

Cours N° 4

Ce cours va vous apprendre comment faire un Keygen pour Hexa.exe.

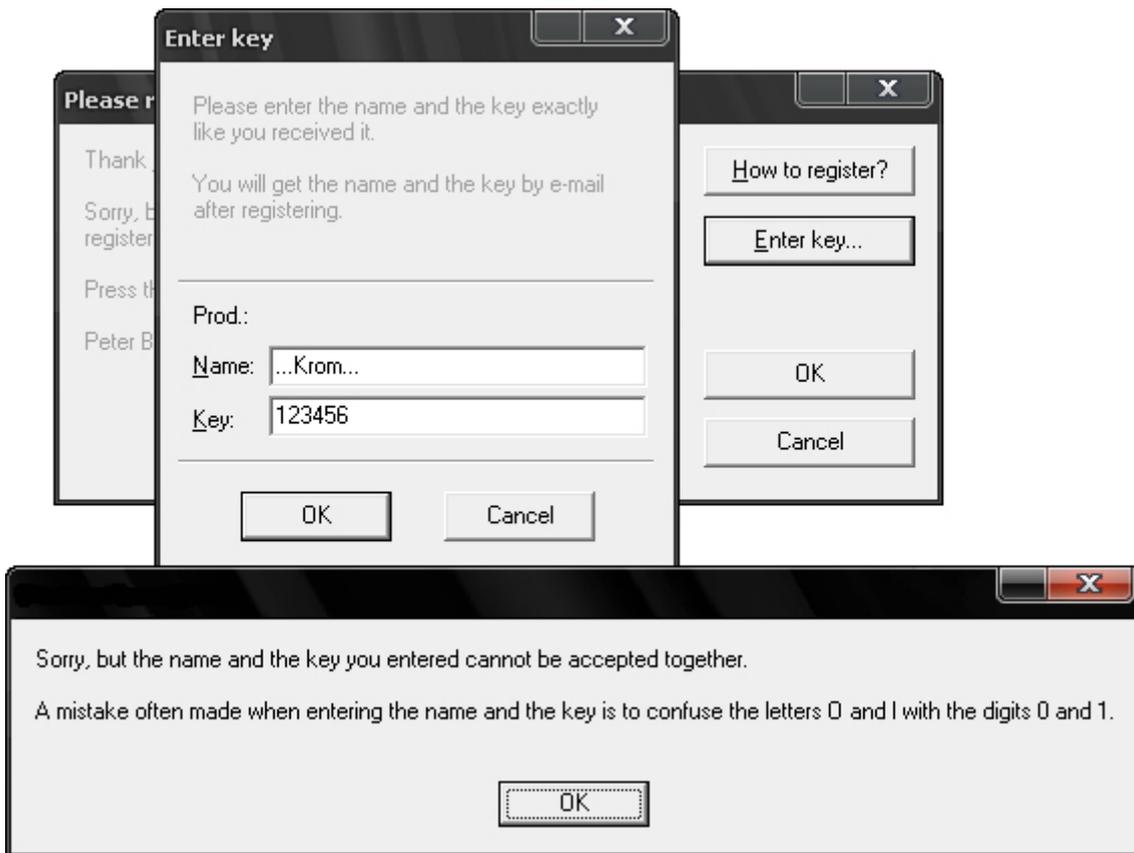
Il est téléchargeable ici :

- <http://www.KromCrack.com/prog/HexaCours4.exe>

(Password : Krommork)

Un Keygen, aussi appelé KeyGenerator ou Générateur de Clés est un logiciel générant des numéros de série afin d'installer / déverrouiller / lancer une application payante. La difficulté de ce type de Crack est de comprendre l'algorithme utilisé par le programme pour ensuite le reproduire dans n'importe quel langage de programmation, (C, C++, ASM ...). La compréhension de l'algorithme et la programmation d'un Keygen est le fin du fin pour un Cracker car il n'y a aucune modification du programme.

La première étape consiste à trouver la routine de calcul du Sérial. Pour la localiser dans le programme, on doit trouver le message d'erreur et remonter ensuite dans le code pour voir à quel moment le code est créé à partir du nom entré.



Alors quand vous entrez un nom de plus de 10 caractères, cf. Cours N° 3

- <http://www.KromCrack.com/cours3.php>

Vous voyez que le message d'erreur indiquant qu'un code est invalide est:

"Sorry, but the name and the key you entered cannot be accepted together".

Alors ouvrez "Hexa.exe" dans OllyDBG et recherchez le message d'erreur : clic droit puis "Search for" ->> "All referenced text strings".

Dans cette nouvelle fenêtre, cherchez le contenu du message d'erreur, a savoir "Sorry, but the name and the key you entered cannot be accepted together".

