DarkCloud

By REXor

Archived: 2026-04-05 15:31:18 UTC

_Overview

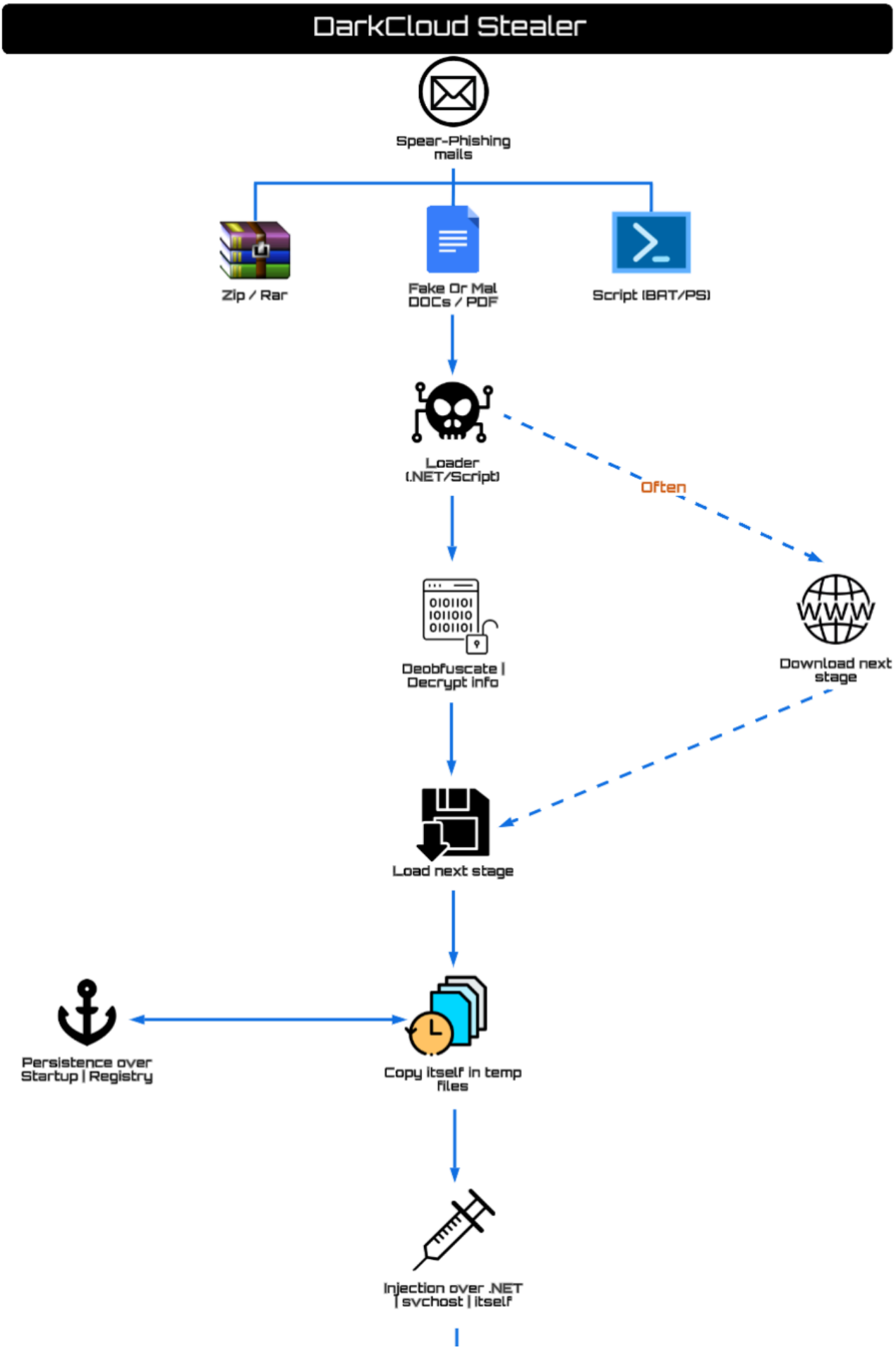
DarkCloud is a type of malware categorized as a Stealer, which has evolved over time and positioned itself as one of the most widely used in its category. DarkCloud began making an impact in 2022 and gained relevance that same year and in 2023 by being promoted in various forums showcasing its functionalities. As usual, it was sold through different sources, often leading buyers to communicate via Telegram for its acquisition.

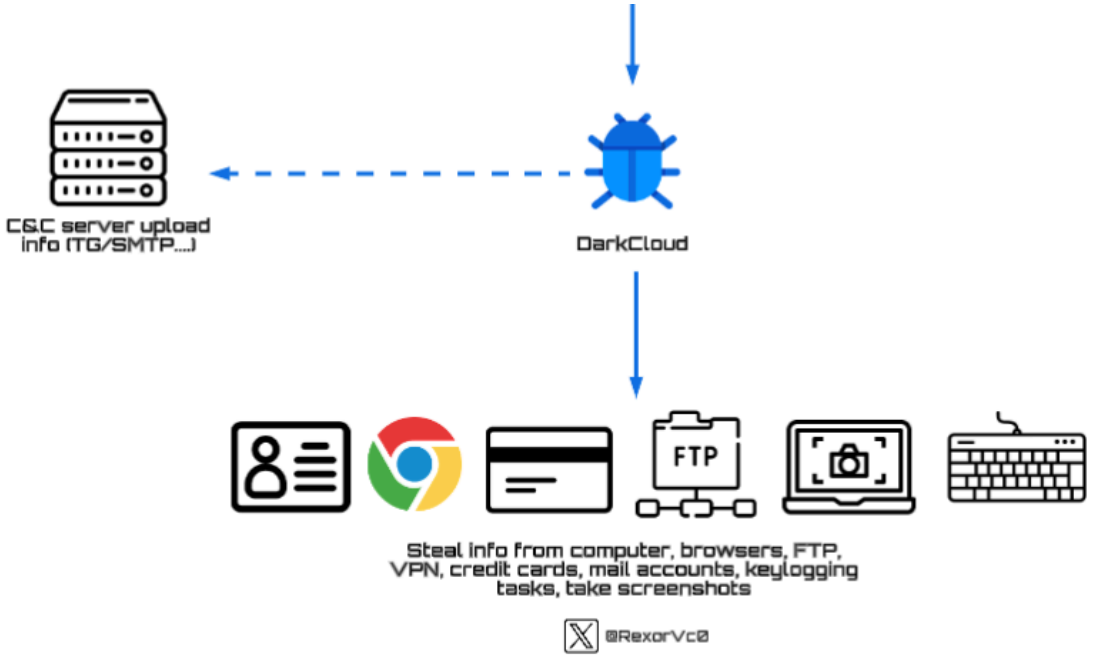
The execution and distribution of this Stealer have been driven by phishing campaigns, where attackers impersonated various companies or disguised their attacks as payment receipts, fines, and other commonly used tactics aimed at compromising HR departments or similar targets. However, it has also been used in other, less frequent campaigns, where users were directed to download or execute malicious samples through infected products or websites, employing techniques such as malvertising, watering hole attacks, or similar methods. Additionally, it has occasionally been deployed alongside other malware acting as loaders or launched in conjunction with similar threats, such as DbatLoader or ClipBanker.

_Technical Analysis

The distribution method, as we mentioned, can vary, but in the vast majority of samples I have encountered, phishing has been the primary focus. Therefore, we can say that phishing is the most common distribution technique for DarkCloud, even though it has been associated with other methods. Once the victim accesses the link or downloads the file, they may encounter compressed files leading to a loader, documents, or even scripts in different languages that initiate execution. After this, a loader is executed, which typically downloads the next stage or contains it within its resources, obfuscated or encrypted, to extract the next module. This module will execute DarkCloud in memory before injecting it. The final step is the theft of various types of information, including browser data, FTP credentials, screenshots, keylogging, and more.

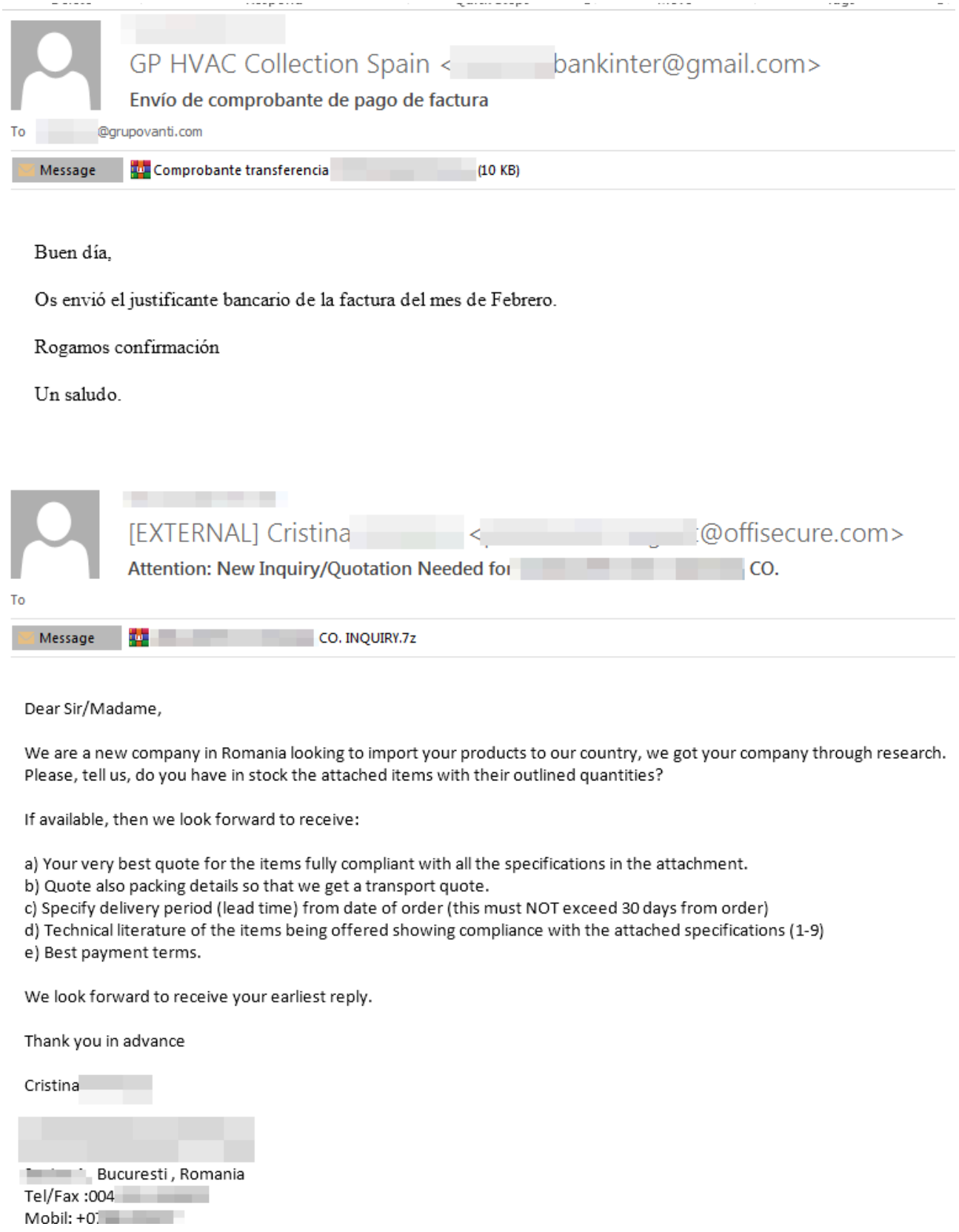
A diagram summarizing the most common versions I have encountered and attempting to unify them into a single representation is as follows:


