IcedID on my neck I'm the coolest

By Axel Zengers

Published: 2021-04-10 · Archived: 2026-04-05 22:32:41 UTC

IntroductionPermalink

With the takedown of Emotet with the Operation Ladybird, there is now room for a new challenger to take the throne of the "Yeah, it's me who delivers the bad stuff". This past few days I saw a new campaign of IcedID and decided to take a closer look.

The goal of this post is to unpack IcedID and recover the C2 url as quickly as possible.

Getting our hands dirtyPermalink

First, we have to find a sample. For this my go-to place is https://bazaar.abuse.ch.

The sample used during this post can be found here.

The tools we will use are :

- PEBear
- x64dbg
- PeStudio
- IDA (optional)

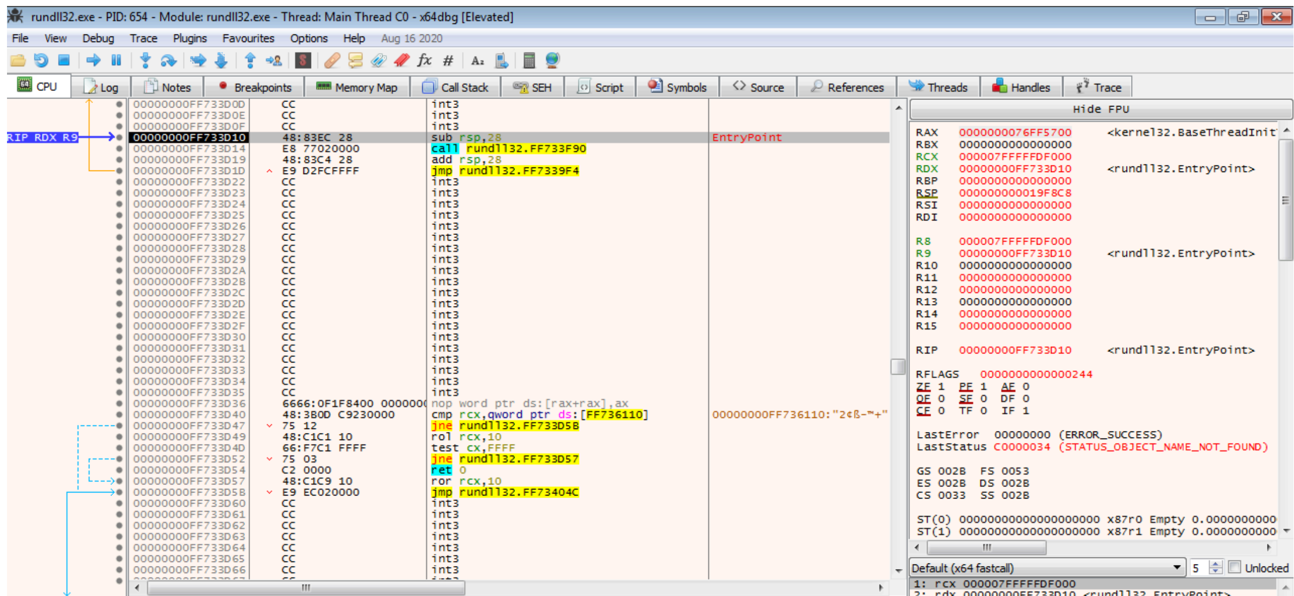
indicators (3/19)
virustotal (disabled)
dos-header (64 bytes)
dos-stub (152 bytes)
file-header (Apr.2021)
optional-header (Native)
directories (time-stamp)
sections (87.82%)
libraries (2)
imports (2/3)
exports (4)
tls-callbacks (n/a)
resources (n/a)
strings (2/732)
debug (time-stamp)
manifest (n/a)
version (n/a)
certificate (n/a)
overlay (unknown)

property	value
md5	7D7BDC559AE699579A700645D0FD5F03
sha1	C4C0CA6B2B7779D870B0B69E5D7001453BABBFF0
sha256	0A0B3D91698A46D409791D4DD866E56DDD70F91A3F1D4557A0CB2899BDA1E524
md5-without-overlay	1FA1859777AB9564F4BD05AD7838BD72
sha1-without-overlay	1F3A9EACECF5153B63C636D2263D3BDD5F3E68FA
sha256-without-overlay	749F002493202F71C0C4B8B8CA71CE3F8E33E138D25188FEE839E9CC8ACA4877
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
first-bytes-text	M Z .. .. @ .. ..
file-size	185404 (bytes)
size-without-overlay	163840 (bytes)
entropy	6.207
imphash	418BB7AFA91EE2677E9770DEEEB77473
signature	n/a
entry-point	48 83 EC 10 83 FA 01 B8 40 11 66 A1 BA 64 B7 48 21 0F 44 D0 B8 DA CF 13 86 41 B8 B1 C3 72 00 3D DA
file-version	n/a
description	n/a
file-type	**dynamic-link-library**
cpu	**64-bit**
subsystem	Native
compiler-stamp	0x6070195F (Fri Apr 09 11:07:43 2021 - UTC)
debugger-stamp	0x6070195F (Fri Apr 09 11:07:43 2021)
resources-stamp	n/a
exports-stamp	0xFFFFFFFF (Sun Feb 07 07:28:15 2106)
version-stamp	n/a
certificate-stamp	n/a

