

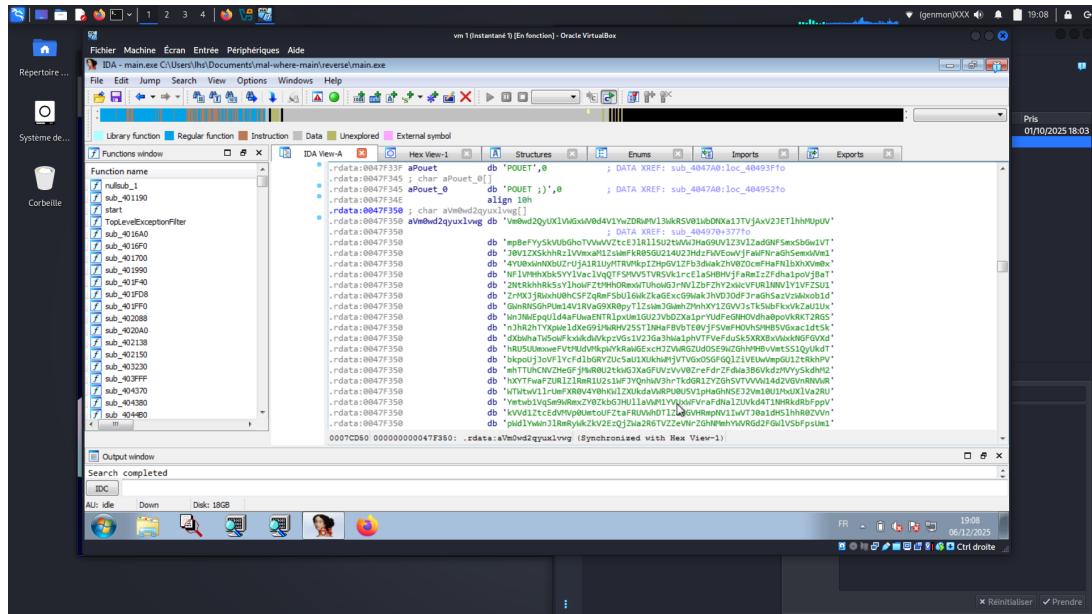
## Très longue chaîne encodée

**Localisation:** 0047F350 **Fonction:** const  
**Type:** Encodage **Sévérité:** Faible

## Analyse

Cette longue chaîne de caractère semble être une chaîne encodée en `base64`. Après avoir mis cette chaîne dans CyberChef celui-ci ne parvenait pas à décoder la chaîne. Nous avons alors construit un petit script Python. Après une première itération, la chaîne semblait toujours être encodé en `base64`. Nous avons alors modifié le script pour y ajouter une analyse d'entropie. Après une trentaine d'itération nous sommes parvenu à décoder la chaîne : **Hop là, on n'oublie pas de s'amuser**

## Screenshot



## Comportement malveillant en fonction de la chaîne d'entrée

**Localisation:** 0x00401BAF, 0x004019EA    **Fonction:** ScreenMelter, loc\_401C94, loc\_401B4F  
**Type:** Malware    **Sévérité:** Moyenne

### Code Assembleur : Extrait

Test 1 (ligne 24)

```
1 cmp      [ebp+var_C4], 2
2 mov      [ebp+nHeight], offset unk_47DFC8
3 jz       loc_401B4F
```

Test 2 (ligne 141)

```
1 cmp      eax, 8
2 mov      [ebp+var_B0], eax
3 jbe     loc_401C94
```

### Analyse

Le *Test 1* vérifie le nombre d'arguments passés en paramètre à l'exécutable. Si celui est égal à 2 le programme exécute la fonction loc\_401B4F

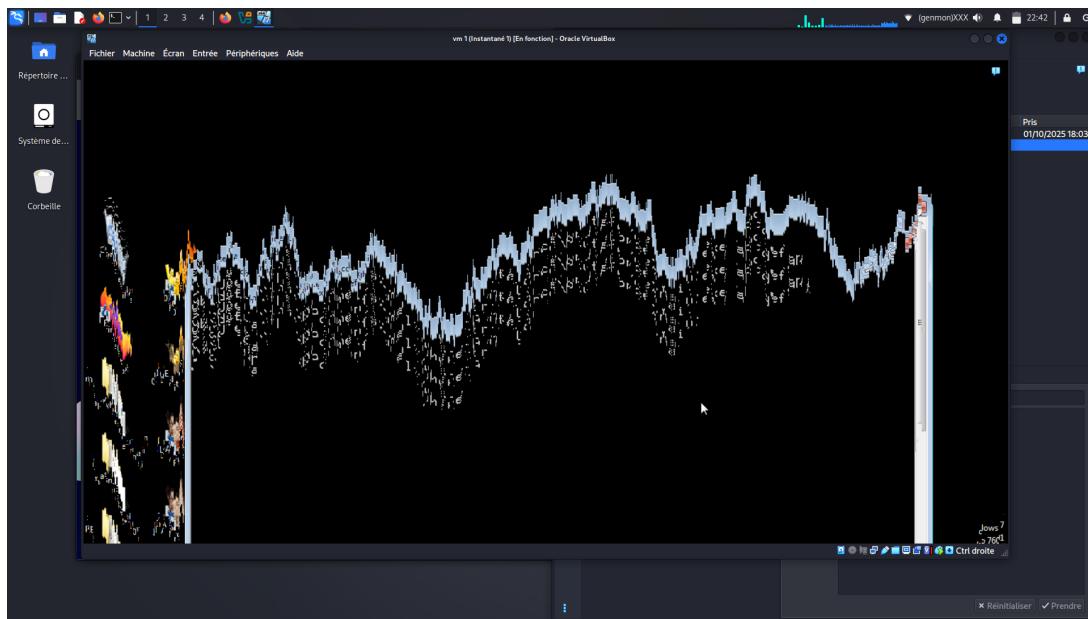
Le *Test 2* vérifie quant à lui si la chaîne passée en entrée à une longueur supérieure à 8. Si tel est le cas alors le programme exécute la fonction loc\_401C94.

Le code assembleur, si dessous, a le comportement suivant :

- Si le programme est exécuté sans argument, il adopte un comportement malveillant : il cache les fenêtres en boucle à chaque clic, provoquant un déni de service sur la machine.
- Si le programme est exécuté avec un argument, il analyse celui-ci et, en fonction de sa longueur, adopte un comportement spécifique :
  - Longueur inférieure égale à 8 : comportement normal
  - Longueur supérieur à 8 : Effet visuel sur les fenêtres (comportement malveillant)

Dans tous les cas de figure, le fichier `astiko.txt` est créé (voir *astiko.pdf*).