DarkCloud Stealer
Spear-Phishing mails
Zip / Rar
Fake Or Mal DOCs / PDF
Script (BAT/PS)
Loader (.NET/Script)
Often
Deobfuscate | Decrypt info
Download next stage
Load next stage
Persistence over Startup | Registry
Copy itself in temp files
Injection over .NET | svchost | itself



During the analysis process, I have come across various samples that all aim to achieve the same goal—executing DarkCloud. While respecting the different versions, I will highlight the most relevant and distinct ones I have found to provide a clearer understanding of the various existing variables. However, I will avoid excessive detail to keep the explanation engaging rather than tedious.

As mentioned earlier, the most common approach involves phishing attempts using documents, compressed files, or other formats.



GP HVAC Collection Spain <bankinter@gmail.com>

Envío de comprobante de pago de factura

To @grupovanti.com

Message Comprobante transferencia (10 KB)

Buen día,

Os envió el justificante bancario de la factura del mes de Febrero.

Rogamos confirmación

Un saludo.

[EXTERNAL] Cristina <@offisecure.com>

Attention: New Inquiry/Quotation Needed for CO.

To

Message CO. INQUIRY.7z

Dear Sir/Madame,

We are a new company in Romania looking to import your products to our country, we got your company through research. Please, tell us, do you have in stock the attached items with their outlined quantities?

If available, then we look forward to receive:

a) Your very best quote for the items fully compliant with all the specifications in the attachment.
b) Quote also packing details so that we get a transport quote.
c) Specify delivery period (lead time) from date of order (this must NOT exceed 30 days from order)
d) Technical literature of the items being offered showing compliance with the attached specifications (1-9)
e) Best payment terms.

We look forward to receive your earliest reply.

Thank you in advance

Cristina

Bucuresti , Romania
Tel/Fax :004
Mobil: +0

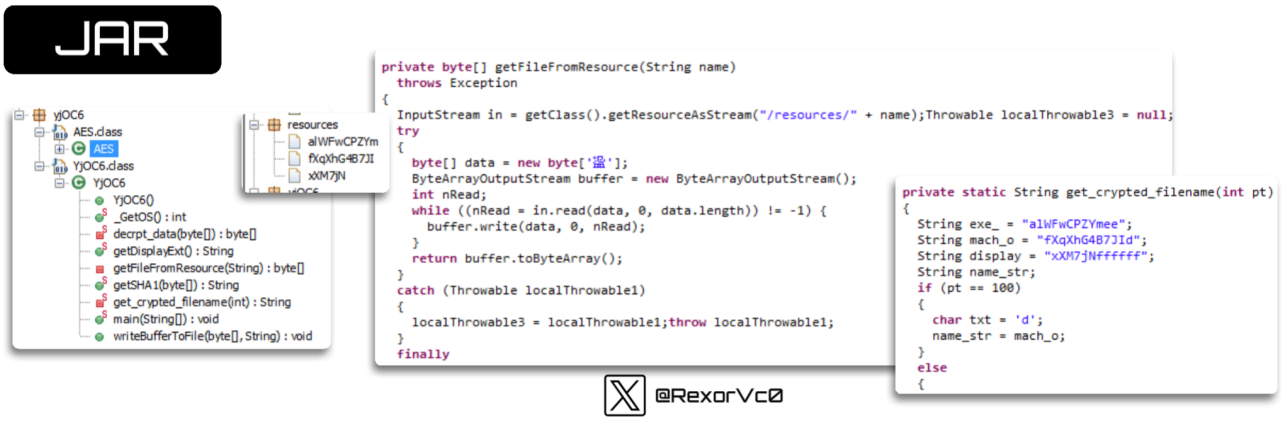
Going straight to the point, in an initial execution of one of the samples I analyzed, we see a rather extensive process—a succession of execution copies launched in temporary paths.

Once the victim falls for the phishing attack, they may be led to different types of loaders. Some of these can be scripts, covering a wide range of formats, from JAR and BAT to PowerShell (PS). It’s quite interesting to see the

variety of types I have encountered, especially considering that I generally do not analyze samples from previous years.

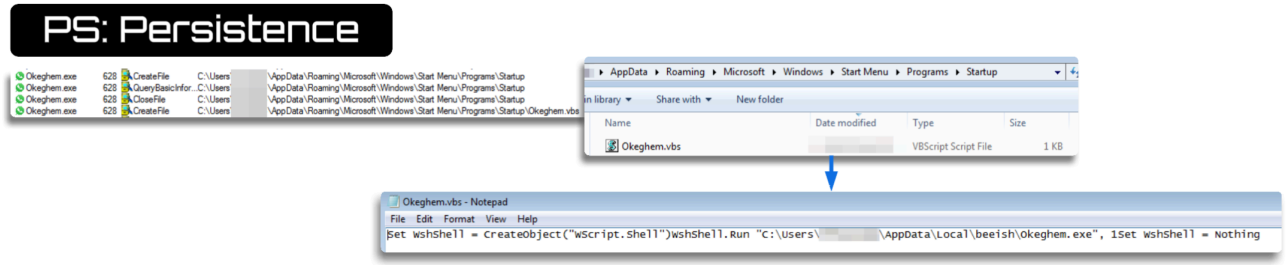
_Script Loader Versions

In the case of JAR files, they retrieve the information for the next stage using embedded resources—something that, as we will see, is not the common trend in DarkCloud.



In other versions, such as those written in PowerShell, which are more common, we can observe executions where, after multiple and varied obfuscation techniques for each type, the process leads either to a download or to the extraction of a binary. This binary, in its resources, contains the next stage, which in this case is an AutoIT script.

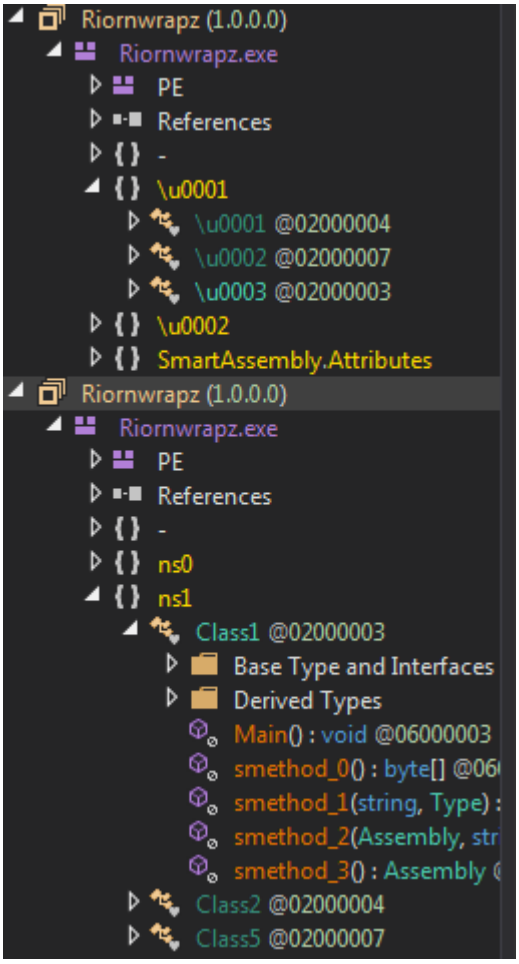
At this stage, several tasks are performed, such as the creation of a VBS file (a very common practice, as you will see) in startup-like folders to establish persistence. Ultimately, this specific sample carried out an injection into svchost, but we will explore that in more depth later.



To wrap up the discussion on scripts, another frequent scenario involves encountering VBS/PowerShell scripts—more or less obfuscated—that establish a connection to download and execute the payload. This method is similar to the one previously observed in PowerShell. If the download link is down, however, the process reaches a dead end—something quite typical for this malware. As I mentioned earlier, a large number of samples follow this step at some point. Here are some additional examples.

_Most common loader stage

Moving on to another category of samples, where I will expand on the information a bit more, we have those typically based on .NET. These can act as loaders or appear in the second stage, depending on the version. Among these, there are many similar samples, often utilizing different obfuscators.



After working with several of these obfuscators, we usually reach an initial function where the samples, in different ways, request the next stage to process it. This poses a problem, as you might expect, because unless the sample is very recent, it is likely already inactive and cannot be retrieved. This means we get stuck at the dynamic analysis stage (debugging and such things), and we will only be able to rely on our sharp eyesight to understand everything statically. Personally, I dislike this approach, as I prefer to examine all (or most) of the functions to gain a deeper understanding of everything the malware does.

As we can see, when we try to load the path and therefore, the binary, it will crash and we will have a dead end.

```
    catch (Exception innerException)
    {
        throw new Exception("Failed to load assembly.", innerException);
    }
    return result;
}
```

_Technical Issues

I wanted to dive deeper into a specific topic in case it helps someone, but it doesn't affect the overall understanding of the analysis. So, if you're not interested, feel free to skip it (You can follow the analysis in "*Following the thread*" section). In this case, we reached a DarkCloud sample which, in most cases (if we want to roll back and don't have fresh samples), makes requests to a specific domain to download the next stage, as we've seen. This might seem like a trivial issue, but it's actually quite serious if you truly want to analyze in detail how a sample works—or, in this case, how multiple variants of the same malware behave.

Sometimes, it's a good strategy to try and obtain the file that is being downloaded. Tools like VT (VirusTotal), Any.Run, JoeSandbox, Tria.ge, and others make this relatively easy, as long as we know the IP or domain the sample is contacting. We can then pivot and take a more h4ck3r-like approach, capturing packets (.pcap) to inspect the received data—how deep you want to go into this is up to you. However, in this case, additional steps were required because the sample requested the file at runtime to modify, process, and then execute or load it. This adds an extra challenge, as we need to understand the sample, debug it up to the right point, and manipulate it so that it processes correctly.

