

# BumbleBee DocuSign Campaign

By 0xToxin

Published: 2023-03-04 · Archived: 2026-04-05 15:49:02 UTC

## Intro [Permalink](#)

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 [Permalink](#)

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

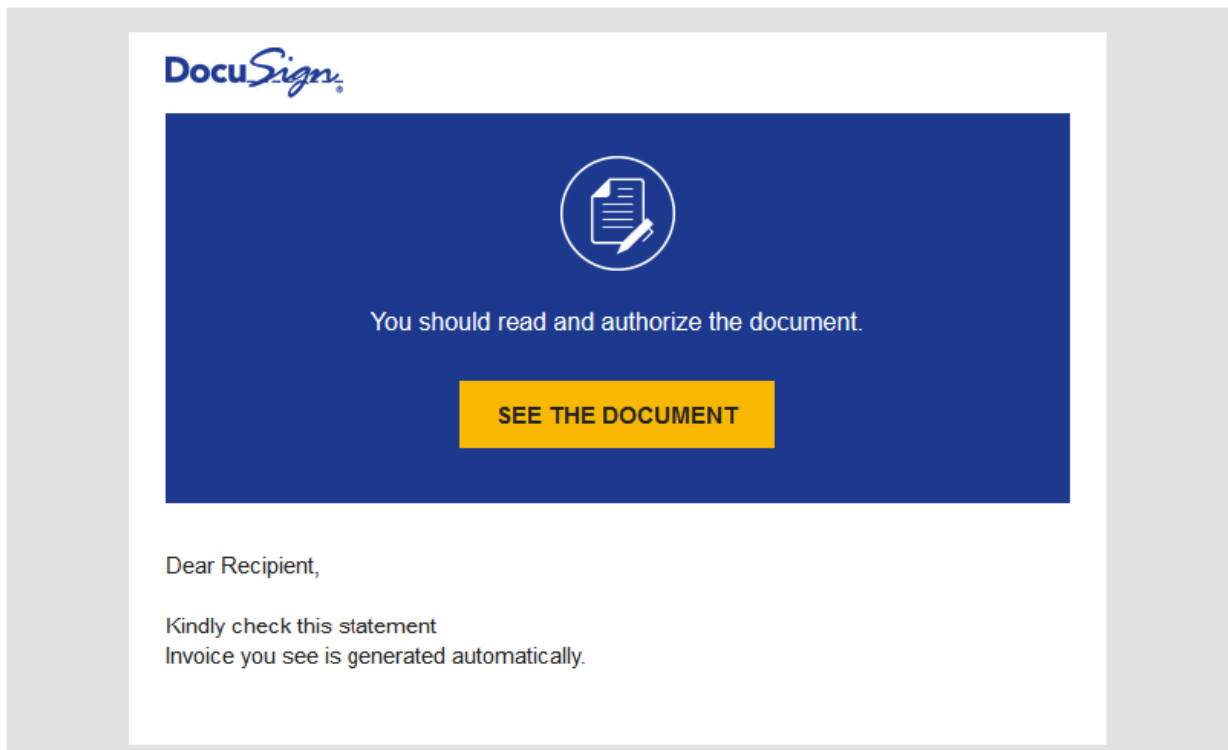
Subject **DocuSign Documents sent**