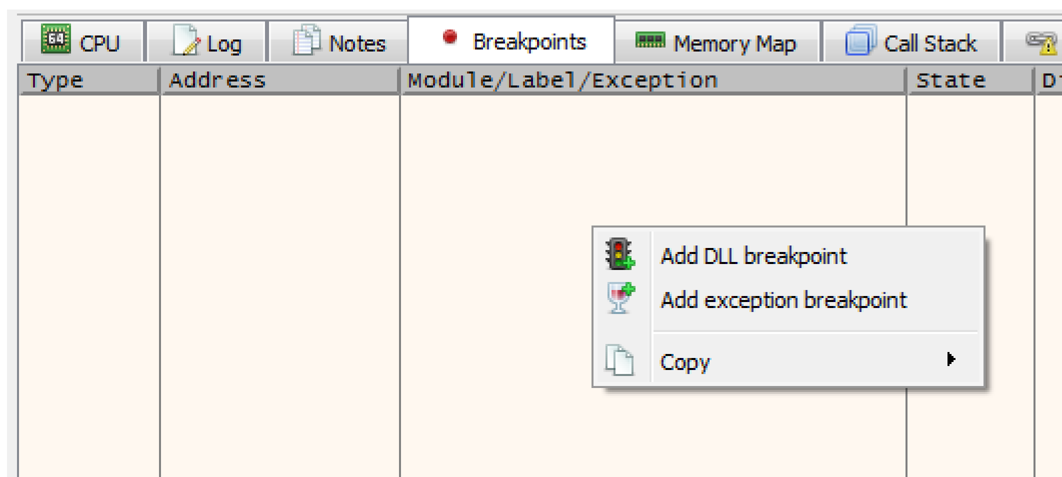
As you may have notice, the file is a dll and not a .exe file, meaning that just running it in the debugger won't work. We need the help of `rundll32` for this. So first we got to open it with x64dbg and change the commandline to : `"C:\Windows\System32\rundll32.exe" PathToSample\0a0b3d91698a46d409791d4dd866e56ddd70f91a3f1d4557a0cb2899bda1e524.bin, DllRegisterServer`

💡 : Rundll32.exe needs to be specified a function for running. DllRegisterServer is the function triggered in the MalDoc and is the EntryPoint of the malicious behavior. If you are using DllMain as an entrypoint, nothing will happens.

Hitting F9 (Run) or clicking on the right arrow places us in the Rundll process



Going to the breakpoint tab, right clicking gives this menu. You can add a "dll breakpoint", so when the debugger enter the dll, it stops



💡 : The expected name is the same as the filename

Executing a couple of time until we hit the entrypoint of our dll

We can now set as many breakpoints that we want. For unpacking this sample, only [VirtualAlloc] is needed, but do not hesitate to add breakpoints on [CreateThread] or [GetProcAddress] if you want to go deeper.

[VirtualAlloc] will be hit 3 times by the sample :

1. Memory allocation for the payload (decrypted 2nd stage)
2. Memory allocation for data in .data (encrypted 2nd stage)
3. Memory allocation for the creation of a new thread (execution of the 2nd stage)