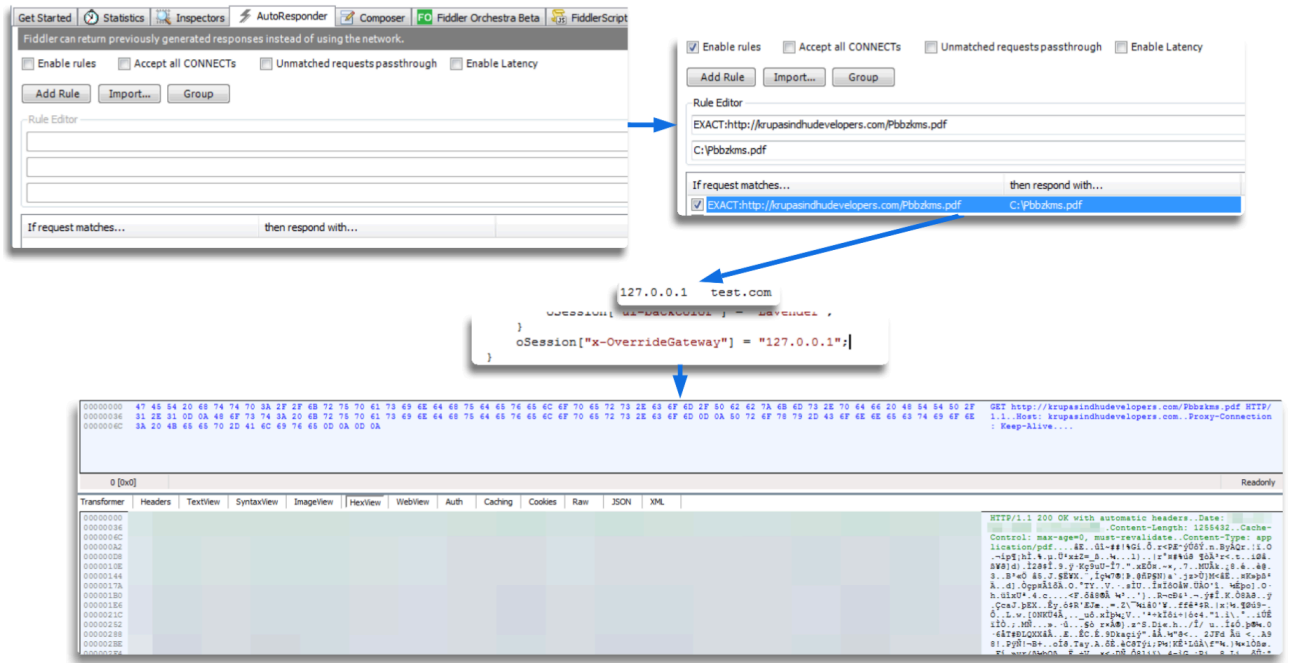
I came up with three ways to approach this (I know there are many more, and yours is probably better), and to keep things simple, I tested them on multiple examples but will only show one to stay focused:

_Setting Up an HTTP Server and Hosting the Sample Locally

This is probably the simplest option. It involves extracting the sample (which we would have retrieved by pivoting and identifying what it wants to download), placing it in a directory like C:\\ , modifying the hosts file to point to our local machine, and starting an HTTP server using Python. I chose port 80 even though the request was originally HTTPS, because changing an "s" in the debugger is obviously much easier. Once we reach the same execution point, we can observe what the sample serves and proceed further.

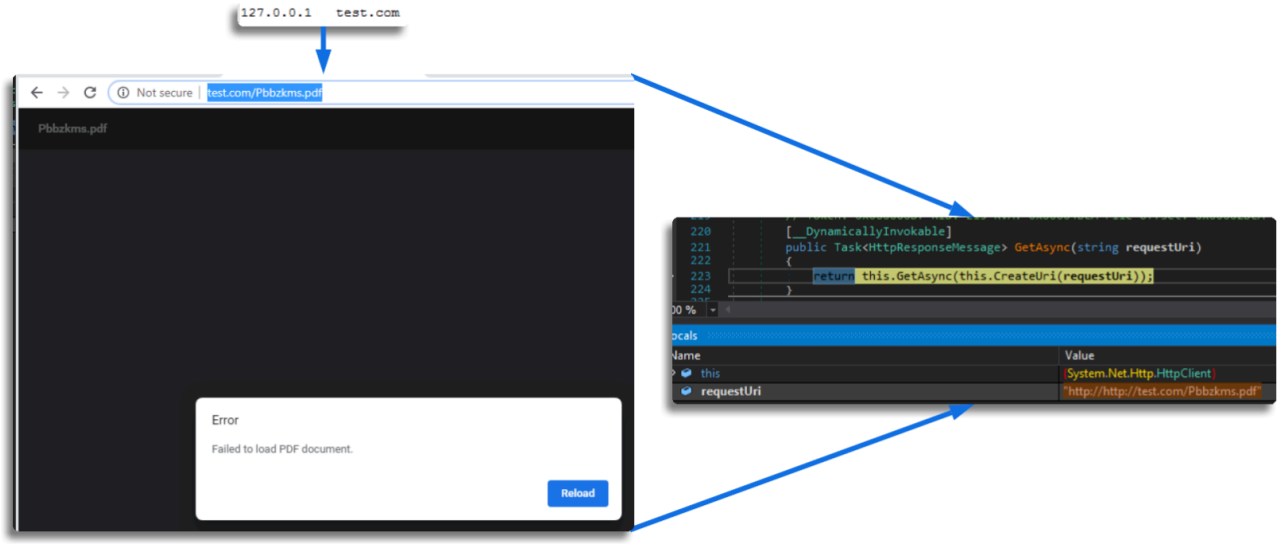
_Modifying Traffic with Fiddler

Sometimes, I like using Fiddler, so I wanted to test it in this scenario. I simply created a rule that, when the specific URL was requested, it would serve the file from my local directory. Some configuration changes were required in Fiddler to redirect traffic to localhost. Of course, I also modified the hosts file. In this case, I set up test.com to point to localhost. Once the sample reached the same function, I made the same modification as before—switching to HTTP—to ensure it received the file at runtime.



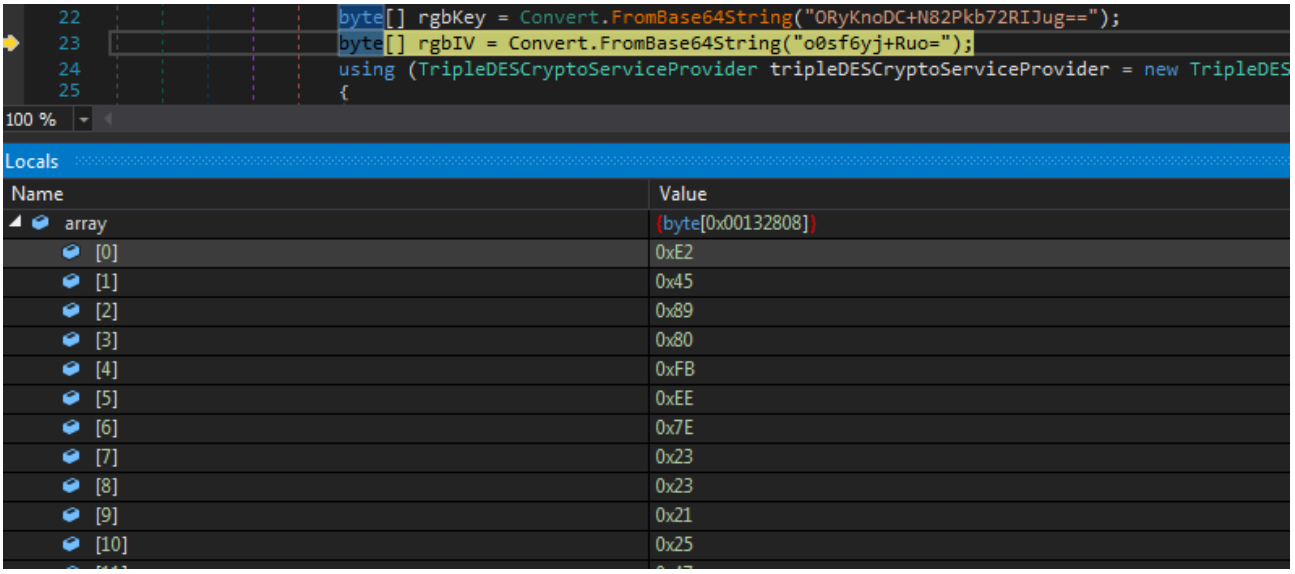
_Combining Techniques

For the third approach, I experimented with a hybrid method. I kept the same rules as before but changed the URL to one I was serving on test.com, with the file located in C:. When the execution reached the crucial point, I changed the URL, and it loaded successfully.



Following the thread

After obtaining and loading the file at runtime, the next step involves key generation. As mentioned earlier, it's very common for pre-loader stages to deobfuscate or decrypt some part of the code—or even an entire file.



At this stage, we can see that the PDF was loaded into an array, and indeed, it matches the one we had retrieved via pivoting.