Vous trouverez 2 occurrences, une en 00435BD0 et une en 004358AA et c'est cette dernière qui nous intéresse :

```

00435734 ASCII " name and the ke"
00435744 ASCII "y by e-mail aft"
00435754 ASCII "r registering",0
0043576C ASCII "Cancel",0
0043577C ASCII "Enter key",0
00435884 MOV EAX,HexDecCh.00435A18
004358AA MOV EAX,HexDecCh.00435BD0 ASCII "Thank you for registering!!!You might want to have a look at my web-site"
00435929 ASCII "und Erfolg bei d"
00435939 ASCII "er Nutzung diese"
00435964 ASCII "chten Sie ab und"
  
```

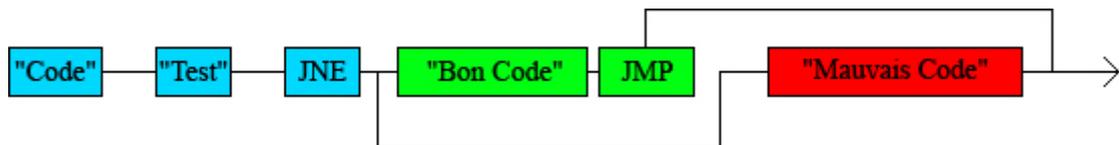
Alors, après avoir double cliquer dessus, vous arriverez là :

```

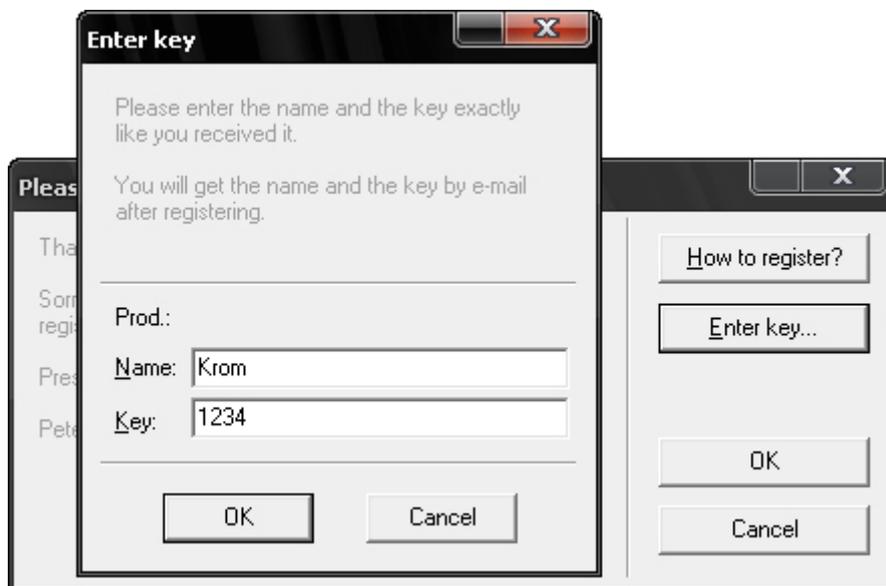
00435849  . 8B93 E0010000  MOV EDX,DWORD PTR DS:[EBX+1E0]
0043584F  . 8B93 D0010000  MOV ERX,DWORD PTR DS:[EBX+1DC]
00435855  . E8 EEC2FFFF  CALL HexDecCh.00431B45
0043585A  . 84C0          TEST AL,AL
0043585C  . 74 32        JE SHORT HexDecCh.00435890
0043585E  . C05 20654500  MOV BYTE PTR DS:[45520],1
00435865  . C783 28010000  MOV DWORD PTR DS:[EBX+1D8],1
0043586F  . 83BB D0010000  CMP DWORD PTR DS:[EBX+1D0],7
00435876  . 75 0C        JNZ SHORT HexDecCh.00435884
00435878  . B8 E5E43000  MOV ERX,HexDecCh.00435883
0043587D  . E8 EE9DFFFF  CALL HexDecCh.0042F670
00435882  . EB 20        JMP SHORT HexDecCh.00435884
00435883  . B8 18504300  MOV ERX,HexDecCh.00435818
00435884  . E8 E29DFFFF  CALL HexDecCh.0042F670
00435889  . EB 24        JMP SHORT HexDecCh.00435884
00435890  . E8 9F23DFFF  CALL HexDecCh.00407C34
00435895  . 83B9 D0010000  CMP DWORD PTR DS:[EBX+1D0],7
0043589C  . 75 0C        JNZ SHORT HexDecCh.00435884
0043589E  . B8 D3504300  MOV ERX,HexDecCh.004354D8
004358A3  . E8 C89DFFFF  CALL HexDecCh.0042F670
004358A8  . EB 0A        JMP SHORT HexDecCh.00435884
004358AF  . B8 D05E4300  MOV ERX,HexDecCh.004358D0
004358B4  . E8 BC9DFFFF  CALL HexDecCh.0042F670
004358B4  . 33C0        XOR ERX,ERX
  
```

<<-- Call de Verification du Serial "Code"
 <<-- Test du Code "Test"
 <<-- Sauter si on entre un mauvais code, sinon ne saute pas "JE"
 <<-- Message d'enregistrement si le code entre est correct "Bon Code"
 <<-- Si le code est correct, saute par dessus le message d'erreur "JMP"
 <<-- Message d'erreur si le code entre est faux "Mauvais Code"

Nous voyons ici que le fait d'être enregistré ou non dépend du saut JE SHORT HexDecCh.00435890 en 0043585C. le fait de sauter ou non dépend du résultat du TEST AL,AL en 0043585A et c'est dans le CALL qui le précède que l'on va trouver notre routine. Mettez un breakpoint sur ce CALL avec F2.



Alors lancez le programme dans OllyDBG avec F9 et enregistrez-vous (Krom / 1234) puis cliquez sur "Ok".



