BumbleBee DocuSign Campaign

March 4, 2023

Breakdown of a BumbleBee PowerShell Dropper & extracting the config of BumbleBee

6 minute read

0xToxin

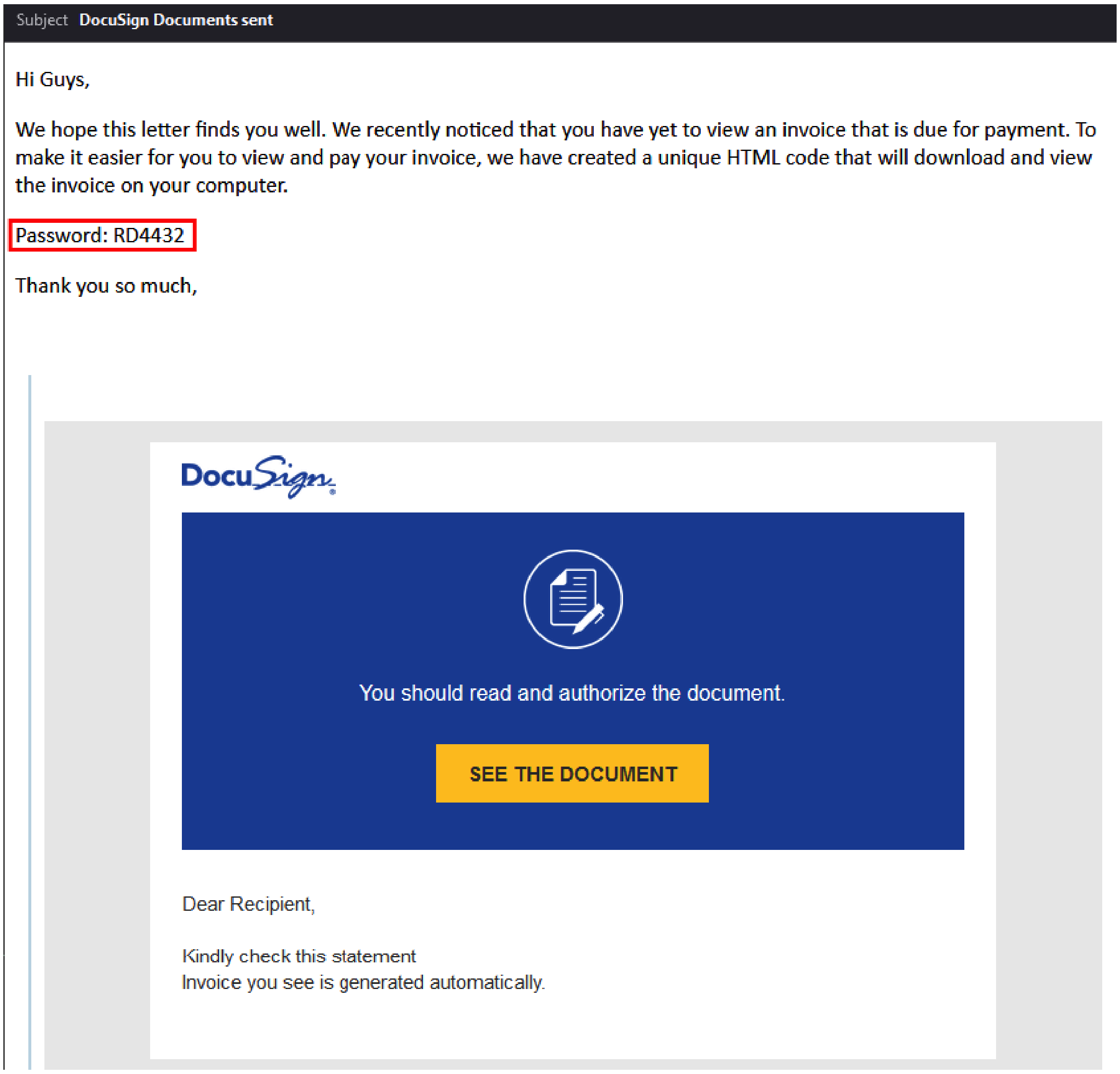
Threat Analyst & IR team leader - Malware Analysis - Blue Team

Intro

In this blog post I will be going through a recent bumblebee camapign that impersonates DocuSign, I will be going through the execution chain, the powershell loader and some IOC extractions

The Phish

The email delivered to the user simply tells the user that an invoice is waiting to be paid and that a “unique HTML code” was created for him to download and view the invoice on the user’s computer. Additionally a password was provided: `RD4432`


Subject DocuSign Documents sent

Hi Guys,

We hope this letter finds you well. We recently noticed that you have yet to view an invoice that is due for payment. To make it easier for you to view and pay your invoice, we have created a unique HTML code that will download and view the invoice on your computer.

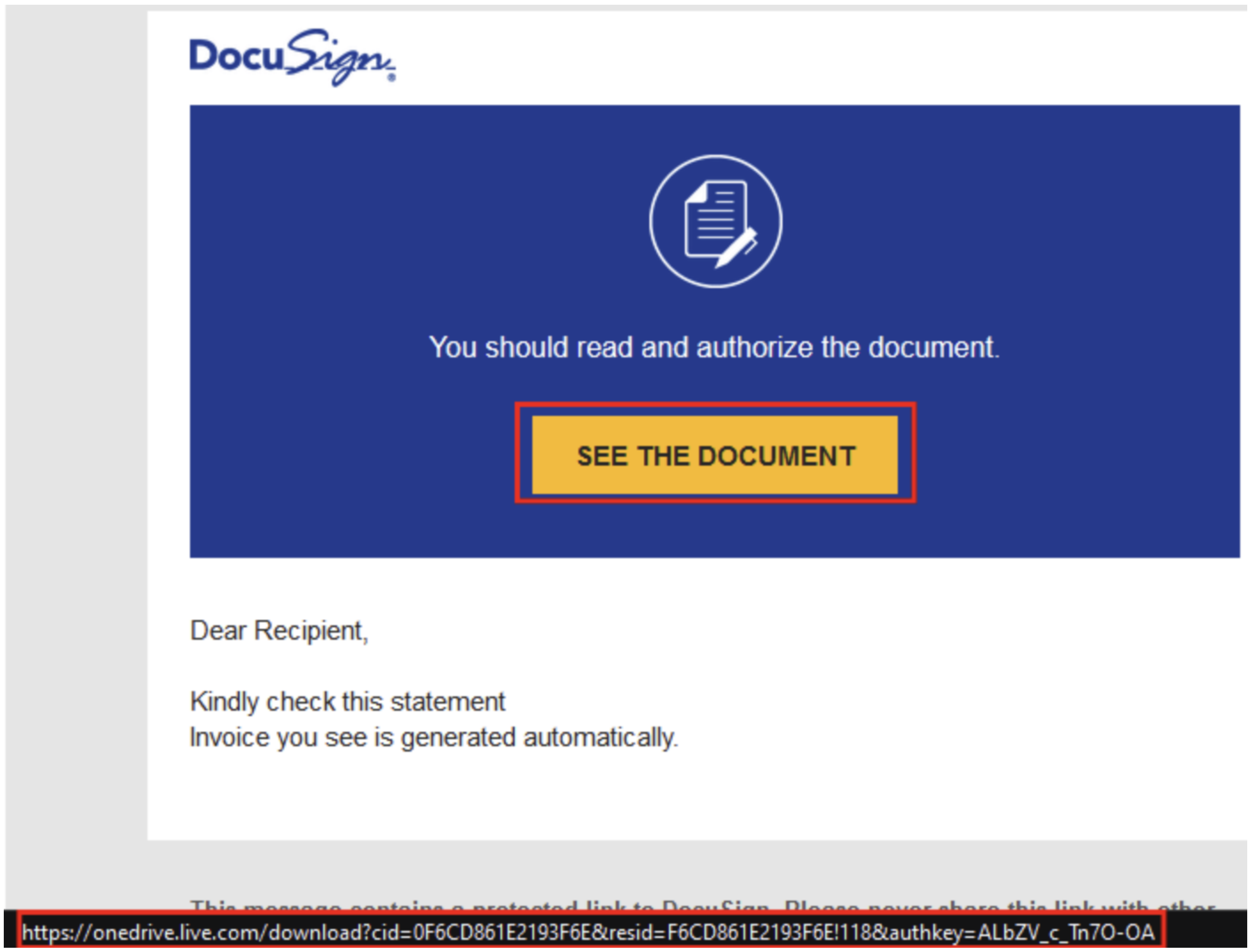
Password: RD4432

Thank you so much,

Dear Recipient,

Kindly check this statement
Invoice you see is generated automatically.

Hovering over the the **"See The Document"** can help us to see what is the click on action URL:


DocuSign

You should read and authorize the document.

SEE THE DOCUMENT

Dear Recipient,

Kindly check this statement
Invoice you see is generated automatically.

