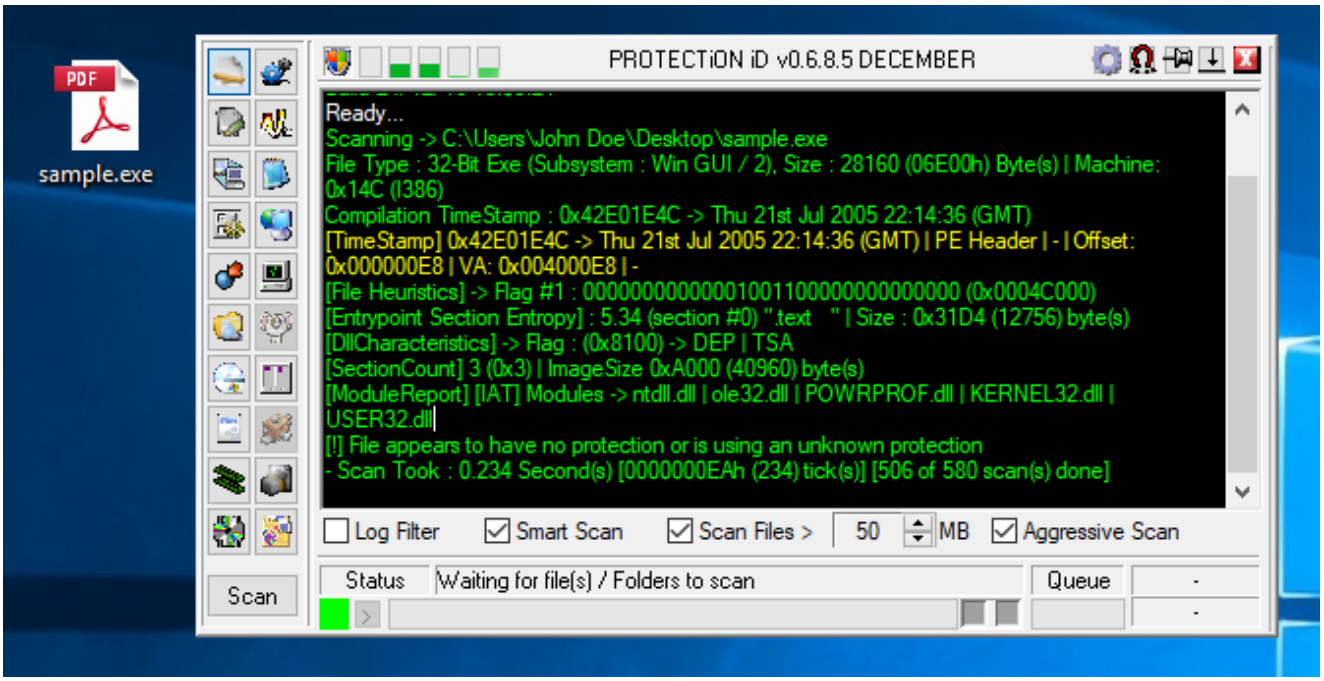
Upatre - Trojan Downloader

secrary.com/ReversingMalware/Upatre/

cd ../reverse_engineering_malware 6 minutes read
You can get the sample from theZoo

SHA-256: `1b893ca3b782679b1e5d1afecb75be7bcc145b5da21a30f6c18dbddc9c6de4e7`

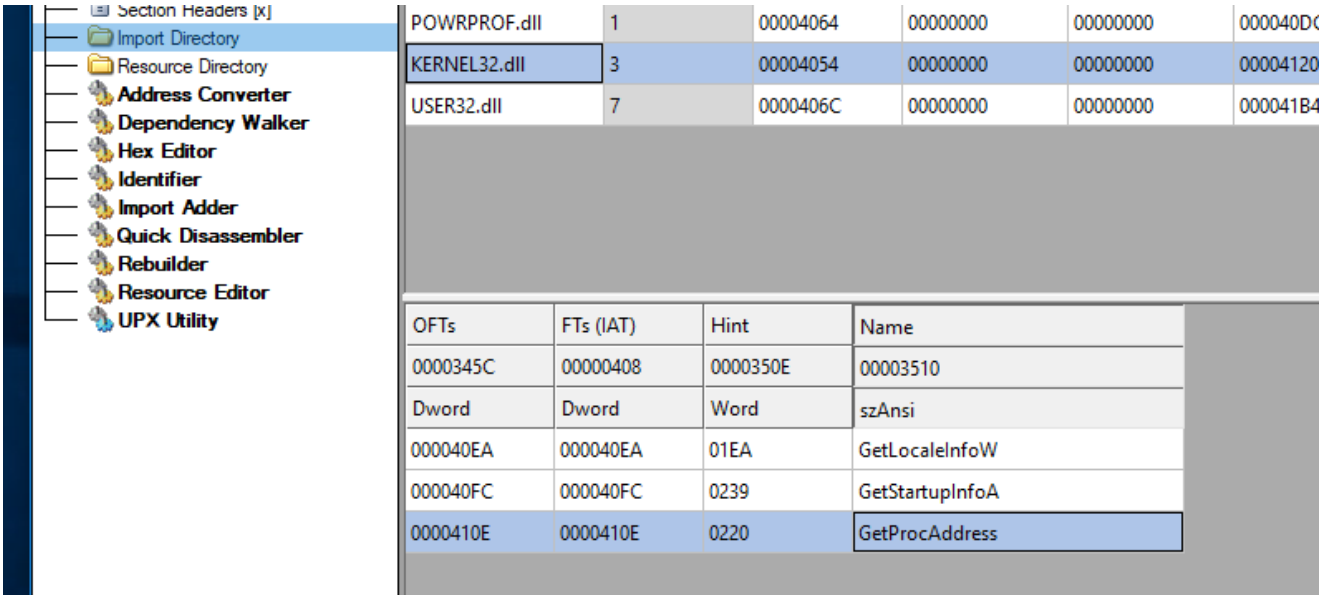
We can use behavior analysis from hybrid-analysis.



Seems like there is no known protection mechanism.

