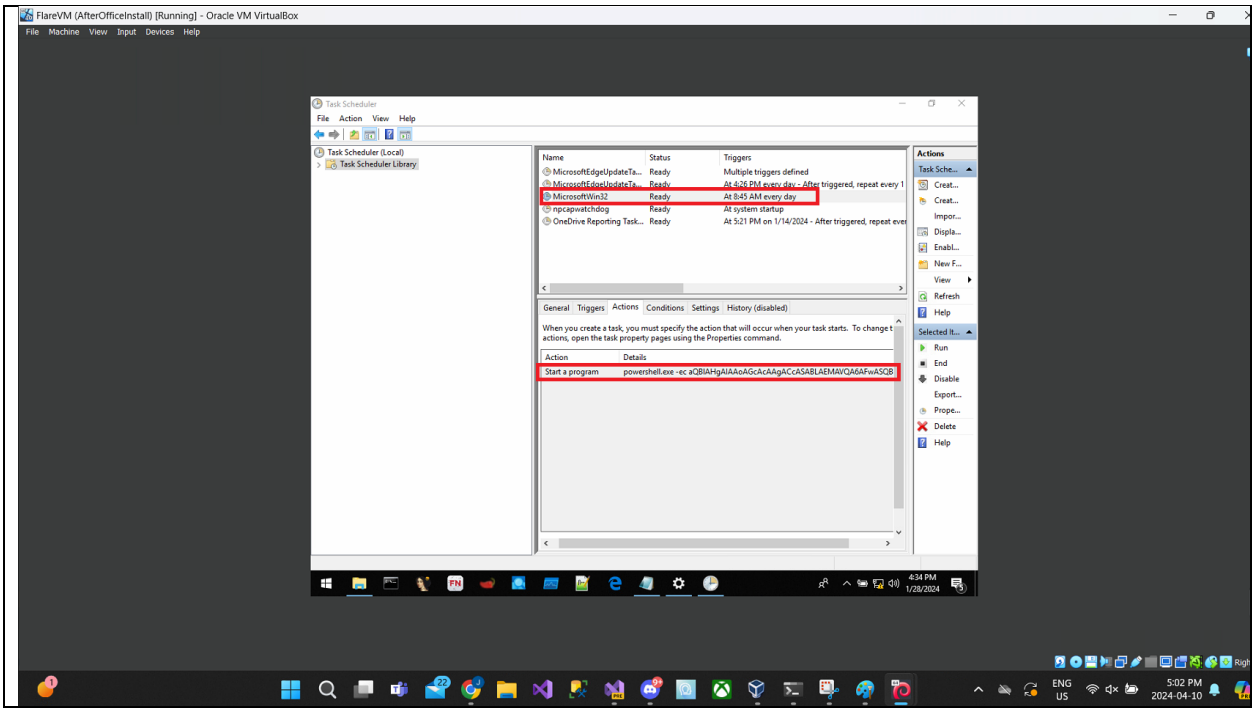

Figure 25 scheduled MicrosoftWin32 task.

