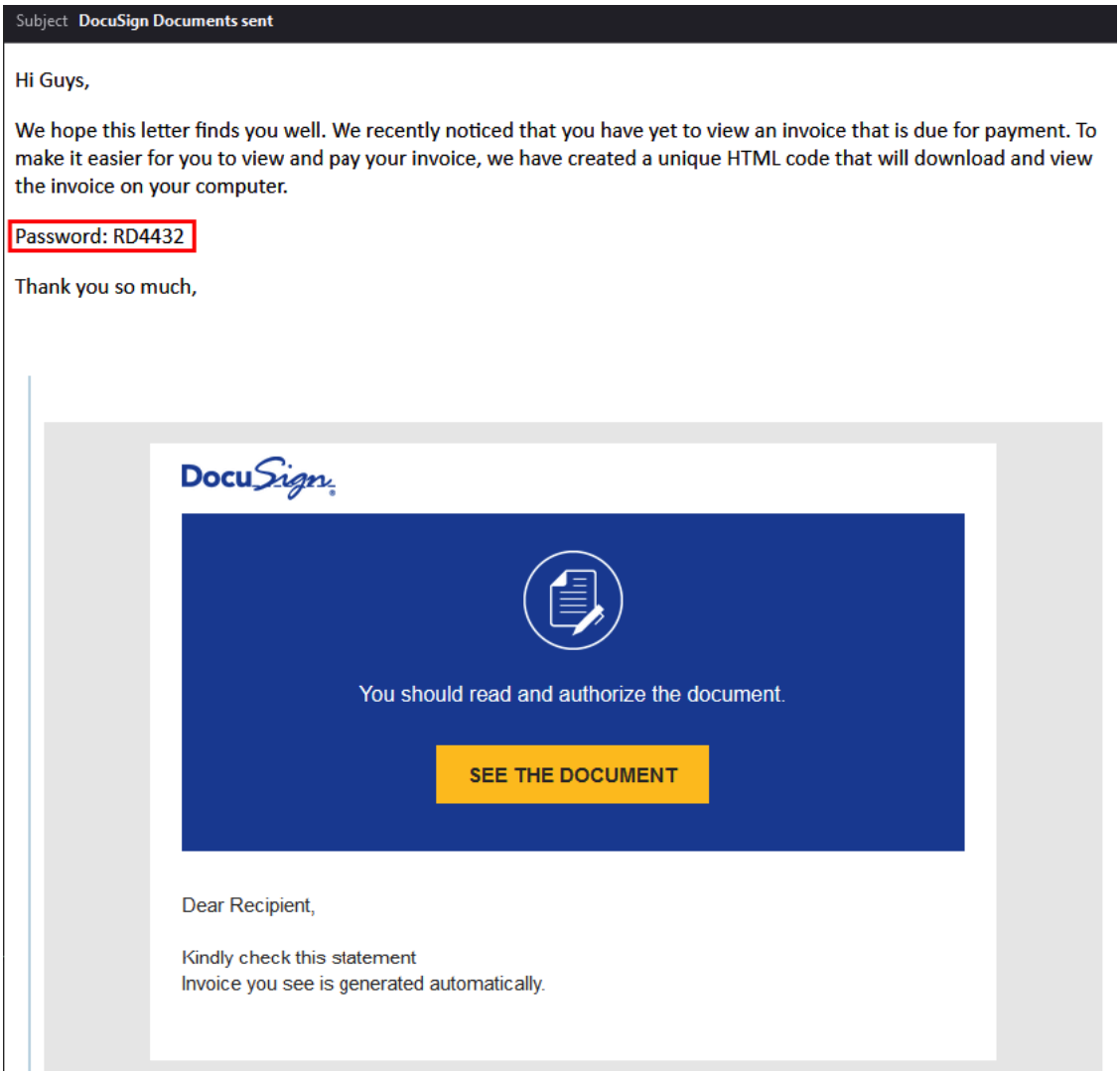
# Bumblebee DocuSign Campaign



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

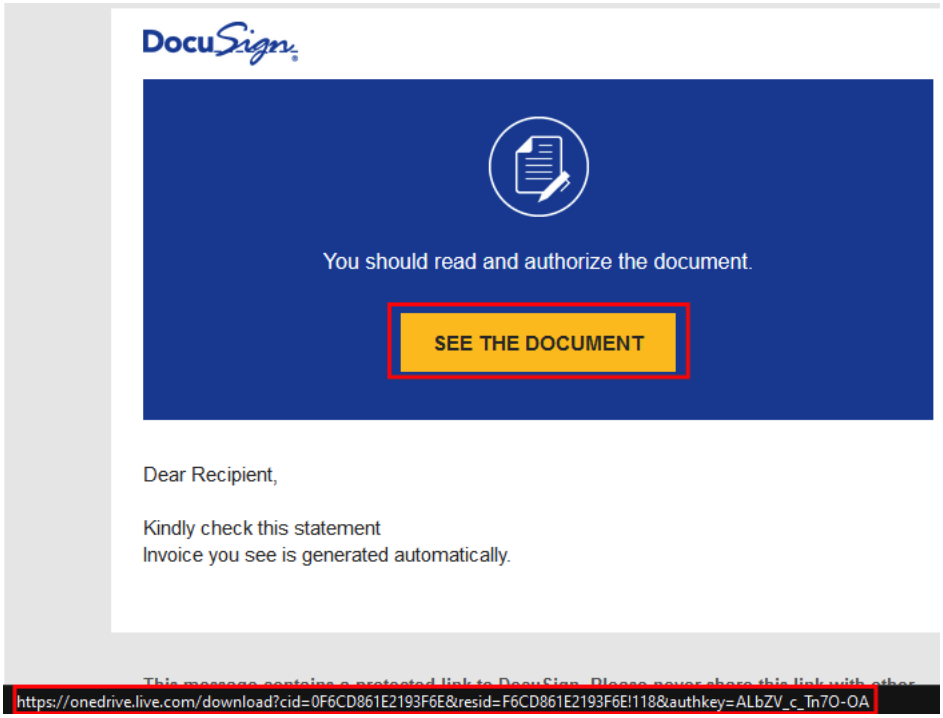
## The Phish

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`




Phishing Mail

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



