SIRAV :
Analyse de Malware

Clément BRUN, Zakaria LEZGHAM, Rémi SALAUN

Détection anti-débug *(401FD - 40111A)*

Le code suivant correspond à un anti-débug. En effet, la fonction *fs:[30]* fait appel au process environment block qui contient comme champ *BeingDebugged*.

Pour contourner l'anti-debug, il suffit de de modifier manuellement la valeur du flag *ZF* à 1, ce qui permet de faire le saut à l'adresse 401130. Autrement, l'argument entré au programme est affiché et le programme s'arrête.

```
.text:004010FD
.text:004010FD loc_4010FD:                             ; CODE XREF: _main+28↑j
.text:004010FD                  mov     eax, [edi+4]
.text:00401100                  mov     ebx, 20h
.text:00401105                  mov     ecx, 10h
.text:0040110A                  mov     edx, fs:[ebx+ecx]
.text:0040110E                  mov     [esp+148h+var_130], edx
.text:00401112                  mov     ecx, [esp+148h+var_130]
.text:00401116                  test    byte ptr [ecx+68h], 70h
.text:0040111A                  jz      short loc_401130
.text:0040111C                  push    eax
.text:0040111D                  push    offset aS       ; "%s"
.text:00401122                  call    ds:printf
.text:00401128                  add     esp, 8
.text:0040112B                  jmp     loc_401540
```

Proposition de code

```
PEB *ppeb;
__asm{
    mov ebx,0x20
    mov ecx,0x10
    mov edx,fs:[ebx+ecx]
    mov ppeb,edx
}
if(ppeb->BeingDebugged != 0){
    printf("%s",entreeUtilisateur);
    while(1);
}
```

Détection SHA256 *(401360 - 401398, 401466 - 40149E)*

Les constantes utilisées dans les codes qui suivent (présents dans deux fonctions différentes et successives) sont caractéristiques d'un chiffrement par SHA256.

Reste donc à savoir à quel endroit du code le chiffrement est utilisé.

```
.text:00401462          lea     esi, [esp+14Ch+var_128]
.text:00401466          mov     [esp+14Ch+var_120], 6A09E667h
.text:0040146E          mov     [esp+14Ch+var_11C], 0BB67AE85h
.text:00401476          mov     [esp+14Ch+var_118], 3C6EF372h
.text:0040147E          mov     [esp+14Ch+var_114], 0A54FF53Ah
.text:00401486          mov     [esp+14Ch+var_110], 510E527Fh
.text:0040148E          mov     [esp+14Ch+var_10C], 9B05688Ch
.text:00401496          mov     [esp+14Ch+var_108], 1F83D9ABh
.text:0040149E          mov     [esp+14Ch+var_104], 5BE0CD19h
.text:004014A6          mov     [esp+14Ch+var_128], ebx
.text:004014AA          mov     [esp+14Ch+var_124], ebx
.text:004014AE          mov     [esp+14Ch+var_C0], bl
.text:004014B5          call    sub_4019A0
```

```
.text:0040135C          lea     esi, [esp+14Ch+var_128]
.text:00401360          mov     [esp+14Ch+var_120], 6A09E667h
.text:00401368          mov     [esp+14Ch+var_11C], 0BB67AE85h
.text:00401370          mov     [esp+14Ch+var_118], 3C6EF372h
.text:00401378          mov     [esp+14Ch+var_114], 0A54FF53Ah
.text:00401380          mov     [esp+14Ch+var_110], 510E527Fh
.text:00401388          mov     [esp+14Ch+var_10C], 9B05688Ch
.text:00401390          mov     [esp+14Ch+var_108], 1F83D9ABh
.text:00401398          mov     [esp+14Ch+var_104], 5BE0CD19h
.text:004013A0          mov     [esp+14Ch+var_128], ebx
.text:004013A4          mov     [esp+14Ch+var_124], ebx
.text:004013A8          mov     [esp+14Ch+var_C0], bl
.text:004013AF          call    sub_4019A0
```