Hi Guys,

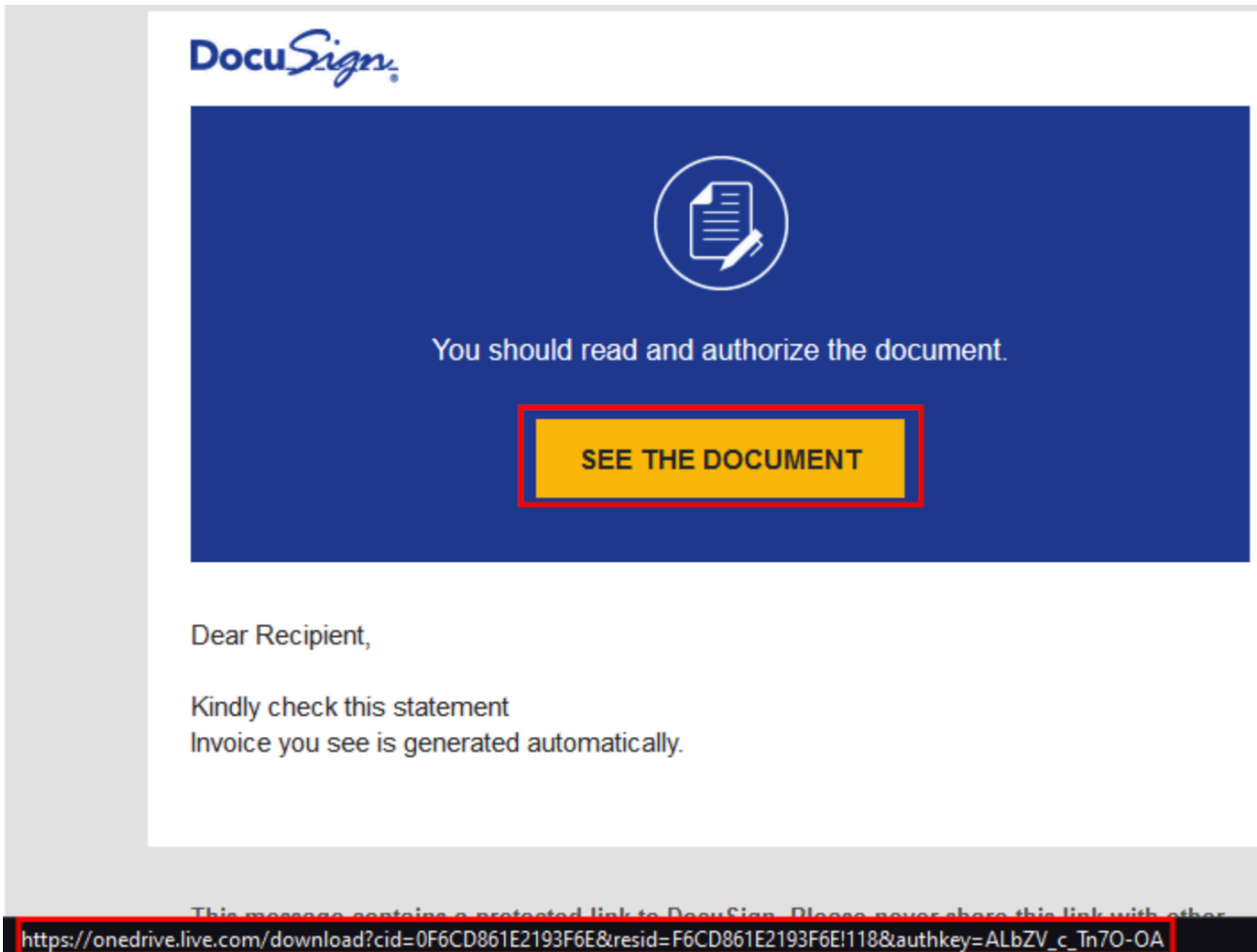
We hope this letter finds you well. We recently noticed that you have yet to view an invoice that is due for payment. To make it easier for you to view and pay your invoice, we have created a unique HTML code that will download and view the invoice on your computer.