This message contains a protected link to DocuSign. Please never share this link with other

https://onedrive.live.com/download?cid=0F6CD861E2193F6E&resid=F6CD861E2193F6E!118&authkey=ALbZV_c_Tn7O-OA

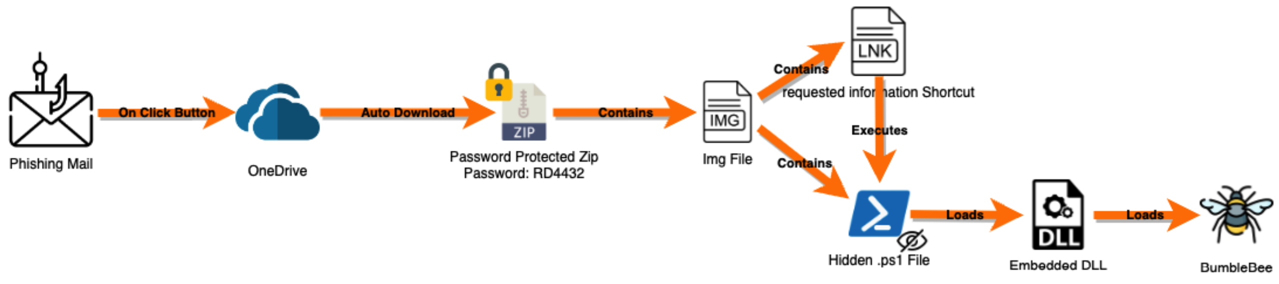
The URL is:

```
https://onedrive.live[.com/download?
cid=0F6CD861E2193F6E&resid=F6CD861E2193F6E%21118&authkey=ALbZV_c_Tn7O-OA
```

so instead of going to the actual DocuSign site, the file will be hosted on onedrive which once clicked will trigger an auto download of an archive file.

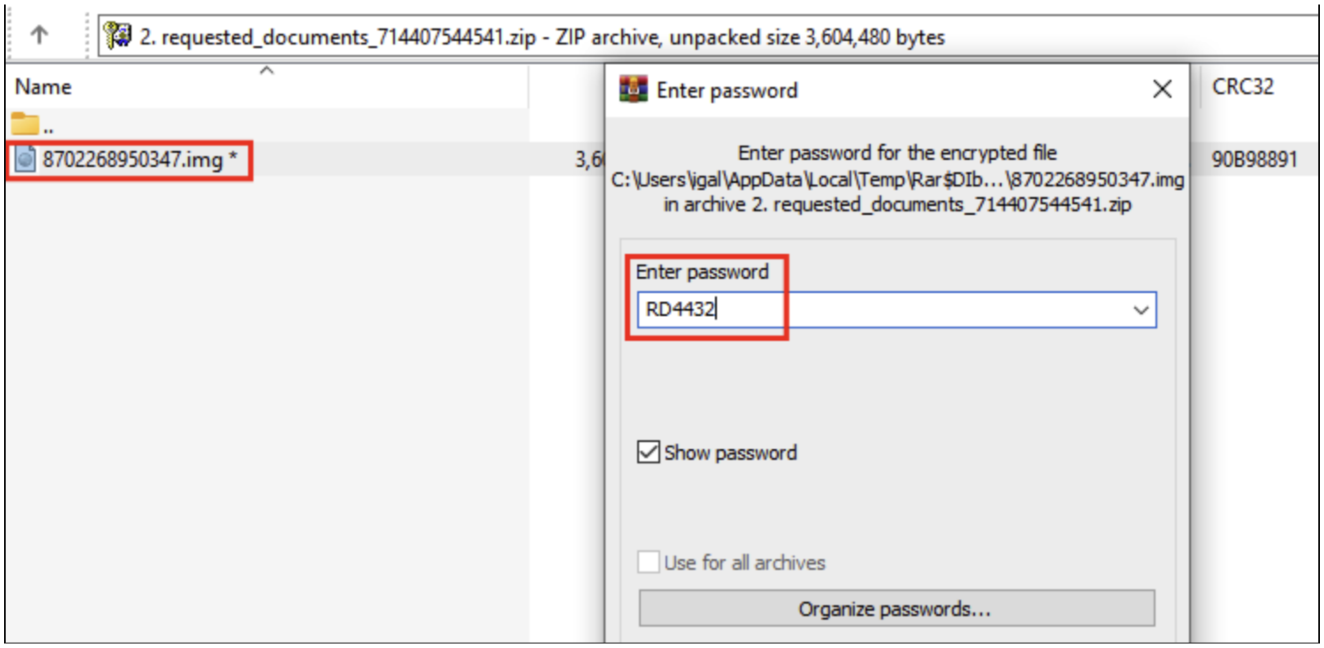
Execution Chain

Below you can see a diagram of the execution chain from the moment the phishing mail was opened:



Lets go quickly through this chains:

> Downloaded archive is being opened by the user, in order to extract the IMG file the user will have to enter the given password: `RD4432`



2. requested_documents_714407544541.zip - ZIP archive, unpacked size 3,604,480 bytes

Name		CRC32
..		
8702268950347.img *	3,6	90B98891

Enter password

Enter password for the encrypted file
C:\Users\igal\AppData\Local\Temp\Rar$DIb...\8702268950347.img
in archive 2. requested_documents_714407544541.zip

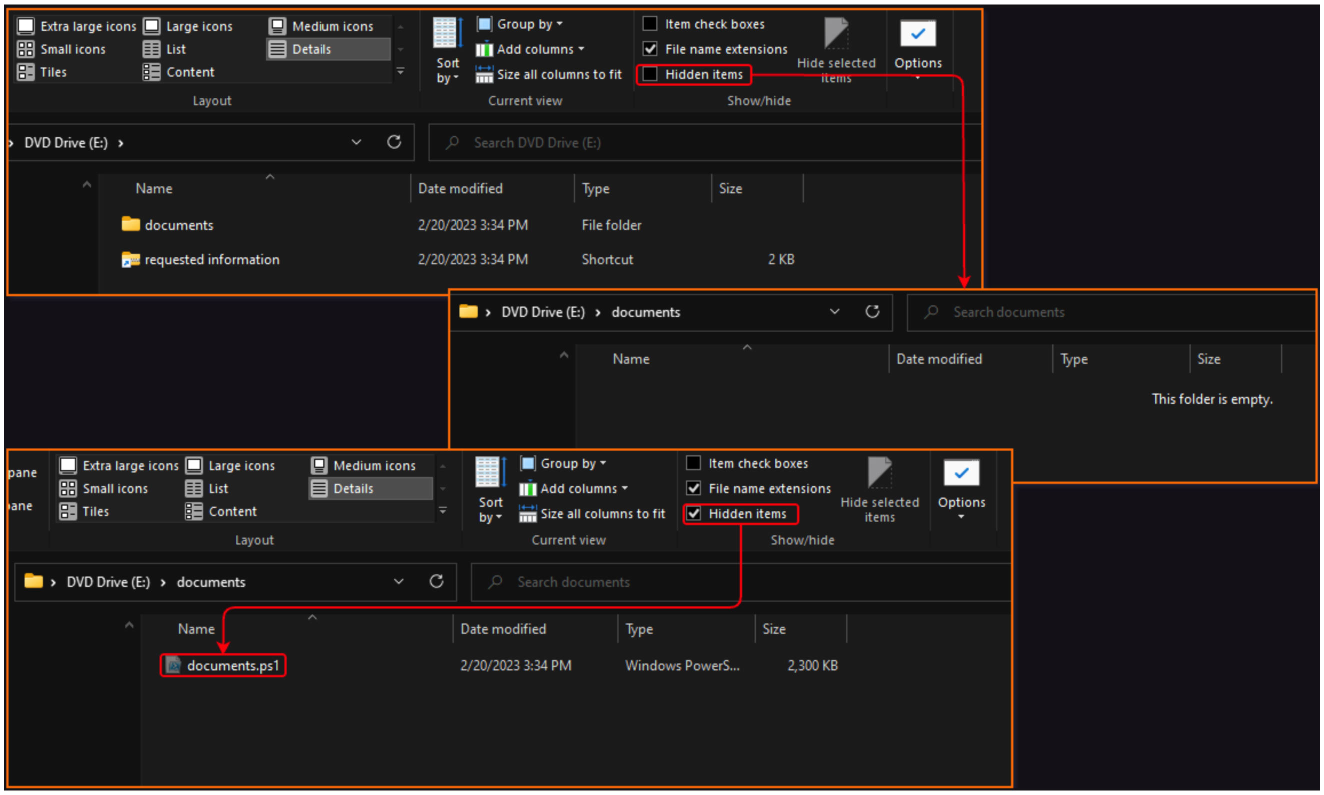
Enter password: RD4432

☑ Show password

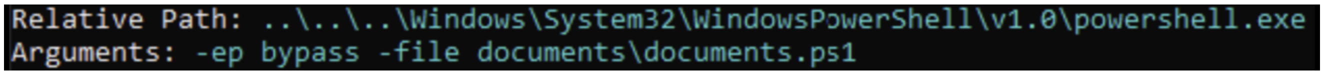
☐ Use for all archives

Organize passwords...