## Screenshot



## Code Assembleur : Entier

```
1      lea      ecx, [esp+4]
2      and      esp, 0FFFFFFF0h
3      push     dword ptr [ecx-4]
4      push     ebp
5      mov      ebp, esp
6      push     edi
7      push     ecx
8      sub      esp, 100h
9      mov      eax, [ecx]
10     mov      ecx, [ecx+4]
11     lea      edx, [ebp+var_9+1]
12     mov      [ebp+var_7C], edx
13     mov      [ebp+var_84], offset sub_474F60
14     mov      [ebp+var_C4], eax
15     lea      eax, [ebp+var_9C]
16     mov      [ebp+var_C8], ecx
17     mov      [esp+108h+var_108], eax
18     mov      [ebp+var_80], offset dword_4755C4
19     mov      [ebp+var_78], offset loc_401A96
20     mov      [ebp+var_74], esp
21     call    _Unwind_Sjlj_Register
22     call    sub_40A780
23     cmp      [ebp+var_C4], 2
24     mov      [ebp+nHeight], offset unk_47DFC8
25     jz      loc_401B4F
26     mov      eax, ds:GetForegroundWindow
27     mov      edx, ds>ShowWindow
28     mov      [ebp+var_A0], eax
29     mov      [ebp+var_A4], eax
30     mov      [ebp+var_AC], edx
31     nop
32     lea      esi, [esi+0]
```

```

34 loc_401A20: ; CODE XREF: sub_401990+104j
35     mov    [ebp+var_98], 6
36     call   [ebp+var_A0]
37     test  eax, eax
38     mov    [ebp+var_A8], eax
39     jz    short loc_401A4E
40     mov    [esp+2Ch+Msg.pt.x], 0 ; nCmdShow
41     mov    [esp+2Ch+Msg.time], eax ; hWnd
42     call   ds>ShowWindow
43     sub    esp, 8
44
45 loc_401A4E: ; CODE XREF: sub_401990+A8j
46     mov    [ebp+var_98], 6
47     call   [ebp+var_A4]
48     cmp   [ebp+var_A8], eax
49     jz    short loc_401A7A
50     mov    [esp+2Ch+Msg.pt.x], 5
51     mov    [esp+2Ch+Msg.time], eax
52     call   [ebp+var_AC]
53     sub    esp, 8
54
55 loc_401A7A: ; CODE XREF: sub_401990+D4j
56     mov    [esp+2Ch+Msg.time], 3E8h ; dwMilliseconds
57     mov    [ebp+var_98], 6
58     call   ds>Sleep
59     sub    esp, 4
60     jmp   short loc_401A20
61
62 ;
-----+
63
64 loc_401A96: ; DATA XREF: sub_401990+460
65     add   ebp, 8
66     mov    eax, [ebp+var_98]
67     mov    edx, [ebp+var_94]
68     cmp   eax, 1
69     mov    [ebp+var_CC], edx
70     jz    loc_401B38
71     cmp   eax, 2
72     jz    short loc_401B21
73     cmp   eax, 3
74     jz    short loc_401B0A
75     cmp   eax, 4
76     jz    short loc_401ADD
77     cmp   eax, 5
78     jz    short loc_401ADD
79     lea    eax, [ebp+File]
80     mov    [esp+4+hInstance], eax
81     mov    [ebp+var_98], 0
82     call  sub_448860
83
84 loc_401ADD: ; CODE XREF: sub_401990+131j
85 ; sub_401990+136j ...
86     lea    eax, [ebp+nHeight]
87     mov    [esp+4+hInstance], eax
88     mov    [ebp+var_98], 0
89     call  sub_448860

```

```
90          mov      edi, [ebp+var_CC]
91          mov      [ebp+var_98], 0FFFFFFFh
92          mov      [esp+4+hInstance], edi
93          call    _Unwind_SjLj_Resume
94
95 loc_401B0A:           ; CODE XREF: sub_401990+12Cj
96          lea      eax, [ebp+nWidth]
97          mov      [esp+4+hInstance], eax
98          mov      [ebp+var_98], 0
99          call    sub_448860
100         jmp     short loc_401ADD
101 ;
102
103 loc_401B21:           ; CODE XREF: sub_401990+127j
104         lea      eax, [ebp+Y]
105         mov      [esp+4+hInstance], eax
106         mov      [ebp+var_98], 0
107         call    sub_448860
108         jmp     short loc_401ADD
109 ;
110
111 loc_401B38:           ; CODE XREF: sub_401990+11Ej
112         lea      eax, [ebp+X]
113         mov      [esp+4+hInstance], eax
114         mov      [ebp+var_98], 0
115         call    sub_448860
116         jmp     short loc_401ADD
117 ;
118
119 loc_401B4F:           ; CODE XREF: sub_401990+68j
120         mov      edi, [ebp+var_C8]
121         lea      eax, [ebp+var_9]
122         mov      [esp+2Ch+Msg.pt.y], eax ; int
123         mov      eax, [edi+4]
124         mov      [ebp+var_98], 5
125         mov      [esp+2Ch+Msg.pt.x], eax ; char *
126         lea      eax, [ebp+nWidth]
127         mov      [esp+2Ch+Msg.time], eax ; int
128         call    sub_448120
129         lea      eax, [ebp+nWidth]
130         mov      [esp+2Ch+Msg.pt.x], eax
131         lea      eax, [ebp+nHeight]
132         mov      [esp+2Ch+Msg.time], eax
133         mov      [ebp+var_98], 4
134         call    sub_446F80
135         lea      eax, [ebp+nWidth]
136         mov      [esp+2Ch+Msg.time], eax
137         mov      [ebp+var_98], 5
138         call    sub_448860
139         mov      eax, [ebp+nHeight]
140         mov      eax, [eax-0Ch]
141         cmp     eax, 8
```

```
142         mov     [ebp+var_B0], eax
143         jbe     loc_401C94
144         mov     [esp+2Ch+Msg.time], 0 ; nIndex
145         mov     [ebp+var_98], 6
146         call    ds:GetSystemMetrics
147         push   edx
148         mov     ds:nWidth, eax
149         mov     [esp+2Ch+Msg.time], 1 ; nIndex
150         call    ds:GetSystemMetrics
151         push   edi
152         mov     ds:nHeight, eax
153         mov     [esp+2Ch+Msg.time], 0 ; lpModuleName
154         call    ds:GetModuleHandleA
155         lea     edx, [ebp+WndClass]
156         mov     [ebp+var_C0], eax
157         mov     edi, edx
158         xor     eax, eax
159         push   ecx
160         mov     ecx, 0Ah
161         rep    stosd
162         mov     eax, [ebp+var_C0]
163         mov     [ebp+WndClass.lpfnWndProc], offset sub_401700
164         mov     [ebp+WndClass.lpszClassName], offset ClassName
165         ; "ScreenMelter"
166         mov     [esp+2Ch+Msg.pt.x], 7F00h ; lpCursorName
167         mov     [ebp+WndClass.hInstance], eax
168         mov     [esp+2Ch+Msg.time], 0 ; hInstance
169         call    ds:LoadCursorA
170         push   edx
171         push   edx
172         mov     [ebp+WndClass.hCursor], eax
173         lea     eax, [ebp+WndClass]
174         mov     [esp+2Ch+Msg.time], eax ; lpWndClass
175         call    ds:RegisterClassA
176         test   ax, ax
177         push   edi
178         jnz    loc_401E39
179
180 loc_401C57:           ; CODE XREF: sub_401990+517j
181                   ; sub_401990+53Dj ...
182         mov     [ebp+var_BC], 1
183
184 loc_401C61:           ; CODE XREF: sub_401990+4A4j
185         lea     eax, [ebp+nHeight]
186         mov     [esp+2Ch+Msg.time], eax
187         mov     [ebp+var_98], 0FFFFFFFh
188         call    sub_448860
189         lea     eax, [ebp+var_9C]
190         mov     [esp+2Ch+Msg.time], eax
191         call    _Unwind_Sjlj_Unregister
192         mov     eax, [ebp+var_BC]
193         lea     esp, [ebp-8]
194         pop    ecx
195         pop    edi
196         pop    ebp
197         lea     esp, [ecx-4]
         retn
```

```
198 ;-----  
199  
200 loc_401C94: ; CODE XREF: sub_401990+228j  
201     lea    eax, [ebp+nHeight]  
202     mov    [esp+2Ch+Msg.pt.y], 0 ; char  
203     mov    [esp+2Ch+Msg.pt.x], 8 ; size_t  
204     mov    [esp+2Ch+Msg.time], eax ; int  
205     mov    [ebp+var_98], 6  
206     call   sub_4475F0  
207     mov    eax, [ebp+var_B0]  
208     mov    [ebp+var_B4], eax  
209     jmp    short loc_401CFB  
210 ;-----  
211  
212 loc_401CC7: ; CODE XREF: sub_401990+372j  
213     mov    eax, [ebp+nHeight]  
214     cmp    dword ptr [eax-4], 0  
215     js    short loc_401CE8  
216     lea    edx, [ebp+nHeight]  
217     mov    [esp+2Ch+Msg.time], edx  
218     mov    [ebp+var_98], 6  
219     call   sub_445EE0  
220     mov    eax, [ebp+nHeight]  
221  
222 loc_401CE8: ; CODE XREF: sub_401990+33Ej  
223     mov    edi, [ebp+var_B4]  
224     mov    byte ptr [eax+edi], 0  
225     add    edi, 1  
226     mov    [ebp+var_B4], edi  
227  
228 loc_401CFB: ; CODE XREF: sub_401990+335j  
229     cmp    [ebp+var_B4], 8  
230     jnz    short loc_401CC7  
231     mov    [esp+2Ch+Msg.time], 7D0h ; dwMilliseconds  
232     mov    [ebp+var_98], 6  
233     call   ds:Sleep  
234     push   eax  
235     mov    [esp+2Ch+Msg.pt.x], offset Mode ; "w"  
236     mov    [esp+2Ch+Msg.time], offset Filename ; "C:\\  
          Users\\lhs\\AppData\\Local\\Temp\\a"...  
237     call   fopen  
238     test   eax, eax  
239     mov    [ebp+var_B8], eax  
240     jz    loc_401F2C  
241     mov    edi, [ebp+var_B8]  
242     mov    [esp+2Ch+Msg.pt.y], 13h ; Count  
243     mov    [esp+2Ch+Msg.pt.x], 1 ; Size  
244     mov    [esp+2Ch+Msg.time], offset aPakboEtLombrik ; "  
          pakbo-et-lombrik.fr"  
245     mov    [esp+2Ch+File], edi ; File  
246     mov    [ebp+var_98], 6  
247     call   fwrite  
248     mov    [esp+2Ch+Msg.time], edi ; File  
249     call   fclose
```

```

250          lea      eax, [ebp+nHeight]
251          mov      [esp+2Ch+Msg.pt.x], eax
252          lea      eax, [ebp+Y]
253          mov      [esp+2Ch+Msg.time], eax
254          call    sub_4481C0
255          lea      eax, [ebp+Y]
256          mov      [esp+2Ch+Msg.time], eax
257          mov      [ebp+var_98], 3
258          call    sub_403230
259          lea      eax, [ebp+Y]
260          mov      [esp+2Ch+Msg.time], eax
261          mov      [ebp+var_98], 6
262          call    sub_448860
263          lea      eax, [ebp+nHeight]
264          mov      [esp+2Ch+Msg.pt.x], eax
265          lea      eax, [ebp+X]
266          mov      [esp+2Ch+Msg.time], eax
267          call    sub_4481C0
268          lea      eax, [ebp+X]
269          mov      [esp+2Ch+Msg.time], eax
270          mov      [ebp+var_98], 2
271          call    sub_404380
272          lea      eax, [ebp+X]
273          mov      [esp+2Ch+Msg.time], eax
274          mov      [ebp+var_98], 6
275          call    sub_448860
276          lea      eax, [ebp+nHeight]
277          mov      [esp+2Ch+Msg.pt.x], eax
278          lea      eax, [ebp+File]
279          mov      [esp+2Ch+Msg.time], eax
280          call    sub_4481C0
281          lea      eax, [ebp+File]
282          mov      [esp+2Ch+Msg.time], eax
283          mov      [ebp+var_98], 1
284          call    sub_404970
285          lea      eax, [ebp+File]
286          mov      [esp+2Ch+Msg.time], eax
287          mov      [ebp+var_98], 6
288          call    sub_448860
289          mov      [ebp+var_BC], 0
290          jmp     loc_401C61
291          ;
-----  

292
293 loc_401E39:           ; CODE XREF: sub_401990+2C1j
294          mov      eax, ds:nHeight
295          mov      edx, [ebp+var_C0]
296          mov      [esp+2Ch+lpParam], 0 ; lpParam
297          mov      [esp+2Ch+var_9+1], 0 ; hMenu
298          mov      [esp+2Ch+nHeight], eax ; nHeight
299          mov      eax, ds:nWidth
300          mov      [esp+2Ch+hInstance], edx ; hInstance
301          mov      dword ptr [esp+20h], 0 ; hWndParent
302          mov      [esp+2Ch+Y], 0 ; Y
303          mov      [esp+2Ch+nWidth], eax ; nWidth
304          mov      [esp+2Ch+X], 0 ; X
305          mov      [esp+2Ch+File], 80000000h ; dwStyle