OllyDBG break en 00435855. Entrez dans le CALL avec F7 :

```

00431B46 8BC0      MOV EAX, EAX
00431B49 8BEC     <<-- Debut du CALL
...
00431B07  E8 EC3EFDFF CALL HexDecCh.004058C8
00431B0C  33F0     CMP ESI, EAX
00431B0E  7405     JNZ HexDecCh.00431D17 <<-- Saut vers le XOR EAX, EAX
00431BE4  8B45 F8  MOV EAX, DWORD PTR SS:[EBP-8]
00431BE7  E8 0819FDFF CALL HexDecCh.00403594
00431BEC  33F0 0A  CMP EAX, 0A
00431BF5  7405     JL HexDecCh.00431D17 <<-- Saut vers le XOR EAX, EAX
...
00431D15  EB F0    JMP SHORT HexDecCh.00431D07 <<-- Arrivée des JNZ et JL
00431D17  33C0     XOR EAX, EAX
00431D19  59      POP EDI
00431D1A  58      POP EAX
00431D1B  59      POP EDI
00431D1C  64:8910  MOV DWORD PTR FS:[EAX], EDI
00431D1F  EB 0A    JMP SHORT HexDecCh.00431D2B
00431D21  6A11FDFF JMP HexDecCh.00402E39
00431D26  E8 8114FDFF CALL HexDecCh.004031AC
00431D2B  33C0     XOR EAX, EAX
00431D2D  59      POP EDI
00431D2F  59      POP ECX
00431D30  64:8910  MOV DWORD PTR FS:[EAX], EDI
00431D33  68 2A1D4300 PUSH HexDecCh.00431D68
00431D38  3D45 E8  LEA EAX, DWORD PTR SS:[EBP-20]
00431D3B  BA 02000000 MOV EDX, 2
00431D40  E8 FB16FDFF CALL HexDecCh.00403440
00431D45  68 3D45 F8  LEA EAX, DWORD PTR SS:[EBP-10]
00431D48  E8 D916FDFF CALL HexDecCh.00403420
00431D4D  3D45 F8  LEA EAX, DWORD PTR SS:[EBP-8]
00431D50  BA 02000000 MOV EDX, 2
00431D55  E8 E616FDFF CALL HexDecCh.00403440
00431D5A  3D45 08  LEA EAX, DWORD PTR SS:[EBP+8]
00431D5D  E8 BE16FDFF CALL HexDecCh.00403420
00431D62  C9      RETN
00431D65  E9 1C13FDFF JMP HexDecCh.00403084
00431D68  EB CE    JMP SHORT HexDecCh.00431D38
00431D6A  3D45 F7  MOV AL, BYTE PTR SS:[EBP-9]
00431D6D  58      POP EAX
00431D6E  58      POP ESI
00431D6F  5B      POP EBX
00431D70  5B      MOV ESP, EBP
00431D72  C2 0400  RETN 4
00431D76  00      DB 00
<<-- Fin du CALL

```

On voit ici que l'on saute en 00431BDE (JNZ HexDecCh.00431D17) et en 00431BEF (JL HexDecCh.00431D17) pour arriver en 00431D17 ce qui a pour but de remettre EAX (donc indirectement) AL à 0. Si AL est à 0 en 0043585C, le JE (Jump if Equal to 0) Sautera par dessus le bon message pour afficher le message d'erreur. Il faut donc que les 2 sauts (JNZ HexDecCh.00431D17 et JL HexDecCh.00431D17) ne sautent pas. Intéressons-nous d'abord au premier saut (JNZ HexDecCh.00431D17).

```

00431B07  E8 EC3EFDFF CALL HexDecCh.004058C8
00431B0C  33F0     CMP ESI, EAX
00431B0E  7405     JNZ HexDecCh.00431D17 <<-- Saut vers le XOR EAX, EAX
00431BE4  8B45 F8  MOV EAX, DWORD PTR SS:[EBP-8]
00431BE7  E8 0819FDFF CALL HexDecCh.00403594
00431BEC  33F0 0A  CMP EAX, 0A
00431BF5  7405     JL HexDecCh.00431D17 <<-- Saut vers le XOR EAX, EAX
00431BF8  B2 01    MOV DL, 1
00431BF9  68 5E743200 MOV EAX, HexDecCh.0042F7E8
00431BFC  E8 5E743200 CALL HexDecCh.0042F79C
00431C01  8945 E8  MOV DWORD PTR SS:[EBP-18], EAX
00431C04  33C0     XOR EAX, EAX
00431C06  55      PUSH ESP
00431C08  68 101D4300 PUSH HexDecCh.00431D10
00431C0C  64:FF30  PUSH DWORD PTR FS:[EAX]
00431C0F  64:8920  MOV DWORD PTR FS:[EAX], ESP
00431C12  BA 02000000 MOV EDI, 00000002
00431C17  8B45 E8  MOV EAX, DWORD PTR SS:[EBP-18]
00431C1A  E8 000DFFFF CALL HexDecCh.0042F92C
00431C1F  3D65 E4  LEA EDI, DWORD PTR SS:[EBP-10]
00431C23  68 2A1D4300 MOV EAX, HexDecCh.00431D38
00431C25  E8 2A1D4300 CALL HexDecCh.00403594
EAX=00000402
ESI=0000000A

```

Pour qu'il ne saute pas, on doit avoir ESI égal à EAX donc comme ca il n'y aura aucune différence entre ESI et EAX et le JNZ ne sautera pas car Jump if Not Zéro, pas 0 -> pas de saut ;).