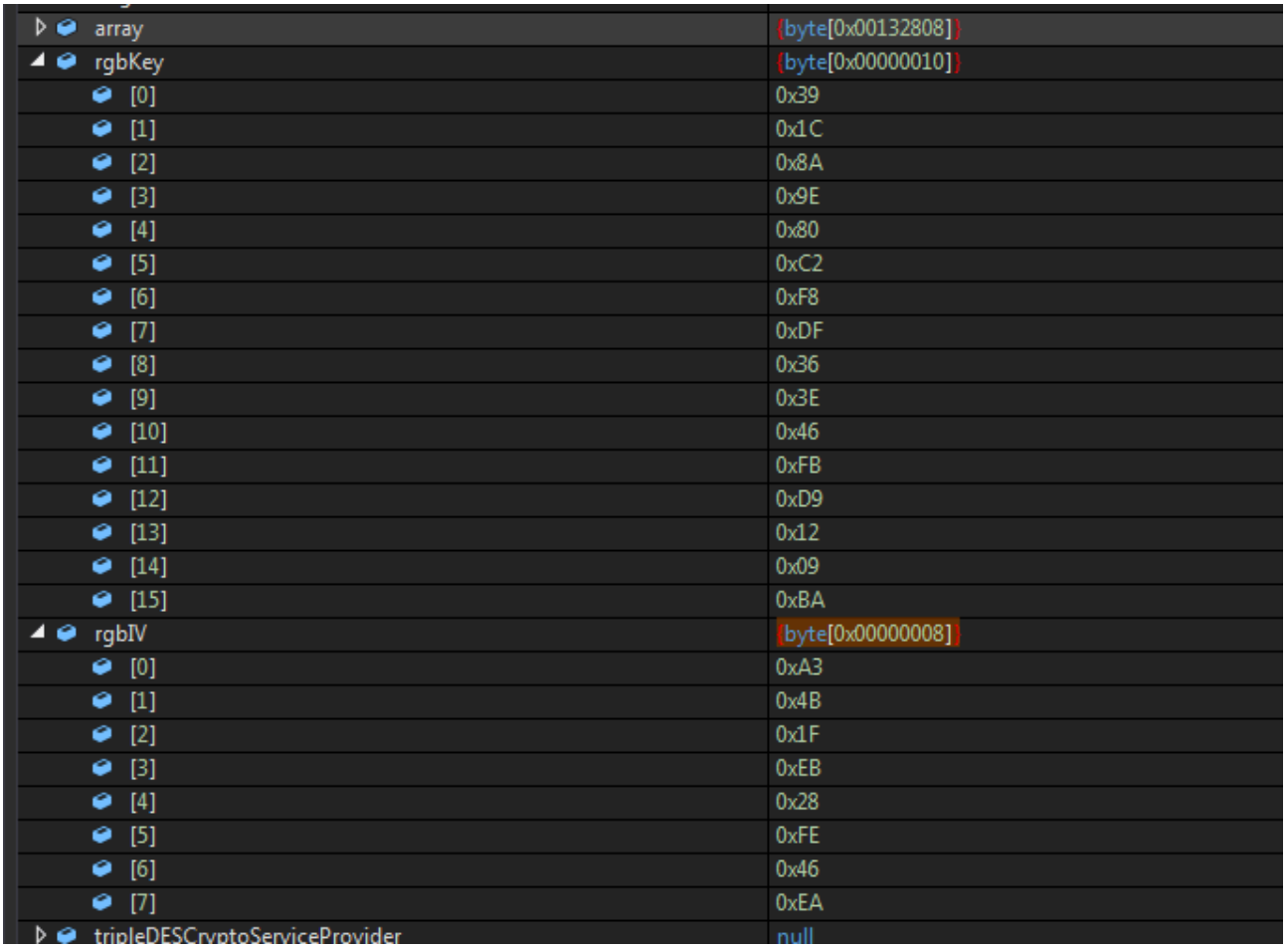
array	byte[0x00132808]
[0]	0xE2
[1]	0x45
[2]	0x89
[3]	0x80
[4]	0xFB
[5]	0xEE
[6]	0x7E
[7]	0x23
[8]	0x23
[9]	0x21
[10]	0x25
[11]	0x47
[12]	0xED
[13]	0x1A
[14]	0xD5
[15]	0x83

```
00000000  E2 45 89 80 FB EE 7E 23 23 21 25 47 ED 1A D5 83
00000010  72 3C 50 C6 5E FD DA F4 DD 92 6E 8A 42 79 C1 51
00000020  72 0F A6 EF 01 4F 15 AC ED 70 B6 A1 68 CC 92 25
00000030  92 B5 08 DC B2 78 B1 5A 3D 5F DF 9C 9A BC 16 17
00000040  85 6C 29 82 82 7C 72 B0 A4 23 25 FA F0 AD B6 F2
00000050  C4 B3 72 3C 01 74 1E 16 69 D8 E2 87 DF A5 F0 5D
```

From here, the usual process follows: the code undergoes processing, whether through deobfuscation or decryption. In this example, the technique used is Base64 + TripleDES.

```
static byte[] smethod_0()
{
    int num = 0;
    bool flag = false;
    while (!flag)
    {
        try
        {
            byte[] array = Class2.smethod_0();
            byte[] rgbKey = Convert.FromBase64String("ORyKnoDC+N82Pkb72RIJug==");
            byte[] rgbIV = Convert.FromBase64String("o0sf6yj+Ruo=");
            using (TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider())
            {
                using (MemoryStream memoryStream = new MemoryStream())
                {
                    using (CryptoStream cryptoStream = new CryptoStream(memoryStream, tripleDESCryptoServiceProvider.CreateDecryptor(rgbKey, rgbIV), CryptoStreamMode.Write))
                    {
                        cryptoStream.Write(array, 0, array.Length);
                        flag = true;
                        return memoryStream.ToArray();
                    }
                }
            }
        }
        catch (WebException)
        {
            num++;
        }
```

We observe how the keys are processed



After this we will see that at the end of the function we can obtain the processed binary

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00  ............€...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  50 45 00 00 4C 01 03 00 57 28 B7 67 00 00 00 00  PE..L...W(·g....
00000090  00 00 00 00 E0 00 0E 21 0B 01 30 00 00 20 13 00  ....à..!..0.. ..
000000A0  00 06 00 00 00 00 00 00 EE 3F 13 00 00 20 00 00  ........î?... ..
000000B0  00 40 13 00 00 00 40 00 00 20 00 00 00 02 00 00  .@....@.. ......
000000C0  04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ................
000000D0  00 80 13 00 00 02 00 00 00 00 00 00 03 00 40 85  .€............@…
000000E0  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
000000F0  00 00 00 00 0F 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  A0 3F 13 00 4B 00 00 00 00 40 13 00 34 03 00 00   ?..K....@..4...
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120  00 60 13 00 0C 00 00 00 00 00 00 00 00 00 00 00  .`..............
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 20 00 00 08 00 00 00  ......... ......
```

At this point, we can go full h4ck3r mode again and verify that we are on the right track and truly understand the process. For example, we can write Python code to handle the extracted file and perform the decryption ourselves.

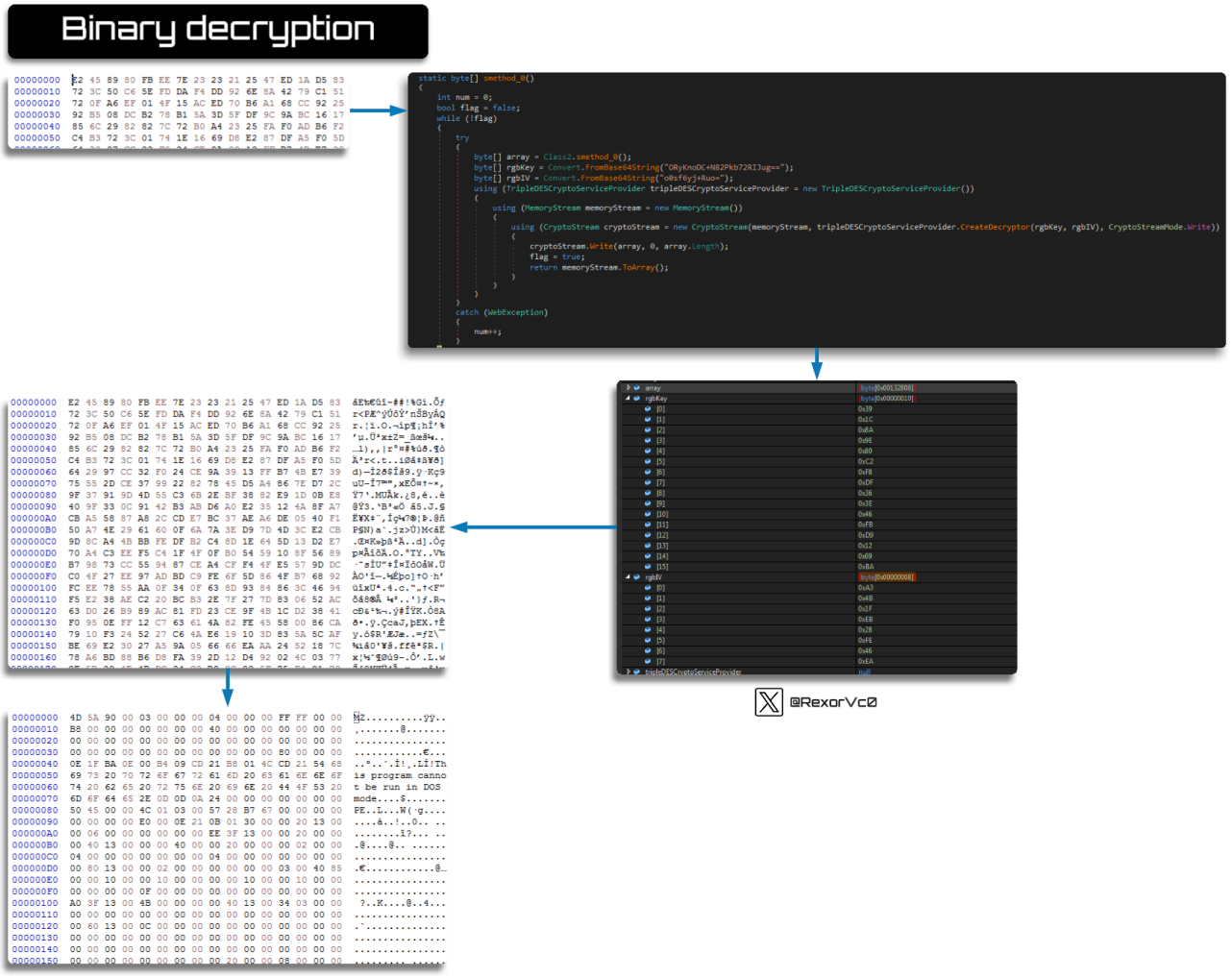
```
 from Crypto.Cipher import DES3
import binascii

#Keys
rgbKey = bytes([0x39, 0x1C, 0x8A, 0x9E, 0x80, 0xC2, 0xF8, 0xDF, 0x36, 0x3E, 0x46, 0xFB, 0xD9, 0x12, 0x09, 0xBA])
rgbIV  = bytes([0xA3, 0x4B, 0x1F, 0xEB, 0x28, 0xFE, 0x46, 0xEA])

#Hex data
hex_data = "<redacted>"
ciphertext = binascii.unhexlify(hex_data)

#Decrypt Triple DES (3DES)
cipher = DES3.new(rgbKey, DES3.MODE_CBC, rgbIV)
plaintext = cipher.decrypt(ciphertext)