```

```

306         mov      [esp+2Ch+Msg.pt.y], 0 ; lpWindowName
307         mov      [esp+2Ch+Msg.pt.x], offset ClassName ; "
308             ScreenMelter"
309         mov      [esp+2Ch+Msg.time], 8 ; dwExStyle
310         call     ds>CreateWindowExA
311         sub     esp, 30h
312         test    eax, eax
313         jz      loc_401C57
314         call     ds:GetTickCount
315         mov      [esp+2Ch+Msg.time], eax ; Seed
316         call     srand
317         lea     edx, [ebp+Msg]
318         mov      ecx, 7
319         xor     eax, eax
320         mov      edi, edx
321         rep stosd

322 loc_401EC9:                      ; CODE XREF: sub_401990+57Ej
323                                     ; sub_401990+59Aj
324         cmp     [ebp+Msg.message], 12h
325         jz      loc_401C57
326         lea     eax, [ebp+Msg]
327         mov     [esp+2Ch+X], 1 ; wRemoveMsg
328         mov     [esp+2Ch+File], 0 ; wMsgFilterMax
329         mov     [esp+2Ch+Msg.pt.y], 0 ; wMsgFilterMin
330         mov     [esp+2Ch+Msg.pt.x], 0 ; hWnd
331         mov     [esp+2Ch+Msg.time], eax ; lpMsg
332         mov     [ebp+var_98], 6
333         call    ds:PeekMessageA
334         sub     esp, 14h
335         test    eax, eax
336         jz      short loc_401EC9
337         lea     edx, [ebp+Msg]
338         mov     [esp+2Ch+Msg.time], edx ; lpMsg
339         call    ds:TranslateMessage
340         lea     edi, [ebp+Msg]
341         push   ecx
342         mov     [esp+2Ch+Msg.time], edi ; lpMsg
343         call    ds:DispatchMessageA
344         push   edx
345         jmp     short loc_401EC9
346     ;

347 loc_401F2C:                      ; CODE XREF: sub_401990+3A8j
348         mov     [esp+2Ch+Msg.time], offset ErrMsg ; "[LOG] [
349             main] fopen"
350         call    perror
351         jmp     loc_401C57
352 sub_401990 endp

```

## Ecriture dans un fichier

**Localisation:** 0x00401D1C    **Fonction:** ?

**Type:** Ecriture    **Sévérité:** Faible

### Code Assembleur

```

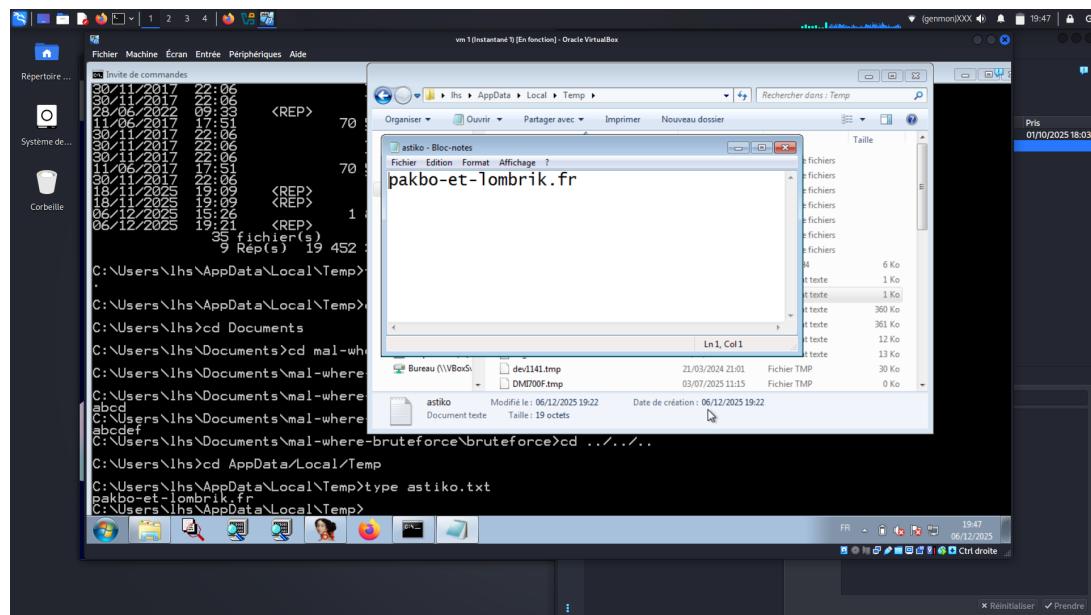
1 mov      [esp+2Ch+Msg.pt.x], offset Mode ; "w"
2 mov      [esp+2Ch+Msg.time], offset Filename ; "C:\\\\Users\\\\lhs\\\\AppData
   \\Local\\\\Temp\\\\a"...
3 call     fopen
4 test    eax, eax
5 mov      [ebp+var_B8], eax
6 jz       loc_401F2C
7 mov      edi, [ebp+var_B8]
8 mov      [esp+2Ch+Msg.pt.y], 13h ; Count
9 mov      [esp+2Ch+Msg.pt.x], 1 ; Size
10 mov     [esp+2Ch+Msg.time], offset aPakboEtLombrik ; "pakbo - et -
   lombrik.fr"
11 mov     [esp+2Ch+File], edi ; File
12 mov     [ebp+var_98], 6
13 call    fwrite
14 mov     [esp+2Ch+Msg.time], edi ; File
15 call    fclose