Password: RD4432

Thank you so much,



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



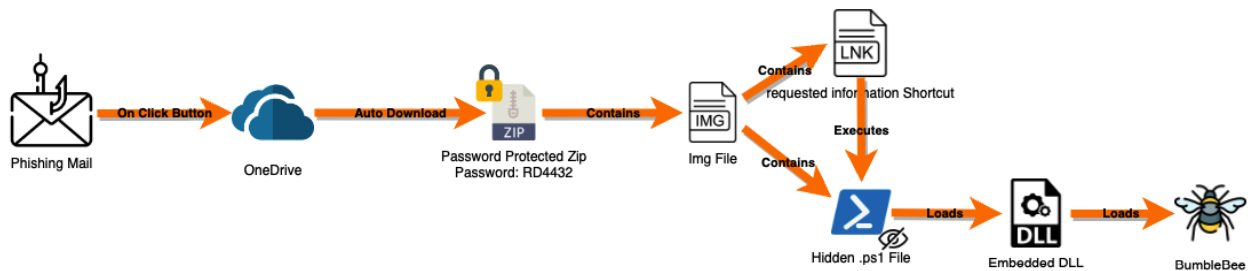
The URL is:

```
https://onedrive.live[.com/download?cid=0F6CD861E2193F6E&resid=F6CD861E2193F6E%21118&authkey=ALbZV_c_Tn70-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.

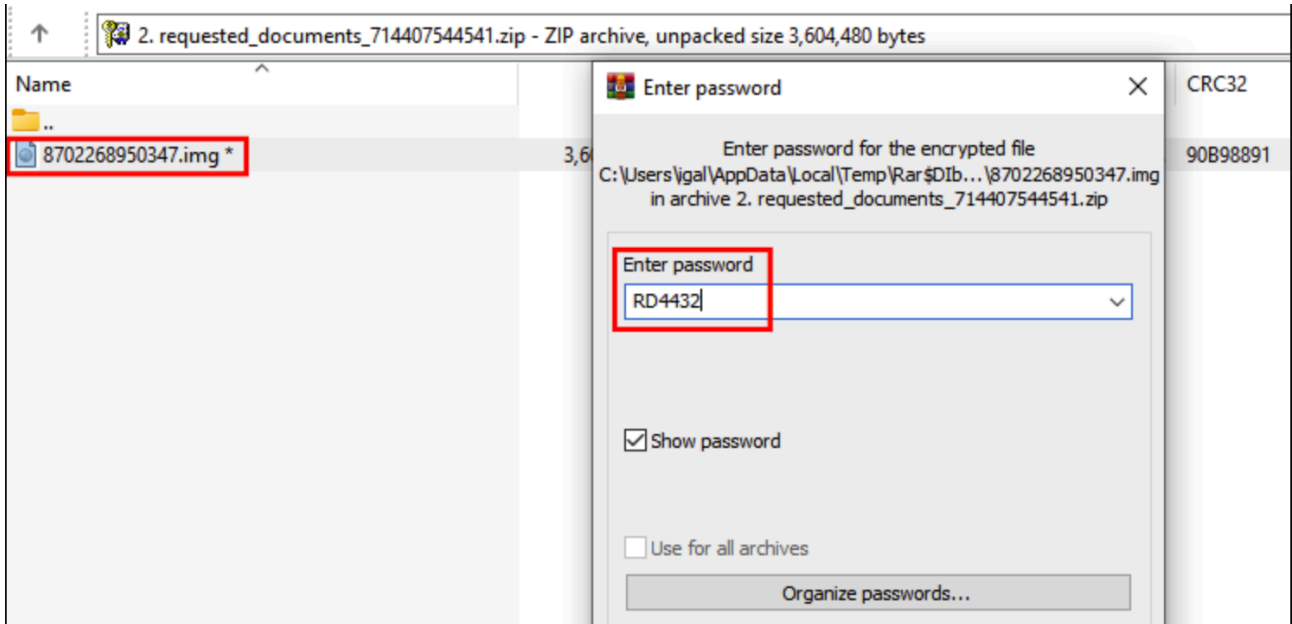
### Execution Chain [Permalink](#)

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

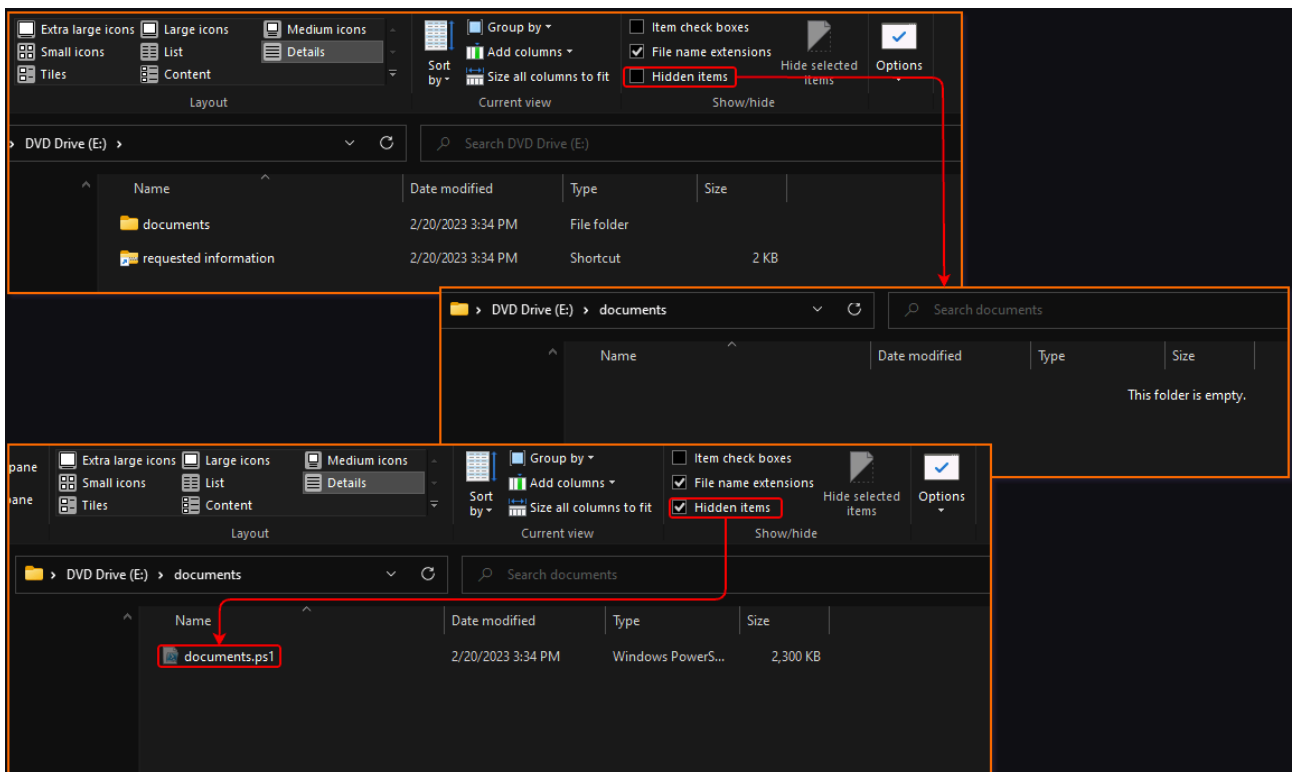


Lets go quickly through this chains:

- 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



- 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 [Permalink](#)

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 =
"b4sIAAAAAAAEA018W28ruXL1+wHOfxgEeUgg5Nuy3LbLA0eBkt1aZUluY7LcDvIgyzJVarViti5t+dd/XJTtPTPZey6YSXJyYAJb26Vis9lkdWgYotKdWc
jPpRSys1v4mV6hrIF5CfG16+aiVKUdlSSi0g97TX0z30qwXkMeQx5M2Nr3S1UP8Rskv97kMvP9PnkIsX6CPotz/Tv0DehftnoyB71aYBfen1zhP615/1L77