OneDrive Embedded URL

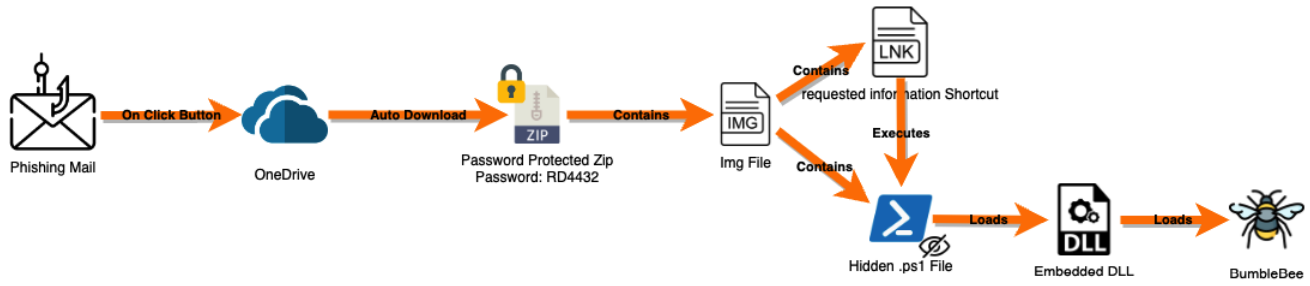
The URL is:

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

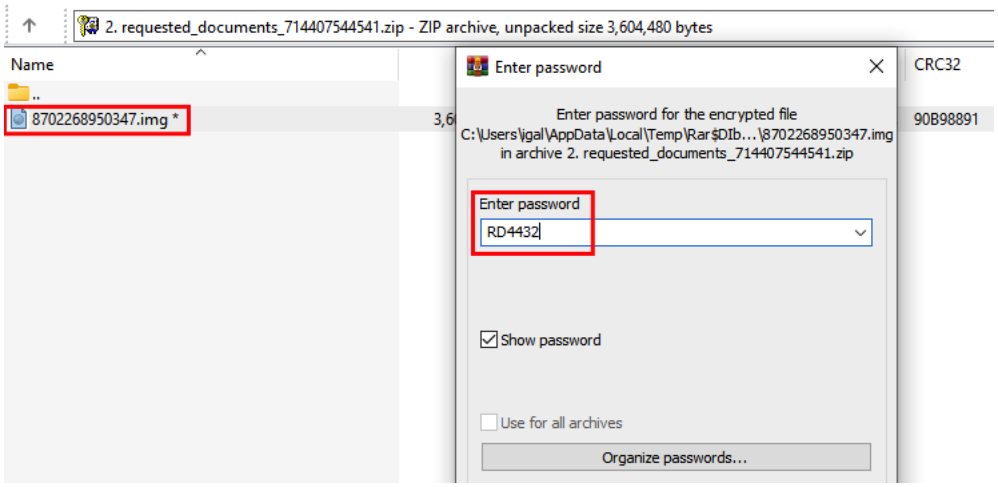
Below you can see a diagram of the execution chain from the moment the phishing mail was opened:




Execution Flow

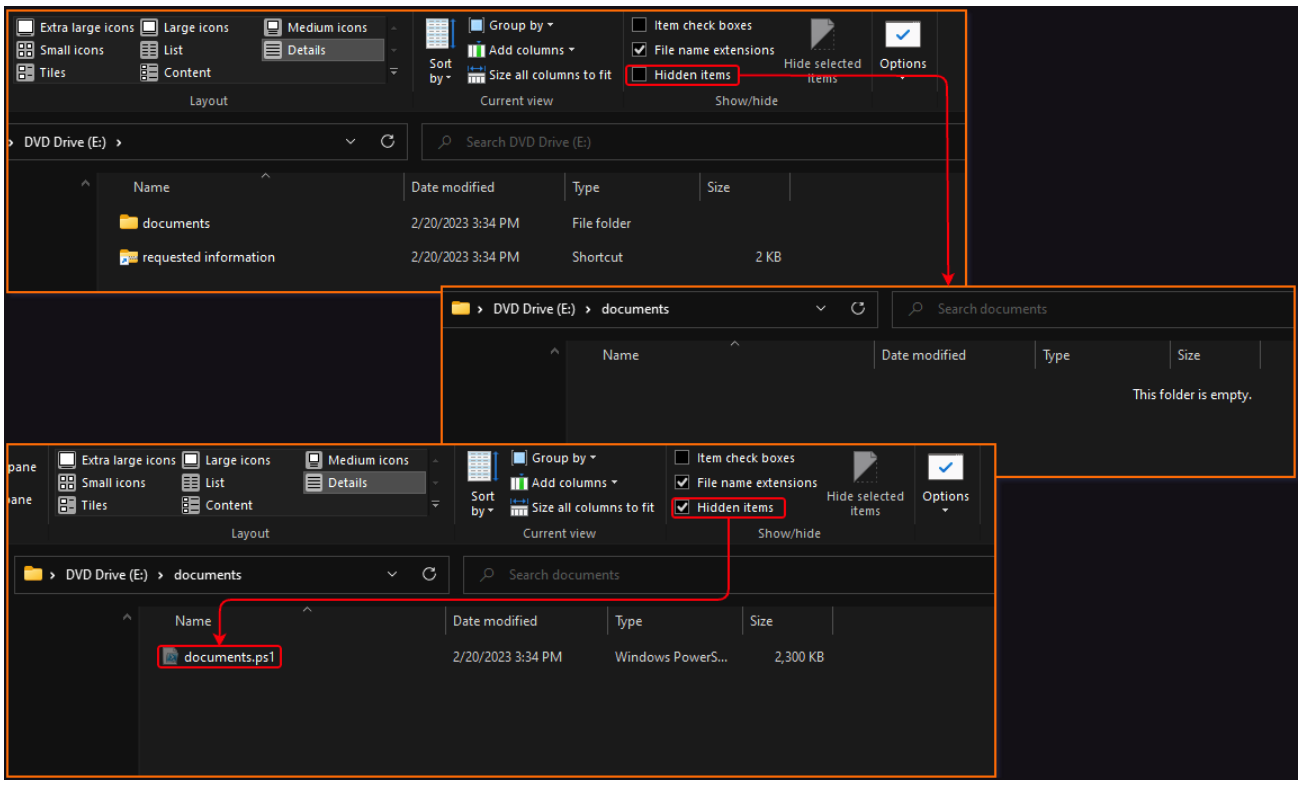
Lets go quickly through this chains:

1. 1.
   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




Password Protected Archive

2. 2.
   Once the IMG file is opened the user will see only the LNK file `requested information` (because the .ps1 is hidden)