```

### Analyse

Ce code ouvre un fichier dans le dossier temporaire `lhs\AppData\Local\Temp\astiko.txt` de l'utilisateur et y écrit la chaîne "`pakbo-et-lombrik.fr`". Si le dossier n'existe pas celui-ci est créé avant l'écriture.

### Screenshot

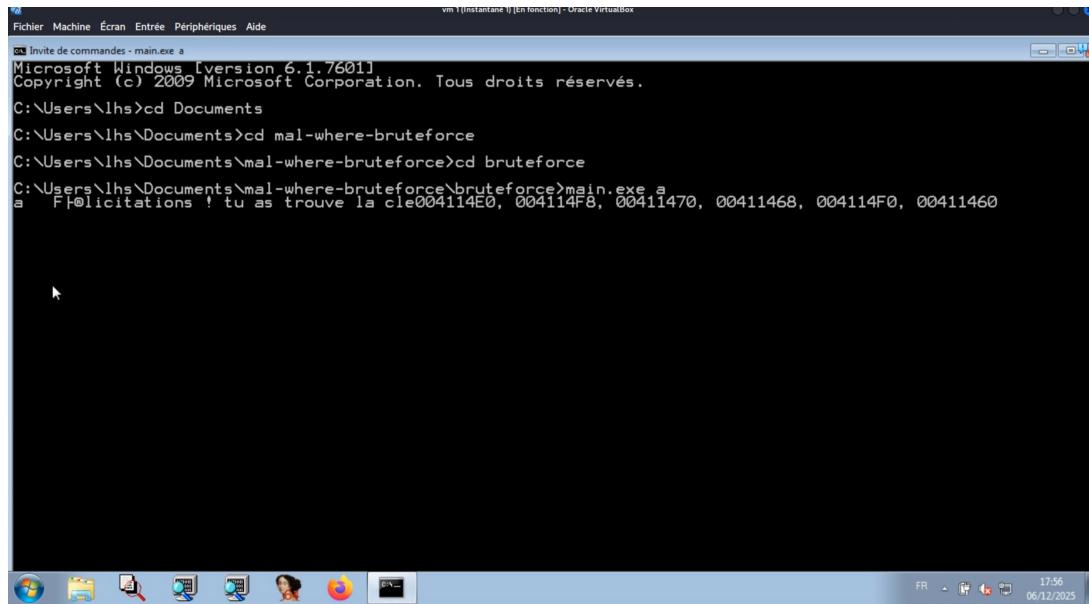


## Affichage furtif et fuite d'adresse mémoire

Localisation: 0x004052BC, 0x0040530C Fonction: ?

Type: ? Sévérité: Moyenne

### Screenshot



### Analyse

En exécutant le fichier `main.exe` de manière légitime (en respectant les contraintes de taille, avec la chaîne `a` par exemple), le programme affiche pendant quelques secondes la chaîne de caractère suivante : `a Félicitations ! tu as trouve la cle04114E0, 004114F8, 00411470, 00411468, 004114F0, 00411460.` En cherchant dans le résultat de `strings main.exe` on trouve la chaîne partielle `licitations !` et la chaîne `%p, %p, %p, %p, %p, %p,` ainsi en cherchant dans les données brutes avec IDA on peut remonter au code assembleur suivant :

### Code Assembleur : Fuite d'adresse mémoire

```

1  mov      [esp+518h+hWnd], eax
2  mov      [esp+518h+lpRect], edx
3  call    __moddi3
4  mov      [ebp+var_54], eax
5  mov      [esp+518h+x1], offset strlen
6  mov      [esp+518h+hdcsrc], offset free
7  mov      [esp+518h+cy], offset malloc
8  mov      [esp+518h+wDest], offset memcpy
9  mov      [esp+518h+y], offset loc_4114F8
10  mov     [esp+518h+lpRect], offset printf
11  mov     [esp+518h+hWnd], offset aPPPPP ; "%p, %p, %p, %p, %p"
12  mov     [ebp+var_44C], 4
13  call    printf

```

## Analyse

Ce code appelle la fonction `printf` et affiche les adresses mémoires des fonctions suivantes :

- `strlen`
- `free`
- `malloc`
- `memcpy`
- `loc_4114F8`
- `printf`

## Détection Manuelle du Débogueur (PEB)

**Localisation:** 0x004044DF    **Fonction:** sub\\_\\_4044B0

**Type:** Anti-debug / Evasion    **Sévérité:** High

### Code Assembleur

```

1 ; Acces manuel au TEB puis au PEB pour verifier le flag BeingDebugged
2 mov    ebx, 28h
3 mov    ecx, 2
4 mov    eax, fs:[ebx+ecx*4]      ; EAX = Adresse du PEB (FS:[30h])
5 mov    edx, eax
6 cmp    byte ptr [edx+2], 0     ; Verifie l'offset +2 (BeingDebugged)
7 jnz    loc_4045F0              ; Saute vers la sortie d'echech si
                                 detecte

```

### Code Décompilé (Reconstitution)

```

1 // Verification combinee API et acces direct memoire
2 CheckRemoteDebuggerPresent(GetCurrentProcess(), &pbDebuggerPresent);
3
4 // Lecture directe du segment FS
5 PEB* peb = (PEB*)__readfsdword(0x30);
6 if (peb->BeingDebugged == 1) {
7     return 0; // Echec silencieux
8 }

```

### Analyse

Le binaire utilise une technique d'anti-débogage classique mais obfuscée par des calculs d'index sur le segment FS. Il récupère l'adresse du **Process Environment Block (PEB)** à l'adresse FS:[0x30] et vérifie le champ **BeingDebugged** (offset 0x02). Si ce champ est à 1 (présence d'un débogueur), le programme branche vers loc\_4045F0, qui nettoie la pile et retourne 0, empêchant l'exécution de la logique de décodage.

## Détection de Débogueur (Anti-Debug)

Localisation: 0x004049C7 Fonction: ?