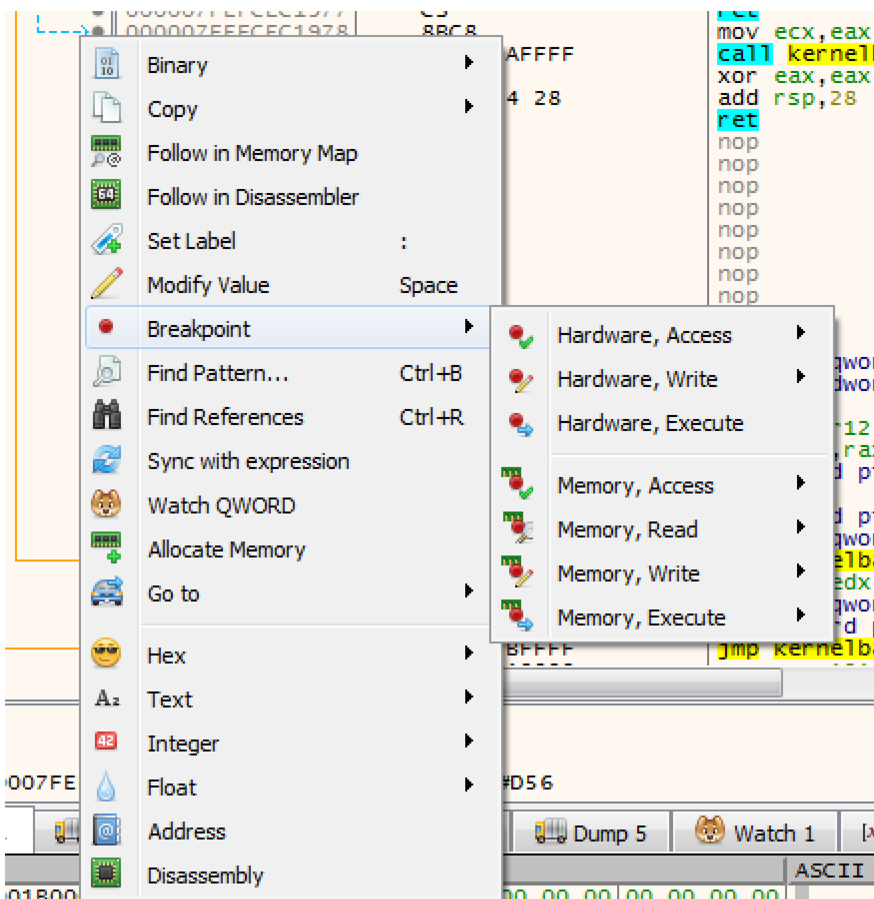
Upon hitting our first breakpoint on [VirtualAlloc]:

Let's hit the "Execute until return" button. The value stored in RAX after this is the address of the allocated memory region. In my case it's 0x01B0000. Following it in dump :

```
Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct
Address           Hex                                              ASCII
00000000001B0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B00E0  00 00                                            ................
00000000001B00F0  00 00 [00000000001B00C1] = 0000000000000000 (User Data)  ................
00000000001B0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000001B0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Now let’s pretend we don’t know what coming. A good way I found is to set breakpoint on the allocated region for access.

💡 : Making a breakpoint on “write” is also a good idea, but for whatever reason I didn’t really work in my case



right click on the memory region in dump

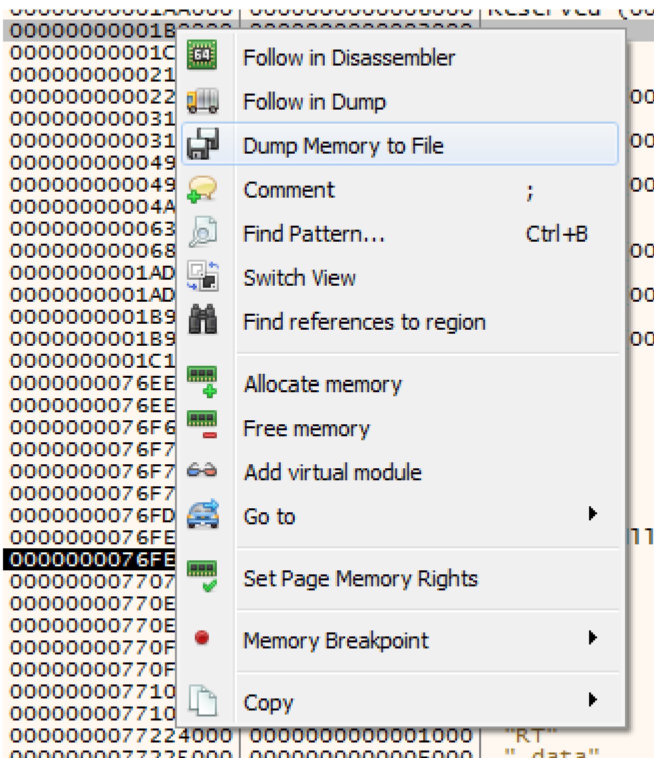
Something that looks like junk is written to the first allocated region of memory