#Save
with open("output.exe", "wb") as f:
    f.write(plaintext)

print("Saved as output.exe")
```

In both cases, we should be able to extract a binary, which is then loaded into memory.

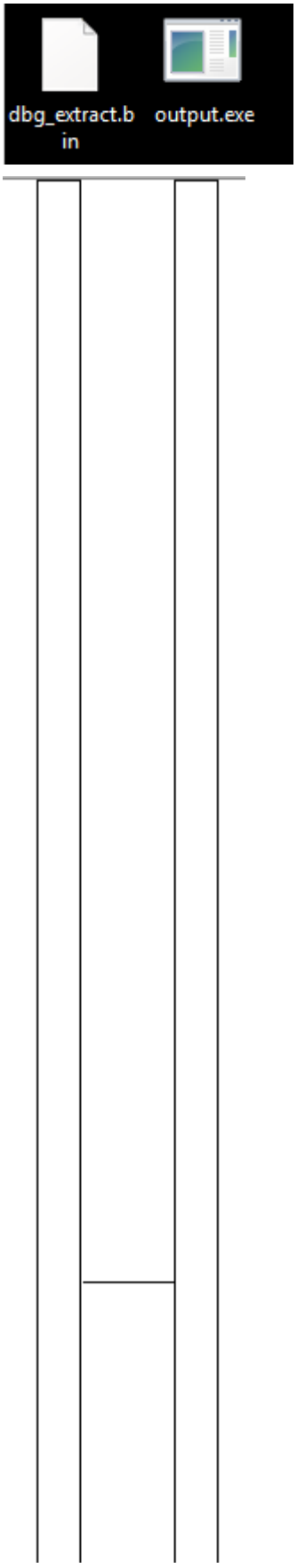
```
Assembly result;
try
{
    byte[] rawAssembly = Class1.smethod_0();
    result = AppDomain.CurrentDomain.Load(rawAssembly);
}
catch (Exception innerException)
{
    throw new Exception("Failed to load assembly.", innerException);
}
```

_buffer	{byte[0x00132800]}
[0]	0x4D
[1]	0x5A
[2]	0x90
[3]	0x00
[4]	0x03
[5]	0x00
[6]	0x00
[7]	0x00
[8]	0x04
[9]	0x00
[10]	0x00
[11]	0x00
[12]	0xFF
[13]	0xFF
[14]	0x00
[15]	0x00
[16]	0xB8
[17]	0x00
[18]	0x00
[19]	0x00
[20]	0x00
[21]	0x00
[22]	0x00

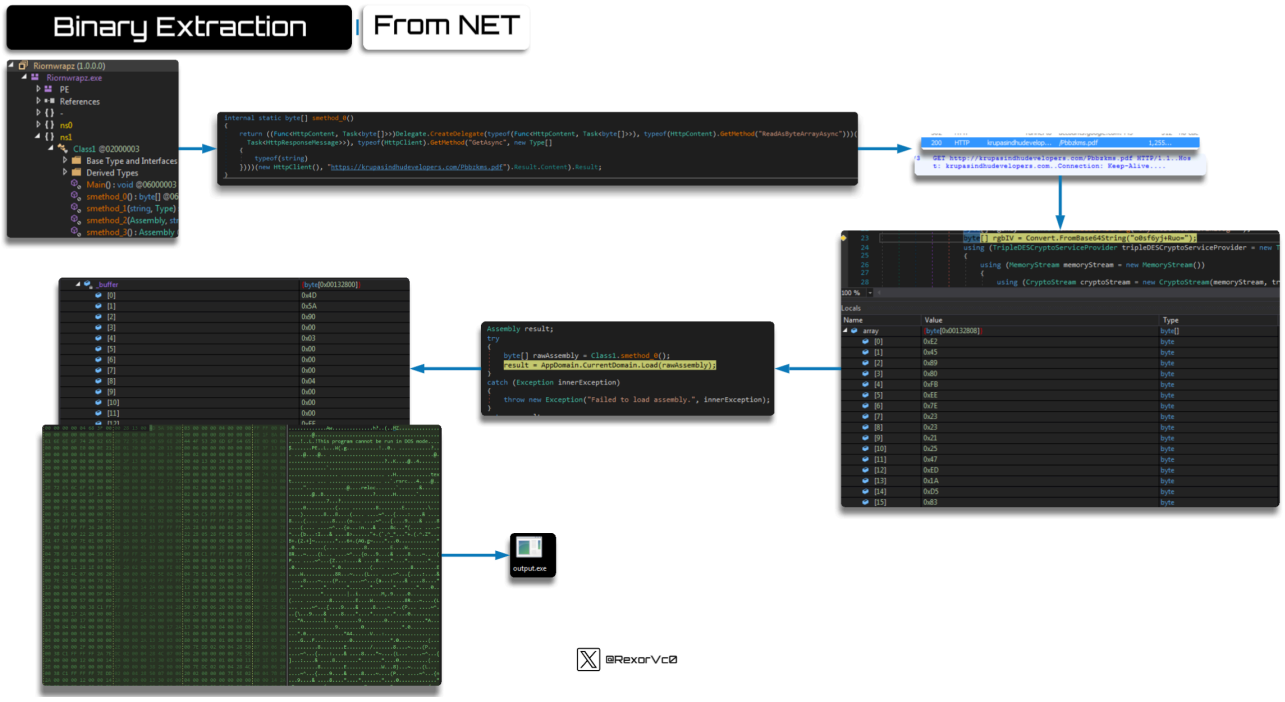
Decryption Result in a Graphical Format



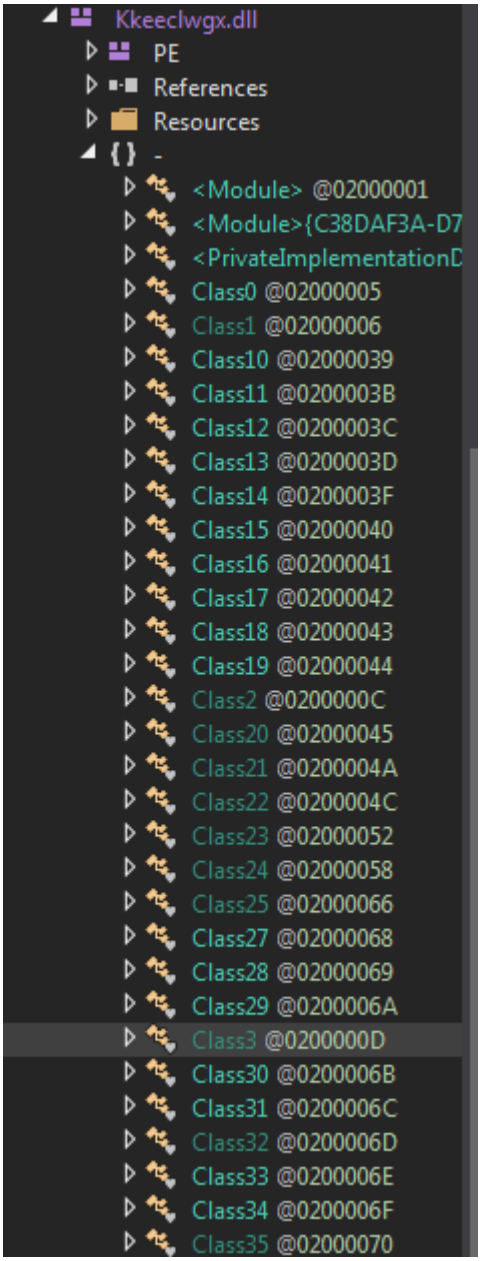
I tested this myself, and as expected, we received the same content in both cases.

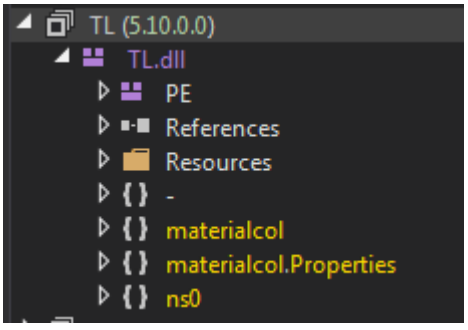


To make the binary extraction process more visual, here’s a summary in graphical form:

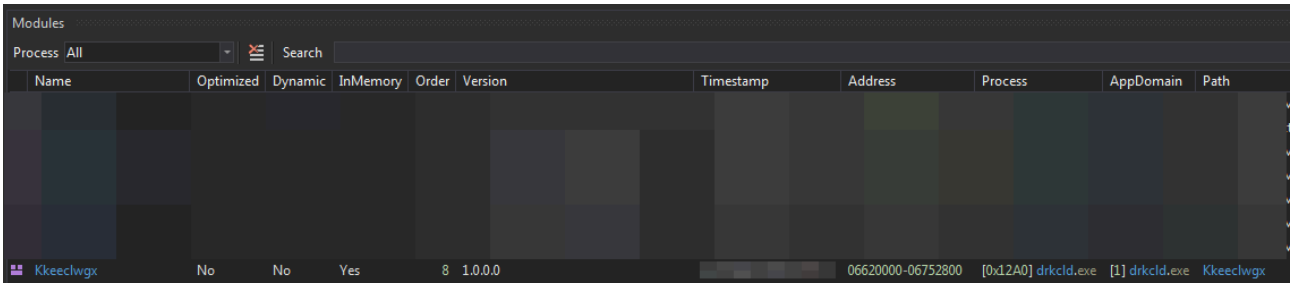


Once we reach this stage, what we obtain in most cases is a library that loads the code into memory. However, it also performs other tasks, such as creating additional persistence mechanisms.



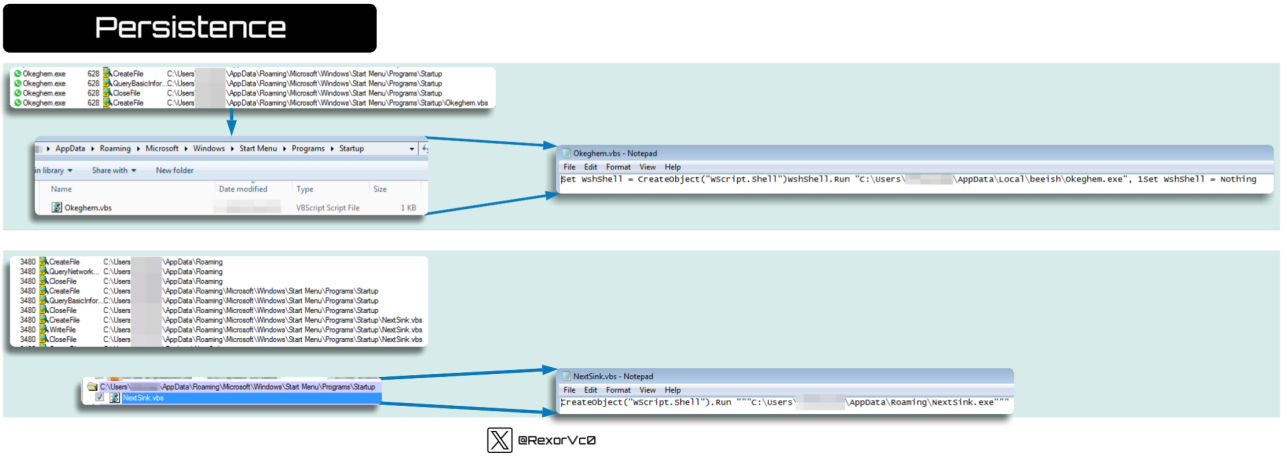


After loading the library module, it carries out actions like creating scripts in temporary folders or startup-type directories. We've seen this before, but it's not the only persistence mechanism I've encountered—some samples also modify registry settings or scheduled tasks.



Examples of Different Persistence Methods

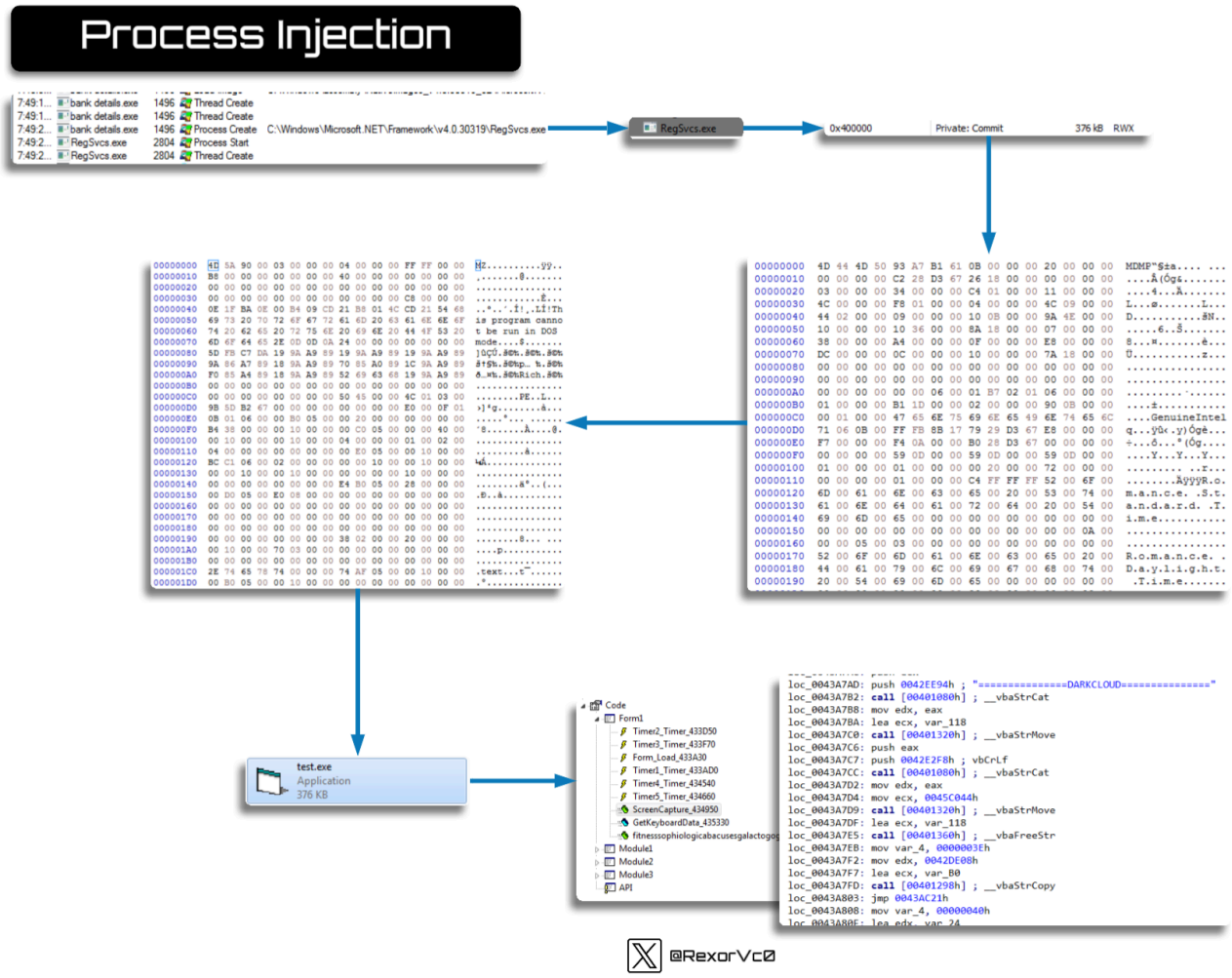
```
 schtasks.exe /Create /TN "Updates\<RandName>" /XML "C:\Users\<user>\AppData\Local\Temp\<DroppedFile>"
C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\<RandName>.vbs
Key: *\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce - Data: *\AppData\Roaming\Microsoft\Windows\Templates\<
```



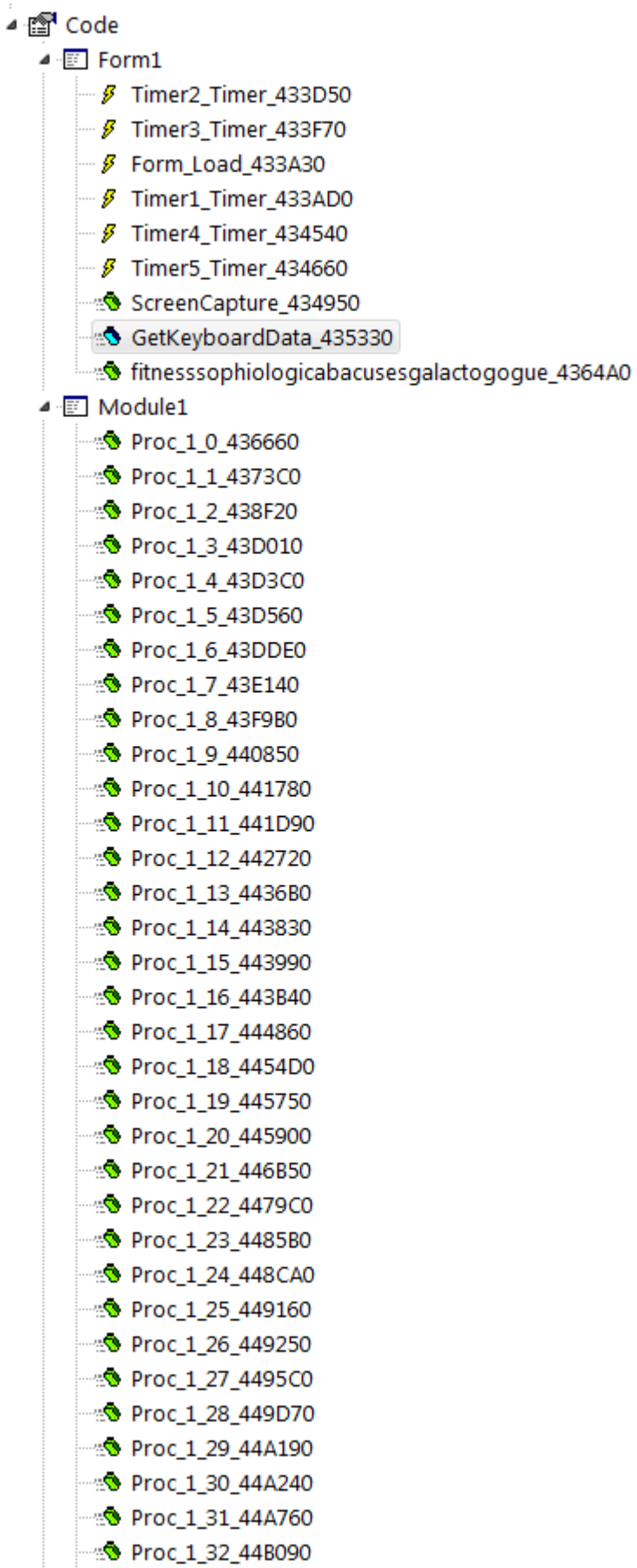
Ultimately, everything culminates in the injection of a VB-based binary, which is the Stealer. This gets injected into a process of its choosing. I've seen a variety of targets, ranging from RegSvc to .NET files, and even the loader process itself. As is common with RATs and Stealers, this technique is designed to evade detection and complicate analysis. It is much easier to spot an unusual process or file than to analyze or detect something that resides in the memory of a legitimate system process.