Type: Evasion Sévérité: Moyen

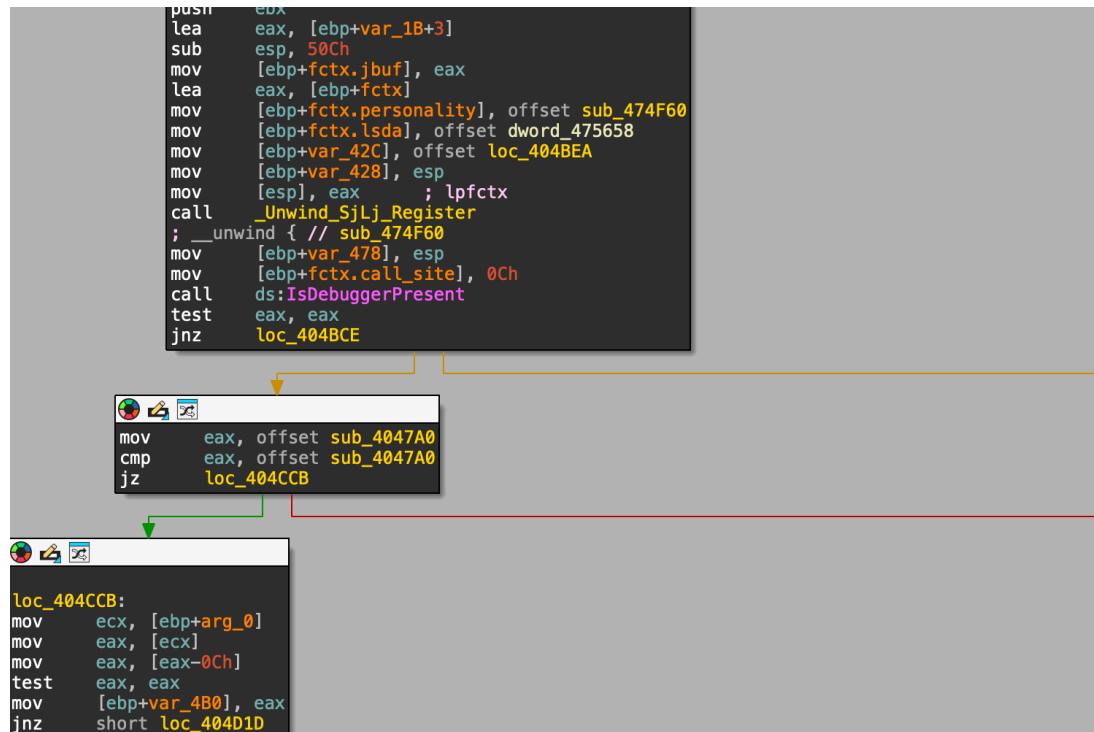
### Code Assembleur

```

1 ; Appel de l'API pour vérifier si un débogueur est attaché
2 call    ds:IsDebuggerPresent
3 test    eax, eax           ; Vérifie le résultat (1 = Debuggué, 0 = Normal)
4 jnz     loc_404BCE        ; Si EAX != 0, saut vers la sortie (Evasion)
5
6 ; ... (Le payload s'exécuterait ici si EAX == 0) ...
7
8 loc_404BCE:             ; Bloc de sortie prématurée
9 mov     esp, [ebp+var_478]
10 lea    eax, [ebp+var_450]
11 mov    [esp+518h+hWnd], eax
12 call   _Unwind_SjLj_Unregister ; Nettoyage contextuel
13 lea    esp, [ebp-0Ch]
14 pop    ebx
15 pop    esi
16 pop    edi
17 pop    ebp
18 retn               ; Fin de la fonction sans exécuter le payload

```

### Screenshot



## Analyse

Le programme implémente une technique d'évasion basique via `IsDebuggerPresent`. Il vérifie le drapeau `BeingDebugged` dans le PEB. Si un analyste est détecté (le registre `EAX` n'est pas nul), l'instruction `jnz` redirige le flux d'exécution vers `loc_404BCE`.

Ce bloc de code restaure simplement la pile et les registres avant de quitter la fonction via `retn`. Cela empêche l'exécution du code malveillant (manipulations graphiques GDI) situé plus bas dans la fonction, rendant l'analyse dynamique silencieuse si la protection n'est pas contournée.

## Opaque Predicate (Saut Impossible)

**Localisation:** 0x00404540    **Fonction:** sub\_4044B0

**Type:** Obfuscation    **Sévérité:** Medium

### Code Assembleur

```
1           ; Calcul préalable: 0x2A (42) * 0x11 (17) = 0x2CA (714)
2 mov      eax, [esp+6Ch+var_30] ; Charge 714
3 test     eax, eax           ; Teste les flags sur un nombre positif
4 js       loc_4046D8          ; "Jump if Sign" (ne saute jamais car
                           positif)
```

### Analyse

Une structure de contrôle trompeuse (Opaque Predicate) est utilisée pour masquer le chemin vers la routine de succès. Le programme effectue une multiplication dont le résultat (714) est toujours positif. L'instruction `js` (Jump if Sign) teste si le résultat est négatif. Dans un flux normal, ce saut n'est jamais pris, cachant ainsi le bloc de code situé en `loc_4046D8` aux outils d'analyse statique. Ce bloc caché contient l'appel à `puts` permettant d'afficher la chaîne secrète.

## Chaîne Chiffrée et Sortie

**Localisation:** 0x004046D8    **Fonction:** loc\_4046D8

**Type:** Data Hiding    **Sévérité:** Faible

### Code Assembleur

```
1 ; Bloc atteint uniquement si l'Opaque Predicate est patché
2 mov      [esp+6Ch+hProcess], offset off_47F307 ; Chaîne Base64
3 call     ds:puts                                ; Affiche le résultat
```

### Analyse

La chaîne de caractères située à l'offset 0x47F307 contient une donnée encodée en Base64 : "V1XTXhXWGhhU0ZaV1lsaFNWW1ZxUmt0WGJGcDBU". L'analyse dynamique a révélé que cette chaîne n'est pas déchiffrée par la fonction `puts` standard, mais dépend probablement des constantes calculées précédemment (0x2A, 0x11) pour un déchiffrement XOR manuel ou une validation via l'algorithme récursif identifié plus loin. En modifiant le .exe nous n'avons pas réussi à obtenir le déchiffrement de la chaîne.

## DeadCode : Petit Prince

**Localisation:** 0x004053D5    **Fonction:** ?

**Type:** Deadcode    **Sévérité:** Moyenne

```

1 .text:00405341          mov     [ebp+var_40], 0Ah
2 .text:00405378          mov     [ebp+var_44], 5

```

```

1 .text:004053D5 loc_4053D5:
2 .text:004053D5          mov     edx, [ebp+var_40]
3 .text:004053D8          mov     eax, [ebp+var_44]
4 .text:004053DB          cmp     edx, eax
5 .text:004053DD          js    loc_405708
6 .text:004053E3          mov     edx, [ebp+var_40]
7 .text:004053E6          mov     eax, [ebp+var_44]
8 .text:004053E9          movsx  ebx, [ebp+var_4A9]
9 .text:004053F0          imul   eax, edx
10 .text:004053F3          lea    eax, [ebx+eax]
11 .text:004053F6          mov     [ebp+var_54], eax
12 .text:004053F9          mov     eax, [ebp+var_54]
13 .text:004053FC          nop
14 .text:004053FD          nop
15 .text:004053FE          nop
16 .text:004053FF          nop
17 .text:00405400          nop
18 .text:00405401          nop
19 .text:00405402          nop

```

```

1 .text:00405708 loc_405708:
2 .text:00405708          mov     [esp+518h+hWnd], offset
   aLePetitPrinceR ; "Le Petit Prince raconte l"
3 .text:0040570F          mov     [ebp+var_44C], 4
4 .text:00405719          call   puts
5 .text:0040571E          movsx  ebx, [ebp+var_4A9]
6 .text:00405725          jmp    loc_405435

```

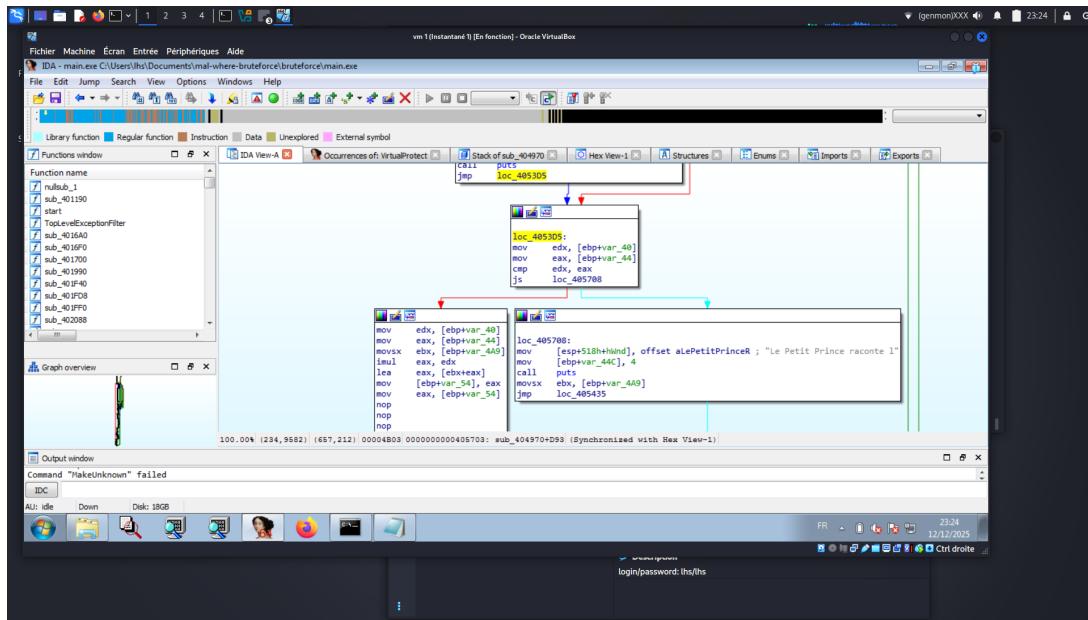
## Analyse

La comparaison effectuée à l'adresse 0x004053DB met en relation les valeurs de `var_40` et `var_44`. Si  $\text{var\_40} - \text{var\_44} < 0$ , l'exécution saute à l'adresse 0x405708, laquelle provoque ensuite l'affichage du texte suivant :