```
Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Stru
Address           Hex                                               ASCII
00000000001B0000  4D 01 BD 01 2F 01 01 A2 01 01 01 01 01 0D 01 2F  M.½./..¢......./
00000000001B0010  01 01 11 A2 01 01 01 41 01 21 5E 01 01 01 01 01  ...¢...A.!^.....
00000000001B0020  B9 01 BD 01 65 01 01 01 73 01 61 BC A3 A3 0B 01  ¹.½.e...s.a¼££..
00000000001B0030  01 01 01 01 01 01 01 73 01 01 01 01 75 01 5F 74  .......s....u._t
00000000001B0040  01 01 01 01 91 01 01 64 3D 01 01 11 01 01 69 01  .......d=.....i.
00000000001B0050  01 D9 BD 21 11 5E 01 01 01 01 01 41 A2 01 01 01  .Ù½!.^.....A¢...
00000000001B0060  01 01 0F 01 01 01 01 01 01 01 41 01 01 01 0A CD  ..........A....Í
00000000001B0070  52 01 0F 01 01 01 01 01 01 11 01 01 01 25 01 01  R............%..
00000000001B0080  01 01 01 64 0C 01 24 0C 01 01 01 01 53 01 01 01  ...d..$.....S...
00000000001B0090  01 B5 11 6E 01 01 01 60 A2 E5 00 3B 71 9C 01 01  .µ.n...`¢å.;q...
00000000001B00A0  01 01 01 01 01 01 01 01 01 01 01 F1 01 01 01 6D  ...........ñ...m
00000000001B00B0  01 01 01 01 01 6E 65 01 01 01 01 01 A2 5B 01 01  .....ne.....¢[..
00000000001B00C0  20 01 08 01 5F 01 01 01 01 01 BC 11 01 62 00 4E   ..._.....¼..b.N
00000000001B00D0  F1 1B 01 01 01 01 01 01 01 6F 51 01 21 01 01 BD  ñ........oQ.!..½
00000000001B00E0  01 01 23 01 01 07 01 6F 01 01 0F 01 01 61 01 01  ..#....o.....a..
00000000001B00F0  81 55 01 21 01 01 5F 01 05 01 2C A3 01 A3 01 A2  .U.!.._...,£.£.¢
00000000001B0100  01 6C 01 A2 8D 66 5F 3B 01 99 05 01 5F 5F 01 6F  .l.¢.f_;....__.o
00000000001B0110  01 01 01 01 D3 01 01 A2 01 45 BD C6 05 45 01 01  ....Ó..¢.E½Æ.E..
00000000001B0120  01 17 01 01 5F 01 01 73 15 01 01 21 6F 60 01 5F  ...._..s...!o`._
```

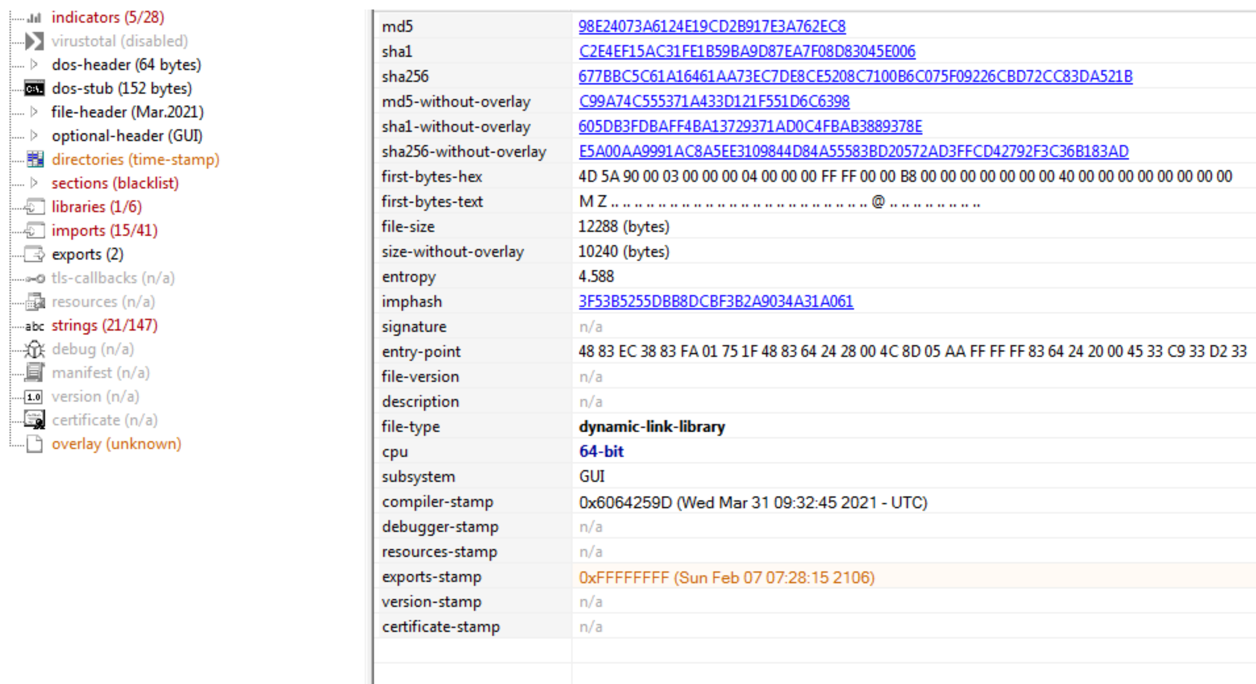

Can you guess what will this become ?