Hidden Powershell Script

3. 3.
   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
```
LNK Target Command

## 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/e4KEIjTDoXm4FObQ3Qv2mwaCdfQyqgv035pive9hLu475PMX8IloJ9qA+5od/1qoP6s2CvMa4/QA760N5RIl+yjW3p7aNRx/bP6s+RmDXIF7Tdhf0f9hQ7
23A32jPsHeQw5gxyuZ/T3aJ9BjtFfakFe4HnioG8GfVhPYf1cHutDDuvnHPIc9ZtVP56NPcbzUftOdDTkA+QHyN0U8hLyU6Jh75DD+KcY/3Yd8jzI0HcjyJe
Qn6GPW16mLuQF2ovRPrWDHPQb1NeQ70u0F65fQP4C+6cIf25RNcitEuvnCvIr7L8Pe24Og71/yMpCz7BfmuH6HHIP643q9F/r/0gOenWB+kN0tZlBjiAPyvf
```
Broken B64 Variables

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")
```
First Char Swap

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//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAA
hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1vZGUuDQ
szm+XLHgxki3ob5cseDGeLe5vlyx4MZct7m+XLH0xaCzub5cseDGVLe
V4GMAAAAAAAAAAPAAIiALAg4AAD4AAAC6FgAAAAAwBAAAAAQAAAAAA
AAAAAAAIAYAEAABAAAAAAAAAAQAAAAAAAAAAAQAAAAAAAEAAAAAAAAA
BAAAAAAAAAAAAAAAwFwDAAAAAgF0AADgAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAudGV4dAAAAFU8AAAAEAAAAD4AAA
```

Broken B64 Strings

```
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}')
```

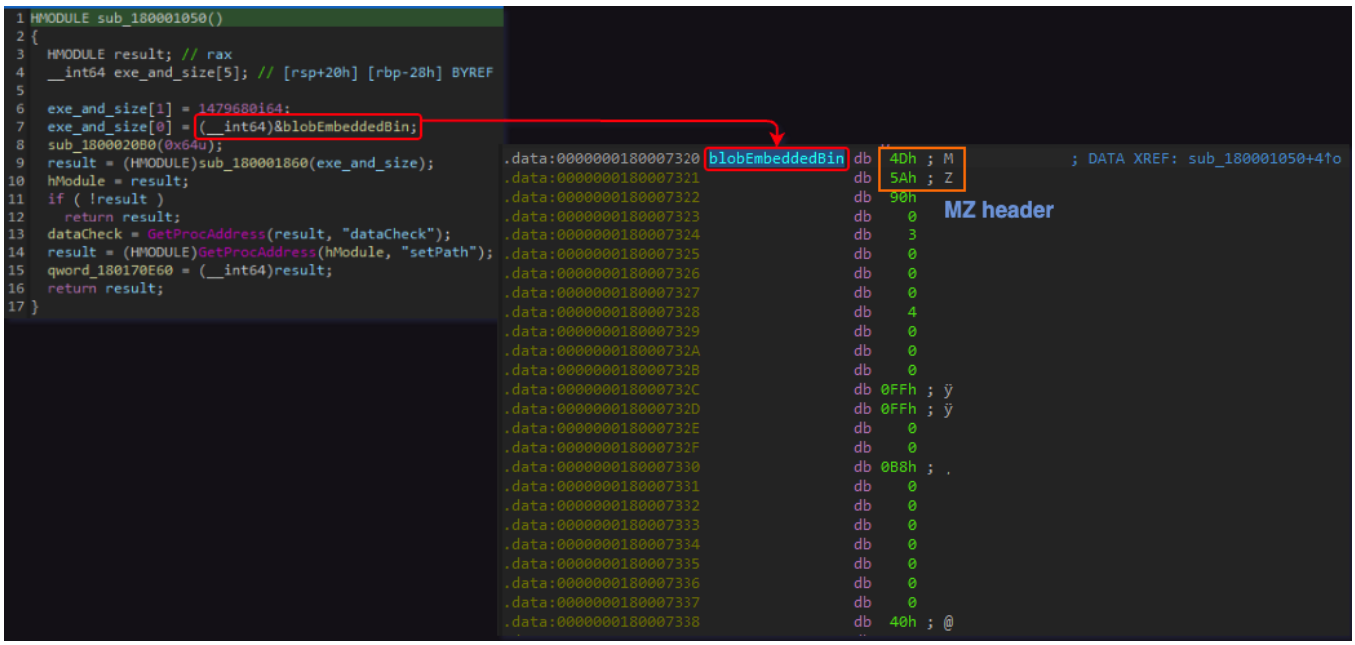
[+] Payload was extracted to the path:/Users/igal/malwares/bumblebee/21-02-2023/payload.bin