*Le Petit Prince raconte l' [...]*

Or, `var_40` et `var_44` valent respectivement 10 et 5. Cette condition n'est donc jamais vérifiée : il s'agit d'un simple leurre.

## Screenshot



## Analyse

Ainsi, sur l'image, le chemin de droite n'est jamais emprunté. Le graphe peut donc être élagué.

## Ecriture dans un fichier

**Localisation:** 0x00401D1C    **Fonction:** VirtualProtect  
**Type:** Anti-debug    **Sévérité:** Élevée

### Code Assembleur

```
.text:00404898      mov     [esp+4Ch+Stream], 40h ; @ ; flNewProtect
.text:004048A0      mov     [esp+4Ch+pbDebuggerPresent], 19Eh ; dwSize
.text:004048A8      mov     [esp+4Ch+hProcess], offset byte_4786A0 ; lpAddress
.text:004048AF      call    ds:VirtualProtect
```

### Analyse

La fonction `sub_4047A0` agit comme un *loader* : son rôle est d'utiliser une donnée "externe" pour reconstruire un morceau de programme. Premièrement la fonction vérifie si un debugger est présent à l'aide de la fonction native `CheckRemoteDebuggerPresent`. Si un debugger est détecté, elle saute à l'adresse `loc_404800` (nous ne détaillons pas le comportement si un debugger est présent). Dans le cas contraire, la fonction va chercher directement sur le disque à la localisation suivante :

"C:\Users\lhs\AppData\Local\Temp\astiko.txt".

La fonction tente d'ouvrir ce fichier avec `fopen`. Si `fopen` échoue, alors la fonction saute à l'erreur "POUET", d'où la présence de cette chaîne de caractères dans les *strings* de l'exe. Ensuite, la fonction lit les 20 (14h) octets présents dans ce fichier texte. Nous avons découvert plus tôt que le fichier texte contient la chaîne de caractère suivante, servant de clé de déchiffrement:

pakbo-et-lombrik.fr

Ensuite le déchiffrement commence. La ligne `xor byte_4786A0[ecx], al` indique que le chiffré se trouve à l'adresse `0x4786A0`. La boucle de déchiffrement tourne 414 fois (19Eh). Cela nous indique que le chiffré est long de 414 octets.

Une fois les 414 octets déchiffrés, le programme fait un appel à `VirtualProtect` afin de changer les permissions d'une zone mémoire. L'appel à `VirtualProtect` est détaillé ci-dessus :

- `lpAddress` (`0x4786A0`) : l'adresse du début de la zone mémoire dont on veut changer les permissions. C'est une adresse statique dans le segment de donnée du programme ;
- `dwSize` (`0x19E = 414 octets`) : la taille de la région à protéger ;
- `flNewProtect` (`0x40`) : la nouvelle protection demandée correspond à `PAGE_EXECUTE_READWRITE`, ce qui signifie que la zone devient lisible, modifiable et exécutable ;
- `pflOldProtect` : un pointeur vers une variable locale sur la *stack* (`[esp+4Ch+f10ldProtect]`) qui recevra l'ancienne valeur de protection.

Ensuite, nous avons extrait les 414 octets chiffrés à l'aide du script python ci-dessous. On obtient le code machine suivant

```
55          push   rbp
89 e5        mov    rbp, rsp
83 ec 24      sub    rsp, 0x24
8b 45 08      mov    eax, DWORD PTR [rbp+0x8]
```

88 45 dc	mov	BYTE PTR [rbp-0x24],al
c6 45 fb 01	mov	BYTE PTR [rbp-0x5],0x1
c6 45 fa 00	mov	BYTE PTR [rbp-0x6],0x0
c6 45 f9 01	mov	BYTE PTR [rbp-0x7],0x1
c7 45 f4 2a 00 00 00	mov	DWORD PTR [rbp-0xc],0x2a
0f b6 45 fb	movzx	eax,BYTE PTR [rbp-0x5]
84 c0	test	al,al
74 0f	je	0x35
0f b6 45 fa	movzx	eax,BYTE PTR [rbp-0x6]
84 c0	test	al,al
74 07	je	0x35
b8 01 00 00 00	mov	eax,0x1
eb 05	jmp	0x3a
b8 00 00 00 00	mov	eax,0x0
88 45 f3	mov	BYTE PTR [rbp-0xd],al
0f b6 45 fb	movzx	eax,BYTE PTR [rbp-0x5]
84 c0	test	al,al
75 08	jne	0x52
0f b6 45 fa	movzx	eax,BYTE PTR [rbp-0x6]
84 c0	test	al,al
74 07	je	0x59
b8 01 00 00 00	mov	eax,0x1
eb 05	jmp	0x5e
b8 00 00 00 00	mov	eax,0x0
88 45 f2	mov	BYTE PTR [rbp-0xe],al
0f b6 45 f9	movzx	eax,BYTE PTR [rbp-0x7]
83 f0 01	xor	eax,0x1
88 45 f1	mov	BYTE PTR [rbp-0xf],al
0f b6 55 f3	movzx	edx,BYTE PTR [rbp-0xd]
0f b6 45 f2	movzx	eax,BYTE PTR [rbp-0xe]
38 c2	cmp	dl,al
0f 95 c0	setne	al
88 45 f0	mov	BYTE PTR [rbp-0x10],al
c7 45 fc 00 00 00 00	mov	DWORD PTR [rbp-0x4],0x0
eb 16	jmp	0x9f
8b 45 fc	mov	eax,DWORD PTR [rbp-0x4]
83 e0 01	and	eax,0x1
85 c0	test	eax,eax
0f 94 c0	sete	al
88 45 eb	mov	BYTE PTR [rbp-0x15],al
0f b6 45 eb	movzx	eax,BYTE PTR [rbp-0x15]
83 45 fc 01	add	DWORD PTR [rbp-0x4],0x1
83 7d fc 04	cmp	DWORD PTR [rbp-0x4],0x4
0f 9e c0	setle	al
84 c0	test	al,al
75 df	jne	0x89
8b 45 f4	mov	eax,DWORD PTR [rbp-0xc]
c1 e8 1f	shr	eax,0x1f
84 c0	test	al,al
74 0c	je	0xba
0f b6 45 dc	movzx	eax,BYTE PTR [rbp-0x24]