Donc pour qu'il n'y ait aucune différence entre le bon et le mauvais code, le code entré doit être bon logique non ? ;) Il ne suffit plus qu'a convertir 4D2 (EAX) et 62A (ESI) en Décimal car les registres sont en Hexadécimal

Registre	Valeur Hexadécimale	Valeur Décimale
EAX	000004D2	1234
ESI	0000062A	1578

Le Sérial pour Krom est donc 1578. Voyons maintenant ou ce Sérial est généré. Pour savoir d'ou viens ce Sérial il suffit de débuzzer ligne par ligne et dès que l'on verra notre Sérial (62A) apparaître dans un EAX (EAX = Sérial entré), on saura que le Sérial aura été généré dans le CALL qui le précède ... C'est assez difficile à expliquer sans exemple, donc en voici un qui vous sera surement utile !

Redémarrez avec CTRL + F2 -> Oui, lancez le ensuite en enregistrez vous avec Krom / 1234. Le programme break sur le CALL HexDecCh.00431B48 en 00435855. Entrez-y avec F7, puis faites une série de F8 jusqu'a voir 62A dans EAX. la première fois que l'on voit 62A est à l'instruction MOV ESI,EAX en 00431BD2, donc le Sérial est généré dans ce CALL :

00431BCD . E8 86FEFFFF CALL HexDecCh.00431A58

```

00431BC8 . 8B55 F8      MOV EDX,DWORD PTR SS:[EBP-8]
00431BCB . 8BC3        MOV EAX,EBX
00431BD0 . E8 86FEFFFF CALL HexDecCh.00431A58
00431BD2 . 8BF0        MOV ESI,EAX
00431BD4 . 8B45 08      MOV EAX,DWORD PTR SS:[EBP+8]
00431BD7 . E8 EC3EFDFF CALL HexDecCh.00405AC8
00431BDC . 3BF0        CMP ESI,EAX
00431BDE . 0F85 33010000 JNZ HexDecCh.00431D17
00431BE4 . 8B45 F8      MOV EAX,DWORD PTR SS:[EBP-8]
00431BE7 . E8 A819FDFF CALL HexDecCh.00403594
00431BEC . 83F8 0A      CMP EAX,0A
00431BEF . 0F8C 22010000 JL HexDecCh.00431D17
00431BF5 . B2 01        MOV DL,1
00431BF7 . B8 F8F74200 MOV EAX,HexDecCh.0042F7E8
    <<-- EAX = 00000001
    <<-- EAX = 0000062A
    <<-- Saut vers le XOR EAX,EAX
    <<-- Saut vers le XOR EAX,EAX

```

Mettez un breakpoint en 00431BDC avec F2 puis redémarrez avec CTRL + F2 -> Oui, lancez le programme avec F9 et enregistrez vous avec Krom / 1234. Le programme break sur le CALL HexDecCh.00431B48 en 00435855. Entrez-y avec F7 puis une série de F8 jusqu'au CALL 00431BDC. Entrez-y avec F7. En faisant quelques F8, vous arriverez à une boucle qui est exécutée 4 fois (Krom contient 4 lettres, la boucle sera exécutée le nombre de fois que de lettre contenue dans le nom) et si vous faites un peu attention à ce qui se passe dans cette boucle (au niveau des registres) vous y verrez Krom lettre par lettre, multiplié, divisé, mis dans un emplacement mémoire ... et donc on est sur que c'est ici qu'on a notre routine de calcul du Sérial.

```

00431B46 8BC0      MOV EAX,EAX
00431B47 $ 55      PUSH EBP
00431B49 . 8BEC     MOV EBP,ESP
<<-- Debut du CALL

...

00431B07 + E9 ED3EFDFF CALL HexDecCh.004058C8
00431B0C + 3BF0     CMP ESI,EBX
00431B0E + 0F85 33010000 JNZ HexDecCh.00431D17 <<-- Saut vers le XOR EAX,EAX
00431BE4 + 0B45 F8  MOV EAX,DWORD PTR SS:[EBP-8]
00431BE7 + E9 0019FDFF CALL HexDecCh.00403594
00431BEF + 3BF8 0A   CMP EAX,0A
00431BF1 + 0F8C 22010000 JL HexDecCh.00431D17 <<-- Saut vers le XOR EAX,EAX
00431BF5 . B2 01    MOV DL,1

...

00431D15 + EB F0    JMP SHORT HexDecCh.00431D07 <<-- Arrivée des JNZ et JL
00431D17 > 33C0     XOR EAX,EAX
00431D19 + 58      POP EDI
00431D1A + 59      POP ECX
00431D1B + 59      POP ECX
00431D1C + 64+8910 MOV DWORD PTR FS:[EAX],EDX
00431D1F + EB 0A   JMP SHORT HexDecCh.00431D2B
00431D21 + E9 6A11FDFF JMP HexDecCh.00402E39
00431D26 + E8 8114FDFF CALL HexDecCh.004031AC
00431D28 > 33C0     XOR EAX,EAX
00431D2A + 5A      POP EDI
00431D2C + 59      POP ECX
00431D2E + 59      POP ECX
00431D2F + 64+8910 MOV DWORD PTR FS:[EAX],EDX
00431D33 + 68 2A1D4300 PUSH HexDecCh.00431D5A
00431D38 > 8D45 E8  LEA EAX,DWORD PTR SS:[EBP-20]
00431D3B + BA 02000000 MOV EDX,2
00431D40 + E8 FB16FDFF CALL HexDecCh.00403448
00431D45 + 8D45 E8  LEA EAX,DWORD PTR SS:[EBP-10]
00431D48 + E8 D916FDFF CALL HexDecCh.00403420
00431D4D + 8D45 F8  LEA EAX,DWORD PTR SS:[EBP-8]
00431D50 + BA 02000000 MOV EDX,2
00431D55 + E8 E516FDFF CALL HexDecCh.00403448
00431D5A + 8D45 08  LEA EAX,DWORD PTR SS:[EBP+8]
00431D5D + E8 BE16FDFF CALL HexDecCh.00403420
00431D62 + C3      RETN
00431D63 + E9 1C13FDFF JMP HexDecCh.00403084
00431D68 + EB CE   JMP SHORT HexDecCh.00431D38
00431D6A + 8D45 F7  LEA EAX,BYTE PTR SS:[EBP-9]
00431D6D + 5E      POP EDI
00431D6E + 5E      POP ESI
00431D6F + 5B      POP EBX
00431D70 + 8BE5   MOV ESP,EBP
00431D72 + 5D     POP EBP
00431D75 + C2 0400 RETN 4 <<-- Fin du CALL
00431D76 . 0B     DB 0B
  