In the strings, there is nothing important other than this base64 encoded string:

…and imports is not eloquent but there is our friend `GetProcAddress`:



Let's open in `IDA`:

`sub_403760` is used to get necessary Win API functions:

```
public start
start proc near

var_1C= dword ptr -1Ch

mov     eax, large fs:18h
mov     eax, [eax+30h]
mov     ecx, [eax+18h]
sub     esp, 1Ch
push    esi
mov     esi, dword_405A38
push    edi
mov     dword_405214, eax
mov     dword_405844, ecx
call    sub_403760
mov     edx, dword_405518
mov     eax, [edx+1]
mov     ecx, dword_405020
mov     dword_405534, eax
call    sub_403380
push    offset unk_40532C
call    sub_401110
mov     edx, dword_405084
add     esp, 4
push    eax
mov     ecx, 2
mov     eax, edx
call    sub_4011D0
mov     edx, 0E40h
sub     edx, esi
push    edx
mov     dword_405A2C, eax
call    sub_4034D0
add     esp, 4
push    0
mov     dword_405510, eax
```

Inside `sub_403760`, malware decrytes strings and uses `GetProcAddress` to get addresses of functions:

```
004030D0        mov     eax, [edx+18h]
004030D3        push    offset aNtallocatevirt ; "4\x17--P{BpDEA^]I[UAK¦¦EIkL"
004030D8        mov     ntdll, eax
004030DD        call    DecryptString_
004030E2        mov     ecx, ntdll
004030E8        add     esp, 4
004030EB        push    eax
004030EC        push    ecx
004030ED        call    ds:GetProcAddress
004030F3        mov     NtAllocateVirtualMemory_, eax
004030F8        sub     dx, 0
004030FC        sub     dl, 0
```

To decrypt strings before call `GetProcAddress`, `Upatre` uses following decryption routine:

```
12
13  v2 = a2;
14  LOBYTE(a1) = *(_BYTE *)a2;
15  v10 = 0;
16  memset(&v11, 0, 0x7Fu);
17  v3 = *(_BYTE *)(a2 + 1);
18  v4 = 0;
19  if ( *(_BYTE *)(a2 + 1) )
20  {
21    v5 = (_BYTE *)(a2 + 5);
22    while ( *v5 )
23    {
24      v7 = (a1 ^ *v5) - v4 - 1;
25      v6 = ((unsigned __int8)a1 ^ *v5) - (_BYTE)v4 == 1;
26      *(&v10 + (_DWORD)v5 - a2 - 5) = v7;
27      if ( !v7 && v6 )
28        a1 = __ROL4__(a1, -32);
29      ++v4;
30      ++v5;
31      if ( v4 >= v3 )
32      {
33        v2 = a2;
34        goto LABEL_10;
35      }
36    }
37    *(_BYTE *)a2 = 0;
38    v2 = a2;
39  }
40LABEL_10:
41  *(&v10 + v3) = 0;
42  *(_BYTE *)(v3 + v2) = 0;
43  for ( i = &v10; v3; ++i )
44  {
45    --v3;
46    i[v2 - (_DWORD)&v10] = *i;
47  }
48  return v2;
49}
```

Inside `sub_402F30` malware uses this teqnique to get addresses for following Win API functions:

`NtAllocateVirtualMemory`, `NtUnmapViewOfSection`, `CreateThread`, `WaitForSingleObject`, `LoadLibraryA`, `HeapAlloc`, `RtlAllocateHeap`, `RtlDecompressBuffer`, `FlushInstructionCache`, `NtGetContextThread`.

The decryption routine is used heavily by malware in different places to get plain text.

```
0040325E call    GetProcAddress__
00403263 or      dword_40502C, 2000h
0040326D push    offset aRtldecompressb ; "RtlDecompressBuffer"
00403272 mov     RtlAllocateHeap_, eax
00403277 call    DecryptString_
0040327C mov     ecx, ntdll
00403282 add     esp, 4
00403285 push    eax
00403286 push    ecx
00403287 call    esi ; GetProcAddress
00403289 push    offset aFlushinstructi ; "FlushInstructionCache"
0040328E mov     RtlDecompressBuffer_, eax
00403293 call    DecryptString_
00403298 mov     edx, ntdll
0040329E add     esp, 4
004032A1 push    eax
004032A2 xor     ecx, ecx
004032A4 mov     eax, edx
004032A6 call    GetProcAddress__
004032AB cmp     _NtGetContextThread, 0
004032B2 mov     off_40500C, eax
004032B7 jnz     short loc_4032D5
```

At `00403572`, `Upatre` decodes base64 encoded string and saves at `004051B0` (I renamed variable as `decrypted_bin`):

```
00403554 DecryptBigString:
00403554 push    0
00403556 push    0
00403558 lea     ecx, [esp+10h+var_4]
0040355C push    ecx
0040355D push    esi
0040355E push    1               ; CRYPT_STRING_BASE64
00403560 push    0E40h
00403565 push    offset aAZmarbmte1mtez ; "a/ZMARbMTE1MTEzOSEx8s7NMTPRMdGFNTAxIdFV"...
0040356A mov     [esp+24h+var_4], 0E41h
00403572 call    CryptStringToBinaryA_
00403578 test    eax, eax
0040357A jnz     short loc_403586

00403586
00403586 loc_403586:
00403586 mov     decrypted_bin, esi
0040358C test    esi, esi
0040358E pop     esi
0040358F jnz     short loc_403535
```

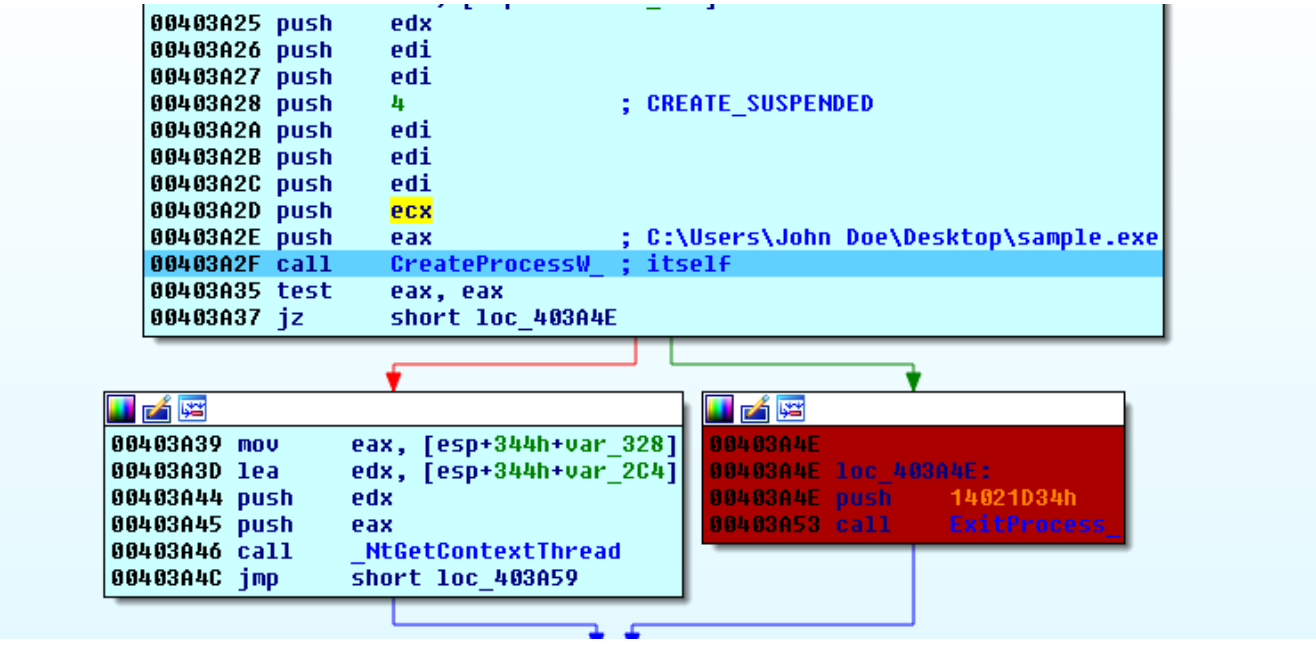
At `0040386D` it creates a new thread:

```
00403852 push    0FFFFFFFFh
00403854 push    0
00403856 push    0
00403858 push    0
0040385A push    offset loc_403F30 ; lpStartAddress
0040385F push    0
00403861 push    0
00403863 mov     dword_405A3C, 1
0040386D call    CreateThread_
00403873 push    eax
00403874 call    off_40563C      ; WaitForSingleObject
0040387A call    IsPwrHibernateAllowed
```