> Once the IMG file is opened the user will see only the LNK file `requested information` (because the .ps1 is hidden)



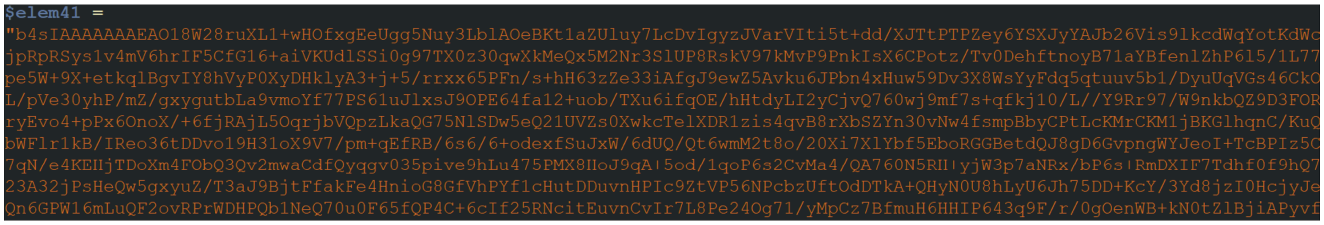
The LNK file will execute the hidden .ps1 script



```
Relative Path: ..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Arguments: -ep bypass -file documents\documents.ps1
```

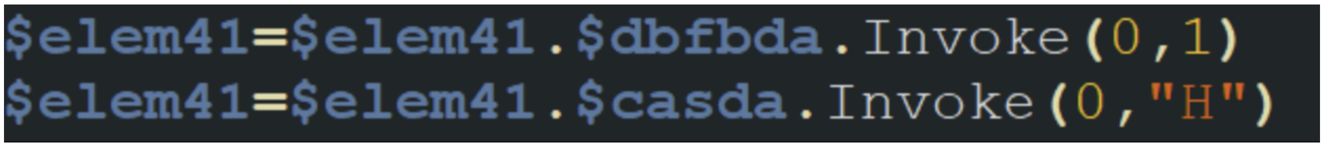
Bumblebee Ps1 Loader

I will be focusing now on what is going on in the script and what I've done to extract the payload out of it. So I know that there are about 42 base64 encoded strings (that are actually archives) each one of them stored in variable with the name **elem{X}** , for example:



```
$elem41 =
"b4sIAAAAAAAEAO18W28ruXL1+wHOfxgEeUgg5Nuy3LblAOeBKt1aZUluy7LcDvIgyzJVarVIti5t+dd/XJTtPTPZey6YSXJyYAJb26Vis9lkcdWqYotKdWc
jpRpRSys1v4mV6hrIF5CfG16+aiVKUdlSSi0g97TX0z30qwXkMeQx5M2Nr3SlUP8RskV97kMvP9PnkIsX6CPotz/Tv0DehftnoyB71aYBfenlZhP6l5/1L77
pe5W+9X+etkqlBgvIY8hVyP0XyDHklyA3+j+5/rrxx65PFn/s+hH63zZe33iAfgJ9ewZ5Avku6JPbn4xHuw59Dv3X8WsYyFdq5qtuuv5b1/DyuUqVGs46CkO
L/pVe30yhP/mZ/gxygutbLa9vmoYf77PS61uJlxsJ9OPE64fa12+uob/TXu6ifqOE/hHtdyLI2yCjvQ760wj9mf7s+qfkj10/L//Y9Rr97/W9nkbQZ9D3FOR
ryEvo4+pPx6OnoX/+6fjRAjL5OqrjbVQpzLkaQG75NlSDw5eQ21UVZs0XwkcTelXDR1zis4qvB8rXbSZYn30vNw4fsmpBbyCPtLcKMrCKM1jBKGlhqnC/KuQ
bWFlr1kB/IReo36tDDvo19H31oX9V7/pm+qEfRB/6s6/6+odexfSuJxW/6dUQ/Qt6wmM2t8o/20Xi7XlYbf5EboRGGBetdQJ8gD6GvpngWYJeoI+TcBPIz5C
7qN/e4KEIIjTDoXm4FObQ3Qv2mwaCdfQyqgv035pive9hLu475PMX8IIoJ9qA|5od/1qoP6s2CvMa4/QA760N5RII|yjW3p7aNRx/bP6s|RmDXIF7Tdhf0f9hQ7
23A32jPsHeQw5gxyuZ/T3aJ9BjtFfakFe4HnioG8GfVhPYf1cHutDDuvnHPIc9ZtVP56NPcbzUftOdDTkA+QHyN0U8hLyU6Jh75DD+KcY/3Yd8jzI0HcjyJe
Qn6GPW16mLuQF2ovRPrWDHPQb1NeQ70u0F65fQP4C+6cIf25RNcitEuvnCvIr7L8Pe24Og71/yMpCz7BfmuH6HHIP643q9F/r/0gOenWB+kN0tZlBjiAPyvf
```

The script then removes the first char in the encoded string and replace it with `H` to match the `.gz` magic bytes: `1f 8b`.



```
$elem41=$elem41.$dbfbda.Invoke(0,1)
$elem41=$elem41.$casda.Invoke(0,"H")
```

This script will extract each string variable, decode it and save in the selected folder

```
from base64 import b64decode
import re
import os

PS1_FILE_PATH = '/Users/igal/malwares/bumblebee/21-02-2023/documents.ps1'
OUTPUT_FOLDER = '/Users/igal/malwares/bumblebee/21-02-2023/archives/'

REG_PATTERN = '^\$elem.*\=\"(.*)\"$'

archiveIndex = 0

if not os.path.exists(OUTPUT_FOLDER):
    os.makedirs(OUTPUT_FOLDER)

ps1File = open(PS1_FILE_PATH, 'rb').readlines()
for line in ps1File:
    regMatch = re.findall(REG_PATTERN, line.replace(b'\x00',b'').decode('iso-8859-
1'))
    if regMatch:
        varData = b64decode('H' + regMatch[0][1:])
        open(f'{OUTPUT_FOLDER}/archive{archiveIndex}.gz', 'wb').write(varData)
        print(f'[+] gz archive was created in:
{OUTPUT_FOLDER}/archive{archiveIndex}.gz')
        archiveIndex += 1
```

```
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive0.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive1.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive2.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive3.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive4.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive5.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive6.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive7.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive8.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive9.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive10.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive11.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive12.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive13.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive14.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive15.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive16.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive17.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive18.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive19.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive20.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive21.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive22.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive23.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive24.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive25.gz
```

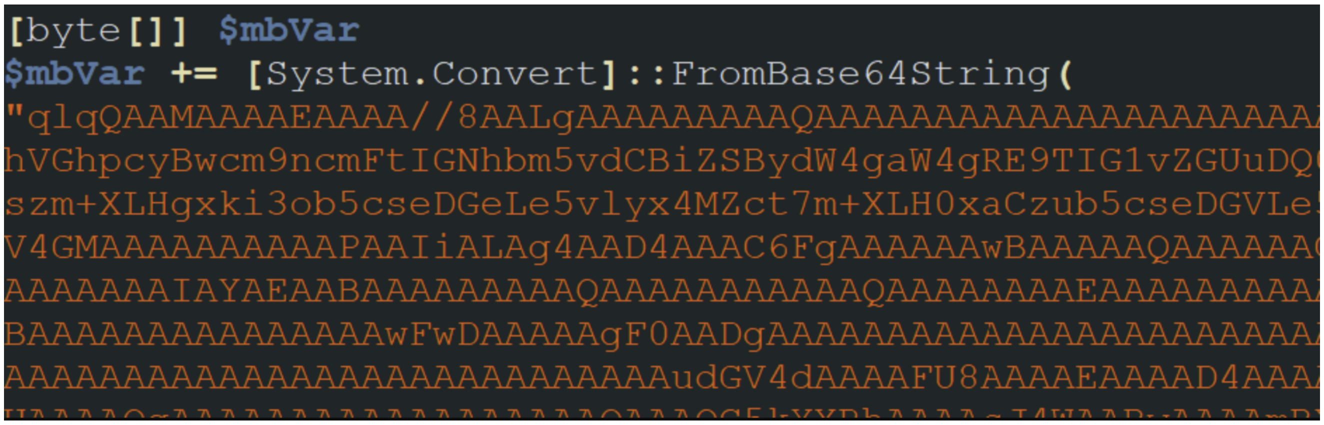
```
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive26.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive27.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive28.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive29.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive30.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive31.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive32.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive33.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive34.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive35.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive36.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive37.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive38.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive39.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive40.gz
[+] gz archive was created in:/Users/igal/malwares/bumblebee/21-02-
2023/archives//archive41.gz
```

Each archive contains code parts of a bigger powershell script, I will extract the content of those archives and concatenate them to one big powershell script.

```
import gzip


ARCHIVES_FOLDER = '/Users/igal/malwares/bumblebee/21-02-2023/archives'
OUTPUT_FILE = '/Users/igal/malwares/bumblebee/21-02-2023/powershellCommand.txt'

countArchives = sum(1 for file in os.scandir(ARCHIVES_FOLDER))

finalString = ''

for x in range(0,countArchives):
    with gzip.open(f'{ARCHIVES_FOLDER}/archive{x}.gz', 'rb') as f:
        finalString += f.read().decode('utf-8')

open(OUTPUT_FILE, 'w').write(finalString)
```

```
2074441
```

Once again the script contains a huge amount of b64 encoded strings that once concatenated they create an executable.

