KeyGenMe for Newbies :: Progressive KeygenMe #1

Publié le 22 décembre 2010 par Ryscrow

Bonjour à tous!

Pour mon premier post sur Re-Xe, je vous propose un petit tuto qui vise surtout les débutants en RE.
Il s'agit ici d'exploiter un simple petit KeyGenMe, en l'analysant puis en codant rapidement un petit KeyGen.

Vous pouvez trouver le KeygenMe ici : http://re-xe.com/uploads/2011/04/progressive_keygenme_1.zip

Si on l'execute, on aperçoit une simple fenêtre, avec deux champs de texte: un pour le nom et un pour le serial, ainsi que deux boutons.
Essayons de rentrer des infos au hasard.

Pour ma part, j'ai testé avec 'ryscrow' comme nom, et '123456789′ en serial. Appuyez sur Check, et normalement (sauf si vous avez une chance d'enfer et que vous avez réussi à trouver le serial correspondant au hasard du premier coup) la réponse ne tarde pas à tomber, avec une fenêtre qui s'ouvre indiquant: « Et non… Retente ta chance ».

Le but va donc être de trouver un serial correspondant au nom entré, et de réussir à trouver la routine pour le générer à partir de n'importe quel nom.

La première chose à vérifier, c'est que l'executable n'est pas packé ni crypté. Pour celà, un simple outil comme PEiD suffit.
On lance PEiD, on ajoute l'executable à tester, et PEiD nous informe que rien n'a été trouvé. (ce qui est normal pour un KeyGenMe destiné aux débutants…)

On va donc ici utiliser OllyDBG pour reverser notre programme.
Une fois lancé, OllyDBG nous place directement sur le point d'entrée du programme, à l'adresse 0×0401000.