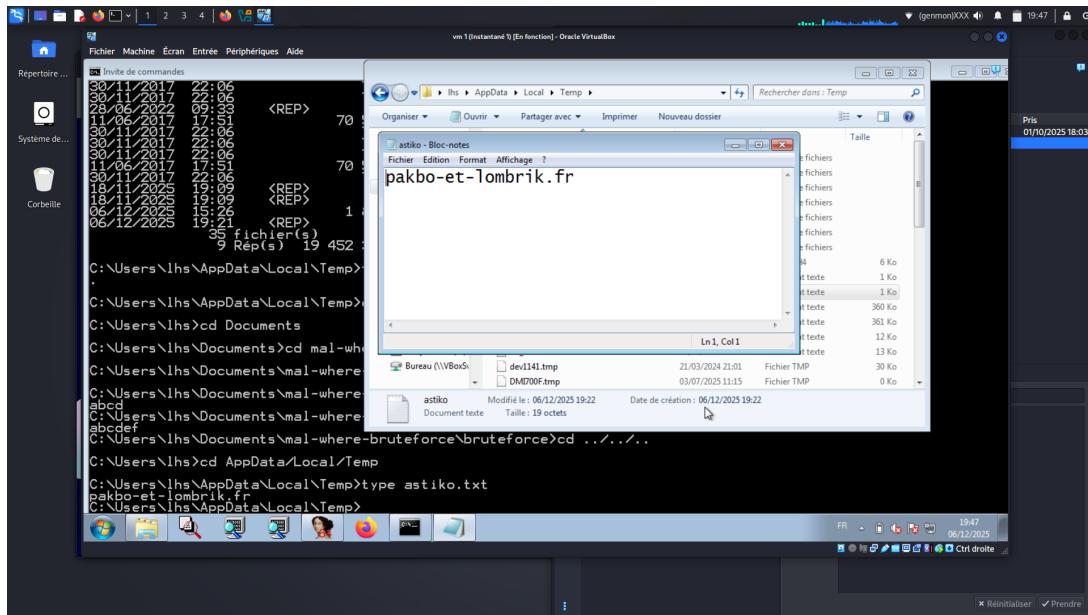
83 f0 01	xor	eax,0x1
e9 e5 00 00 00	jmp	0xa4
8b 55 f4	mov	edx,DWORD PTR [rbp-0xc]
8b 45 f4	mov	eax,DWORD PTR [rbp-0xc]
0f af c2	imul	eax,edx
85 c0	test	eax,eax
0f 9f c0	setg	al
88 45 ef	mov	BYTE PTR [rbp-0x11],al
0f b6 45 ef	movzx	eax,BYTE PTR [rbp-0x11]
83 f0 01	xor	eax,0x1
84 c0	test	al,al
74 0a	je	0xe6
b8 00 00 00 00	mov	eax,0x0
e9 bf 00 00 00	jmp	0xa4
0f b6 45 dc	movzx	eax,BYTE PTR [rbp-0x24]
88 45 ee	mov	BYTE PTR [rbp-0x12],al
0f b6 45 ee	movzx	eax,BYTE PTR [rbp-0x12]
84 c0	test	al,al
74 07	je	0xfa
b8 01 00 00 00	mov	eax,0x1
eb 05	jmp	0xff
b8 00 00 00 00	mov	eax,0x0
88 45 ee	mov	BYTE PTR [rbp-0x12],al
0f b6 45 ee	movzx	eax,BYTE PTR [rbp-0x12]
84 c0	test	al,al
75 07	jne	0x10f
b8 00 00 00 00	mov	eax,0x0
eb 05	jmp	0x114
b8 01 00 00 00	mov	eax,0x1
88 45 ee	mov	BYTE PTR [rbp-0x12],al
0f b6 45 ee	movzx	eax,BYTE PTR [rbp-0x12]
88 45 ee	mov	BYTE PTR [rbp-0x12],al
0f b6 45 ee	movzx	eax,BYTE PTR [rbp-0x12]
88 45 ee	mov	BYTE PTR [rbp-0x12],al
90	nop	
0f b6 45 fb	movzx	eax,BYTE PTR [rbp-0x5]
84 c0	test	al,al
74 12	je	0x14b
0f b6 45 fb	movzx	eax,BYTE PTR [rbp-0x5]
83 f0 01	xor	eax,0x1
84 c0	test	al,al
74 07	je	0x14b

```
b8 01 00 00 00      mov    eax,0x1
eb 05                jmp    0x150
b8 00 00 00 00      mov    eax,0x0
84 c0                test   al,al
74 07                je     0x157
b8 01 00 00 00      mov    eax,0x1
eb 48                jmp    0xa4
0f b6 45 dc          movzx  eax,BYTE PTR [rbp-0x24]
88 45 ed            mov    BYTE PTR [rbp-0x13],al
0f b6 45 ed          movzx  eax,BYTE PTR [rbp-0x13]
83 f0 01            xor    eax,0x1
88 45 ed            mov    BYTE PTR [rbp-0x13],al
0f b6 45 ed          movzx  eax,BYTE PTR [rbp-0x13]
83 f0 01            xor    eax,0x1
88 45 ed            mov    BYTE PTR [rbp-0x13],al
90                  nop
90                  nop
90                  nop
90                  nop
90                  nop
0f b6 45 dc          movzx  eax,BYTE PTR [rbp-0x24]
88 45 ec            mov    BYTE PTR [rbp-0x14],al
0f b6 45 ec          movzx  eax,BYTE PTR [rbp-0x14]
84 c0                test   al,al
75 06                jne   0x18e
80 7d dc 00          cmp    BYTE PTR [rbp-0x24],0x0
74 07                je    0x195
b8 01 00 00 00      mov    eax,0x1
eb 05                jmp    0x19a
b8 00 00 00 00      mov    eax,0x0
88 45 ec            mov    BYTE PTR [rbp-0x14],al
0f b6 45 dc          movzx  eax,BYTE PTR [rbp-0x24]
c9                  leave
c3                  ret
```

## Analyse

Ce code ne fait rien, c'est un leurre

## Screenshot



```

import struct
import os

FILENAME = "main.exe"
TARGET_VA = 0x4786A0
SIZE = 0x19E

def get_file_offset(pe_file, virtual_address):
    """
    Convertit une Virtual Address (VA) en File Offset (Raw Address)
    en analysant les sections du fichier PE.
    """
    try:
        pe_file.seek(0x3C)
        pe_header_offset = struct.unpack('<I', pe_file.read(4))[0]

        # Vérification signature PE
        pe_file.seek(pe_header_offset)
        if pe_file.read(4) != b'PE\x00\x00\x00\x00':
            print("[-] Erreur : Ce n'est pas un fichier PE valide.")
            return None

        # Lecture du File Header (nombre de sections)
        pe_file.seek(pe_header_offset + 6)
        num_sections = struct.unpack('<H', pe_file.read(2))[0]

        # Lecture de l'Optional Header
        pe_file.seek(pe_header_offset + 24)
    
```

```
magic = struct.unpack('<H', pe_file.read(2))[0]

# Calcul de la taille de l'Optional Header
pe_file.seek(pe_header_offset + 20)
opt_header_size = struct.unpack('<H', pe_file.read(2))[0]

# Recuperation de l'ImageBase
pe_file.seek(pe_header_offset + 24)
if magic == 0x10b: # PE32 (32-bit)
    pe_file.seek(pe_header_offset + 24 + 28)
    image_base = struct.unpack('<I', pe_file.read(4))[0]
elif magic == 0x20b: # PE32+ (64-bit)
    pe_file.seek(pe_header_offset + 24 + 24)
    image_base = struct.unpack('<Q', pe_file.read(8))[0]
else:
    print("[-] Format PE inconnu.")
    return None

print(f"[i] Image Base detectee : {hex(image_base)}")

# Calcul de la RVA (Relative Virtual Address)
target_rva = virtual_address - image_base
print(f"[i] RVA Cible : {hex(target_rva)}")

# Debut de la table des sections
section_table_offset = pe_header_offset + 24 + opt_header_size
pe_file.seek(section_table_offset)

# Parcours des sections pour trouver celle qui contient notre adresse
for i in range(num_sections):
    # Structure Section Header
    pe_file.seek(section_table_offset + (i * 40))
    sec_name = pe_file.read(8).strip(b'\x00').decode(errors='ignore')
    v_size, v_addr, r_size, r_ptr = struct.unpack('<IIII', pe_file.read(16))

    # Verification si l'adresse est dans cette section
    if v_addr <= target_rva < (v_addr + max(v_size, r_size)):
        print(f"[+] Adresse trouvée dans la section : .{sec_name}")
        # Formule magique : Offset = RVA - VirtualAddress + PointerToRawData
        file_offset = target_rva - v_addr + r_ptr
        return file_offset

print("[-] Adresse introuvable dans les sections.")
return None

except Exception as e:
    print(f"[-] Erreur lors de l'analyse : {e}")
    return None

def extract_data():
```

```
global FILENAME
with open(FILENAME, 'rb') as f:
    print(f"[*] Analyse de {FILENAME}...")
    offset = get_file_offset(f, TARGET_VA)

    if offset is not None:
        print(f"[+] Offset fichier calcule : {hex(offset)}")

        f.seek(offset)
        data = f.read(SIZE)

        if len(data) == SIZE:
            print(f"[+] {len(data)} octets extraits avec succes !")

            # Sauvegarde dans un fichier blob
            output_file = "encrypted_blob.bin"
            with open(output_file, "wb") as out:
                out.write(data)

            print(f"[i] Donnees sauvegardees dans '{output_file}'")

    else:
        print("[-] Erreur : Lecture incomplete.")

if __name__ == "__main__":
    extract_data()
```