```
byte ptr ds:[rcx+rax*1]=[00000000001B0003]=1
dl=0

.text:000007FEF7A72E6E 0a0b3d91698a46d409791d4dd866e56ddd70f91a3f1d4557a0cb2899bda1e

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | S
Address           Hex                                               ASCII
00000000001B0000  4D 5A BD 01 2F 01 01 A2 01 01 01 01 01 0D 01 2F  MZ½./..¢......./
00000000001B0010  01 01 11 A2 01 01 01 41 01 21 5E 01 01 01 01 01  ...¢...A.!^.....
00000000001B0020  B9 01 BD 01 65 01 01 01 73 01 61 BC A3 A3 0B 01  ¹.½.e...s.a¼££..
00000000001B0030  01 01 01 01 01 01 01 73 01 01 01 01 75 01 5F 74  .......s....u._t
00000000001B0040  01 01 01 01 91 01 01 64 3D 01 01 11 01 01 69 01  .......d=.....i.
00000000001B0050  01 D9 BD 21 11 5E 01 01 01 01 01 41 A2 01 01 01  .Ù½!.^.....A¢...
00000000001B0060  01 01 0F 01 01 01 01 01 01 01 41 01 01 01 0A CD  ..........A....Í
00000000001B0070  52 01 0F 01 01 01 01 01 01 11 01 01 01 25 01 01  R............%..
00000000001B0080  01 01 01 64 0C 01 24 0C 01 01 01 01 53 01 01 01  ...d..$.....S...
00000000001B0090  01 B5 11 6E 01 01 01 60 A2 E5 00 3B 71 9C 01 01  .µ.n...`¢å.;q...
00000000001B00A0  01 01 01 01 01 01 01 01 01 01 01 F1 01 01 01 6D  ...........ñ...m
00000000001B00B0  01 01 01 01 01 6E 65 01 01 01 01 01 A2 5B 01 01  .....ne.....¢[..
00000000001B00C0  20 01 08 01 5F 01 01 01 01 01 BC 11 01 62 00 4E   ..._.....¼..b.N
00000000001B00D0  F1 1B 01 01 01 01 01 01 01 6F 51 01 21 01 01 BD  ñ........oQ.!..½
00000000001B00E0  01 01 23 01 01 07 01 6F 01 01 0F 01 01 61 01 01  ..#....o.....a..
00000000001B00F0  81 55 01 21 01 01 5F 01 05 01 2C A3 01 A3 01 A2  .U.!.._...,£.£.¢
00000000001B0100  01 6C 01 A2 8D 66 5F 3B 01 99 05 01 5F 5F 01 6F  .l.¢.f_;....__.o
00000000001B0110  01 01 01 01 D3 01 01 A2 01 45 BD C6 05 45 01 01  ....Ó..¢.E½Æ.E..
00000000001B0120  01 17 01 01 5F 01 01 73 15 01 01 21 6F 60 01 5F  ...._..s...!o`._