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`:

```
1 BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
2 {
3   if ( fdwReason )
4   {
5     if ( fdwReason == 1 )
6     {
7       sub_180001050();
8       return 1;
9     }
10  }
11  else
12  {
13    sub_180002020(hinstDLL, fdwReason, lpReserved);
14  }
15  return 1;
16 }
```

DllMain

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:

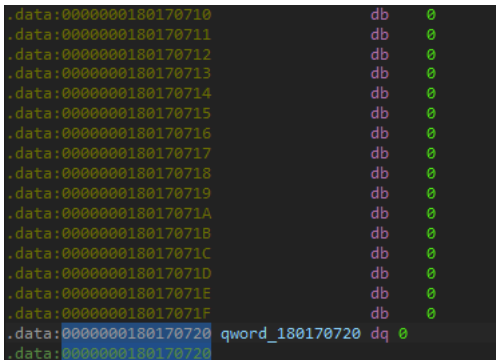


MZ Blob

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

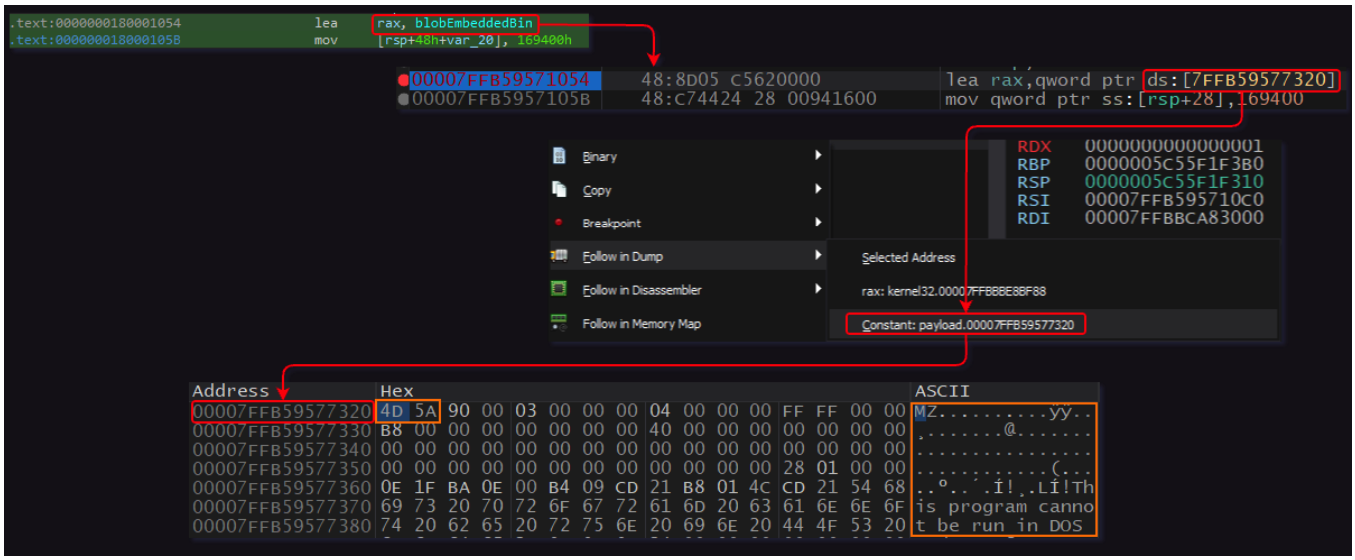
print(hex(0x180007320 + 0x169400))