Here's a list of the most commonly used processes I've observed:

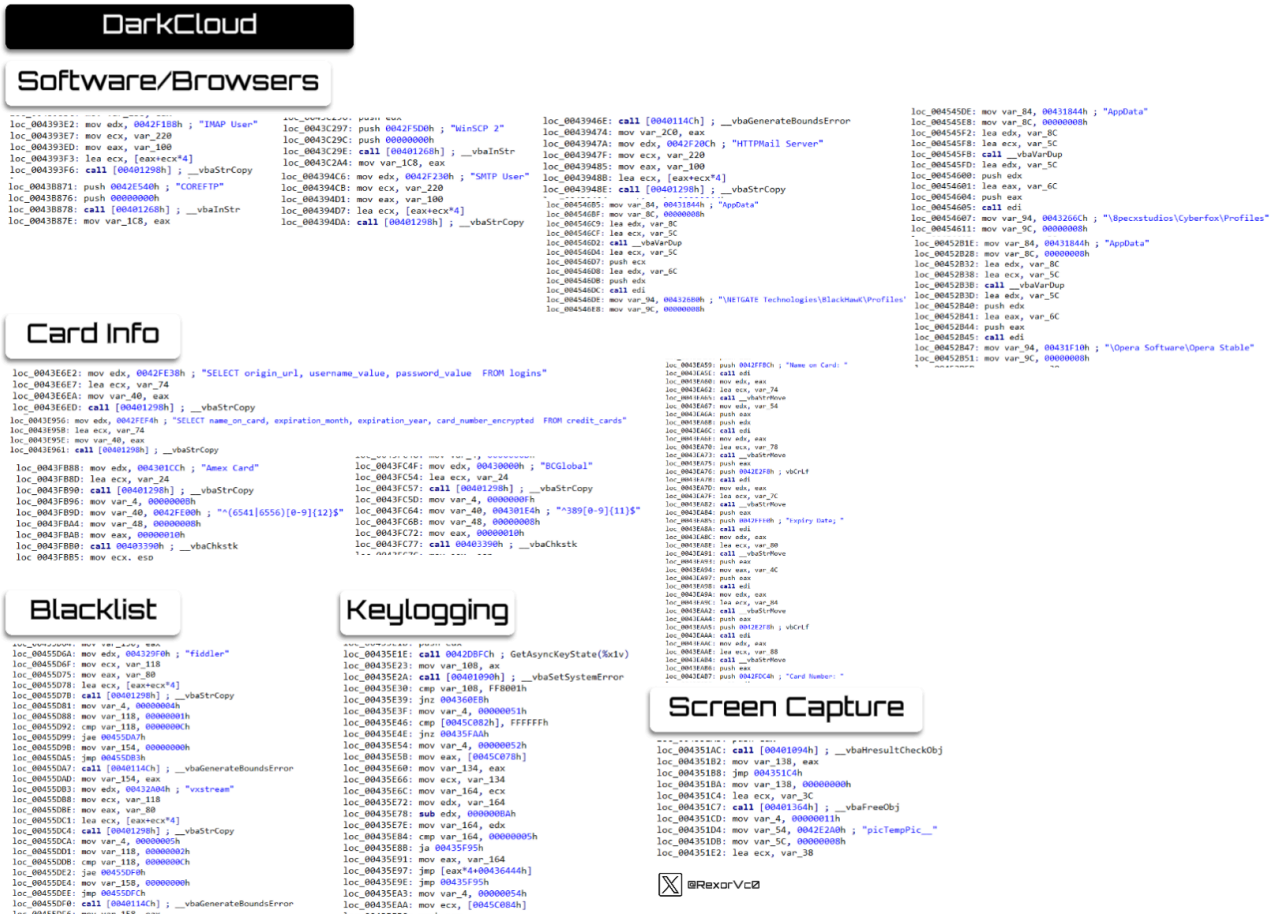
- Svchost
- InstallUtil
- MSBuild
- Itself



I didn’t want to go too deep into the extraction of the second binary, since it follows a similar process but from a second-stage library. The end result is a VB-based binary with extensive capabilities, which I will now present in a more accessible format.

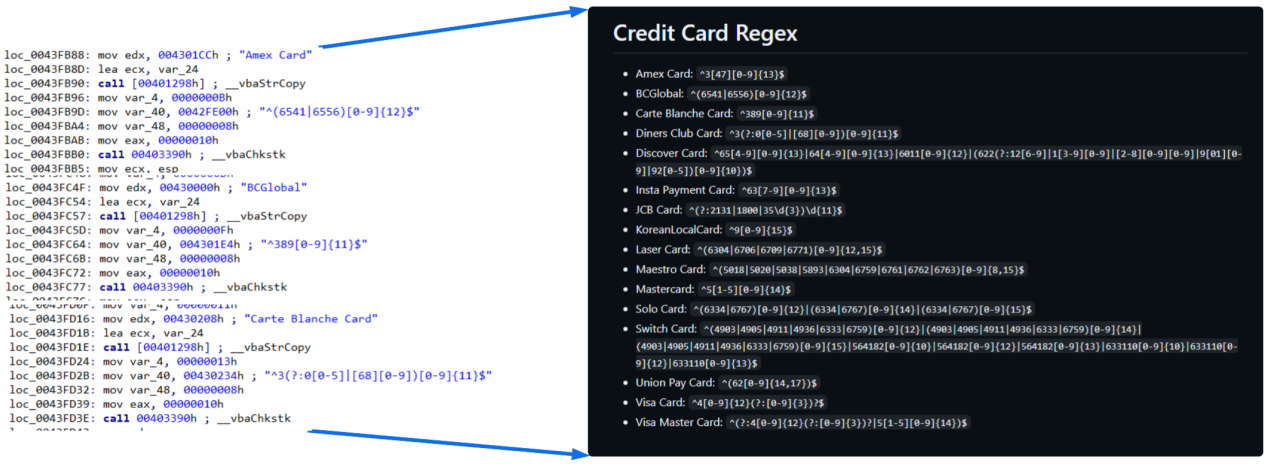

Code
Form1
Timer2_Timer_433D50
Timer3_Timer_433F70
Form_Load_433A30
Timer1_Timer_433AD0
Timer4_Timer_434540
Timer5_Timer_434660
ScreenCapture_434950
GetKeyboardData_435330
fitnesssophiologicabacusesgalactogogue_4364A0
Module1
Proc_1_0_436660
Proc_1_1_4373C0
Proc_1_2_438F20
Proc_1_3_43D010
Proc_1_4_43D3C0
Proc_1_5_43D560
Proc_1_6_43DDE0
Proc_1_7_43E140
Proc_1_8_43F9B0
Proc_1_9_440850
Proc_1_10_441780
Proc_1_11_441D90
Proc_1_12_442720
Proc_1_13_4436B0
Proc_1_14_443830
Proc_1_15_443990
Proc_1_16_443B40
Proc_1_17_444860
Proc_1_18_4454D0
Proc_1_19_445750
Proc_1_20_445900
Proc_1_21_446B50
Proc_1_22_4479C0
Proc_1_23_4485B0
Proc_1_24_448CA0
Proc_1_25_449160
Proc_1_26_449250
Proc_1_27_4495C0
Proc_1_28_449D70
Proc_1_29_44A190
Proc_1_30_44A240
Proc_1_31_44A760
Proc_1_32_44B090

Once injected, DarkCloud exhibits a wide range of malicious activities. It systematically gathers information from web browsers, collects system data, and even harvests credit card details. Additionally, it closely monitors running processes and logs keystrokes, making it a highly capable and well-rounded piece of malware.



As a curiosity, while I was collecting all the command lines it executes, as well as the registry entries and other accessed locations, I noticed that the credit card-related regex used by DarkCloud had been copied in order from a site on GitHub (@michaelkeevildown)

[michaelkeevildown Github]



I leave you with the compilation of interesting paths/logs/cmd that I have compiled based on the extracted samples.

> Features

- Screenshot
- Keylogger
- Get credentials
- Get Computer Name/Users
- Mail Info
- IMAP/HTTP/SMTP/FTP/NNTP/NNTP/POP3 Info
- Password managers
- Router info

> Paths/Files accessed