00000000001B0130  01 01 01 01 0F 01 01 01 01 01 63 01 01 01 01 68  ..........c....h
00000000001B0140  01 64 81 01 4D 01 02 6C 01 01 01 20 01 01 5F 01  .d..M..l... .._.
00000000001B0150  11 01 01 01 76 78 21 69 11 54 01 21 01 01 71 01  ....vx!i.T.!..q.
00000000001B0160  01 01 01 BD 01 17 01 01 68 01 BD 01 A2 01 01 B9  ...½....h.½.¢..¹
00000000001B0170  01 01 01 01 A2 01 FE CC 01 01 01 01 01 01 01 11  ....¢.þÌ........
00000000001B0180  01 01 01 01 01 01 01 4C 01 BD 01 01 01 01 5F 01  .......L.½...._.
00000000001B0190  01 79 01 00 03 21 65 01 01 01 01 01 01 07 01 01  .y...!e.........
00000000001B01A0  5F 03 01 1E 62 01 57 01 5F BF BB 01 01 BD 01 FE  _...b.W._¿»..½.þ
```

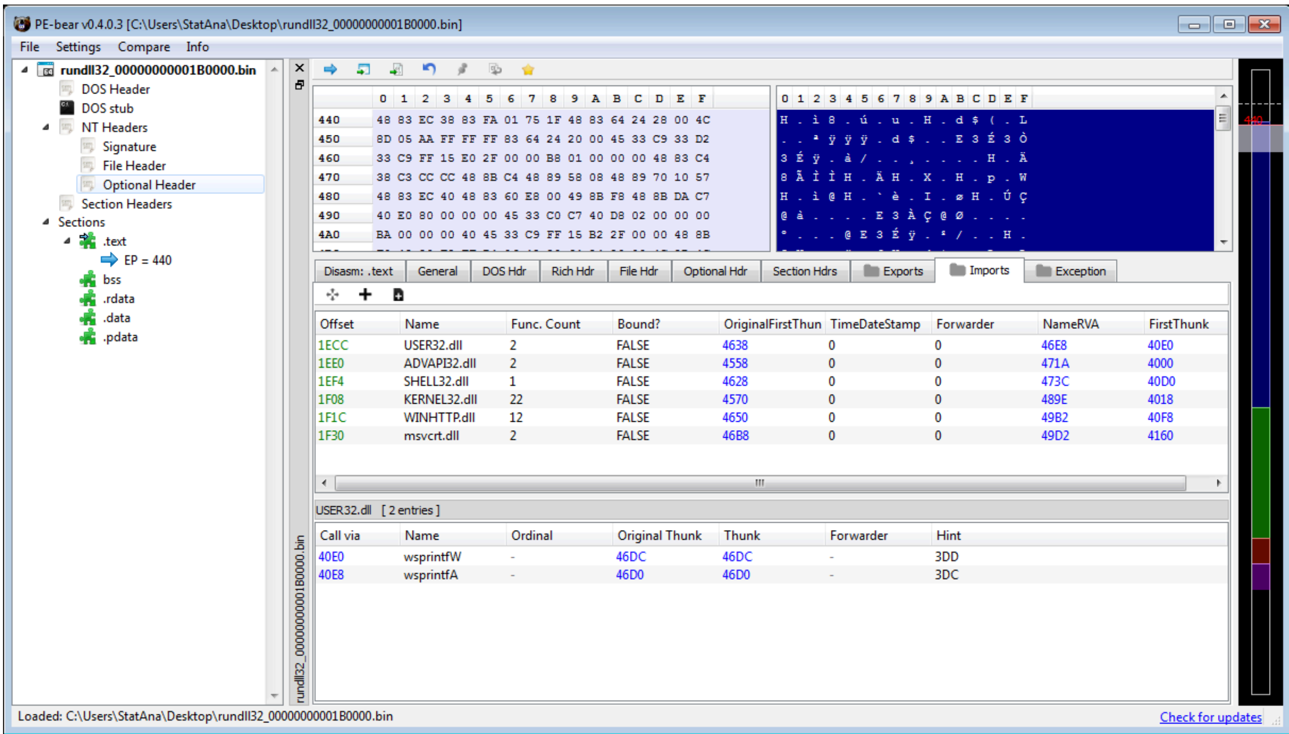

Sounds familiar isn’t it ? That’s actually the 2nd stage which is responsible of the C2 communication and that’s where we will find the C2 config. Now we just got to dump the memory to a file



Opening it with PeStudio :



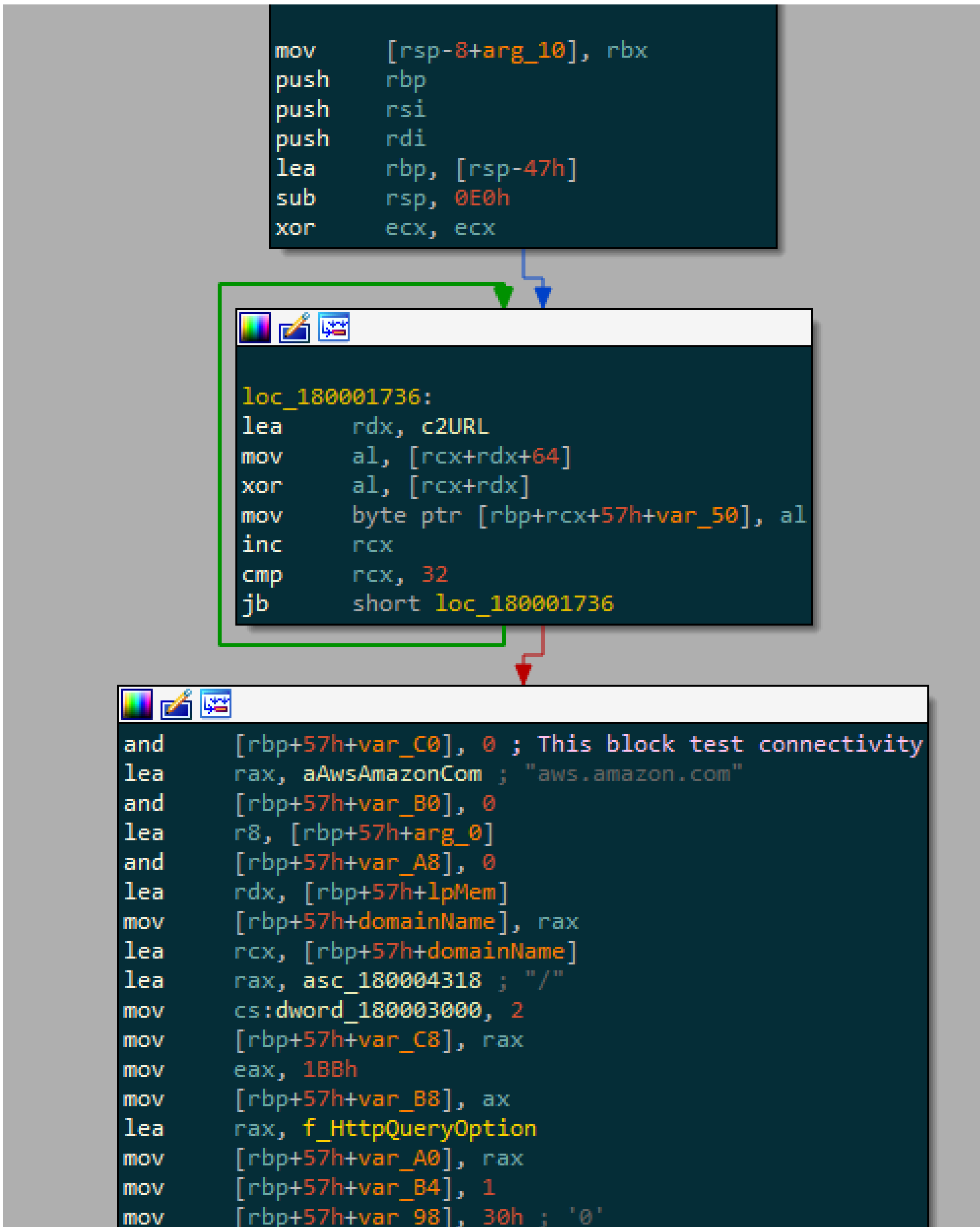
All imports are resolved, no need to remap of anything



Opening it with IDA, we only got a small set of functions

Function name	Segment	Start
StartAddress	.text	00000
PluginInit	.text	00000
DllEntryPoint	.text	0000
sub_180001074	.text	00000
sub_180001100	.text	00000
sub_1800013B8	.text	00000
sub_1800014C4	.text	00000
sub_18000159C	.text	00000
sub_180001658	.text	00000
sub_180001720	.text	00000
sub_180001920	.text	00000
sub_180001998	.text	00000
sub_180001A24	.text	00000
sub_180001B94	.text	00000
sub_180001D48	.text	00000
sub_180001F94	.text	00000
sub_1800020DC	.text	00000
sub_180002170	.text	00000
sub_180002314	.text	00000
memcpy	.text	0000
memset	.text	0000

Nothing is obfuscated and you can quite easily find the function responsible for the C2 communication :



You can also notice the making of the cookie that will be sent to the C2 :