pe5W+9X+etkq1BgvIY8hVp0XyDHklyA3+j+5/rrxx65PFn/s+hH63zZe33iAfgJ9ewZ5Avku6JPbn4xHuw59Dv3X8WsYyFdg5qtuuv5b1/DyuUqVGs46CkC
L/pVe30yhP/mZ/gxygutbLa9vmoYf77PS61uJlxsJ9OPE64fa12+uob/TXu6ifqOE/hHtdyLI2yCjvQ760wj9mf7s+qfj10/L//Y9Rr97/W9nkbQZ9D3FOE
ryEvo4+pPx6OnoX/+6fjRAjL50qrjbvQpzLkaQG75N1SDw5eQ21UVZs0XwkcTelXDR1zis4qvB8rXbSZYn30vNw4fsmPbByCptLcKMrCKM1jBKG1hqnc/KuQ
bWFlr1kB/IReo36tDDvo19H31oX9V7/pm+qEERB/6s6/6+odexfSuJxW/6dUQ/Qt6wmM2t8o/20Xi7X1Ybf5EboRGGBetdQJ8gd6GvpngWYJeoI+TcBPIz5C
7gN/e4KEHjTDoXm4FObQ3Qv2mwaCdfQyggv035pive9hLu475PMX8HloJ9gA15od/lqoP6s2CvMa4/QA760N5RHlyjW3p7aNRx/bP6s1RmDXIF7Tdhf0f9hQ7
23A32jPsHeQw5gxyuZ/T3aJ9BjtfFakFe4HnioG8GfVhPYf1cHutDDuvnHPIC9ZtVP56NPcbzUftOdtTKA+QHyN0U8hLyU6Jh75DD+KcY/3Yd8jzI0HcjyJe
Qn6GPW16mLuQF2ovRPrWDHPQb1NeQ70u0F65EQP4C+6cIf25RncitEuvnCVIr7L8Pe240g71/yMPCz7BfmuH6HHIP643q9F/r/0gOenWB+kN0tZ1BjiAPvff
```