```
 \Default\Login Data\
\User Data\
\WebData\
\Logins.json
\key3.db
\key4.db
\keyDBPath.db
\signons.sqlite
\keyDBPath.sqlite
Storage\
mail\
Data\
\Accounts\Account.rec0
\Accounts\Account.tdat
\Account.stg
\Account.rec0
\Local State
```

```
\Microsoft\Windows\Templates\
\accounts.xml
\recentservers.xml
\sitemanager.xml

Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook\
Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\
Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\
Software\Microsoft\Windows Messaging Subsystem\Profiles\
Software\FTPWare\COREFTP\Sites
Software\Martin Prikryl\WinSCP 2\Sessions

\Google\Chrome\User Data
\Opera Software\Opera Stable
\Yandex\YandexBrowser\User Data
\360Chrome\Chrome\User Data
\Comodo\Dragon\User Data
\MapleStudio\ChromePlus\User Data
\BraveSoftware\Brave-Browser\User Data
\7Star\7Star\User Data
\CocCoc\Browser\User Data
\uCozMedia\Uran\User Data
\Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer
\CatalinaGroup\Citrio\User Data
\NETGATE Technologies\BlackHawK\Profiles
\8pecxstudios\Cyberfox\Profiles
```

Software

```
 Outlook
foxmail
pidgin
CoreFTP
WinSCP
FTPWare
FileZilla
Chrome
Opera
Yandex
360Chrome
Comodo
MapleStudio
Chromium
Torch
Brave
Iridium
```

```
7Star
Amigo
CentBrowser
Chedot
CocCoc
Elements Browser
Epic Privacy Browser
Kometa
Orbitum
Sputnik
uCozMedia
Vivaldi
Fenrir Inc
Citrio
Coowon
liebao
QIP Surf
Microsoft Edge
Mozilla
Waterfox
K-Meleon
Thunderbird
Cyberfox
BlackHawK (NetGate)
```

> Regex

```
 Proto

^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$
^(?!:\/\/)([a-zA-Z0-9-_]+\.)[a-zA-Z0-9][a-zA-Z0-9-_]+\.[a-zA-Z]{2,11}?$

Accounts (I don't put more because you have them in the github)

^3[47][0-9]{13}$

Amex
^(6541|6556)[0-9]{12}$

BCGlobal
^389[0-9]{11}$

Carte Blanche Card
^3(?:0[0-5]|[68][0-9])[0-9]{11}$
...
```

> Queries

```
 SELECT origin_url, username_value, password_value  FROM logins
SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted  FROM credit_cards
SELECT origin_url, username_value, password_value, length(password_value)  FROM logins
SELECT hostname, encryptedUsername, encryptedPassword  FROM moz_logins
SELECT item1 FROM metadata WHERE id = 'password';
SELECT a11,length(a11) FROM nssPrivate
Select * from Win32_Process
Select * from Win32_LogicalDisk
Select * from Win32_ComputerSystem
SELECT * FROM Win32_Processor
```

> RegKeys

```
 HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Foxmail.url.mailto\Shell\open\command\
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\
```

> BlackList Process

```
 fiddler
vxstream
tcpview
procexp
vmtools
autoit
wireshark
procmon
idaq
autoruns
apatedns
windbg
```

> IP public

```
 http://showip.net
http://www.mediacollege.com/internet/utilities/show-ip.shtml
```

_Outro

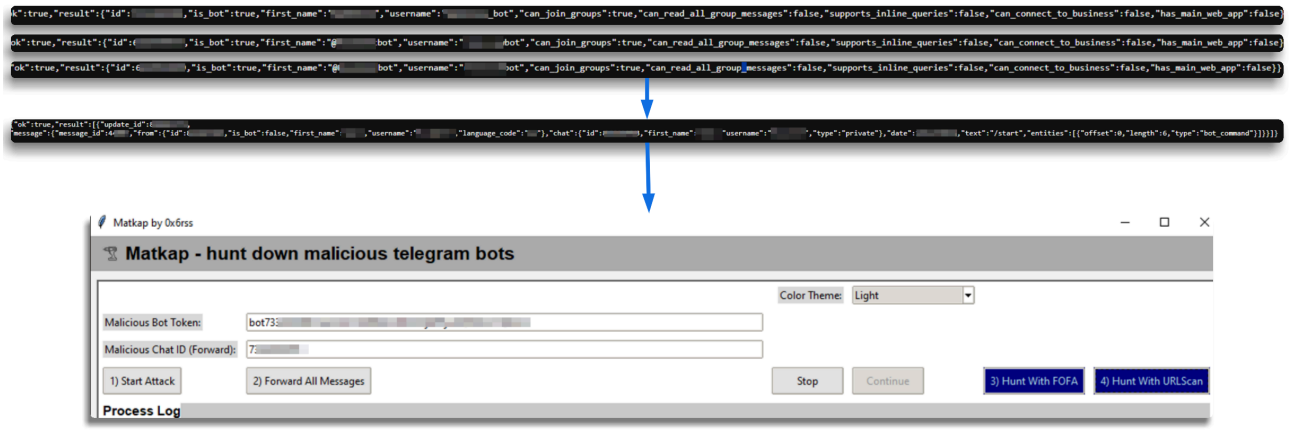
After the analysis, I checked what was going on in Telegram groups as well as in more exclusive groups. The truth is, they don’t have much activity or don’t present themselves as openly as other groups do. Moreover, there is quite a bit of confusion in the logs when differentiating between the botnet and the Stealer, which appears in almost all forums.

It is interesting to see how many of the samples communicate via bots, making it relatively easy to find information about them.