Dans ce cas précis, le code désassemblé est suffisament clair pour qu'on puisse se passer de debug.
Les choses intéressantes commencent lors de l'appel à la fonction GetDlgItemTextA.

```
1  0040106B   |.  83F8 05            CMP EAX,5
2  0040106E   |.  72 7E              JB SHORT 004010EE
3  00401070   |.  83F8 14            CMP EAX,14
4  00401073   |.  0F87 8C000000      JA 00401105
5  00401079   |.  89C1               MOV ECX,EAX
6  0040107B   |.  8D35 00204000      LEA ESI,[402000]
7  00401081   |.  31DB               XOR EBX,EBX
8  00401083   |&gt;  0FB606           /MOVZX EAX,BYTE PTR [ESI]
```

```
9  00401086   |.  01C3              |ADD EBX,EAX
1000401088   |.  46                |INC ESI
1100401089   |.^ E2 F8             \LOOPD SHORT 00401083
120040108B   |.  53                PUSH EBX                 ; /&lt;%i&gt;
130040108C   |.  68 63204000       PUSH 00402063            ; |Format = "%i"
1400401091   |.  68 42204000       PUSH 00402042            ; |s = progress.00402042
1500401096   |.  FF15 C2304000     CALL [&lt;&amp;USER32.wsprintfA&gt;]   ; \wsprintfA
160040109C   |.  6A 20             PUSH 20                  ; /Count = 20 (32.)
170040109E   |.  68 21204000       PUSH 00402021            ; |Buffer = progress.00402021
18004010A3   |.  6A 66             PUSH 66                  ; |ControlID = 66 (102.)
19004010A5   |.  FF75 08           PUSH DWORD PTR [EBP+8]   ; |hWnd
20004010A8   |.  FF15 BE304000     CALL [&lt;&amp;USER32.GetDlgItemTex&gt;; \GetDlgItemText
21004010AE   |.  68 21204000       PUSH 00402021            ; /String2 = ""
22004010B3   |.  68 42204000       PUSH 00402042            ; |String1 = ""
23004010B8   |.  FF15 6A304000     CALL [&lt;&amp;KERNEL32.lstrcmp&gt;]   ; \lstrcmpA
24004010BE   |.  75 17             JNZ SHORT 004010D7
25004010C0   |.  6A 00             PUSH 0                   ; /Style = MB_OK|MB_APPLMODAL
26004010C2   |.  68 66204000       PUSH 00402066            ; |Title = "Progressive KeygenMe #1"
27004010C7   |.  68 7E204000       PUSH 0040207E            ; |Text = "Bravooo !!!",LF,CR,"Maintenan
28t'es",LF,CR,"motiv",E9," un tutorial est le bienvenu."
29004010CC   |.  FF75 08           PUSH DWORD PTR [EBP+8]   ; |hOwner
30004010CF   |.  FF15 C6304000     CALL [&lt;&amp;USER32.MessageBoxA&gt;] ; \MessageBoxA
31004010D5   |.  EB 67             JMP SHORT 0040113E
32004010D7   |&gt;  6A 00           PUSH 0                   ; /Style = MB_OK|MB_APPLMODAL
33004010D9   |.  68 66204000       PUSH 00402066            ; |Title = "Progressive KeygenMe #1"
34004010DE   |.  68 D5204000       PUSH 004020D5            ; |Text = "Et non...",LF,CR,"Retente ta ch
35004010E3   |.  FF75 08           PUSH DWORD PTR [EBP+8]   ; |hOwner
36004010E6   |.  FF15 C6304000     CALL [&lt;&amp;USER32.MessageBoxA&gt;] ; \MessageBoxA
37004010EC   |.  EB 50             JMP SHORT 0040113E
38004010EE   |&gt;  6A 00           PUSH 0                   ; /Style = MB_OK|MB_APPLMODAL
39004010F0   |.  68 66204000       PUSH 00402066            ; |Title = "Progressive KeygenMe #1"
40004010F5   |.  68 21214000       PUSH 00402121            ; |Text = "Le nom doit faire au moins 5 car
41004010FA   |.  FF75 08           PUSH DWORD PTR [EBP+8]   ; |hOwner
42004010FD   |.  FF15 C6304000     CALL [&lt;&amp;USER32.MessageBoxA&gt;] ; \MessageBoxA
4300401103   |.  EB 39             JMP SHORT 0040113E
4400401105   |&gt;  6A 00           PUSH 0                   ; /Style = MB_OK|MB_APPLMODAL
4500401107   |.  68 66204000       PUSH 00402066            ; |Title = "Progressive KeygenMe #1"
460040110C   |.  68 F2204000       PUSH 004020F2            ; |Text = "Le nom ne doit pas faire plus de
4700401111   |.  FF75 08           PUSH DWORD PTR [EBP+8]   ; |hOwner
4800401114   |.  FF15 C6304000     CALL [&lt;&amp;USER32.MessageBoxA&gt;] ; \MessageBoxA
490040111A   |.  EB 22             JMP SHORT 0040113E
500040111C   |&gt;  6A 40           PUSH 40                  ; /Style = MB_OK|MB_ICONASTERISK|MB
510040111E   |.  68 49214000       PUSH 00402149            ; |Title = "About"
5200401123   |.  68 4F214000       PUSH 0040214F            ; |Text = "Progressive KeygenMe
53#1",LF,CR,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"",AF,"
   00401128   |.  FF75 08           PUSH DWORD PTR [EBP+8]   ; |hOwner
   0040112B   |.  FF15 C6304000     CALL [&lt;&amp;USER32.MessageBoxA&gt;] ; \MessageBoxA
```

En analysant un peu, on se rend compte que le gros de l'algorithme de génération du serial tient en 11 lignes de code:

```
1  0040106B   |.  83F8 05            CMP EAX,5
2  0040106E   |.  72 7E              JB SHORT 004010EE
3  00401070   |.  83F8 14            CMP EAX,14
4  00401073   |.  0F87 8C000000      JA 00401105
5  00401079   |.  89C1               MOV ECX,EAX
6  0040107B   |.  8D35 00204000      LEA ESI,[402000]
7  00401081   |.  31DB               XOR EBX,EBX
8  00401083   |&gt;  0FB606           /MOVZX EAX,BYTE PTR [ESI]
9  00401086   |.  01C3               |ADD EBX,EAX
1000401088   |.  46                 |INC ESI
1100401089   |.^ E2 F8              \LOOPD SHORT 00401083
```

Les 4 premières lignes contrôlent le nombre de caractères du « Name », nombre stocké dans le registre EAX.
Si EAX < 5 (donc 5 en décimal), on saute vers l'instruction qui affiche un message d'erreur. Si EAX > 14 (donc 20 en décimal), on saute vers l'instruction qui affiche un autre message d'erreur.