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')
```



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'

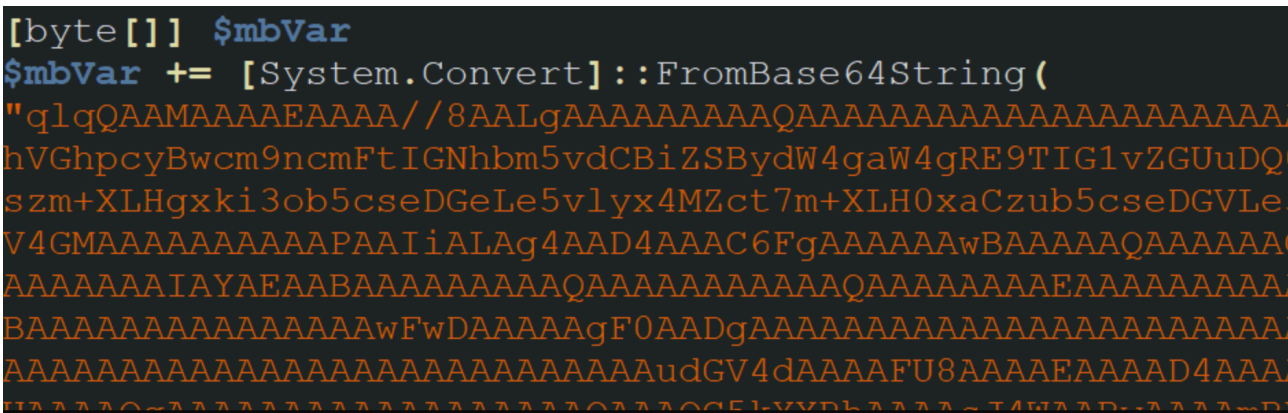
countArchives = sum(1 for file in os.scandir(ARCHIVES_FOLDER))

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)
```

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



```
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 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:

```
1 HMODULE sub_180001050()
2 {
3     HMODULE result; // rax
4     __int64 exe_and_size[5]; // [rsp+20h] [rbp-28h] BYREF
5
6     exe_and_size[1] = 1479688164;
7     exe_and_size[0] = (__int64)&blobEmbeddedBin;
8     sub_180002080(0x640);
9     result = (HMODULE)sub_180001860(exe_and_size);
10    hModule = result;
11    if ( !result )
12        return result;
13    dataCheck = GetProcAddress(result, "dataCheck");
14    result = (HMODULE)GetProcAddress(hModule, "setPath");
15    qword_180170E60 = (__int64)result;
16    return result;
17 }
```

.data:0000000180007320	blobEmbeddedBin	db	4Dh ; M	; DATA XREF: sub_180001050+4to
.data:0000000180007321		db	5Ah ; Z	
.data:0000000180007322		db	90h	
.data:0000000180007323		db	0	MZ header
.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 ; y	
.data:000000018000732D		db	0FFh ; y	
.data:000000018000732E		db	0	
.data:000000018000732F		db	0	
.data:0000000180007330		db	088h ; .	
.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 added the possible length ( `0x169400` ) (wrote it in the IDA output window)

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

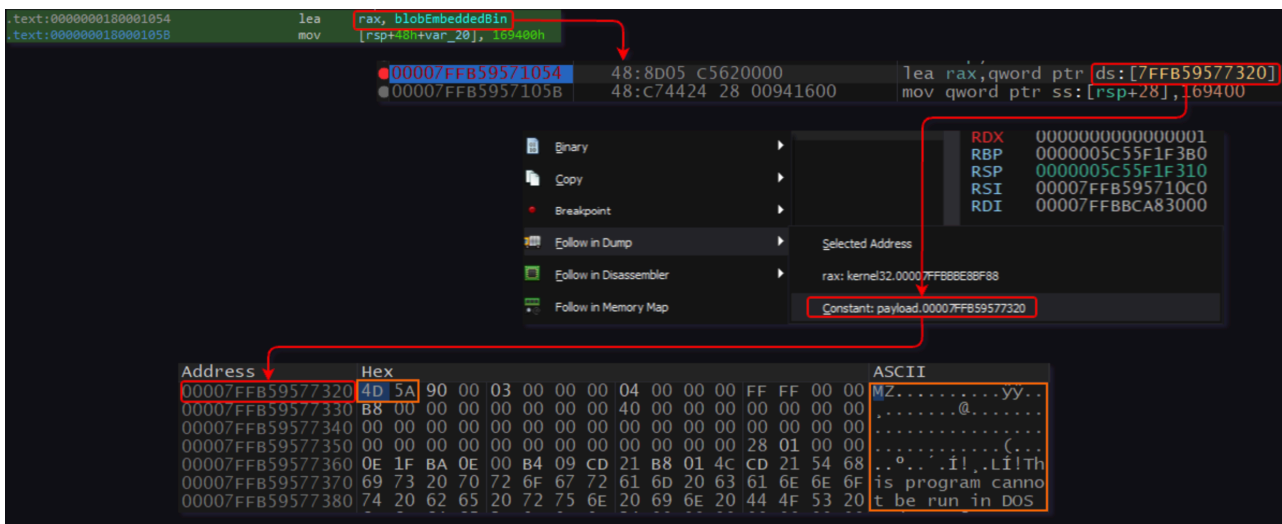
And by double-clicking on the printed value it jumped to the offset which was the actual end of the blob data:

```

.data:00000000180170710 db 0
.data:00000000180170711 db 0
.data:00000000180170712 db 0
.data:00000000180170713 db 0
.data:00000000180170714 db 0
.data:00000000180170715 db 0
.data:00000000180170716 db 0
.data:00000000180170717 db 0
.data:00000000180170718 db 0
.data:00000000180170719 db 0
.data:0000000018017071A db 0
.data:0000000018017071B db 0
.data:0000000018017071C db 0
.data:0000000018017071D db 0
.data:0000000018017071E db 0
.data:0000000018017071F db 0
.data:00000000180170720 qword_180170720 dq 0
.data:00000000180170720

```

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



Now we can investigate the embedded binary.

## BumbleBee Payload [Permalink](#)

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:

The screenshot shows a submission interface for a malware sample. The submission is titled "Submission" and has a target of "payload\_00007FFAE987000.bin" with a file size of 1.5MB. It was completed on 6-2-2023 at 10:0. The submission is associated with the botnet "bumblebee" and the botnet ID "202lg". A score of 10/10 is displayed. The file tree shows the file "payload\_00007FFAE987000.bin" with a ".dll" extension. The interface includes buttons for "Analyze" and "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