Obfuscation de la fonction printf *(401166 - 401178)*

Dans le code qui suit, le pointeur associé à la fonction *printf_s* a été remplacé par celui de la fonction printf. A partir de là, appeler la fonction *printf_s* revient à appeler la fonction *printf*.

```
.text:00401166                push    edx             ; lpflOldProtect
.text:00401167                push    40h             ; flNewProtect
.text:00401169                push    6               ; dwSize
.text:0040116B                push    esi             ; lpAddress
.text:0040116C                call    ds:VirtualProtect
.text:00401172                mov     dword ptr [esi], 0B056B468h
.text:00401178                mov     word ptr [esi+4], 0C378h
```

```
RAX 0000000000404030 ↳ .data:00404030
RBX 00000000FF006A50 ↳
RCX 0000000000000000 ↳
RDX 000000000012FE4C ↳ Stack[00000EA4]:000000000012FE4C
RSI 0000000078B057AE ↳ msvcr100.dll:msvcr100_printf_s
```

Le code de la fonction

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *p = (char *) printf_s;
    DWORD old;
    VirtualProtect(p,6,PAGE_EXECUTE_READWRITE,&old);
    p[0] = '\x68';
    p[1] = '\xB4';
    p[2] = '\x56';
    p[3] = '\xB0';
    p[4] = '\x78';
    p[5] = '\xC3';
    printf_s("Bravo !");
    while(1);
}
```

Fonctions de comparaison de chaînes de caractères

(sub_40100, sub_401050)

Ces deux fonctions assurent la comparaison des registres *ecx* et *esx*, sachant que *ecx* est une entrée et *edx* une chaîne de caractère propre à chaque fonction.

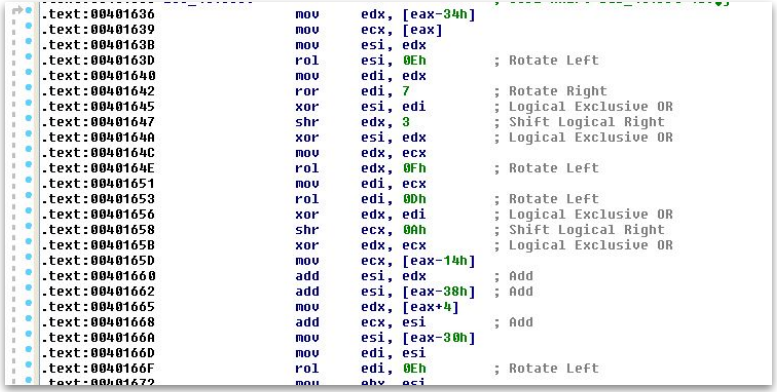
La première est appelée à l'adresse 401181 et la seconde est appelée à l'adresse 40125D. Ces deux fonctions renvoient que la comparaison n'est pas bonne

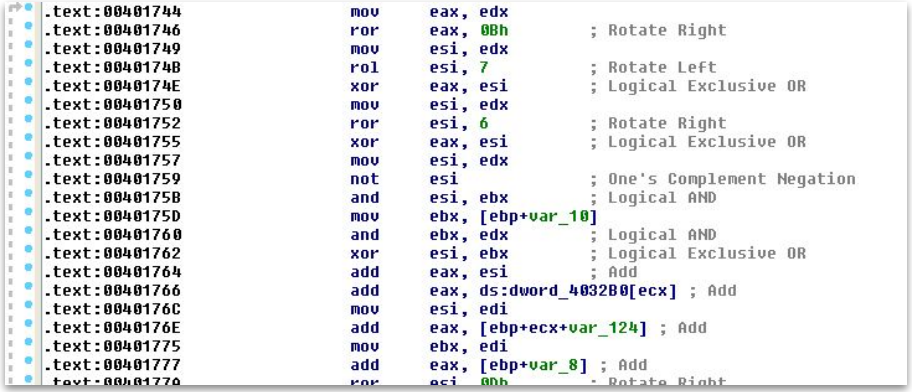
```
.text:00401000 sub_401000      proc near               ; CODE XREF: _main+E1↓p
.text:00401000                 xor     eax, eax
.text:00401002                 mov     edx, offset a6e756ff100dcbfb3d879b01fec7e715ac985772"...
.text:00401007                 push    ebx
.text:00401008
.text:00401008 loc_401008:                             ; CODE XREF: sub_401000+22↓j
.text:00401008                 mov     bl, [ecx]
.text:0040100A                 cmp     bl, [edx]
.text:0040100C                 jnz     short loc_401028
.text:0040100E                 test    bl, bl
.text:00401010                 jz      short loc_401024
.text:00401012                 mov     bl, [ecx+1]
.text:00401015                 cmp     bl, [edx+1]
.text:00401018                 jnz     short loc_401028
.text:0040101A                 add     ecx, 2
.text:0040101D                 add     edx, 2
.text:00401020                 test    bl, bl
.text:00401022                 jnz     short loc_401008
```