```
lea     r8, aCookieGads ; "Cookie: __gads="
lea     rdx, aSU         ; "%s%u"
mov     rcx, rax         ; LPWSTR
call    cs:wsprintfW
movsxd  rbx, eax
mov     r9d, esi
lea     rdx, aSU         ; "%s%u"
lea     rsi, asc_18000427C ; ":"
mov     r8, rsi
lea     rcx, [rdi+rbx*2] ; LPWSTR
call    cs:wsprintfW
movsxd  rcx, eax
add     rbx, rcx
call    cs:GetTickCount64
lea     rcx, [rdi+rbx*2] ; LPWSTR
mov     r8, rsi
mov     r9, rax
mov     rax, 624DD2F1A9FBE77h
mul     r9
sub     r9, rdx
shr     r9, 1
add     r9, rdx
lea     rdx, aSU         ; "%s%u"
shr     r9, 9
call    cs:wsprintfW
movsxd  rcx, eax
add     rbx, rcx
call    getSystemInfo
lea     rcx, [rdi+rbx*2] ; LPWSTR
mov     r9d, eax
mov     r8, rsi
lea     rdx, aSU         ; "%s%u"
call    cs:wsprintfW
movsxd  rcx, eax
add     rbx, rcx
lea     rcx, [rdi+rbx*2] ; LPWSTR
call    getVersionInfo
add     rbx, rax
lea     rcx, [rdi+rbx*2]
call    getProcInfo
add     rbx, rax
lea     rcx, [rdi+rbx*2]
call    getAccountAndPcName
add     rbx, rax
lea     rcx, [rdi+rbx*2]
call    getAdaptaterInfo
lea     r9, [rbp+57h+arg_0]
mov     rdx, rdi
lea     r8, [rbp+57h+lpMem]
lea     rcx, [rbp+57h+var_4C]
call    sendInfoToC2
test    eax, eax
```

Here we are interested in the config, so let's see how this is stored and decrypted. First it loads the address of an array located in the .data section. Then the array is decrypted in a for loop with a xor. Translating this in python gives :