```

On se concentrera sur la boucle au milieu du CALL, qui est celle qui génère le Sériale. Voici le détail de la boucle de calcul du Sériale :

```

00431AD0 + 8F6 TEST ESI,ESI
00431AD2 + 7E 9F JLE SHORT HexDecCh.00431B03
00431AC4 + BB 01000000 MOV EBX,1
00431AD6 + 49 DEC ECX
00431ADC + 81E1 0F000000 AND ECX,0000000F
00431AD2 + 79 05 JNS SHORT HexDecCh.00431AD9
00431AD4 + 49 DEC ECX
00431AD5 + 83C9 F0 OR ECX,FFFFFFF0
00431AD8 + 41 INC ECX
00431AD9 + 8043B LEA EAX,DWORD PTR DS:[EBX+EDI]
00431AD0 + 51 PUSH EAX
00431AD0 + B9 12010000 MOV ECX,112
00431AE2 + 99 CDQ
00431AE3 + F7F9 IDIV ECX
00431AE6 + 8882 0C484500 MOV AL,BYTE PTR DS:[EDX+45480C]
00431AE6 + 8B56 FC MOV EDX,DWORD PTR SS:[EBP-4]
00431AE7 + 8B511A FF MOV DL,BYTE PTR DS:[EDX+EBX-1]
00431AF3 + 3C02 XOR AL,DL
00431AF5 + 25 FF000000 AND EAX,0FF
00431AF6 + D3EB SHL EAX,CL
00431AF6 + 0145 F8 ADD DWORD PTR SS:[EBP-0],EAX
00431AFC + 43 INC EBX
00431AFC + 4E DEC ESI
00431B00 + 75 C6 JNC SHORT HexDecCh.00431AC9
00431B03 + 3300 XOR EAX,EAX
00431B05 + 5A POP EBX
<<-- EBX est mis a 1
<<-- Arrivée du JNZ en 00431B01 et Debut de la boucle
<<-- ECX = 1
<<-- Compare ECX a 0000000F en binaire, et met 1 si les 2 operandes sont 1 sinon met 0
<<-- Saut vers 00431AD9
<<-- ECX = 1
<<-- Compare ECX a FFFFFFF0 en binaire, et met 0 si les 2 operandes sont 0 sinon met 1
<<-- ECX = 1
<<-- Place dans EAX, le contenu de [EBX+EDI]
<<-- Met EAX dans la Pile
<<-- Met 112 dans ECX
<<-- Convertit EAX en un quadruple mot dans <EDX:EAX>
<<-- Division signée de ECX
<<-- Récupère la valeur au haut de la pile et la met dans ECX
<<-- Met dans AL, le contenu de [EDX+45480C]
<<-- Met dans EDX le contenu de [EBP-4] !!! ( [EBP-4] = Non entre = Krom )
<<-- Met a chaque passage une lettre du nom dans DL !!! ( 1.k 2.r 3.o 4.m )
<<-- Compare AL et DL en binaire, met 1 si différent et 0 si égale et place le resultat dans AL
<<-- Compare EAX a 0FF en binaire, et met 1 si les 2 operandes sont 1 sinon met 0
<<-- Multiplie EAX par CL !!! ( SHL ne multiplie que par puissance de 2 )
<<-- Ajoute EAX dans [EBP-0] !!! ( [EBP-0] = 0012F456 = Serial )
<<-- EBX = 1
<<-- ESI = 1
<<-- Fin de la boucle ->> Saute en 00431AC9

```

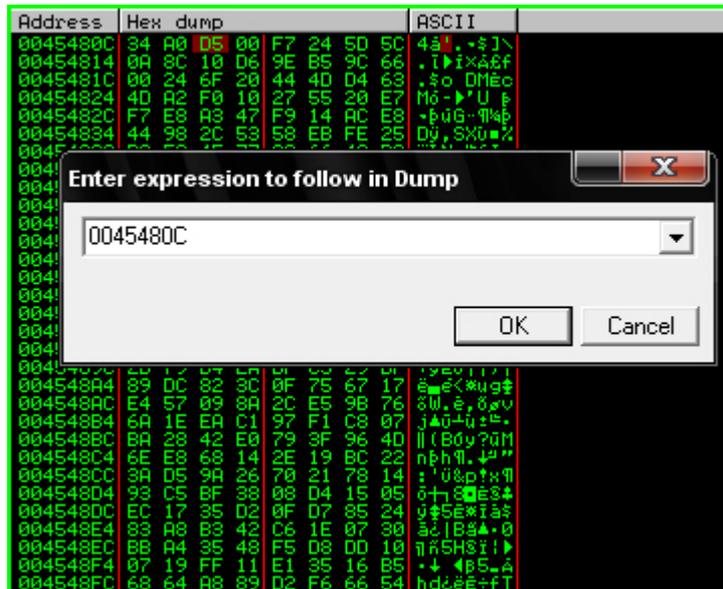
C'est maintenant la partie la plus délicate, comprendre l'algorithme et dans ce cas, il vous faut des notions en Assembleur. Je vous conseille de lire mon cours ASM :

- <http://www.kromCrack.com/pdf/Assembleur.pdf>

Il est indispensable de comprendre les instructions pour pouvoir les reproduire. Alors on va débogger le programme ligne par ligne et regarder ce qu'il fait a chaque fois.