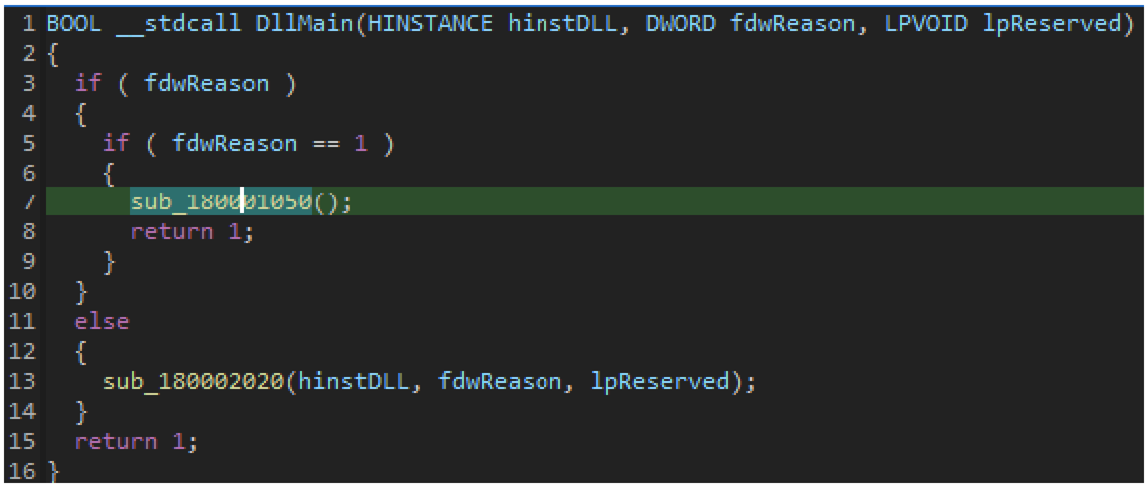


```
[byte[]] $mbVar
$mbVar += [System.Convert]::FromBase64String(
"qlqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAA
hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1vZGUuDQ
szm+XLHgxki3ob5cseDGeLe5vlyx4MZct7m+XLH0xaCzub5cseDGVLe
V4GMAAAAAAAAAPAAIiALAg4AAD4AAAC6FgAAAAAAwBAAAAAQAAAAAA
AAAAAAAIAYAEAABAAAAAAAAQAAAAAAAAAAQAAAAAAAEAAAAAAAAA
BAAAAAAAAAAAAAAwFwDAAAAAgF0AADgAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAudGV4dAAAAFU8AAAAEAAAAD4AAA
```

```
ps1FileContent = open(OUTPUT_FILE, 'r').readlines()
REG_PATTERN = '^\$mbVar.*FromBase64String\(\"(.*)\"\)$'
OUTPUT_PAYLOAD = '/Users/igal/malwares/bumblebee/21-02-2023/payload.bin'
finalPayload = b''
for line in ps1FileContent:
    regMatch = re.findall(REG_PATTERN, line)
    if regMatch:
        finalPayload += b64decode(regMatch[0])

open(OUTPUT_PAYLOAD, 'wb').write(b'\x4d' + finalPayload[1:])
print(f'[+] Payload was extracted to the path:{OUTPUT_PAYLOAD}')
```

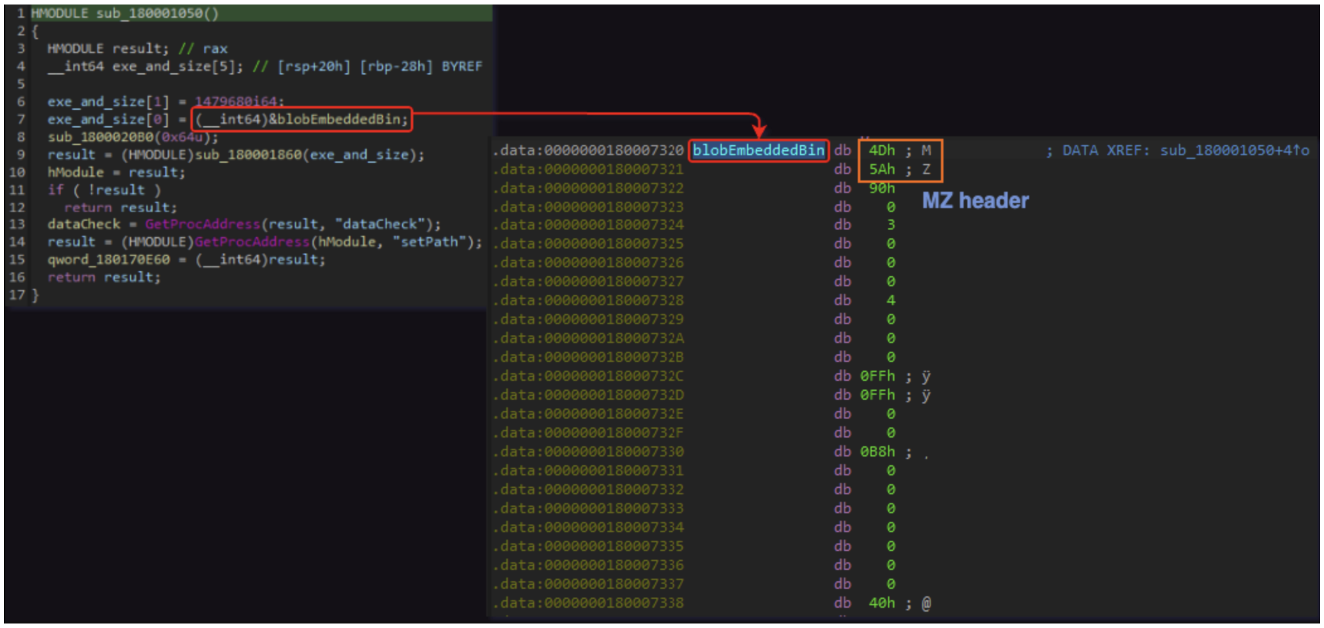
```
[+] Payload was extracted to the path:/Users/igal/malwares/bumblebee/21-02-
2023/payload.bin
```

Investigating the extracted binary, I found out it’s 64bit DLL, I’ve opened the DLL in IDA to see what is being executed from `DLLMain`:



```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  if ( fdwReason )
  {
    if ( fdwReason == 1 )
    {
      sub_180001050();
      return 1;
    }
  }
  else
  {
    sub_180002020(hinstDLL, fdwReason, lpReserved);
  }
  return 1;
}
```

DLLMain will execute the function `sub_180001050` which contains interesting array variable, which has in it's first value a pointer to `MZ` blob and in the second value what seems like the size of the blob:



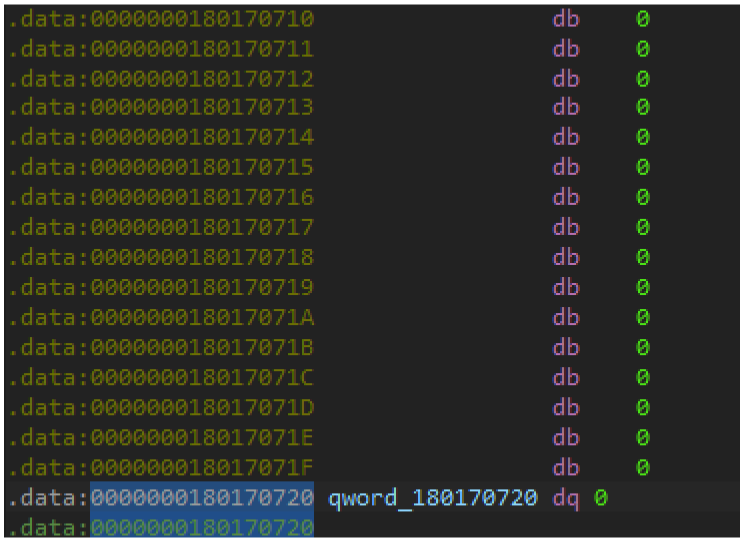
```
HMODULE sub_180001050()
{
  HMODULE result; // rax
  __int64 exe_and_size[5]; // [rsp+20h] [rbp-28h] BYREF

  exe_and_size[1] = 1479680i64;
  exe_and_size[0] = (__int64)&blobEmbeddedBin;
  sub_1800020B0(0x64u);
  result = (HMODULE)sub_180001860(exe_and_size);
  hModule = result;
  if ( !result )
    return result;
  dataCheck = GetProcAddress(result, "dataCheck");
  result = (HMODULE)GetProcAddress(hModule, "setPath");
  qword_180170E60 = (__int64)result;
  return result;
}
```

```
.data:0000000180007320 blobEmbeddedBin db  4Dh ; M          ; DATA XREF: sub_180001050+4↑o
.data:0000000180007321                 db  5Ah ; Z
.data:0000000180007322                 db  90h
.data:0000000180007323                 db    0
.data:0000000180007324                 db    3
.data:0000000180007325                 db    0
.data:0000000180007326                 db    0
.data:0000000180007327                 db    0
.data:0000000180007328                 db    4
.data:0000000180007329                 db    0
.data:000000018000732A                 db    0
.data:000000018000732B                 db    0
.data:000000018000732C                 db 0FFh ; ÿ
.data:000000018000732D                 db 0FFh ; ÿ
.data:000000018000732E                 db    0
.data:000000018000732F                 db    0
.data:0000000180007330                 db 0B8h ; .
.data:0000000180007331                 db    0
.data:0000000180007332                 db    0
.data:0000000180007333                 db    0
.data:0000000180007334                 db    0
.data:0000000180007335                 db    0
.data:0000000180007336                 db    0
.data:0000000180007337                 db    0
.data:0000000180007338                 db  40h ; @
```

I took the starting offset of the blob (`0x180007320`) and addded the possible length (`0x169400`) (wrote it in the IDA output window)

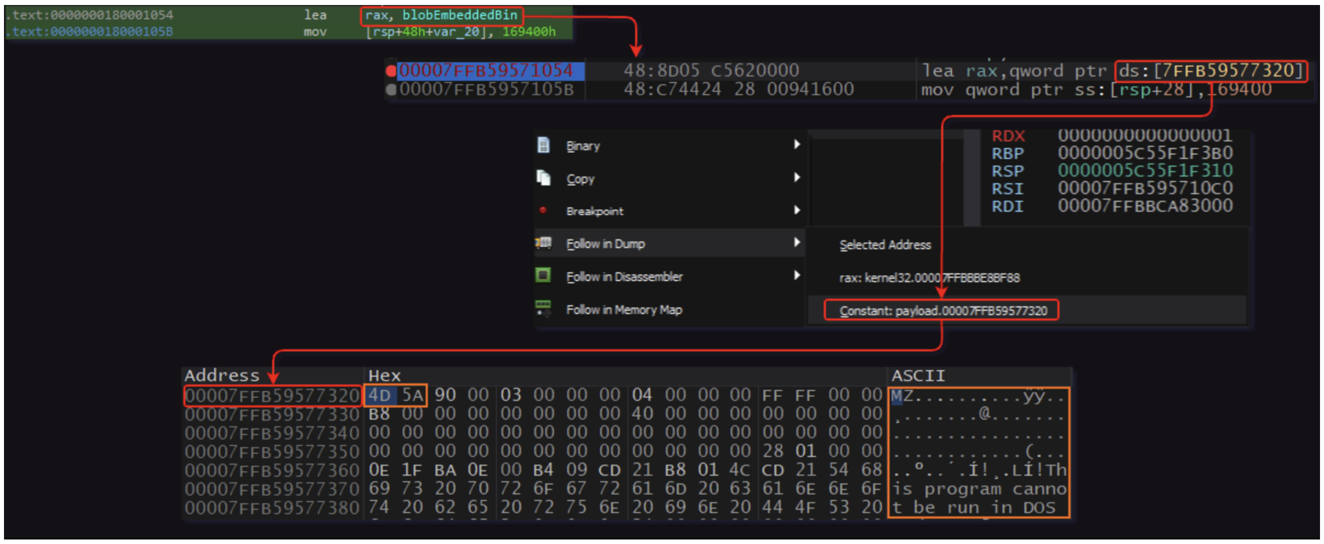
```
print(hex(0x180007320 + 0x169400))
```

And by double-clicking on the printed value it jumped to the offset which was the actual end of the blob data:



```
.data:0000000180170710                 db    0
.data:0000000180170711                 db    0
.data:0000000180170712                 db    0
.data:0000000180170713                 db    0
.data:0000000180170714                 db    0
.data:0000000180170715                 db    0
.data:0000000180170716                 db    0
.data:0000000180170717                 db    0
.data:0000000180170718                 db    0
.data:0000000180170719                 db    0
.data:000000018017071A                 db    0
.data:000000018017071B                 db    0
.data:000000018017071C                 db    0
.data:000000018017071D                 db    0
.data:000000018017071E                 db    0
.data:000000018017071F                 db    0
.data:0000000180170720 qword_180170720 dq 0
.data:0000000180170720
```

I've opened the binary in x64Dbg and set a breakpoint at the array assign of the blob and dumped the embedded binary:



```
.text:0000000180001054    lea    rax, blobEmbeddedBin
.text:000000018000105B    mov    [rsp+48h+var_20], 169400h

00007FFB59571054   48:8D05 C5620000     lea rax,qword ptr ds:[7FFB59577320]
00007FFB5957105B   48:C74424 28 00941600   mov qword ptr ss:[rsp+28],169400

RDX   0000000000000001
RBP   0000005C55F1F3B0
RSP   0000005C55F1F310
RSI   00007FFB595710C0
RDI   00007FFBBCA83000
```

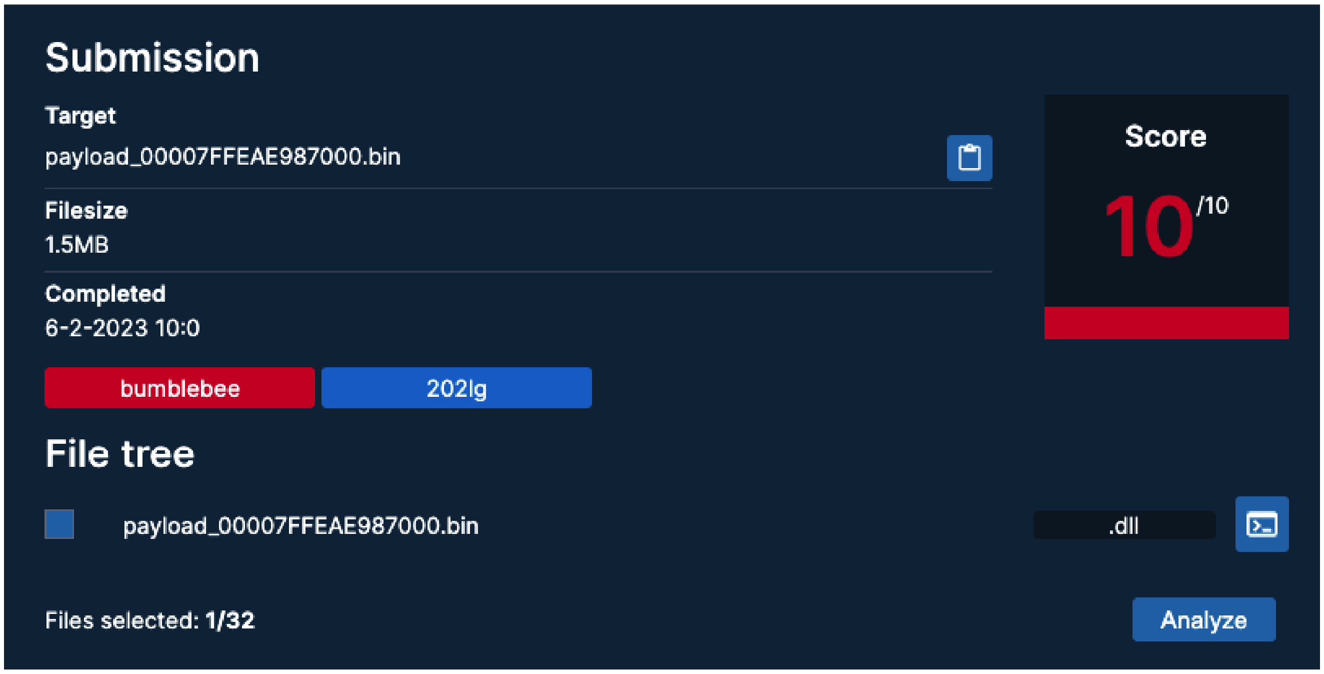
```
Address           Hex                                              ASCII
00007FFB59577320  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00007FFB59577330  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00007FFB59577340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FFB59577350  00 00 00 00 00 00 00 00 00 00 00 00 28 01 00 00  ............(...
00007FFB59577360  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00007FFB59577370  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00007FFB59577380  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
```

Now we can investigate the embedded binary.

