



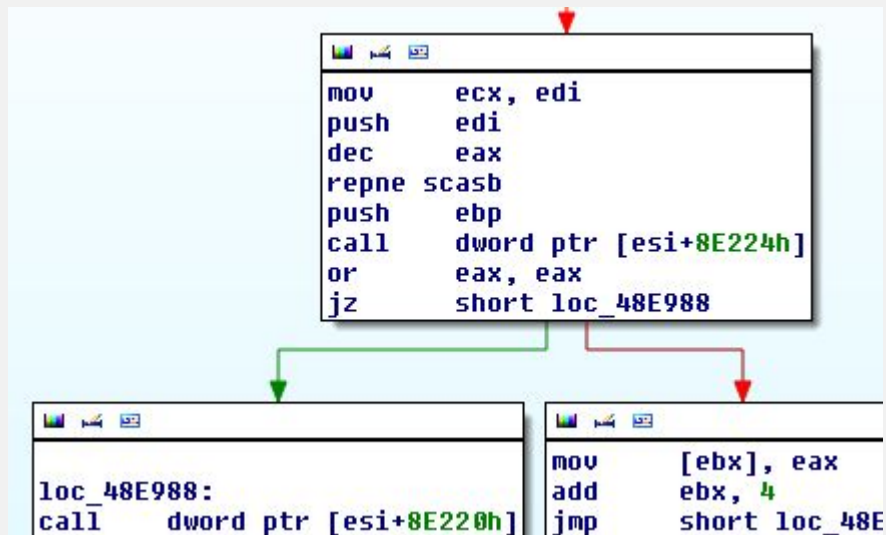
Présentation Malware Analysis

Luc Nicaud - Benjamin Dumont

Recherche statique avec IDA

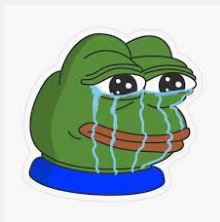
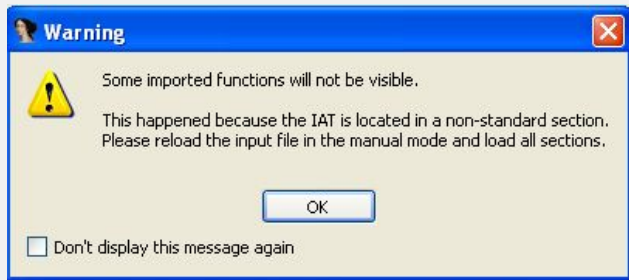
Chaque appel de fonction est obfusqué :

```
mov     ebp, [esi+8E228h]
lea     edi, [esi-1000h]
mov     ebx, 1000h
push    eax
push    esp
push    4
push    ebx
push    edi
call    ebp
```



Recherche statique avec IDA

Une zone de code exécutable qui n'est pas correctement analysé par IDA:

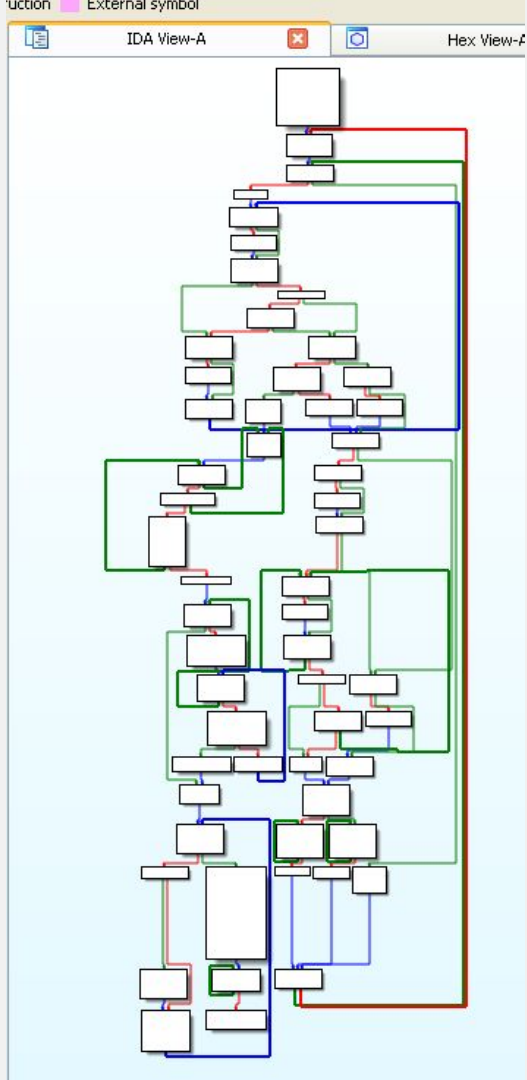
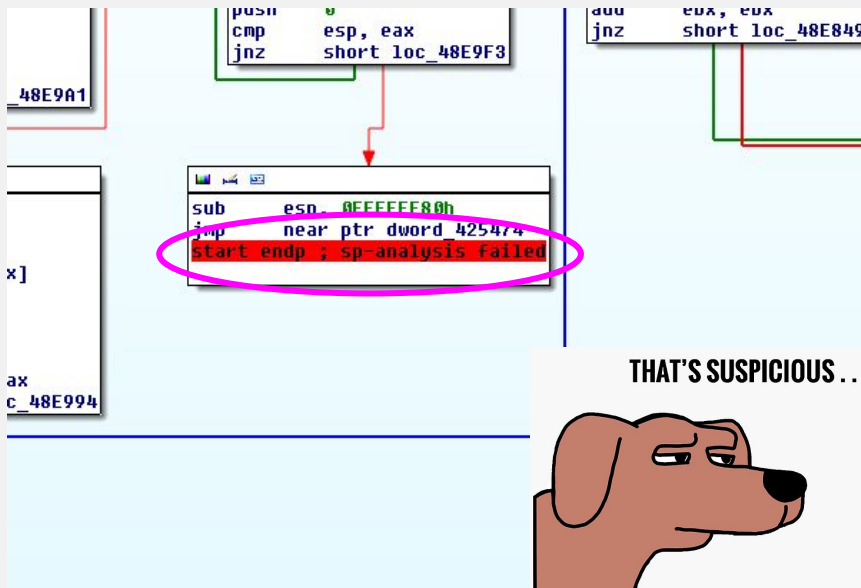


```
UPX0:00425474 dword_425474 dd 13EE3h dup(?) ; CODE XREF: start+1DC1j
UPX0:00425474 UPX0 ends
UPX0:00425474
UPX1:00475000 ; Section 2. (virtual address 00075000)
UPX1:00475000 ; Virtual size : 0001A000 ( 106496.)
UPX1:00475000 ; Section size in file : 00019C00 ( 105472.)
UPX1:00475000 ; Offset to raw data for section: 00000400
UPX1:00475000 ; Flags E0000040: Data Executable Readable Writable
UPX1:00475000 ; Alignment : default
UPX1:00475000 ; =====
UPX1:00475000 ; Segment type: Pure code
UPX1:00475000 ; Segment permissions: Read/Write/Execute
UPX1:00475000 segment para public 'CODE' use32
UPX1:00475000 assume cs:UPX1
UPX1:00475000 ;org 475000h
UPX1:00475000 assume es:nothing, ss:nothing, ds:UPX0, fs:nothing, gs:nothing
UPX1:00475000 dd 902A0008h, 54014200h, 0A8028420h, 50050940h, 0A00A1281h
UPX1:00475000 ; DATA XREF: start+140
UPX1:00475000 dd 40142402h, 80284805h, 50900A0h, 0A12015h, 142402A0h, 2848154h
UPX1:00475000 dd 50902A8h, 0A120550h, 1424A2A0h, 0A195ED40h, 0AE9CC02h
UPX1:00475000 dd 9108F102h, 0FE6E6E6Fh, 0E900046Eh, 8B03921Ch, 0E7820490h
UPX1:00475000 dd 0A0040A02h, 00FFCAC7Ch, 0E93C623Fh, 0A30FCE3h, 0E9BCAC02h
UPX1:00475000 dd 0A0038889h, 7E6CAE9Ch, 0E9DC4EFFh, 1562C89Ah, 0DB300394h
UPX1:00475000 dd 0FDEC9803h, 8F6DFF7h, 0E0516268h, 0E98C444Fh, 0B0130DE7h
UPX1:00475000 dd 865DE94Ch, 0F7DDCFEEh, 87908B2h, 259FCF3h, 0C39ACBFh
UPX1:00475000 dd 0EFE5A8E9h, 0CFDF6DF3h, 0C65A4F82h, 0E01C8485h, 27120883h
UPX1:00475000 dd 00EDDD6DFh, 6F4E0C86h, 49ADEE4h, 52C68A6Dh, 7C671387h
UPX1:00475000 dd 52BFCAD8h, 0E98CD71Ch, 1B037463h, 0BC7F899Eh, 0B3E0CD94h
```