Redémarrez avec CTRL + F2 ->> Oui, puis F9. enregistrez vous avec Krom / 1234 puis "Ok". OllyDBG Break en 00435855, entrez dans le CALL avec F7, puis F9. Là, Olly break en 00431BCD (Le CALL de calcul du Sériale) entrez dans ce CALL avec F7 et faites une série de F8 jusqu'à arriver au MOV EBX,1 en 00431AC4. C'est maintenant qu'il faudra de la concentration et de la patience car nous allons le faire ensemble et lignes par lignes ce qui prendra sûrement un moment ...

- Ligne 00431AE5, 00000000 est mis dans ECX à cause du POP ECX.
- Ligne 00431AE6, Voila une ligne intéressante ! Elle a pour but de mettre dans AL le contenu de [EDX+45480C]. Pour voir ce qu'il y a à cette adresse faites CTRL + G dans la fenêtre de Dump en bas à gauche et entrez-y 0045480C.



- On voit que 45480C vaut 34 mais comme EDX vaut 2 on regarde en fait sur l'adresse 0045480E ($0045480C + 2 = 0045480E$), on a donc AL qui vaut D5.
- Ligne 00431AEC, met [EBP-4] dans EDX, [EBP-4] est l'emplacement mémoire où est stocké le nom entré "Krom".
- Ligne 00431AEF, met dans DL, [EDX+EBX-1], EDX vaut " Krom ", EBX vaut 1 -1 donc met 4B (K) dans DL.
- Ligne 00431AF3, voici sûrement une des lignes les plus importantes, car elle compare AL et DL en binaire, et met dans AL (EAX) 1 si les 2 opérandes sont différentes sinon met 0.

Registre	Hexadécimal	Décimal	Binaire
AL	000000D5	213	11010101
DL	0000004B	75	01001011
EAX	0000009E	158	10011110

- Ligne 00431AF5, on a un AND EAX,0FF qui compare EAX (9E) à 0FF et met 1 uniquement quand les 2 opérandes sont 1 sinon met 0.

Registre	Hexadécimal	Décimal	Binaire
EAX	0000009E	158	10011110
	000000FF	255	11111111
EAX	0000009E	158	10011110

- Ligne 00431AFA, EAX est multiplié par CL. SHL est une multiplication par puissances de 2 mais comme CL vaut 0 $2^0 = 1$ donc $1 \times 9E = 9E$.
- Ligne 00431AFC, ajoute le contenu de EAX (9E) dans [EBP-8] et c'est à cette adresse que le Sérial sera stocké.
- Ligne 00431AFF, INC EBX = EBX + 1 donc EBX vaut 2.
- Ligne 00431B00, DEC ESI = ESI - 1 donc ESI vaut 3.
- Ligne 00431B01, et pour finir cette boucle, le JNZ SHORT HexDecCh.00431AC9 en 00431B01 saute en 00431AC9 pour recommence la boucle.

Boucle N° 2

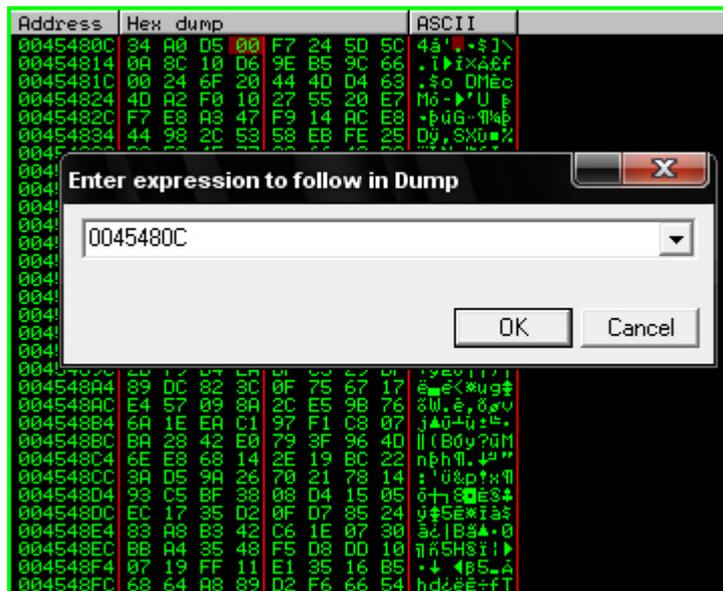
- Ligne 00431AC9, EBX (2) est mis dans ECX (0) on a donc EBX = 2 et ECX = 2.
- Ligne 00431ACB, on a un DEC ECX, ECX = 1.
- Ligne 00431ACC, on a ici un AND ECX, 8000000F, on compare ECX (1) et 8000000F et met 1 si les 2 opérandes sont 1 sinon met 0 et stock le résultat dans la première opérande.