BumbleBee Payload

In this part I will going over a quick triage process of extracting encrypted configs located in the BumbleBee payload.

First of all by simply uploading the payload to Tria.ge we get a static incrimination that the payload is indeed BumbleBee payload:



Submission

Target
payload_00007FFEAE987000.bin

Filesize
1.5MB

Completed
6-2-2023 10:0

bumblebee 202lg

Score
10/10

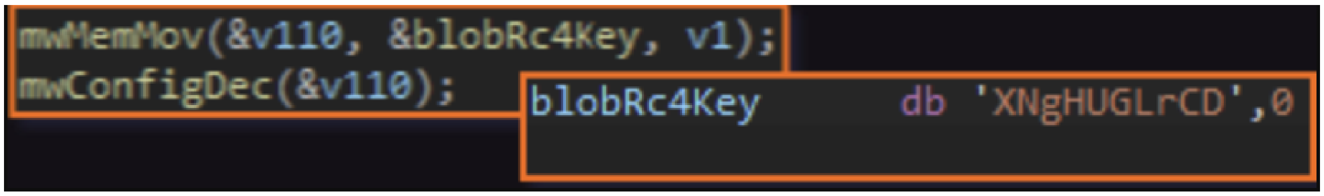
File tree

payload_00007FFEAE987000.bin .dll

Files selected: 1/32

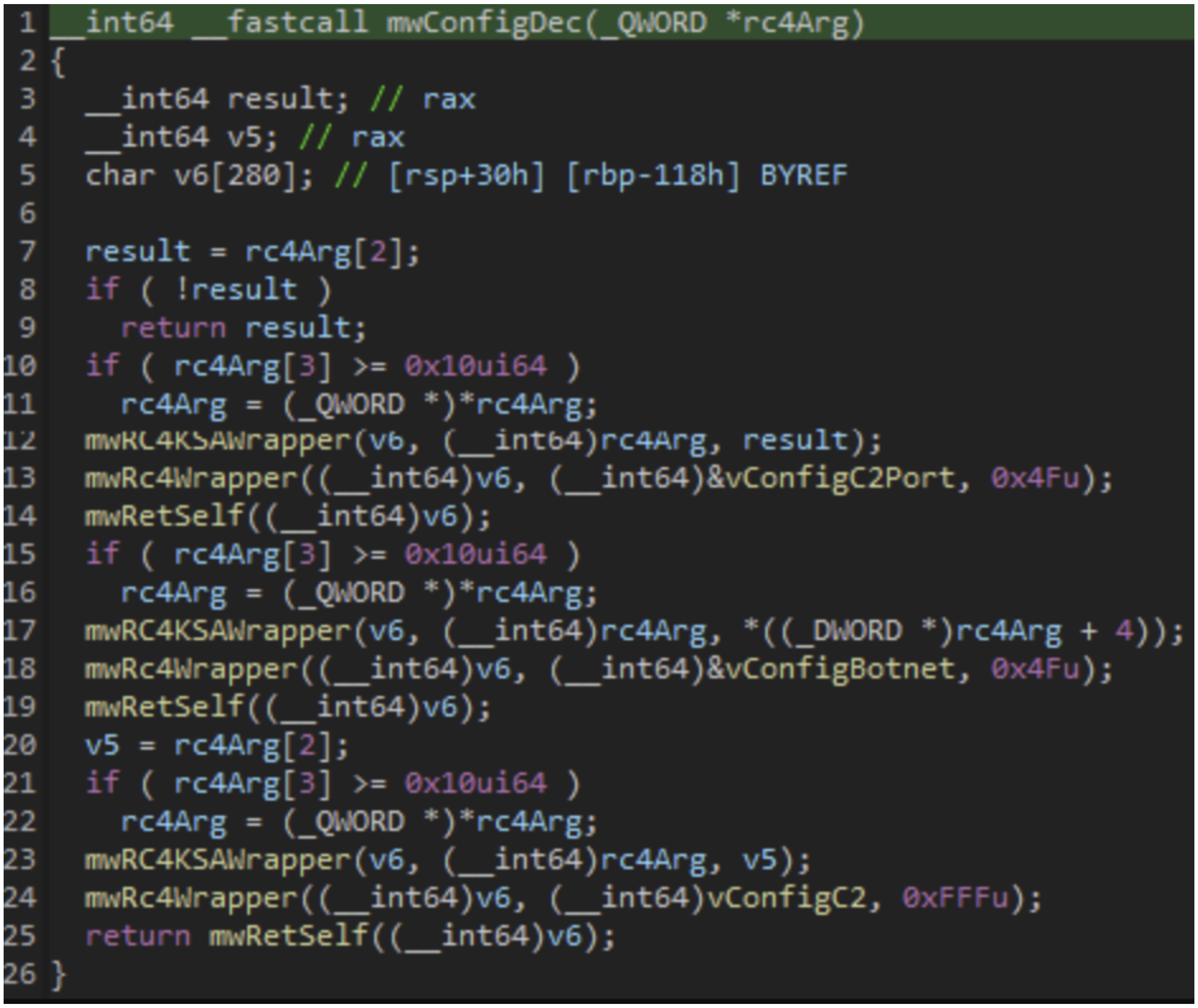
Additionally Tria.ge shows us the botnet ID which is: `202lg`.

Going through what possibly can be the main function of the loader I saw pretty at the beginning of the function a call to a function which pass as an argument an hardcoded strange looking string:



```
mwMemMov(&v110, &blobRc4Key, v1);
mwConfigDec(&v110);
blobRc4Key      db 'XNgHUGLrCD',0
```

The function contains inside of it RC4 encryptions routines that will use the hardcoded passed argument as a key and will pass alongside with it encrypted blob of data and the length of the data



```
__int64 __fastcall mwConfigDec(_QWORD *rc4Arg)
{
  __int64 result; // rax
  __int64 v5; // rax
  char v6[280]; // [rsp+30h] [rbp-118h] BYREF

  result = rc4Arg[2];
  if ( !result )
    return result;
  if ( rc4Arg[3] >= 0x10ui64 )
    rc4Arg = (_QWORD *)*rc4Arg;
  mwRC4KSAWrapper(v6, (__int64)rc4Arg, result);
  mwRc4Wrapper((__int64)v6, (__int64)&vConfigC2Port, 0x4Fu);
  mwRetSelf((__int64)v6);
  if ( rc4Arg[3] >= 0x10ui64 )
    rc4Arg = (_QWORD *)*rc4Arg;
  mwRC4KSAWrapper(v6, (__int64)rc4Arg, *((_DWORD *)rc4Arg + 4));
  mwRc4Wrapper((__int64)v6, (__int64)&vConfigBotnet, 0x4Fu);
  mwRetSelf((__int64)v6);
  v5 = rc4Arg[2];
  if ( rc4Arg[3] >= 0x10ui64 )
    rc4Arg = (_QWORD *)*rc4Arg;
  mwRC4KSAWrapper(v6, (__int64)rc4Arg, v5);
  mwRc4Wrapper((__int64)v6, (__int64)vConfigC2, 0xFFFu);
  return mwRetSelf((__int64)v6);
}
```