Main work starts inside the thread at `00403900`, Where it decryptes and gets addresses for several Win API functions: `CreateProcessW`, `ExitProcess`, `NtWriteVirtualMemory`, `NtSetContextThread`, etc.

```
0040390B push    offset enc_CreateProcessW ; "CreateProcessW"
00403910 mov     [esp+344h+var_2CC], 10007h
00403918 mov     [esp+344h+var_324], eax
0040391C call    DecryptString_
00403921 mov     edx, ntdll
00403927 add     esp, 4
0040392A push    eax
0040392B xor     ecx, ecx
0040392D mov     eax, edx
0040392F call    GetProcAddress__
00403934 mov     CreateProcessW_, eax
00403939 mov     eax, 0x4E6587D2
```

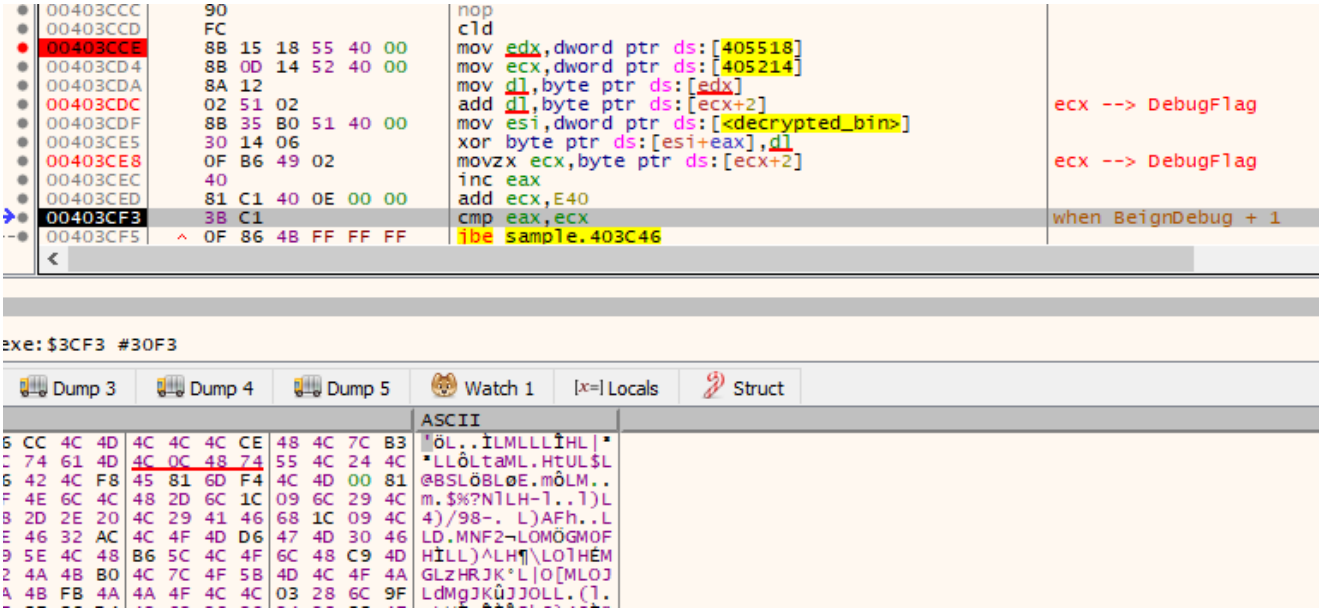
Creates itself as a new process in suspended mode and saves `Context`:



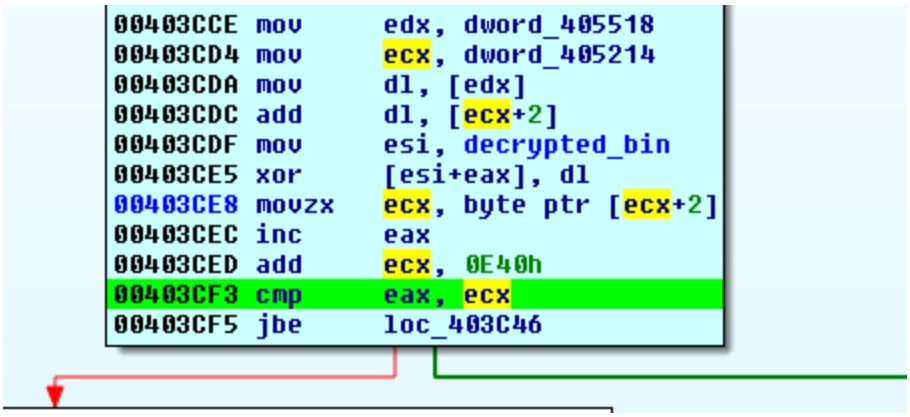
Anti-Debug:

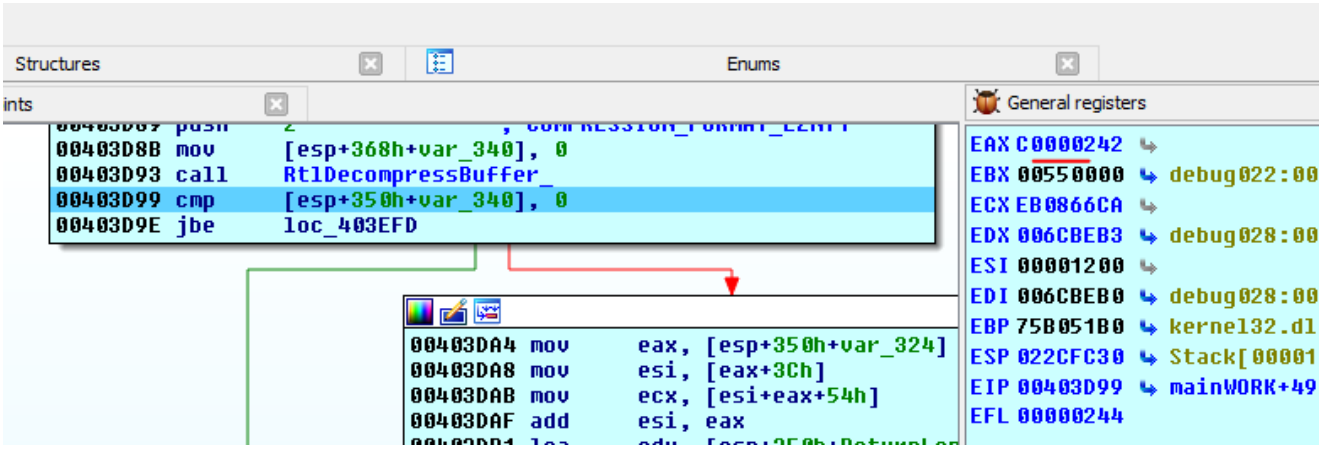
There is one interesting anti-debug trick, at the start, it saves `PEB` and uses `BeingDebug` value `[PEB+2]` in XOR decryption routine, outside of a debugger this value is `0` and adding `0` don't cause any error, but if we try to add `1` (which is the value of `[PEB+2]` if the executable is inside a debugger) it may cause error. In this case `RtlDecompressBuffer` returns `0xC0000242` (STATUS_BAD_COMPRESSION_BUFFER) error.

The reason of this error is that before calling `RtlDecompressBuffer`, malware decrypts(with XOR) decoded strings using `0x4C+[PEB+2]` which is `0x4D` inside a debugger instead of `0x4C`, because of this result is corrupted output.



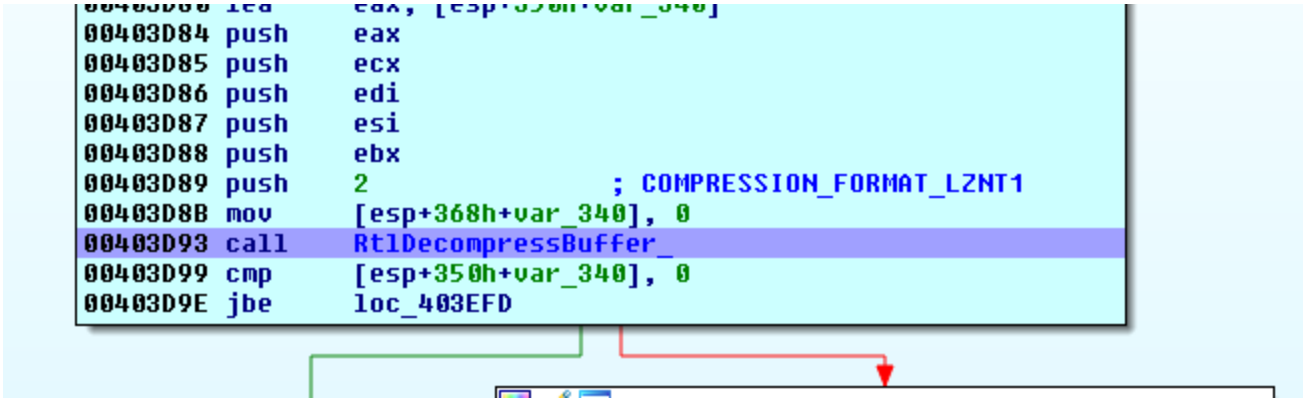
`[eax+2]` is the value of `BeingDebug`:



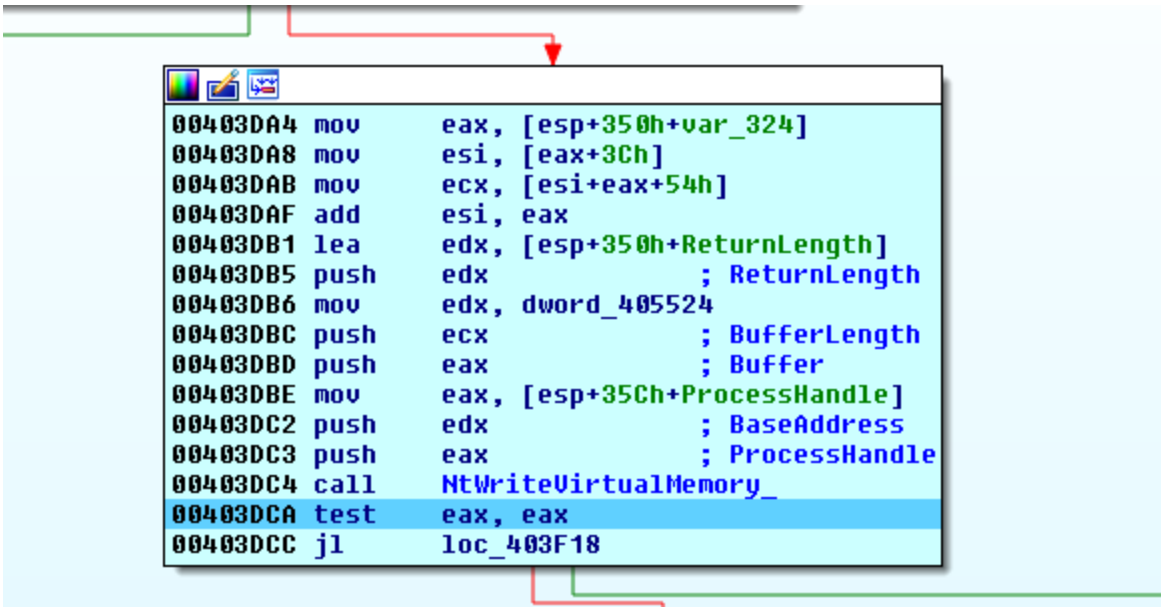


We can use `ScyllaHide` plugin for `IDA` to defeat this anti-debug method.