Registre	Hexadécimal	Décimal	Binaire
ECX	00000000	0	00000000000000000000000000000000
	8000000F	2'147'483'663	100000000000000000000000000000001111
ECX	00000000	0	00000000000000000000000000000000

- Ligne 00431AD2, JNS SHORT HexDecCh.00431AD9 va sauter vers le LEA EAX,DWORD PTR DS:[EBX+EDI] en 00431AD9.
- Ligne 00431AD9, arrivée du JNS SHORT HexDecCh.00431AD9 en 00431AD2. Cette instruction LEA EAX,DWORD PTR DS:[EBX+EDI] est comme un MOV XXX, XXX, et comme EAX vaut 9E, EBX = 2 et EDI = 1 on aura EAX qui vaudra 3.
- Ligne 00431ADC, met ECX (00000001) sur la pile.
- Ligne 00431ADD, met 112 dans ECX.
- Ligne 00431AE2, convertit EAX en un quadruple mot dans <EDX:EAX>.
- Ligne 00431AE3, IDIV ECX à pour but de diviser EAX par ECX donc $3:112 = 0$.

Registre	Hexadécimal	Décimal	Binaire
EAX	00000003	3	000000011
ECX	00000112	274	100010010
EAX	00000000	0	000000000

- Ligne 00431AE5, récupère la valeur au haut de la pile et la met dans ECX (00000001) ECX = 1.
- Ligne 00431AE6, cette ligne a pour but de mettre dans AL le contenu de [EDX+45480C]. Pour voir ce qu'il y a à cette adresse faites CTRL + G dans la fenêtre de Dump en bas à gauche et entrez-y 0045480C.



On voit que 45480C vaut 34 mais comme EDX vaut 3 on regarde en fait sur l'adresse 0045480F (0045480C + 3 = 0045480F), on a donc AL qui vaut 00.

- Ligne 00431AEC, met [EBP-4] dans EDX, [EBP-4] est l'emplacement mémoire où est stocké le nom entré "Krom".
- Ligne 00431AEF, met dans DL, [EDX+EBX-1], EDX vaut "Krom", EBX vaut 2 -1 donc met 72 (r) dans DL (à chaque boucle, EBX vaut 1 de plus pour prendre une lettre différente à chaque passages).
- Ligne 00431AF3, voici surement une des lignes les plus importantes, car elle compare AL et DL en binaire, et met dans AL (EAX) 1 si les 2 opérandes sont différentes sinon met 0.

Registre	Hexadécimal	Décimal	Binaire
AL	00000000	0	0000000
DL	00000072	114	1110010
EAX	00000072	114	1110010

- Ligne 00431AF5, on a un AND EAX,0FF qui compare EAX (72) à 0FF et met 1 uniquement quand les 2 opérandes sont 1 sinon met 0.

Registre	Hexadécimal	Décimal	Binaire
EAX	00000072	114	01110010
	000000FF	255	11111111
EAX	00000072	114	01110010

- Ligne 00431AFA, à cette ligne, on multiplie EAX (72) par CL (1) en puissance de ^2.
Voici le calcul : $72 \times 2^1 = E4$. donc on a un EAX = E4.
- Ligne 00431AFC, c'est à cette ligne que le Sériel est stocké, on met EAX (E4) dans [EBP-8]. ([EBP-8] contient déjà 9E, il nous suffira de l'additionner à notre E4 pour avoir [EBP-8] = 182).
- Ligne 00431AFF, ici on incrémente EBX de 1 donc EBX = 3.
- Ligne 00431B00, ici on décrémente ESI de 1 donc ESI = 2.
- Ligne 00431B01, et pour finir cette boucle, le JNZ SHORT HexDecCh.00431AC9 en 00431B01 saute en 00431AC9 pour recommence la boucle.

Boucle N° 3

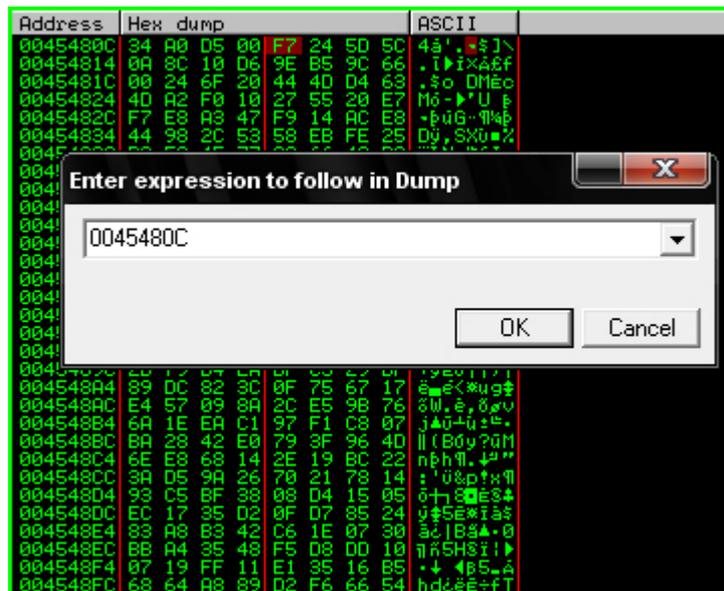
- Ligne 00431AC9, MOV ECX,EBX (EBX = 3 et ECX = 1), ECX va passer à 3.
- Ligne 00431ACB, DEC ECX, ECX = 2.
- Ligne 00431ACC, AND ECX,8000000F, compare ECX (2) à 8000000F en binaire, et met 1 si les 2 opérandes sont 1 sinon met 0. (Le résultat est mis dans la première opérande).