## Test SHA-256

**Localisation:** 0040331B    **Fonction:** ?

**Type:** Hash    **Sévérité:** Moyenne

```

1 .text:00401F6F ; __ unwind { // sub_474F60
2 .text:00401F6F             lea     eax, [ebp+var_A]
3 .text:00401F72             mov     [esp+8], eax    ; int
4 .text:00401F76             mov     dword ptr [esp+4], offset
5     aHelloWorld ; "Hello, World!"
6 .text:00401F7E             mov     dword ptr [esp], offset
7     dword_565030 ; int
8 .text:00401F85             mov     [ebp+fctx.call_site], 2
9 .text:00401F8C             call    sub_448120
10 .text:00401F91            mov     dword ptr [esp], offset
11     sub_4020A0 ; _onexit_t
12 .text:00401F98             call    sub_409C20
13 .text:00401F9D             lea     eax, [ebp+var_A+1]
14 .text:00401FA0             mov     [esp+8], eax    ; int
15 .text:00401FA4             mov     dword ptr [esp+4], offset
16     aBea8e217036cb3 ; "bea8e217036cb3b738e207fe5d40266828bc196"...
17 .text:00401FAC             mov     dword ptr [esp], offset
18     dword_565034 ; int
19 .text:00401FB3             mov     [ebp+fctx.call_site], 1
20 .text:00401FBA             call    sub_448120
21 .text:00401FBF             mov     dword ptr [esp], offset
22     sub_401FF0 ; _onexit_t
23 .text:00401FC6             call    sub_409C20
24 .text:00401FCB             lea     eax, [ebp+fctx]
25 .text:00401FCE             mov     [esp], eax    ; lpfctx
26 .text:00401FD1             call    _Unwind_SjLj_Unregister
27 .text:00401FD6             leave
28 .text:00401FD7             retn

```

### Analyse

Dans la fonction ci-dessus, le programme stocke la valeur

bea8e217036cb3b738e207fe5d40266828bc1969fd8538d533ea39f4e40ffc8f

qui semble être un digest (SHA-256) en `dword_565034`.

```

1 .text:00403770 loc_403770:
2 .text:00403770             cmp     ecx, ecx
3 .text:00403772             repe   cmpsb
4 .text:00403774             jnz    loc_403333
5 .text:0040377A             mov     eax, [ebp+var_24]
6 .text:0040377D             mov     edi, [eax-0Ch]
7 .text:00403780             lea     edx, [eax-0Ch]
8 .text:00403783             test   edi, edi
9 .text:00403785             jnz    loc_403996

```

### Analyse

Ensuite, le programme compare le SHA-256 de l'entrée utilisateur avec la valeur de hash. Si les deux sont identiques, le programme saute à la fonction située en `loc_403996`

```

1 loc_403996:
2 .text:00403996          mov     esi, [edx+8]
3 .text:00403999          test    esi, esi
4 .text:0040399B          js      short loc_4039B5
5 .text:0040399D          lea     eax, [ebp+var_24]
6 .text:004039A0          mov     [esp], eax
7 .text:004039A3          mov     [ebp+fctx.call_site], 25h ; '%'
8 .text:004039AD          call    sub_445EE0
9 .text:004039B2          mov     eax, [ebp+var_24]
10 .text:004039B5
11 .text:004039B5 loc_4039B5:
12 .text:004039B5          movzx  eax, byte ptr [eax]
13 .text:004039B8          mov     dword ptr [esp], offset Format
   ; "Byte 0: %02x\n"
14 .text:004039BF          mov     [ebp+fctx.call_site], 25h ; '%'
15 .text:004039C9          mov     [esp+4], eax
16 .text:004039CD          call    printf
17 .text:004039D2          jmp    loc_40378B

```

## Analyse

La fonction loc\_4039B5 permet d'afficher le premier octet de l'entrée.

## Screenshot

```

.text:004032C4          mov     dword ptr [esp+4], offset dword_565030
.text:004032CC          mov     [esp], eax
.text:004032CF          mov     [ebp+fctx.call_site], 26h ; '&'
call    sub_402150
mov     [ebp+var_30], 7
mov     edi, ds:dword 565034
sub    esp, 4
mov     [ebp+var_34], 3
mov     esi, [ebp+var_30]
mov     edx, [ebp+var_30]
mov     ecx, [ebp+var_34]
mov     eax, [ebp+var_34]
mov     [ebp+var_38], offset unk_47DFC8
imul   edx, esi
mov     esi, [ebp+var_28]
mov     [ebp+var_3C], offset unk_47DFC8
imul   eax, ecx
mov     ecx, [esi-0Ch]
cmp    edx, eax
setne  al
cmp    ecx, [edi-0Ch]
mov     byte ptr [ebp+var_C+3], al
mov     [ebp+var_40], 0
jz     loc_403770
; CODE XREF: sub_403230+544+j
; sub_403230+5F51j
loc_403333:
mov     eax, [ebp+var_24]
mov     eax, [eax+5Ch]

```

## Analyse

Après avoir identifié ce potentiel comportement, nous avons tenté de le produire. Pour cela nous avons cherché le hash directement dans l'executable afin de le modifier afin de le remplacer par un hash de notre choix : le SHA-256 de 1234

03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4

## Screenshot

```

010480 01000374 03725f08 0100c98c 05720000 0c696267 05085f73 2209a6c0 002f9470 3f520507 09737465 72456c61 73736373
510528 00000000 53637265 656e4065 6c746572 00770000 433a5c55 73657273 56c6b873 5c417070 44617461 5c46f63 616c5c54
510576 656f0795 61737469 686f2e74 78740b58 4c4f475d 5b606169 6502066 6f706566 00706168 626f2065 742d6cf 60627269
510624 682f6672 00000000 00000000 00000000 00000000 00000000 00000000 48656c6c 6f2c2057 6f726c64 21000000
510672 62656138 65323137 30333663 62336237 33386532 30376665 35643430 32363638 3236263 31393639 66643835 33386435
510720 33336561 33396634 65343066 66633866 00000000 62617369 635f7374 72696e67 3a3a5f53 5f636f6e 73747275 63742046
510768 554c4c20 666f7420 76616c69 64000000 58444542 55475d20 48617368 2076616c 69646174 696f6e3a 20504153 53454420
510816 7c204368 65636873 75603a20 30784445 41444245 45460042 79746520 3032780a 00646563 6f790075 6e757365
510864 64006661 68655f64 6174615f 31006661 68655f64 6174615f 3200606f 72655f66 616b655f 64617461 00666966 616c5f64

```

## Screenshot

```

J40710 0360013c 0113f763 080f1717 16f1003b 50660103 0c3b2060 0f760501 00700510 020f2063 74200c0f 080f1703
510624 682f6672 00000000 00000000 00000000 00000000 48656c6c 6f2c2057 6f726c64 21000000
510672 30336163 36373432 31366633 65313563 37363165 65323535 66303637 39353336 32336338 62333838 62343435
510720 39651313 63397378 64376533 34366634 00000000 62617369 635f7374 72696e67 3a3a5f53 5f636f6e 73742046
510768 554c4c20 666f7420 76616c69 64000000 58444542 55475d20 48617368 2076616c 69646174 696f6e3a 20504153 53454420

```

## Analyse

Cependant, en effectuant `m.exe 1234` le comportement reste inchangé. Deux hypothèses sont donc à considérer :

- le programme effectue un test d'intégrité sur son contenu ;
- le hash identifié n'impacte en aucun cas la fonction legitimate `echo`.