La 5ème instruction place le contenue du registre EAX dans ECX.

La 6ème place dans ESI un pointeur vers l'adresse où est stocké « Name » (402000)

La 7ème instruction remet à 0 le registre EBX.

Les 4 dernières instructions forment une boucle qui pourrait se traduire ligne par ligne par:

```
Placer l'octet vers lequel pointe ESI dans EAX (implicitement: Tant
que celà est possible, la boucle se termine à la fin de la chaine de
caractères.)
     EBX = EBX + EAX // Ajouter à EBX la valeur de EBX
     Incrémenter i
Retourner au début
```

La suite du code se contente de comparer la valeur obtenue par traitement du Name avec le sérial donné, et de nous rediriger en fonction du résultat de cette comparaison.

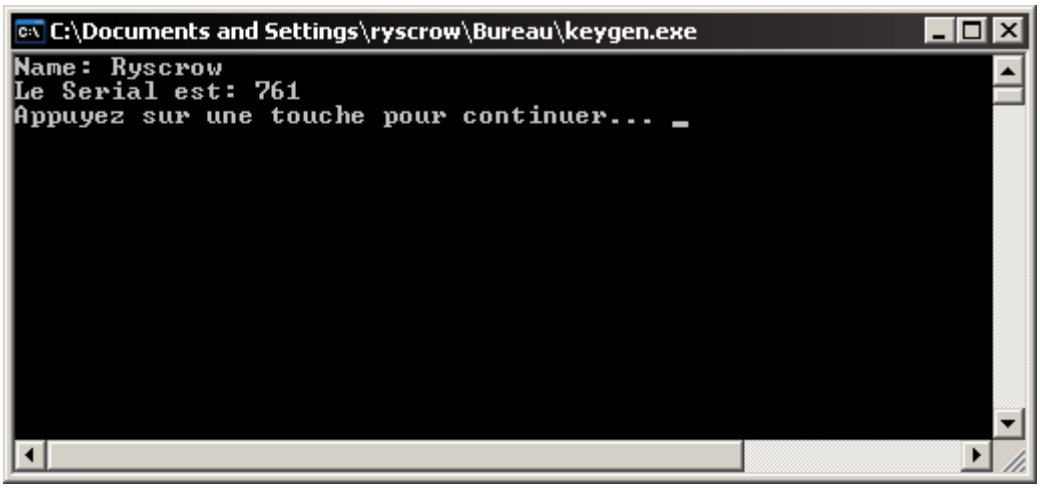
Donc après réflexion, l'algo n'est pas si compliqué que ça: il se contente d'additioner les valeurs décimales de chaque caractères du « Name ».

Voici un petit KeyGen que j'ai codé en C:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6  char name[20];
7  printf("Name: ");
8  scanf("%s", name);
9
```

```
10int somme = 0;
11int i = 0;
12
13while(name[i] != '\0')
14{
15somme = somme + name[i];
16i++;
17}
18
19printf("Le Serial est: %d", somme);
20system("PAUSE");
21return 0;
22}
```

Et voici l'utilisation :



Et donc :

Et voilà, cet article est fini J'espère qu'il vous aura aidé.

Si vous avez la moindre question, n'hésitez pas à poster des commentaires !

Cette entrée a été publiée dans Keygen. Vous pouvez la mettre en favoris avec ce permalien.

2 réponses à *KeyGenMe for Newbies :: Progressive KeygenMe #1*

1. *Xylitol* dit :

   25 décembre 2010 à 15 h 58 min

   Yo nigga
   les valeurs décimales de chaque caractères du « Name ». // loop -> addition de chaque chars du nom en hex !

   le décimal que tu parle de c'est %i
   i: signed decimal integer. This value is equivalent to d.
   (le output du serial et en décimal)
   pour résumé: les calcules sont en hex mais le output et en dec

   allez un keygen en assembleur car le c en console ça sucks:
   tapz.asm:

```
1  .486
2  .model  flat, stdcall
3  option  casemap :none   ; case sensitive
4
5  include windows.inc
6
7  uselib  MACRO   libname
8     include     libname.inc
9     includelib  libname.lib
10ENDM
11
12uselib  user32
13uselib  kernel32
14
15DlgProc     PROTO :DWORD,:DWORD,:DWORD,:DWORD
16
17IDC_OK          equ 1003
18IDC_IDCANCEL    equ 1004
19
20.data
21szFormat db "%i",0
22
23szSizeMin   db "Le nom doit faire au moins 5 caractères",0
24szSizeMax   db "Le nom ne doit pas faire plus de 20 caractères",0
25szCap       db "Progressive KeygenMe #1 KEYGEN",0
26
27.data?
```

```
28hInstance       dd      ?   ;dd can be written as dword
29
30szName db 256 dup(?)
31szCode db 256 dup(?)
32.code
33start:
34    invoke  GetModuleHandle, NULL
35    mov hInstance, eax
36    invoke  DialogBoxParam, hInstance, 101, 0, ADDR DlgProc, 0
37    invoke  ExitProcess, eax
38; --------------------------------------------------------------------
39DlgProc proc    hWin    :DWORD,
40        uMsg    :DWORD,
41        wParam  :DWORD,
42        lParam  :DWORD
43
44    .if uMsg == WM_COMMAND
45        .if wParam == IDC_OK
46; --------------------------------------------------------------------
47;           TODO
48; --------------------------------------------------------------------
49
50        invoke GetDlgItemText,hWin,1001,addr szName,sizeof szName
51        CMP EAX,5
52        JB @MinSize
53        CMP EAX,014h
54        JA @MaxSize
55        MOV ECX,EAX
56        LEA ESI,offset szName
57        XOR EBX,EBX
58
59@progress_00401083:
60
61        MOVZX EAX,BYTE PTR DS:[ESI]
62        ADD EBX,EAX
63        INC ESI
64        LOOPD @progress_00401083
65        PUSH EBX
66        PUSH offset szFormat              ; ASCII "%i"
67        PUSH offset szCode
68        CALL wsprintf
69
70        invoke SetDlgItemText,hWin,1002,addr szCode
71        ret
72
73@MinSize:
74invoke MessageBox,hWin,addr szSizeMin,addr
75szCap,MB_ICONEXCLAMATION
76RET
77@MaxSize:
```

```
78invoke MessageBox,hWin,addr szSizeMax,addr
79szCap,MB_ICONEXCLAMATION
80RET
81
82        .elseif wParam == IDC_IDCANCEL
83            invoke EndDialog,hWin,0
84        .endif
85    .elseif uMsg == WM_CLOSE
86        invoke  EndDialog,hWin,0
87    .endif
88
89    xor eax,eax
90    ret
91DlgProc endp

    end start
```

tapz.rc:

```
    ;This Resource Script was generated by WinAsm Studio.
1
2  #define IDC_OK 1003
3  #define IDC_CANCEL 1004
4
5  101 DIALOGEX 0,0,169,44
6  CAPTION "Base"
7  FONT 8,"Tahoma"
8  STYLE 0x80c80880
9  EXSTYLE 0x00000000
10BEGIN
11    CONTROL
12"OK",IDC_OK,"Button",0x00000001,110,5,50,14,0x00000000
13    CONTROL
14"Cancel",IDC_CANCEL,"Button",0x00000000,110,23,50,14,0x00000000
15    CONTROL "",1001,"Edit",0x00000080,7,7,90,12,0x00000200
16    CONTROL "",1002,"Edit",0x00000080,7,24,90,12,0x00000200
    END
```

see ya!