Recherche statique avec IDA

Arbre des possibilités très chaotique 🤯 →

Et qui ne constitue qu'une partie de l'arbre réel ↓



Recherche dynamique avec debugger sur IDA

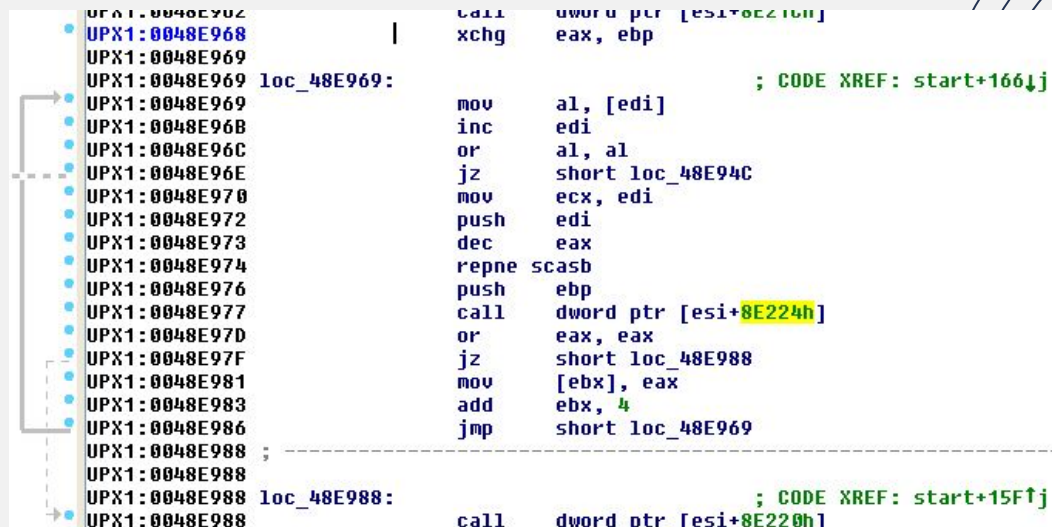
Import dynamique de kernel32

breakpoint en 0x48E977 :

call à $0x401000 + 0x8e224h = 0x48e224 \Rightarrow$
kernel32 GetProcAddress

de 0x48C000 à 0x48C5B0 \Rightarrow liste de nom
de fonction de kernel32

les fonctions sont enregistrées dans EBX à
partir de 0x4861EC



```
UPX1:0048E969 | call     dword ptr [esi+0E210h]
UPX1:0048E969 | xchg     eax, ebp
UPX1:0048E969 | loc_48E969:                                     ; CODE XREF: start+166↓j
UPX1:0048E969 | mov      al, [edi]
UPX1:0048E96B | inc      edi
UPX1:0048E96C | or       al, al
UPX1:0048E96E | jz       short loc_48E94C
UPX1:0048E970 | mov      ecx, edi
UPX1:0048E972 | push     edi
UPX1:0048E973 | dec      eax
UPX1:0048E974 | repne    scasd
UPX1:0048E976 | push     ebp
UPX1:0048E977 | call     dword ptr [esi+8E224h]
UPX1:0048E97D | or       eax, eax
UPX1:0048E97F | jz       short loc_48E988
UPX1:0048E981 | mov      [ebx], eax
UPX1:0048E983 | add      ebx, 4
UPX1:0048E986 | jmp      short loc_48E969
UPX1:0048E988 | ; -----
UPX1:0048E988 | loc_48E988:                                     ; CODE XREF: start+15F↑j
UPX1:0048E988 | call     dword ptr [esi+8E220h]
```

Recherche dynamique avec debugger sur IDA

Import dynamique de kernel32

Script python pour récupérer les adresses des fonctions :

```
address = 0x48C008
end = 0x48C5B0
size = end - address

dico = dict()
index = 0

while(address < end):
    function_name = ""
    while(Byte(address) != 0x00):
        #print(Byte(address))
        function_name += chr(Byte(address))
        address += 1
    dico[index] = function_name
    address += 1
    index += 1

print(dico)
```

=> en 0x48C108 : IsDebuggerPresent

Recherche dynamique avec debugger sur IDA

Modification à la volée du code :

```
loc_48E9BF:
mov     ebp, [esi+8E228h]
lea     edi, [esi-1000h]
mov     ebx, 1000h
push    eax
push    esp
push    4
push    ebx
push    edi
call    ebp
lea     eax, [edi+207h]
and     byte ptr [eax], 7Fh
and     byte ptr [eax+28h], 7Fh
pop     eax
push    eax
push    esp
push    eax
push    ebx
push    edi
call    ebp
pop     eax
popa
lea     eax, [esp+2Ch+var_AC]
```

bp en 0x48E9D6 et 0x48E9EB :

=> call VirtualProtect
de 400000 à 401000 en read write

puis
=> call VirtualProtect
et de 400000 à 401000 en read only



Recherche dynamique avec debugger sur IDA

Utilisation de l'heure pour détecter un potentiel debugger ? :