```
.text:00401050 sub_401050      proc near               ; CODE XREF: _main+1BD↓p
.text:00401050                 xor     eax, eax
.text:00401052                 mov     edx, offset a85f40efa926898 ; "85f40efa9268987839fa5ed422bde5b1fd3d73f"...
.text:00401057                 push    ebx
.text:00401058
.text:00401058 loc_401058:                             ; CODE XREF: sub_401050+22↓j
.text:00401058                 mov     bl, [ecx]
.text:0040105A                 cmp     bl, [edx]
.text:0040105C                 jnz     short loc_401078
.text:0040105E                 test    bl, bl
.text:00401060                 jz      short loc_401074
.text:00401062                 mov     bl, [ecx+1]
.text:00401065                 cmp     bl, [edx+1]
.text:00401068                 jnz     short loc_401078
.text:0040106A                 add     ecx, 2
.text:0040106D                 add     edx, 2
.text:00401070                 test    bl, bl
.text:00401072                 jnz     short loc_401058
```

XOR en boucles *(401636 - 401709, 401744 - 401959)*

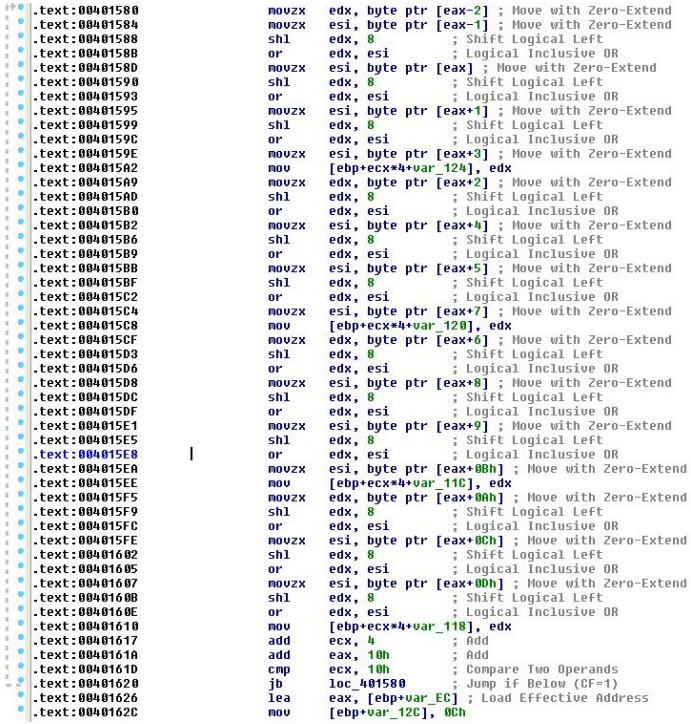
A l'aide de scripts python, des commandes *xor* ont été repérées à l'intérieur de boucles, témoignant peut-être de fonction de de hachage. Cependant, cette hypothèse n'a pas encore été confirmée.





Empilement des caractères - SHA 256 *(401580 - 40162C)*

Ce code permet visiblement de stocker les l'entrée utilisateur dans la pile en entrant octet par octet les caractères : un octet est entré, puis décalé vers les bits de poids fort afin de laisser de la place pour l'octet suivant, et ainsi de suite jusqu'à ce que toute la chaîne de caractères ait été déplacée.



Résumé du code

1. Entrée utilisateur
2. Comparaison avec une chaîne de caractères qui est forcément fausse
3. SHA-256 et comparaison avec une chaîne de caractère
4. Comparaison avec une chaîne de caractère forcément fausse
5. SHA-256 et comparaison avec une chaîne de caractères