```
"ok":true,"result":[]}
```

Additionally, there are various tools that I am using for these tasks which, although they don't need much introduction in my opinion, I'll list here for reference.

@Gi7w0rm - Teletoken Web
@0x6rss - Matkap Github & New Matkap in Web



Finishing the analysis, I must say that I was surprised by the great variety of loaders I found for DarkCloud. It has been a challenge to unify all the versions I encountered into a coherent explanation, but it was truly interesting. I will continue working on the bot-related aspects to develop further analyses with a stronger focus on this area and gain deeper insight into the infrastructure used by those who deploy it

Finally, I would like to thank you for reading this analysis and for supporting me :)

_Detection Opportunities

- [TA0005][T1036] Duplication of original files or loaders in temporary paths

```
 (WriteFile) C:\Users\<user>\AppData\Local\Temp\*.exe|.vbs
(WriteFile) C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Templates\*.exe|.vbs
(WriteFile) C:\Users\<user>\AppData\Roaming\*.exe|.vbs
(WriteFile) C:\Users\Public\*.exe|.cmd|.vbs
```

- [TA0003][T1547.001] Startup vbs loader creation to persistence

```
(WriteFile) C:\Users\<user>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\<RandName>.vbs
```

- [TA0003][T1547.001] Registry RunOnce persistence

```
 (Registry) *\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
(ValueData) *\AppData\Roaming\Microsoft\Windows\Templates\<RandName>.exe
```

- [TA0007][T1217] Queries to browser paths or third-party software to obtain information

```
(Registry/Path query) \Default\Login Data\ | \User Data\ | \WebData\ | \Logins.json | \key3.db | \key4.db | \k
```

- [TA00011][T1071] Connection via noncommon process to TG bots

We can take into account the injected processes I mentioned before or processes in strange paths (Svchost, InstallUtil, MSBuild, injected Itself)

```
(OutBound connection) https?:\/\/api\.telegram\.org\/bot\d+:[A-Za-z0-9_-]+\/
```

_TTP

```
 [TA0001][T1566.001] SpearPhishing
[TA0002][T1059] Command and Scripting Interpreter
[TA0002][T1129] Shared Modules
[TA0002][T1204] User Execution
[TA0003][T1053] Scheduled Task/Job
[TA0003][T1547.001] Registry Run Keys / Startup Folder
[TA0005][T1027] Obfuscated Files or Information
[TA0005][T1027.002] Software Packing
[TA0005][T1036] Masquerading
[TA0005][T1055] Process Injection
[TA0005][T1140] Deobfuscate/Decode Files or Information
[TA0005][T1497] Virtualization/Sandbox Evasion
[TA0007][T1016] System Network Configuration Discovery
[TA0007][T1033] System Owner/User Discovery
[TA0007][T1057] Process Discovery
[TA0007][T1082] System Information Discovery
[TA0007][T1518] Software Discovery
[TA0009][T1005] Data from Local System
[TA0009][T1056.001] Keylogging
[TA0009][T1113] Screen Capture
[TA0009][T1114] Email Collection
[TA0009][T1115] Clipboard Data
[TA0009][T1560] Archive Collected Data
[TA0011][T1071] Application Layer Protocol
[TA0011][T1105] Ingress Tool Transfer
```

_Some Any.Run samples

https://app.any.run/tasks/f3c56abe-04b8-432a-8011-4135871daedc
https://app.any.run/tasks/b3aa2be4-c06e-4bfc-a05a-00b6b5a32c88
https://app.any.run/tasks/4e1b98c5-2d9d-4eca-80cf-42d7de800376

https://app.any.run/tasks/a2aa4e72-1e7b-4819-8a85-77f5145b884b
https://app.any.run/tasks/a36a7b27-c600-4512-92f8-a50e3b43b9f2
https://app.any.run/tasks/62189938-c9aa-4faf-af83-6f34934fc0e0
https://app.any.run/tasks/44386e5b-f252-4cd3-a759-9cea86266838
https://app.any.run/tasks/358ae682-9667-4538-8e03-231f8b550c08
https://app.any.run/tasks/53830d3d-1413-417e-bcc2-02266e731c1a
https://app.any.run/tasks/1564deb3-8c01-4334-a392-3579ff05e5e7
https://app.any.run/tasks/9f89708d-7f12-4fc0-9cb3-2c37bb40db63
https://app.any.run/tasks/74e6dfbb-7997-416b-af72-276e06692a7c
https://app.any.run/tasks/216ec201-b361-4b74-a063-0c7ed1736cbc
https://app.any.run/tasks/6787a231-8ae4-40f6-b9b6-9b6f8c5fdc99
https://app.any.run/tasks/ca51f009-cf34-4bc6-b382-69ea5165ca73
https://app.any.run/tasks/7a152247-6048-4721-96cf-df6266638678
https://app.any.run/tasks/e692ced4-8033-4021-943a-a7bbae338678

_IOC

I’m sorry because there is a lot, but I review many samples, it is interesting to see how they have affected different websites in different parts of the world to host the first stage (the section where loader download the sample).

```
 970791d129721273f0d53c1b4c352c689d3992ec3a236bfa5b4c39804821f64d
20dc4ffc31f978e2c822878b11a4d59c3ad6da9898a7028d75d3c9079598de18
5e45f400b15d22d484430e07d36c9b01eef5bc3069faa8c9fa8b1b88be93ef9a
ecbdaea604f259a7535b32c7abcd25139af9b8abe04a45d24614212c9d8ea67c
930715d4311f4b71e77c5baf6a90d5e9383dbb8d2557e8316054d4e46f019404
8eed39069dfa0574b41e3e3f426809e29f9b108d5106670e07b2c1524085b6e6
869ab9858bfbbb0e479ec4f2f3776ec952cdada4898e9f26d3322cb931c75cab
78c596ef6b627c0210c528adda497162ef323dcb880edc19a1baa67c34e04906
3477d3838f634d4e5489d7d16de56b91bf0bf31b32ac7e2dbccf657a24d487bc
9687a5f5f5b2625642144b433b807c37f968ba696722c1c173a0d7d5ece4d0d9
89b0fa84da8c99eed3aba8a931afb5a032ea0a47cb4b692a15e7b3aef7104ba8
b07a092954cc148342cdd4725f5e9d487155d22b794b273dc874dcf5f144de4e
e25450236e49bbb61e175e39eb46a2f3a1258e8ba8fe7a2c6d8b2f17e85cc167
f18d276ce3ec088f6d446e409489bad5c9b613d6c1f10a6538724749c7cf2af6
73ea9482307e2db84538256bcee3a207fbc8bf512715316c82f3b4cebe46d8b1
d148996c99c0e59c40792ce17a1531480168eb0e05d015ddd6f0fed9ce9478e5
f18d276ce3ec088f6d446e409489bad5c9b613d6c1f10a6538724749c7cf2af6
73ea9482307e2db84538256bcee3a207fbc8bf512715316c82f3b4cebe46d8b1
d59dbca0d404218e0f1729294f30c85d7f61b6e28cc7872d8ce1afb94cb3be42
aa653ad0d107b2d7ab98d4ede0eef147b73fbd7eb2f522f0bf608f833daebe34
688c929b7be5c31a2a5410394024f9dea1bcfc62af0c24237d2b23b8fea70055
80742a25d1550dd0f7ccb299672a5d9de889f57c0e53e3e8eea0e50d6b7ae33b
b7d2a499355e0ff23ee3abd5dcf62b27bfa1a993cb8e53b8a0600c0e4d4e4c56
093f9c611518f42f00826d8134b4860080b690678b27a469a33a529f7706b87f
dda65ed23373504099b4ef1fd883eca088c8664db398909dda87af9d4a02bdf1
```

```
a23430fb24ff7cf5506c58a46a33a1572a5c724fd9b4e0e88228584f6ec6001e
250fa7da2da2ff7e1e1f7e3bb2ff9ae8b363facb00973455e9fe1900e6195c41
8306710e0292b8c88b537bf427a9d651cfdf4d86ad2f7690b47f766b4354faaa
45b104579f856aba3d99b5370360240a3f9f95b82969e3f7ba12c14b0d7abb7f
7c8bcd4345ec5eb572bf25a4831ebf09c98bc92c0847662fd68f7cfefce4a27f
afce524a6ff12252be335bfa95209bc37ca8de86e45b2c34634cc6223fe83263
8616ff3374bcb45c5eda66c11ffa3fbef9e7e00e2ad21e752c10bf94bdd7a28d
db03cf79dd756e39c7036b3541c2b560b4b3a7f0ccd4b896de47ba743745726d
B4bb90f3a33ac5c78672efc80987f80e0885753b8559bf1d258acf99d04dfd1b
B2fe3bf20cb54f650d4b191315cf9247b653a5c3552f9f3d6598990b58139114
B4bb90f3a33ac5c78672efc80987f80e0885753b8559bf1d258acf99d04dfd1b
Dd12a49d57b0dbc1acbfc1493529128a8c9017835a5a008397be8ba2641cbfee
9726f460c40a7274eb35203cec6400f3d3de9dcf5e841ffc78bec26057cc2d6e
782529daacf0b40364707f5b98f01b13a2a5f3d2b48fee351723495affc03940


196[.]251[.]92[.]64
144[.]91[.]79[.]54
87[.]120[.]120[.]56
161[.]132[.]98[.]130
165[.]154[.]217[.]184
204[.]44[.]192[.]90
190[.]171[.]170[.]94
161[.]132[.]98[.]130
176[.]65[.]144[.]3
144[.]91[.]79[.]54


//Affected websites

krupasindhudevelopers[.]com
www[.]arandelasespeciales[.]com
alcomax[.]com[.]co
toyscenter[.]cl
centuryharvestlink[.]com
gugaequiposyservicios[.]com[.]mx
hngandpartners[.]com
```

Source: https://rexorvc0.com/2025/03/31/DarkCloud/