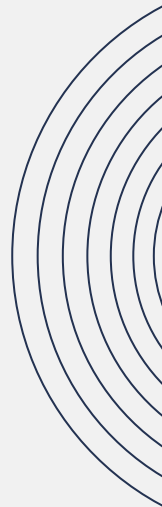
0x433475 : call GetSystemTimeAsFileTime

0x43348A : call with add de GetCurrentProcessId

0x433496 : call GetCurrentThreadId

0x4334A2 : call GetTickCount

0x4334B2 : call QueryPerformanceCounter



Recherche dynamique avec un pintool

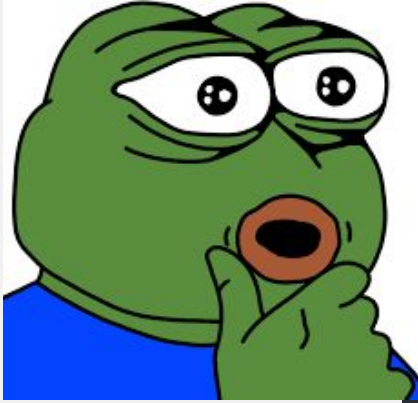
```
1 // Structure pour stocker les informations sur les symboles des fonctions
2 struct FunctionInfo {
3     std::string name;
4 };
5
6 // Table de hachage pour stocker les informations sur les symboles des fonctions
7 std::unordered_map<ADDRINT, FunctionInfo> functionInfoTable;
8
9 // Fonction pour obtenir les informations sur les modules chargés et les symboles des fonctions
10 VOID ImageLoad(IMG img, VOID *v)
11 {
12     std::cout << "Image loaded: " << IMG_Name(img) << std::endl;
13
14     // Parcours de tous les symboles de la fonction dans l'image chargée
15     for (SYM sym = IMG_RegsymHead(img); SYM_Valid(sym); sym = SYM_Next(sym)) {
16         FunctionInfo info;
17         info.name = SYM_Name(sym);
18         functionInfoTable.insert(std::make_pair(SYM_Address(sym), info));
19     }
20 }
21
```

Recherche dynamique avec un pintool



```
1  VOID PrintFunctionName(INS ins, VOID *v)
2  {
3      ADDRINT address = INS_Address(ins);
4      auto it = functionInfoTable.find(address);
5      if (it != functionInfoTable.end()) {
6          std::cout << "Function at address " << std::hex << address << ": " << it->second.name << std::endl;
7      }
8  }
9
10 VOID InstructionTrace(INS ins, VOID *v)
11 {
12     std::cout << "0x" << std::hex << INS_Address(ins) << ": ";
13     std::cout << INS_Disassemble(ins) << std::endl;
14 }
15
```

Recherche dynamique avec un pintool



```
0x7c876134: ret 0x4
0x4275c7: cmp esi, esp
0x4275c9: call 0x425b9a
0x4275ce: mov esi, esp
0x4275d0: call dword ptr [ebp-0x60]
Function at address 7c82f6ef: IsDebuggerPresent
0x7c82f6ef: mov eax, dword ptr fs:[0x18]
0x7c82f6f5: mov eax, dword ptr [eax+0x30]
0x7c82f6f8: movzx eax, byte ptr [eax+0x2]
0x7c82f6fc: ret
0x4275d3: cmp esi, esp
0x4275d5: call 0x425b9a
0x4275da: mov esi, eax
```

Miscellanées

```
UPX1:0048BD59 db 78h, 79h, 7Ah
UPX1:0048BD5C db 58h ; [
UPX1:0048BD5D db 5Ch, 5Dh, 5Eh
UPX1:0048BD60 db 5Fh ; _
UPX1:0048BD61 db 60h, 3Fh, 7Bh
UPX1:0048BD64 db 7Ch ; |
UPX1:0048BD65 db 7Dh, 7Eh, 7Fh
UPX1:0048BD68 db 55h ; U
UPX1:0048BD69 db 0Ch, 0A1h, 2Ah
UPX1:0048BD6C db 0FFh
UPX1:0048BD6D db 0FFh, 0FEh, 0FFh
UPX1:0048BD70 db 0FFh
UPX1:0048BD71 db 1Fh, 41h, 42h
UPX1:0048BD74 db 'CDEFGHIJKLMNOPQRSTUVWXYZ',0
UPX1:0048BD80 db 43h, 8Ah, 45h
UPX1:0048BD90 dd 45FF3Fh, 622F0617h, 1F860284h, 3
UPX1:0048BD98 dd 4E634318h, 0B13757D6h, 27A25232F
UPX1:0048BD98 dd 1ABF57C2h, 0DFDAD068h, 606A42C7F
UPX1:0048BD98 dd 0C69DFA4h, 0B6D0A6A0h, 0FF415756
```



```
C:\>Virus.exe aCdefghijklmnop
aCdef Vous avez gagné
Vous avez gagné
Vous avez gagné
Vous avez gagné
Vous avez gagné
Vous avez gagné
Vous avez gagné
Vous avez gagné
Vous avez gagné
Vous avez gagné
C:\>Virus.exe xxx
Vous avez gagné
Vous avez gagné
Vous avez gagné
C:\>
```

fausse_joie.png