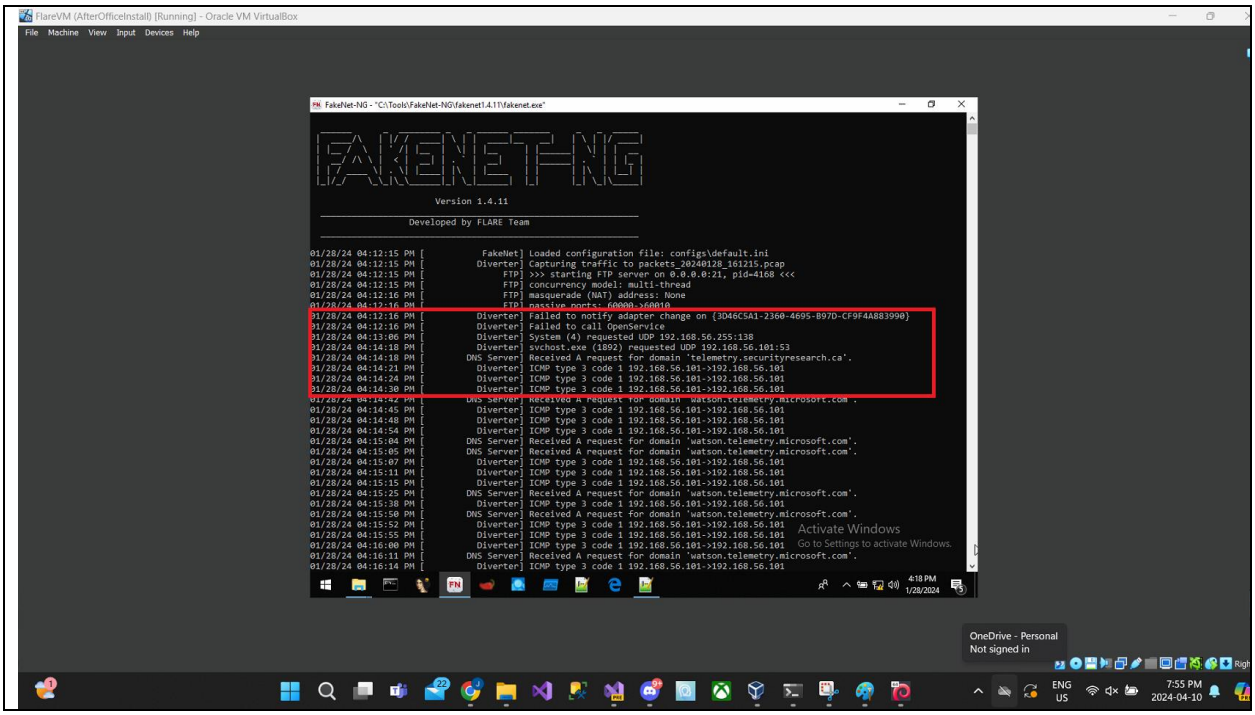

Figure 26 q.exe attempting to sending back to telemetry.securityresearch.ca

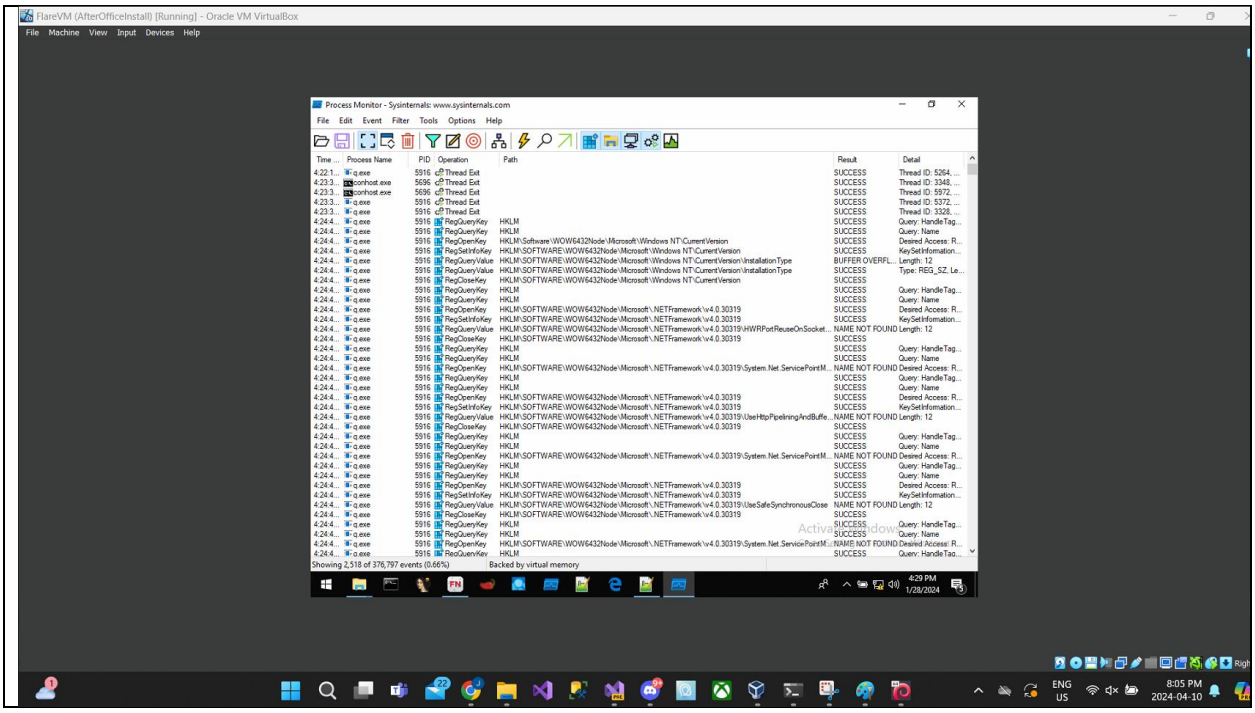

Figure 27 procmon display of q.exe.