So now that we know what the data is let’s implement a quick decryption script:

```
from Crypto.Cipher import ARC4
import binascii

KEY = "XNgHUGLrCD"
BLOB_CONFIG_PORT =
"0b002425baa537efd52cf61f683f8116bc994d01c892b9c140f4a29c3f8a0b823f5a65b8dc08bb73c1e7
ec5f5cb40ca4a45ea741c5367ad2368ea826d4e90a4c2f986b4cfd78e1038028d261f872279b"
BLOB_CONFIG_BOTNET =
"0d042549dda537efd52cf61f683f8116bc994d01c892b9c140f4a29c3f8a0b823f5a65b8dc08bb73c1e7
ec5f5cb40ca4a45ea741c5367ad2368ea826d4e90a4c2f986b4cfd78e1038028d261f872279b"
BLOB_CONFIG_C2 =
"0e00260b8b9306c1e418c531590cb72c8eae7f2dfaa38def77c38ca50ca439b30a60578eef248a43f5c9
dd69649a3d9193709574f60c4ee605a2991fe2c7387e1bb6597ccd56d033ba1cea53d44012afc739f7fc6
717d5691f8421ce19ceff182407a705418cd7b6ac92b685308988032dee724be6d2d897fdb031ca1fe74a
115369eba75a7d1daa2cdab50ad2dc4920229b4a2d03204dff76d0787efee3b3f6129e23c56f0db2638f5
4548b8eeb0e33634b56f78e218b807a7d8088e19961165acaa738d226d897cdeb9ee1e1361a7f4eefbabb
f8f3c4d4d23dc724df1db8b62a51edea15eaf11988094fc0d172e91da11c5121a0e663522d7a06e51ea6d
db531b4a89de69b8bb3a0622046d2bd26577e763a52bd55c9fc33855585a1cadd57f1167c3ce0a60068cc
0aacfb5ddfa59baa6cae138f29e554ea51317e6f23f9a70339816c413b1b8501e2a1231b602666b3a5619
06f752cca682fd8c48c3256c148ccddb89e0199b2122fe3c06d8f23d727215431e8358692c59cf291d03e
b57a92c0d031d604e79b10bd0c5231ccd940ab30ece0d0e33aa12062c0bc48bd49b223de7088768a9a3a8
90e1277bba2006b42eaedca10a57d85a306f8b3560b6f79a95e51732c109a26c47086f31ba8456cb182be
49f2f2bf11d8fb1971e2b296495eb463a018612c4465ebbd0b097fd0c9566f75aa7c643926b4dd7966df7
5b5957445af826e48e0084efb68c73aaaed0dc2b47e7ef70673ee1d8c812bde7c77ba274b8348beceac75
93dd444f513072ec832407173ccff8f46f1fa224bbe2f96990d41e837c37278688270e4c706085ec36c48
02787c168637654543efccf22cd3a1b4561064c301d5fe5fa9add6281e396c7b6b6053a109a6e12cbb9b6
7b34559fb0cd8834136cec8ffca715cae19cf831d8f592ca87196008a89178522ccee395fbfbc1407843d
df216566a8cc35b05b548f09616aeb896757389e526744fb9f5a83e9f338d890418acb698fd882ea2acae
9c82f5d943f579452ab185c2bb8f8a80eb294efd9b70636f0fadc1b8f5055e23dd9c6f5687e6f149797b7
1a95beeb6dc9f4dafa3a9d6cc8eb77557e87f09bf6f96f581dba6bf2f41c0641a43c065c992efd9df4b4a
58ae807bbb46d3a83181705ebd3f95f3d869fb485ba7274526560dc2479a35c664fb55ae74612a767a5fc
2b50c3034d5bab9092bccb4c9ad0bc5e654481e4e5b1f85f8a4fbe2cc26e1442a808bf6b5f90564733329
b4a5a62d815175a15c8b4718f6c16d474957f014d2b930fd7eea94ce485d0a9ecbd36a0391205b4cecbe3
9b2e39ed4a7b1e2c3698b0a66ba9594fe35cd21abdb578c984bcbed482fce3f5bb209ed7f8e8009238656
fd5eda5a5e2cfec507d065960a01d051f4f7af0378714c4fb8040a46fea6b74a6278f36cdac1867a7a429
c4aff551a381df09492aeca2bfb3e0774a3160f21ad9b82a984590e7fa84b6bb01a0a59f882d316203c7d
910a4c27a26d30babb8cbbf4997c47a5b3721c7fa578a0e458834bb27ffbe71d22ede304d8d89077e4d69
dd7c215451cae24ebc9459d7e6ddbea97cd307fac9f9d8d0e4657528ce78054494b85f25b49bde3e691cf
cd6c30015b241a782c9e78cd3c180db8a48625973490740b4fd3b435e1215339ae43ce15c984cf80d2166
6d46b55a99e51f676e12540074283c95ec682cdccf4e06c4336e03acd670123785bb42111d954737829a2
8170c52b9c8ce9c3a1df1547ae5fa63d8df26bc0262fb97be82adc1938fc19287d0704e36e7d41fcc8620
5bcc0d2dc4dfb4e1d304a44bc74e57346a5b92921c9828eac6612f2d0d7c7a2e168c4a77a0e4ef27928d2
01016c246a84af104396622d7d2cd59279aac6c35b50cfcb12990d323d8d03ca2d380d2058e00491ff410
c478fda4eb77e59a1d611c8a2b12091c3515301d60ee95ce950aad32948924f87063860ea50820f4bffc8
8ceae22b403a7d13d8dd0644a68c6e1ce6fb13ed8381da44e2f1e263fba3440626de00a33808ceb4a4a54
8138ccab58f4e1b9fac1deab5e753fe3b851567d6f0c76f03d0461fbf64858abe8d59efe5175b28bc2c0c
88eb16f8e7507ed16390f0a8161bf77db73b886907d266ad71666e166b084aa3e4be41bf4c86af2aea8d3
9c425fa76af5498bf4cf2a7c238ea34658302822e88ad5e807b1bbac59450ec7c3c0bd16fdc7220540148
aaf1d9f11a8df7fae8131a1042c33541e396b875a3a7c05d0d3c485305feb036087d2d1115ae007c7133e
3521e19c6a3fa885491d053cec36f979ae3c174082b19a33c4d0e3c0fe1cd7b76a1faaf2307180f7eb45e
97390fb49a89ee2f05d36feac4862dee22df20aef615cc20c685b5052c681d402abbed1a528db0cae366e
c8cd1ffa79920f55f3d7b51dd0f91fe339a0c2d94e179d26746d60311980b68782bda4b1dc9415a87bed6
```

```
4a97af03d2df234a73fa6e08e25c6d365c54c5b3223a7ed10d48713fa307fb9dced9e9fcb50050f60db52
efd1c0affebddfd4f67ed9154a8607e3f4582a1bf45914e58953ad6d4d96201cc8633c05940a34de55d28
8eae99d3a7bb4a50621702e22816811a4ea8147f8dd6756462b21fd678e37b841d864c9033aab5a482398
e23d643f42a1b11c866b69928e4478c89090e37c4f70bed141dfed9ea3df4c181de244452a185d76c7153
53ca0e33dcafb90fd9bb6642d5af89774e5fc8746ef8c68054bb392a90bd6d72967977554c39fbbaf9767
295049c09c25bc82b1bb4f7fc3b83a02bf722b33a54332dfd0bae6aad5236db428e94eef3b76394e35aaf
ab43a7ce376feced0b33c2a9d2d5fd77177745515df5d98124cde965e5177c4c064f94f92fe1b311c67aa
c88eaddc1bff648547d311c099b3560f40e244506d195aab67bf36b635ff7d30495a88b75864263f04004
049d3580a41666a79f295acaebae1a0e4a916547e52287af8e0e7b49473918afff0f66cd4e19da9d043f8
923aab8f4df541fc0e5a5409cf6a695039b52f1e1479fc20560e7a76dcf56dbd8c4e8c7864c5c48d3f777
cd030ab40dd8e1935579c21552c2dd0bb05b71f8eeb9615dd969c860d273d3db1e02897d57485890d1970
dcc1b5041ec40ed78849367d66d839c38f324e30211fd092fb54df386bd8b244e57fc233b3c970e9d387c
ce7cd4d1c807215ca1f9568be0a52767c2b98a0953ec514d52e4e5d8abef3c6d0940688e88b1998334360
48690dabb30bd602020edf380c7d997a39edc49909f0bf78da88e0e8d445c74bac96c5cd2e2d0aa14875a
8b0c3c696ddbba8a3a9413427f8639219326eddc0c237f52c407d4dc5532e49577fddd20f949fdcc51e15
2231f0f1d6a4c4200286a74645d2b80782228abd2d19d56c3bb2d4a6bc370924c776ed1cefe2aa54dd720
4fc6983eefaa523a9a40fbbc658fbc14f118e3464ca7a91de1b4c37c6299aee6c9921be8c13b4250ffe17
b7225bf6c620c560333f0c7fbb1ca4438da82f1b200602c2c2b1a1f20b70f1a342df54c04acbada99b22e
3211769c8ad317c4d70d86755868dbbbea185328539fa4082d27860bac3d96bc467958960ce9797c1c298
9723e9f0c61c4da4fd2b4ef69e3538adf88dd8500f75c6f35691e9a7c8625b19d48ff9f13b5bbadf0a6c9
cd913625f366bcf038be0fbde2bf68aa56b90e747e41c759ef200fae4ba87077f2c712091cd2b1d21cfc8
7125db90f61483985276e1abad88a36bb16336cad2ae5728300aa64527447c7445c99e1f64eb2fa6f3fd0
36fa684b1e116c0f2d88f809d641903a477e1e25abe2cb9a8497a27602295e55ce20257f83496b74d3897
787ffb4a50f3f076f83017f5c4144841fa70b5e88697afb0e7a6b98b2a487d00de2eab366389682165fb2
0c27c0aaf14a03ab46b0a622c0ac3e90f58e505f1caed2738b69f2b992f8026e35ea5b594545dccdffa1d
1c9c22e692ad4940c42821821aa0f28abf72010c4655bd46bf703cbc26fa83665df8b08bc144227d241f4
9877f38d424db6bfc64f61b8fd1edd4dcc353a38c60a8671fff32a39dec6c2cbafd5344d7393ded7faec3
520d605d420bad79b8a1c7103d9a924c9ece97c77d6bd63b7318f8d903126cec0e24e9a13c3b935180e7b
61d78d42e5dcdc9a5efce858bfd068ea9e2c5ec036eb29e5204084d496167822341375e1b45f9fb6e2a34
f2a29fd217a748a6d2c469c28943b84cd9d7f7c711068d1da87e41d7e27bb395dbc5515d1a0f9d4eff78c
bc4e7a66abf3bfacc073b35e98e4bacdd9e967d3e5db5fcb1ad9f69b67e0cebefe984b604a02f48f5d03c
d5c3459f111514499679d4cbf3e245fdf6560b8d94ce998b186b0f31526ecdf9ca49c5c6dd3fc3084488d
2140d0ffa2165a4e26f54a3de813bb01b42510898520aff1cc727a21430addec70dbf8b9b86ddc720017d
6beb4f807bc2295d00108eb84bdd3c70f8c7320130781bb755fcf95afecaf4446edb59b800f6aee2bda11
f237ee6540d09008713a31cf8bd3067f3dc0b71f2ba4b225799729f5987911308d6086f9ed1ea6344aaae
a794cbb63f15197bc8df2957d775f9db43b6669cf7d128ebc4ee38d90ac32d1fa22f06ec595201c524692
59cb054761b583cb8cf30856aaca94946f076d39fb8995e43733d311d51dafcaffaa42ecec44bf74749fa
36d5b4bfd8f1229af37d6794d53f234ad1e434c3388fef13a4a1030c0aa5050ff46152f2d858cfc8f464c
ceb33887ed950a117589474a70eca2a0b419adfd8de8e492b65efd3f9da0fc31b69bd1808dc8108a59822
9d43e0e86c3bdfb3cb4e6ad7921b07a3c4f3918ab44d95424c2061ece4f5f5004500503924b88e4e9846a
1194d3f70c690abc78b3ea894a3fd3a08785856e9056a36699c585eae0221eda810e32deaae3a44ccbfef
b387c78096a650301eb6b5387501d554d0eba6e6ceef514d3a42050c0cf150658f4fe6dac85ef5b6d3440
c88308a8137e92f5bf3025cce853f7bafce424232a6899a95060e401db71249a3d6d4f62ed64e30ab7814
9f4cfd32833d11798142ce1c80ba76ea9894d21247836ee9339754497e0253ed62016de1ef1ae91284cfe
7c51d88141c4376c9b405cf1e927a1420a38b6c9ac15021810968c6406f9b41ec869421262c2efe0806b6
ac2ff329a8cb688326bdbfcbdf123d88fece510c2f211e823fcbfb7e4160a486383320389a4bad5197eb5
0fdc04c5d2e2751e5a30aeef1a8af37b7e6b67e63ae22705dfbc78b19ed32e8d7cdbb1166b29f0428806f
3b8e83440efd893525e7fefef10911cee4195f5bfa93f2b454112d47f436367a8e87f5e7780dc1ecbe202
d52b398ec81eb51dd340b19992d45a0a2d6fae6538d3b6e68b9068d5af5037004c4175f291dea3aa35d17
2ad4bccee6858738011bae78f1a5ef4ee3de767996298a6278eea9d565b88f6f906f93ad80f0378be4f4d
26f5eae0091f614add12fd7a96d531f2316a23af49b06b0363e88d2069aca3c335c8dc466cb281a55bfe7
365e3755f3f93ba669c8350b7094764121bd5a6203dcb4fbcc75f1170a28fe35f03ddb000c72dca1e0024
```

```
d61ea590f7067237f30fd997408230fac900755cda8e75a4492fb65d465df772ef0435d13084519abc405
fc72a2e804b21361629643e1bbfbf8f232c389725c2cd7f7c8235a482ed4c328deb1e9149310fe4bad15c
df94019dca679f5f718d879c44a44c572f2bb9676086446ada3b67e5406d2aef8b6184b1e2a5bb6229158
ee7bc4191e7bb4b7fdd54faf1892aa8b8"

def toRaw(hexVal):
    return binascii.unhexlify(hexVal.encode())

def initCipher():
    return ARC4.new(KEY.encode())

cipher = initCipher()
plainPort = cipher.decrypt(toRaw(BLOB_CONFIG_PORT)).split(b'\x00\x00\x00\x00')
[0].decode()
cipher = initCipher()
plainBotnet = cipher.decrypt(toRaw(BLOB_CONFIG_BOTNET)).split(b'\x00\x00\x00\x00')
[0].decode()
cipher = initCipher()
plainC2List = cipher.decrypt(toRaw(BLOB_CONFIG_C2)).split(b'\x00\x00\x00\x00')
[0].decode().split(',')

print(f'[+] Botnet:{plainBotnet}')
print(f'[+] Port:{plainPort}')
print('[+] C2 List:')
for c2 in plainC2List:
    print(f'\t[*] {c2}')
```

```
[+] Botnet:202lg
[+] Port:443
[+] C2 List:
        [*] 141.161.143.136:272
        [*] 214.77.93.215:263
        [*] 104.168.157.253:443
        [*] 196.224.200.10:482
        [*] 254.65.104.229:127
        [*] 209.141.40.19:443
        [*] 107.189.5.17:443
        [*] 44.184.236.94:128
        [*] 60.231.88.20:422
        [*] 210.38.79.54:319
        [*] 23.254.167.63:443
        [*] 91.206.178.234:443
        [*] 72.204.201.249:374
        [*] 146.19.173.86:443
        [*] 103.175.16.104:443
        [*] 138.133.49.46:211
        [*] 150.18.156.130:256
        [*] 93.216.14.249:213
        [*] 73.73.80.51:127
        [*] 216.73.114.69:379
        [*] 58.249.161.153:350
        [*] 140.157.121.40:433
        [*] 194.135.33.85:443
        [*] 6.66.255.6:433
        [*] 173.234.155.246:443
        [*] 179.55.218.145:322
        [*] 241.163.228.200:362
        [*] 38.174.252.233:131
        [*] 146.29.236.141:457
        [*] 32.234.39.72:191
        [*] 181.87.160.175:479
        [*] 114.70.235.72:357
        [*] 51.68.144.43:443
        [*] 172.86.120.111:443
        [*] 160.20.147.242:443
        [*] 207.12.58.212:419
        [*] 51.75.62.204:443
        [*] 174.72.94.173:309
        [*] 205.185.113.34:443
        [*] 194.135.33.184:443
        [*] 246.6.106.79:340
        [*] 23.82.140.155:443
        [*] 185.173.34.35:443
        [*] 255.115.3.251:370
        [*] 177.232.32.155:257
        [*] 122.125.104.16:475
        [*] 24.64.127.190:229
```

