Elooo #2

par mimamasse

Comme dans le premier exercice on repère l'appel à l'API qui va récupérer notre serial

```
004010A7  |. 6A 2D          PUSH 2D                                  ; /Count = 2D (45.)
004010A9  |. 68 00304000    PUSH elooo2.00403000                     ; |Buffer = elooo2.00403000
004010AE  |. 68 EB030000    PUSH 3EB                                 ; |ControlID = 3EB (1003.)
004010B3  |. FF75 08        PUSH [DWORD SS:EBP+8]                    ; |hWnd
004010B6  |. E8 85000000    CALL <JMP.&user32.GetDlgItemTextA>       ; \GetDlgItemTextA
004010BB  |. 33F6           XOR ESI,ESI
004010BD     8D05 00304000  LEA EAX,[DWORD DS:403000]
004010C3  |> 83FE 08        /CMP ESI,8
004010C6  |. 7C 02          |JL SHORT elooo2.004010CA
004010C8  |. 33F6           |XOR ESI,ESI
004010CA  |> 8A96 32304000  |MOV DL,[BYTE DS:ESI+403032]
004010D0  |. 3010           |XOR [BYTE DS:EAX],DL
004010D2  |. 46             |INC ESI
004010D3  |. 40             |INC EAX
004010D4  |. 8038 00        |CMP [BYTE DS:EAX],0
004010D7  |.^75 EA          \JNZ SHORT elooo2.004010C3
004010D9  |. 68 3B304000    PUSH elooo2.0040303B     ; /String2 = "cl?v!es7iUchieTOUJ&-XoU=&+ethl"
004010DE  |. 68 00304000    PUSH elooo2.00403000                     ; |String1 = ""
004010E3  |. E8 82000000    CALL <JMP.&kernel32.lstrcmpA>            ; \lstrcmpA
004010E8  |. 85C0           TEST EAX,EAX
004010EA  |. 75 14          JNZ SHORT elooo2.00401100
004010EC  |. 6A 00          PUSH 0                                   ; /Style = MB_OK|MB_APPLMODAL
004010EE  |. 68 5D304000    PUSH elooo2.0040305D                     ; |Title = "... "
004010F3  |. 68 5A304000    PUSH elooo2.0040305A                     ; |Text = "OK"
004010F8  |. FF75 08        PUSH [DWORD SS:EBP+8]                    ; |hOwner
004010FB  |. E8 4C000000    CALL <JMP.&user32.MessageBoxA>           ; \MessageBoxA
```

On voit une boucle (0x004010C3 à 0x004010D7) qui XOR chaque caractère du serial (adresse initiale 0x403000) par la valeur pointée par ESI + 0x403032. Un test fait que la valeur de ESI va rester dans la plage [0,7], ce qui implique qu'une clé de chiffrement de 8 caractères sert à chiffrer notre serial pour le comparer à la chaîne à l'adresse 0x0040303B

L'opération xor étant réversible nous allons utiliser cette propriété pour trouver le bon serial à partir de la chaîne servant à la comparaison. Il nous suffira juste de remplacer l'adresse de la chaîne qui se fait « xoriser » (notre serial) par celle de la chaîne déjà « xorisée » (la chaîne de référence).

Le patch se fait à l'entrée de la boucle

```
004010BD     8D05 00304000  LEA EAX,[DWORD DS:40303B]
```

Voilà, le crackme génère le serial que nous devons trouver.

Réalisation d'un keygen (optionnel).

Il serait ridicule de faire un keygen en utilisant une boucle qui xor une chaîne, on va faire mieux, plus compliqué et surtout totalement inadapté pour ce travail :) : le keygen MMX. On est pas là pour rigoler !

Vous avez sûrement entendu parler du mmx, l'extension multimédia qui rend Internet plus rapide et plus beau. Enfin ça ce sont les gars d'Intel, ceux qui dansent sur du funk en tenue de cosmonaute (astronaute ou taïkonaute suivant où l'on se trouve), qui le disent. Nous on va « xorer » une chaîne de caractères avec ça.

MMX est capable de faire plusieurs opérations à la fois. Ses registres sont ceux de la FPU (64 bits

ou 8 octets comme notre clé. Le hasard fait bien les choses :)) mais qui travaillent dans un autre mode.

Je ne vais pas tout expliquer sur le MMX mais le résultat des opérations peut être :
- saturé - si le résultat d'une opération est supérieur à la limite définie par le mode, le registre contient cette limite.
- Wraparound (rebouclé ?) – si la valeur du résultat dépasse la limite définie par le mode, le registre contient comme valeur, la limite opposé + le dépassement de la limite.

Nous allons travailler avec la saturation des registres car ça va nous simplifier la vie.

Étape 1 : calcul de la longueur de la chaîne.

Cette longueur est connue mais nous allons quand même voir comment déterminer facilement une longueur de chaîne (plus exactement l'adresse de son zéro terminal).

```
or cl, 0xFF           ; pour la répétion avec rep. CL, le nombre d'itérations, est mis à 256.
mov al, 0             ; AL = code ASCII à trouver
mov edi, chaîne       ; EDI= adresse de la chaîne
repne scasb           ; On cherche la valeur 0 à l'adresse pointé par EDI. À chaque itération EDI
                      ; est incrémenté et ECX décrémenté. On termine si [EDI]=0 ou ECX = 0.
```

Étape 2 : « Xorisation » de la chaîne.

```
mov eax, chaîne       ; EAX = adresse chaîne.
movq mm1, qword[mask] ; Registre mm1 (64 bits) contient le masque (clé du xor).
@@:                   ; Début boucle
movq mm0, qword[eax]  ; On met les octets [EAX] à [EAX+7] dans mm0.
pxor mm0, mm1         ; On xor mm1 et mm0.
movq mm2, mm0         ; On met mm1 « xoré » dans mm2.
pcmpeqb mm2, mm1      ; On compare les octets de clé et  les octets « xorés » pour savoir si il y
                      ; avait des 0 dans mm0. (si les octets comparés sont égaux, l'octet de mm2
                      ; est mis à 0. Et dans le cas contraire, mis à 0xFF).
psubusb mm0, mm2      ; On soustrait avec saturation chaque octets de mm0 par l'octet
                      ; correspondant de mm2 (le résultat de la comparaison et la clé). Si la
                      ; comparaison précédente à mis l'octet à 0xFF, l'octet résultant de la
                      ; soustraction sera bloqué à la limite basse (0).
movq qword[eax], mm0  ; On écrit les caractères « xorés » et nettoyés des opérations parasites
                      ; à l'adresse pointée par EAX.
add eax, 8            ; 0n incrémente EAX de 8.
cmp eax, edi          ; Si EAX < EDI (fin de chaîne) alors
jl @b                 ; on reboucle
emms                  ; On « nettoie » les registres mmx
```

Si vous n'avez pas tout compris, faîtes le avec un xor tout simple, sur des bytes, words ou doublewords (8, 16 ou 32 bits). Personne ne vous en voudra :)