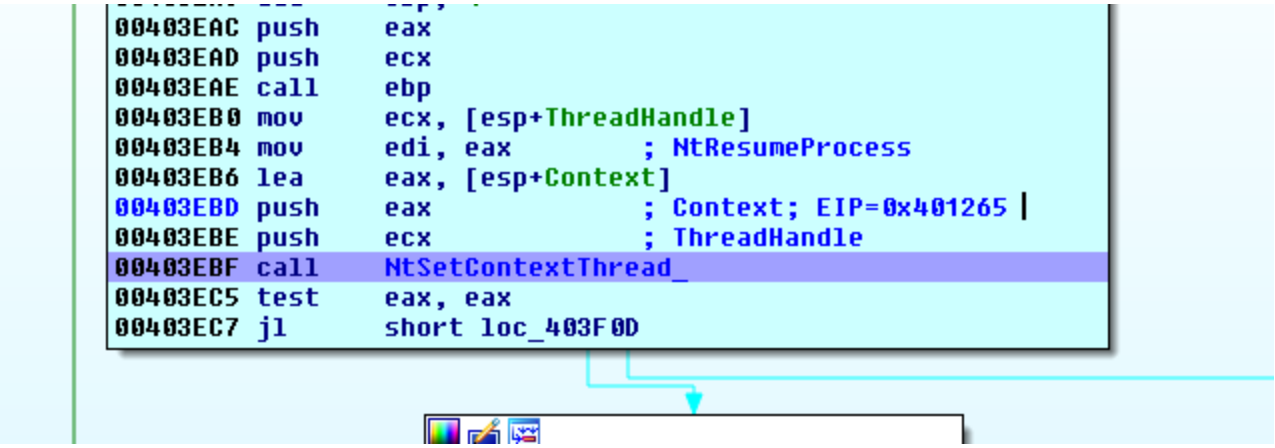
Decompresses decoded and decrypted base64 string using `RtlDecompressBuffer` (format `COMPRESSION_FORMAT_LZNT1`):



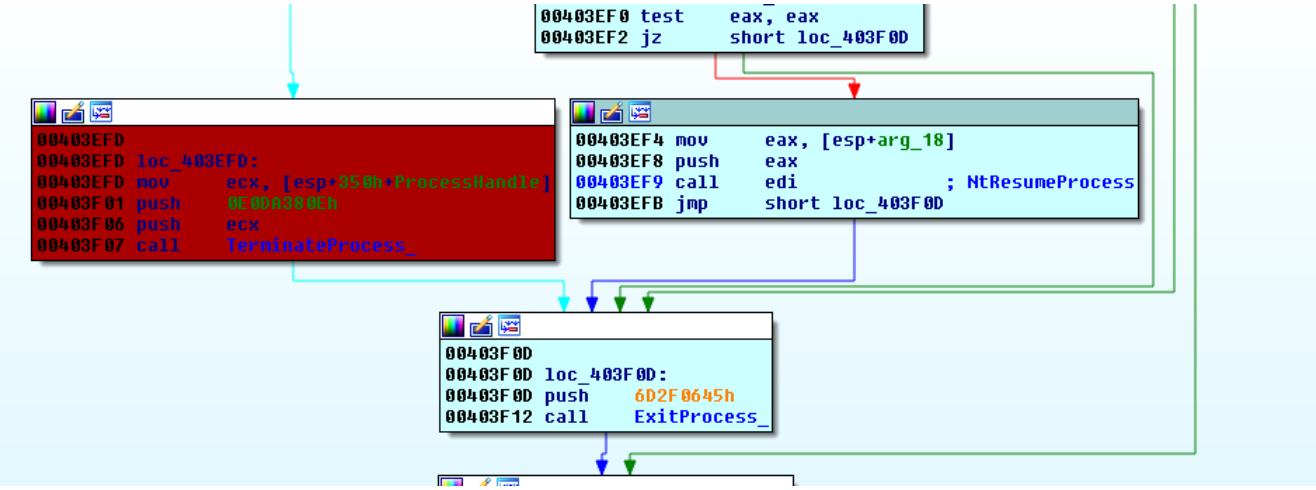
…and writes into suspened process:



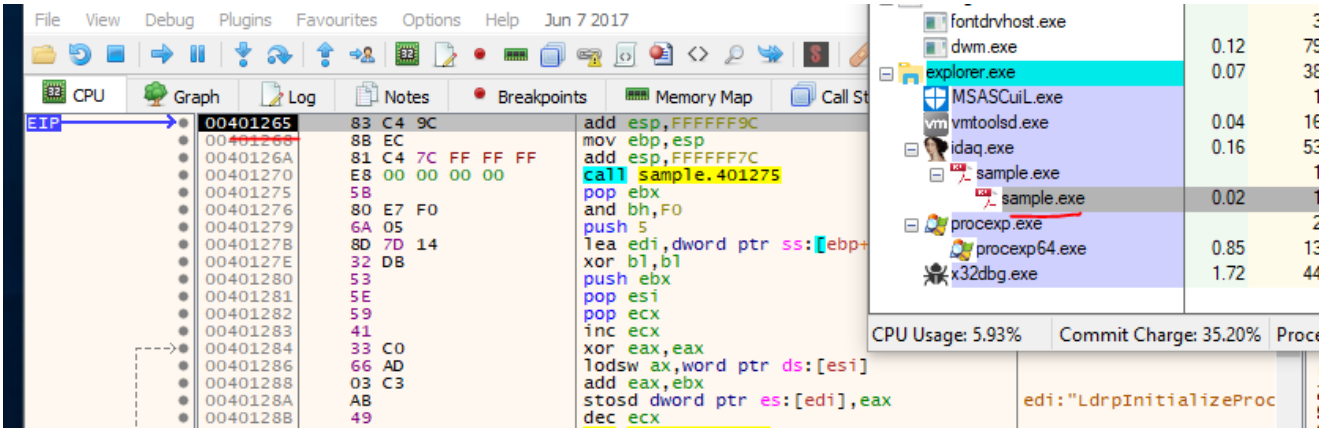
After decompress it calls `NtSetContextThread`, value of `EIP` is `401265`:



Resumes thread and exits:

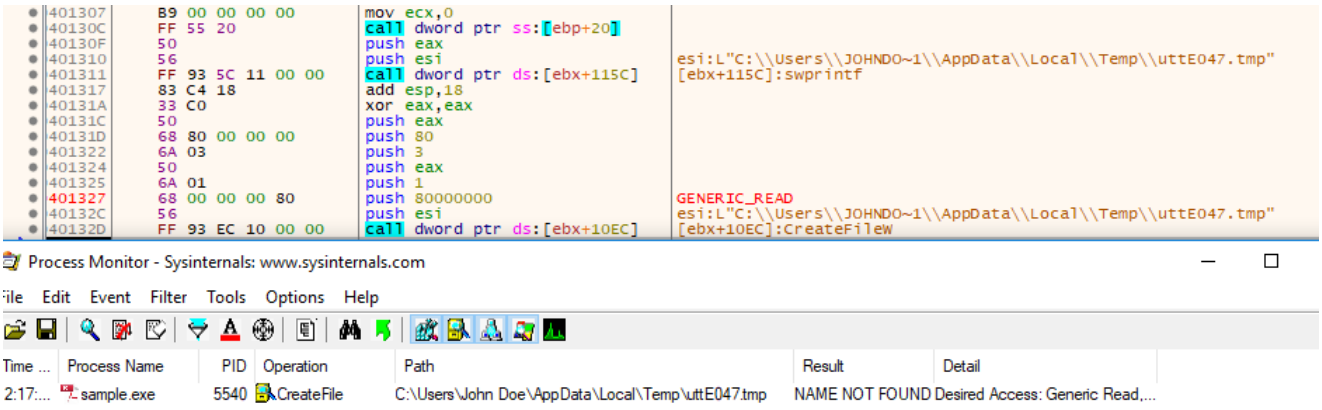


Before `NtResumeProcess` call attach `x32dbg` to child process and set `EIP` to `401265`:

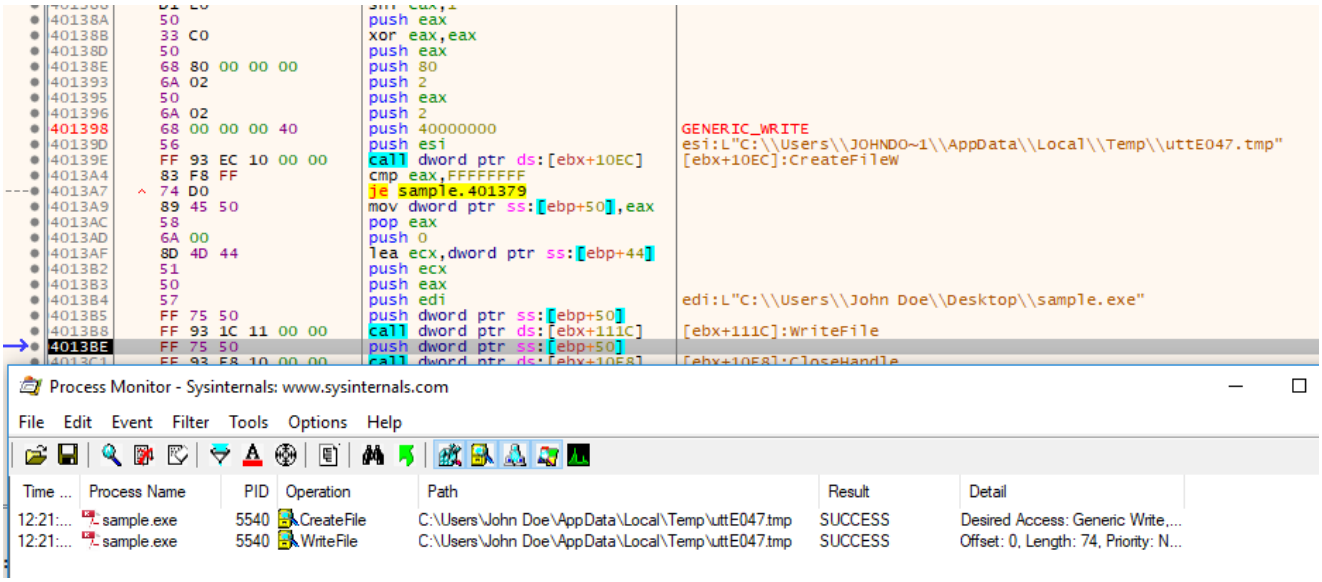


Close `IDA` and start analyzing of the child process.

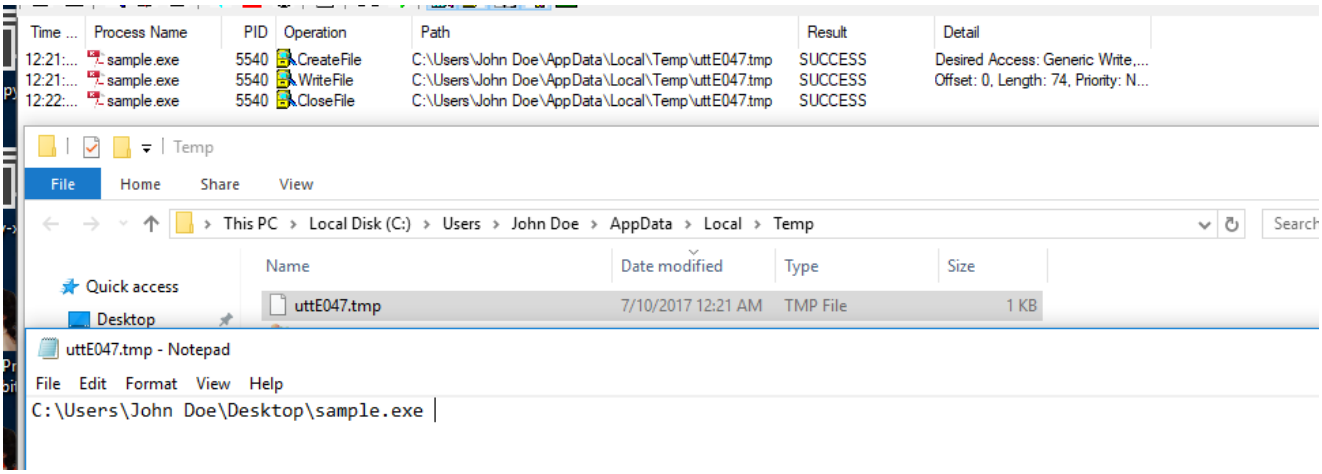
Tries to read `uttE047.tmp` file from `%TEMP%` directory without success:



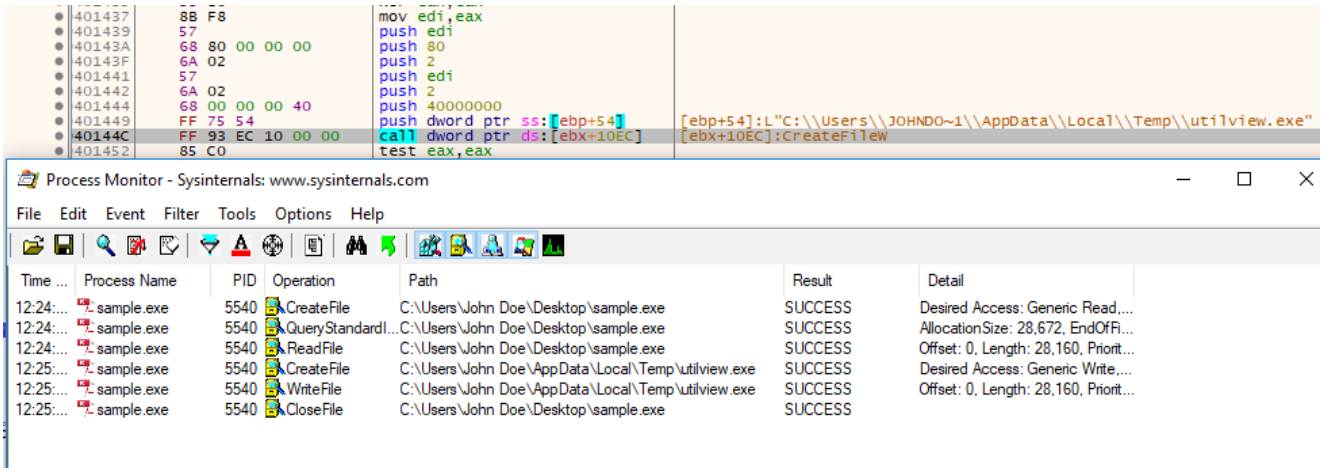
Creates one and writes location of the executable:



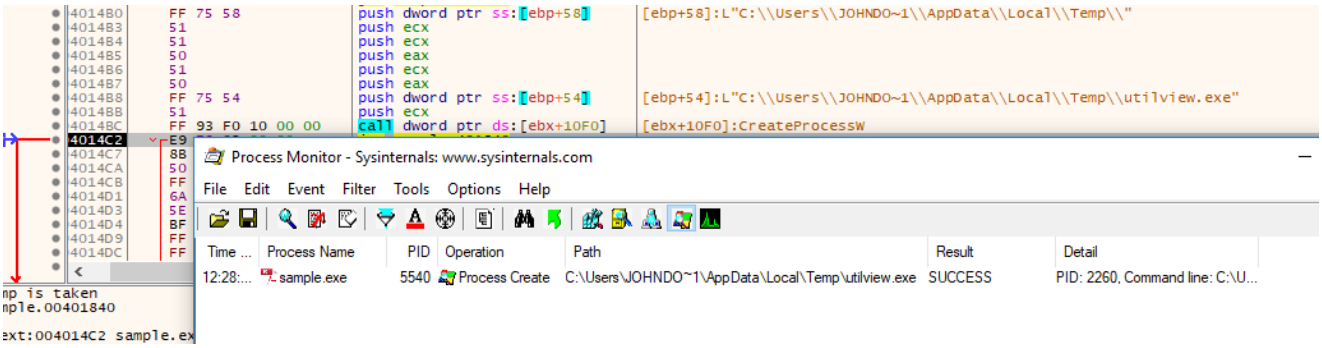
Inside of `uttE047.tmp` file:



Copies executale to `%TEMP%` directory as `utilview.exe`:



…and creates as new process:

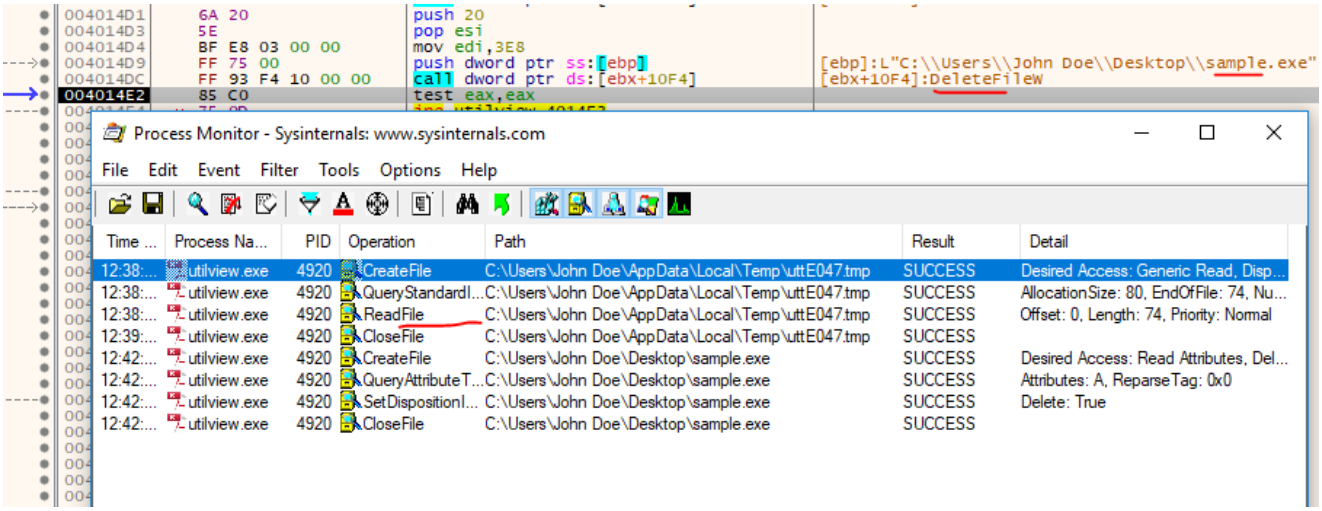


This process is exactly same as the first process, creates a new process and injects decoded and decompressed code.

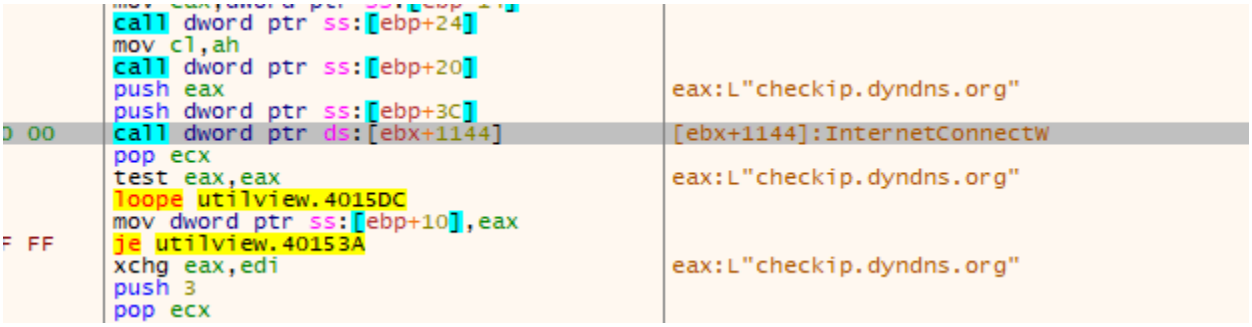
Let's reverse last part (injected code) a little bit higher level.