```
1 __int64 __fastcall mwConfigDec(_QWORD *rc4Arg)
2 {
3     __int64 result; // rax
4     __int64 v5; // rax
5     char v6[280]; // [rsp+30h] [rbp-118h] BYREF
6
7     result = rc4Arg[2];
8     if ( !result )
9         return result;
10    if ( rc4Arg[3] >= 0x10ui64 )
11        rc4Arg = (_QWORD *)*rc4Arg;
12    mwK4KSAWrapper(v6, (__int64)rc4Arg, result);
13    mwRc4Wrapper((__int64)v6, (__int64)&vConfigC2Port, 0x4Fu);
14    mwRetSelf((__int64)v6);
15    if ( rc4Arg[3] >= 0x10ui64 )
16        rc4Arg = (_QWORD *)*rc4Arg;
17    mwRC4KSAWrapper(v6, (__int64)rc4Arg, *((_DWORD *)rc4Arg + 4));
18    mwRc4Wrapper((__int64)v6, (__int64)&vConfigBotnet, 0x4Fu);
19    mwRetSelf((__int64)v6);
20    v5 = rc4Arg[2];
21    if ( rc4Arg[3] >= 0x10ui64 )
22        rc4Arg = (_QWORD *)*rc4Arg;
23    mwRC4KSAWrapper(v6, (__int64)rc4Arg, v5);
24    mwRc4Wrapper((__int64)v6, (__int64)vConfigC2, 0xFFFu);
25    return mwRetSelf((__int64)v6);
26 }
```

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 = "0b002425baa537efd52cf61f683f8116bc994d01c892b9c140f4a29c3f8a0b823f5a65b8dc08bb73c1e7ec5f5cb4
BLOB_CONFIG_BOTNET = "0d042549dda537efd52cf61f683f8116bc994d01c892b9c140f4a29c3f8a0b823f5a65b8dc08bb73c1e7ec5f5c
BLOB_CONFIG_C2 = "0e00260b8b9306c1e418c531590cb72c8eae7f2dfaa38def77c38ca50ca439b30a60578eef248a43f5c9dd69649a3c

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:

## Summary [Permalink](#)

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 than focusing on the loader capabilities, if you want to learn more about the BumbleBee Loader, check this [blog](#) written by [Eli Salem](#)

## Update 1 [Permalink](#)

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 [Permalink](#)

- **Samples:**

- requested\_documents\_714407544541.zip - [d4a358c875ab55c811368eabe8fa33d09fe67f2d3beafa97b9504bf800a7a02d](#)
- 8702268950347.img - [a55979165779c3c4fc1bc80b066837df206d9621b0162685ed1a6f6a5203d8af](#)
- requested information.lnk - [6fb690fbeb572f4f8f0810dd4d79cff1ca9dbd2caa051611e98d0047f3f2aa56](#)
- documents.ps1 - [b6d05d8f7f1f946806cd70f18f8b6af1b033900cfaa4ab7b7361b19696be9259](#)
- LoaderDLL.bin - [2d5c9b33ed298f5fb67ce869c74b2f2ec9179a924780da65fcbc1a0e0463c5d0](#)
- BumbleBeeLoader.bin - [4a5d5e6537044cdbf8de9960d79c85b15997784ba1b74659dbfcb248ccc94f59](#)

Source: <https://0xtoxin.github.io/malware%20analysis/Bumblebee-DocuSign-Campaign/>