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



End Of Blob

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



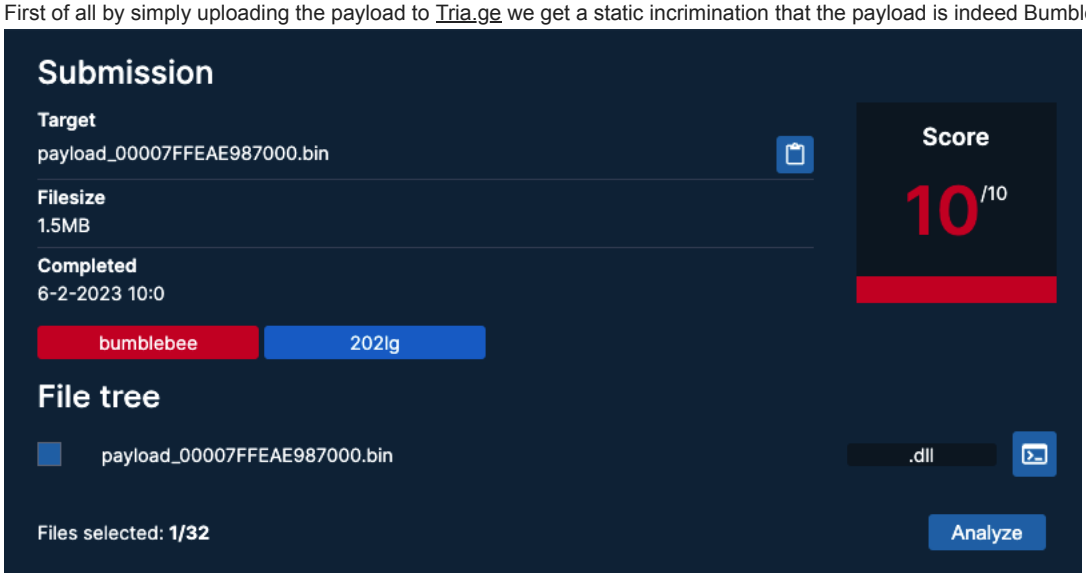
Payload Dumping

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.

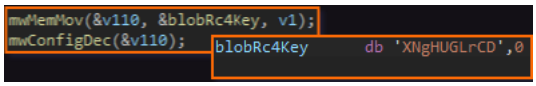
First of all by simply uploading the payload to Tria.ge we get a static incrimination that the payload is indeed BumbleBee payload:



Tria.ge Incrimination

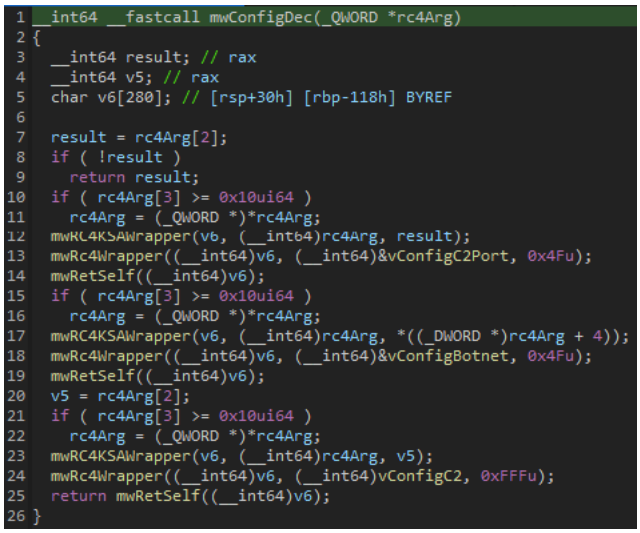
Additionally Tria.ge shows us the botnet ID which is: `202lg`.

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:



Rc4 Key

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);
}
```

Config Decryption Function

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 =
"0b002425baa537efd52cf61f683f8116bc994d01c892b9c140f4a29c3f8a0b823f5a65b8dc08bb73c1e7ec5f5cb40ca4a45ea741c5367ad2368ea826d

BLOB_CONFIG_BOTNET =
"0d042549dda537efd52cf61f683f8116bc994d01c892b9c140f4a29c3f8a0b823f5a65b8dc08bb73c1e7ec5f5cb40ca4a45ea741c5367ad2368ea826d

BLOB_CONFIG_C2 =
"0e00260b8b9306c1e418c531590cb72c8eae7f2dfaa38def77c38ca50ca439b30a60578eef248a43f5c9dd69649a3d9193709574f60c4ee605a2991f


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: `202lg` which is fairly correlated with a recent tweet coming from k3dg3 regarding BumbleBee activity utilized by TA579:

K3dg3 Tweet

## 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