The retrieved botnet ID is: `2021g` which is fairly correlated with a recent tweet coming from k3dg3 regarding BumbleBee activity utilized by TA579:

> #TA579 dropping #Bumblebee "132lg" via #DocuSign lures
>
> Email -> URL -> Zip -> Password -> IMG -> LNK -> BAT -> DLLhttps://t.co/iD7ip1nqpS
>
> — Kelsey (@k3dg3) February 13, 2023

Summary

In this blogpost we went over a recent BumbleBee campaign that uses multi layered powershell script in order to load the BumbleBee loader.

I've mainly focused on breaking down the powershell script part rather then focusing on the loader capabilities, if you want to learn more about the BumbleBee Loader, check this blog written by Eli Salem

Update 1

During my writing i found yet another campaign with the botnet ID of lg0203 I've run my scripts on the hidden powershell script and managed to extract the DLL without any problem :)

IOC's

Samples:

- requested_documents_714407544541.zip - d4a358c875ab55c811368eabe8fa33d09fe67f2d3beafa97b9504bf800a7a02d
- 8702268950347.img - a55979165779c3c4fc1bc80b066837df206d9621b0162685ed1a6f6a5203d8af
- requested information.lnk - 6fb690fbeb572f4f8f0810dd4d79cff1ca9dbd2caa051611e98d0047f3f2aa56
- documents.ps1 - b6d05d8f7f1f946806cd70f18f8b6af1b033900cfaa4ab7b7361b19696be9259
- LoaderDLL.bin - 2d5c9b33ed298f5fb67ce869c74b2f2ec9179a924780da65fcbc1a0e0463c5d0
- BumbleBeeLoader.bin - 4a5d5e6537044cdbf8de9960d79c85b15997784ba1b74659dbfcb248ccc94f59