Now we are here: sample.exe -> sample.exe -> utilview.exe -> **utilview.exe**

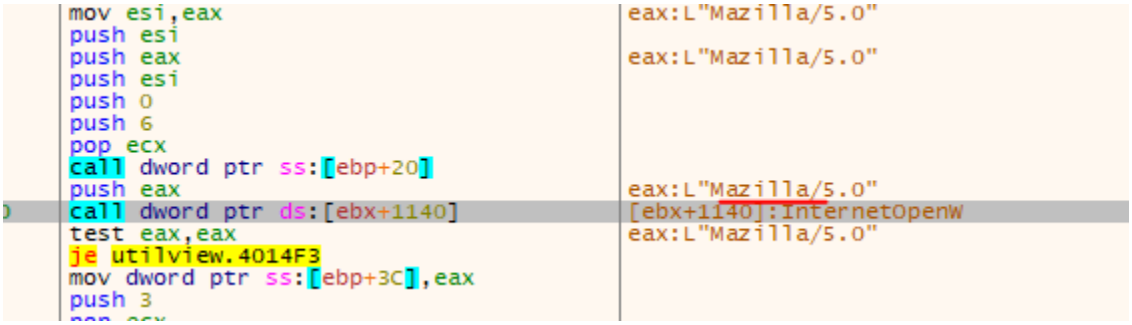
The injected code is also same as before it checks `uttE047.tmp` file, but this time there is `uttE047.tmp` in `%TEMP%` directory and malware goes a different direction, reads the content of `uttE047.tmp`, which is the location of the executable and removes that executable:



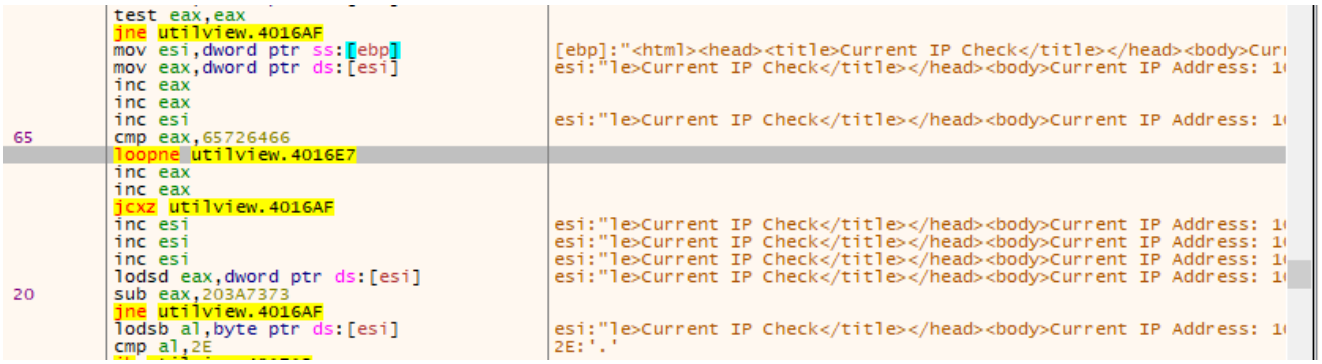
After this it gets IP of the victim using `checkip.dyndns.com`:



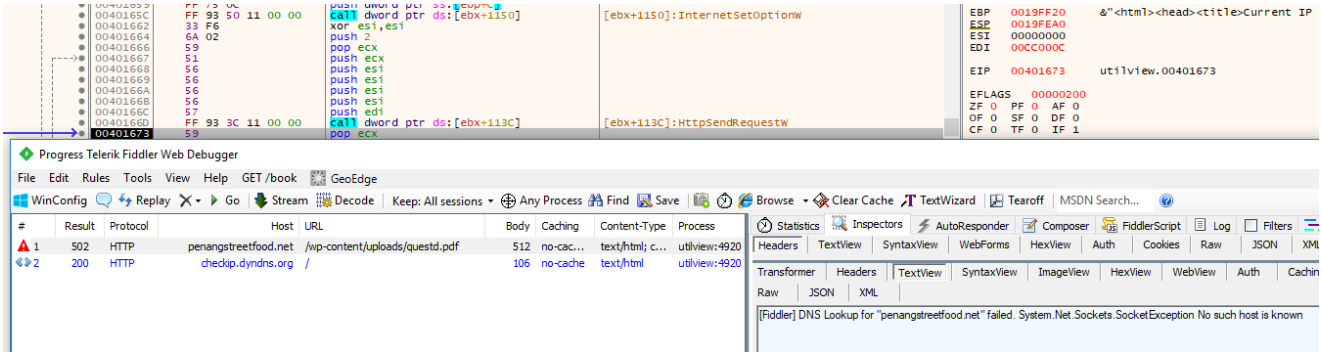
Also, there is a typo in user-agent string:



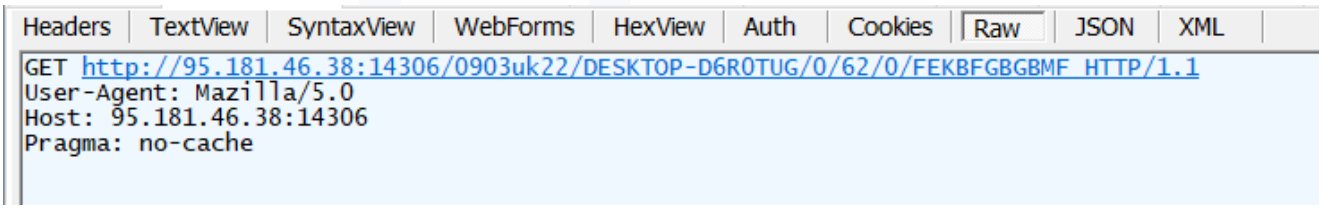
and parses IP from returned file:



It tries to download `questd.pdf` from `http://penangstreetfood.net/wp-content/uploads/questd.pdf` and `http://yumproject.com/wp-content/uploads/2014/11/questd.pdf` without success.



Sends `GET` requests to `95.181.46.38` with client related information, last string derives from victim's IP address, `B` is instead of `.`



That's all… `Upatre`'s main function is to download malicious files.

Note

If you prefer you can use my script to extract payload instead of doing it manually:

I know, I overlook many things related to `Upatre`, due to my limited knowledge, if you find something interesting please contact me.

I'm new to reversing malware and any kind of feedback is helpful for me.

Twitter: @_qaz_qaz