```
decrypted = ""
for i in range(32):
```

```
        decrypted += chr(payload[i+64] ^ payload[i])
```

I guess we are lucky because that's not that difficult. Even more simple for you, I made a script that extract the payload and decode the config, you can find it here

You can also got the domain name easily by setting a breakpoint on "WinHttpConnect" and looking at the RDX register value

```
0000000001E9F7C0 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | 00 00 00 00 | ................
0000000001E9F7D0 | 70 00 72 00 | 6F 00 76 00 | 6F 00 6B 00 | 6F 00 72 00 | p.r.o.v.o.k.o.r.
0000000001E9F7E0 | 64 00 69 00 | 6E 00 6F 00 | 2E 00 73 00 | 70 00 61 00 | d.i.n.o...s.p.a.
0000000001E9F7F0 | 63 00 65 00 | 00 00 00 00 | CA 17 00 77 | 00 00 00 00 | c.e.....Ê..w....
0000000001E9F800 | 00 00 2C 00 | 00 00 00 00 | 01 52 B5 02 | 00 00 00 00 | ..,.....Rµ......
```

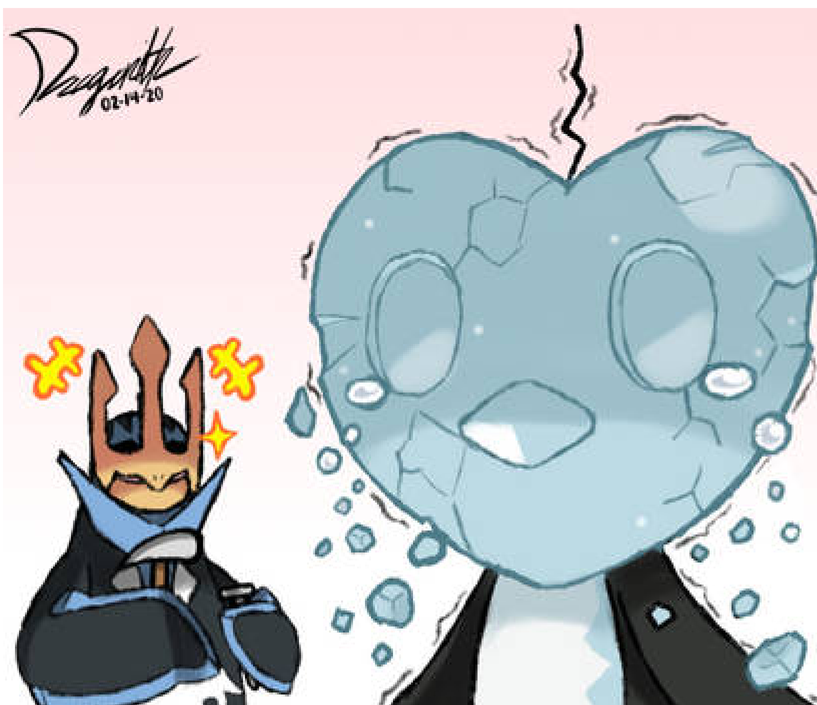
💡 : There is two call to this API, the first one is to "aws.amazon.com" in order to check if there is an internet connection (and also an anti sandbox)

Cleaning our handsPermalink

To be honest I wanted to have a fully automated script with Qiling but due to the emulation and all of the calculation done my script takes literally hours to hit the VirtualAlloc call, so that's pretty useless. If you got any idea on how to extract the payload quicker, do not hesitate to hit me up on Twitter.

I didn't make a deep dive on every routine and functions of the two executable because I don't think this is really interesting as this is something pretty common and I would like my posts to give as much value as possible and not enumerating everything if it doesn't help in our mission.

With this, you can extract the C2 domain in less than 3 minutes, which is not that bad no ?



As always, thanks for taking the time to read this, hope you learned something ! 😇

Source: https://4rchib4ld.github.io/blog/IcedIDOnMyNeckImTheCoolest/