Registre	Hexadécimal	Décimal	Binaire
ECX	00000002	2	00000000000000000000000000000010
	8000000F	2'147'483'663	1000000000000000000000000000001111
ECX	00000002	2	00000000000000000000000000000010

- Ligne 00431AD2, le JNS SHORT HexDecCh.00431AD9 en 00431AD2 saut vers le LEA EAX,DWORD PTR DS:[EBX+EDI] en 00431AD9.
- Ligne 00431AD9, LEA EAX,DWORD PTR DS:[EBX+EDI], place dans EAX, le contenu de [EBX+EDI] donc place dans EAX, 3 (EBX) + 1 (EDI), EAX = 4.
- Ligne 00431ADC, PUSH ECX, place ECX (2) sur la pile.
- Ligne 00431ADD, MOV ECX,112, met 112 dans ECX.
- Ligne 00431AE2, CDQ, convertit EAX en un quadruple mot dans <EDX:EAX>.
- Ligne 00431AE3, IDIV ECX à pour but de diviser EAX par ECX donc $4:112 = 0$.

Registre	Hexadécimal	Décimal	Binaire
EAX	00000004	4	000000100
ECX	00000112	274	100010010
EAX	00000000	0	000000000

- Ligne 00431AE5, POP ECX, récupère la valeur qui est en haut de la pile et la met dans ECX = 2.
- Ligne 00431AE6, cette ligne a pour but de mettre dans AL le contenu de [EDX+45480C]. Pour voir ce qu'il y a à cette adresse faites CTRL + G dans la fenêtre de Dump en bas à gauche et entrez-y 0045480C.



On voit que 45480C vaut 34 mais comme EDX vaut 4 on regarde en fait sur l'adresse 00454810 (0045480C + 4 = 00454810), on a donc AL qui vaut F7.

- Ligne 00431AEC, MOV EDX,DWORD PTR SS:[EBP-4], met dans EDX le contenu de [EBP-4] ([EBP-4] = Nom entré = Krom).
- Ligne 00431AEF, met dans DL, [EDX+EBX-1], EDX vaut " Krom ", EBX vaut 3 -1 donc met 6F (o) dans DL. (à chaque boucle, EBX vaut 1 de plus pour prendre une lettre différente à chaque passages).
- Ligne 00431AF3, voici sûrement une des lignes les plus importantes, car elle compare AL et DL en binaire, et met dans AL (EAX) 1 si les 2 opérandes sont différentes sinon met 0.

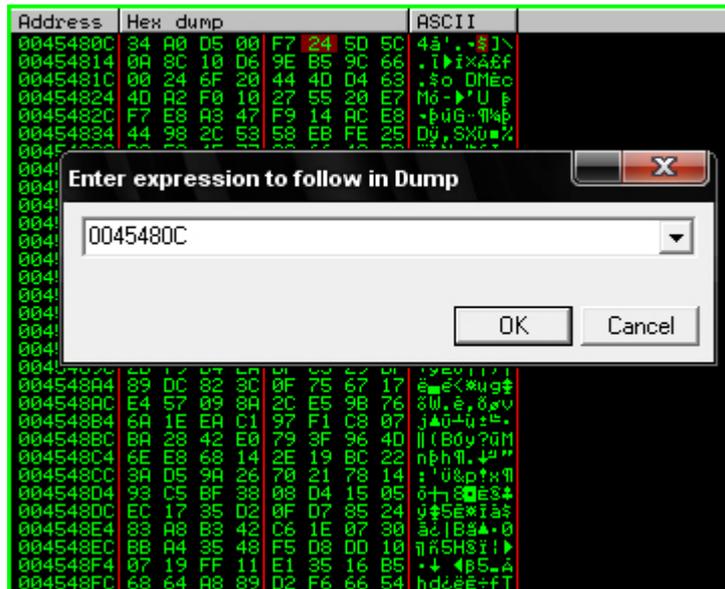
Registre	Hexadécimal	Décimal	Binaire
AL	00000F7	274	11110111
DL	000006F	111	01101111
EAX	00000098	152	10011000

- Ligne 00431AF5, on a un AND EAX,0FF qui compare EAX (98) à 0FF et met 1 uniquement quand les 2 opérandes sont 1 sinon met 0.

Registre	Hexadécimal	Décimal	Binaire
EAX	00000098	152	10011000
	000000FF	255	11111111
EAX	00000098	152	10011000

- Ligne 00431AFA, à cette ligne, on multiplie EAX (98) par CL (2) en puissance de ^2.
Voici le calcul : $98 \times 2^2 = 260$. donc on a un EAX = 260.
- Ligne 00431AFC, c'est à cette ligne que le Sériat est stocké, on met EAX (260) dans [EBP-8]. ([EBP-8] contient déjà 182, il nous suffira de l'additionner à notre 260 pour avoir [EBP-8] = 3E2).
- Ligne 00431AFF, ici on incrémente EBX de 1 donc EBX = 4.
- Ligne 00431B00, ici on décrémente ESI de 1 donc ESI = 1.
- Ligne 00431B01, et pour finir cette boucle, le JNZ SHORT HexDecCh.00431AC9 en 00431B01 saute en 00431AC9 pour recommencer la boucle.

- Ligne 00431AE5, récupère la valeur qui est en haut de la pile et la met dans ECX (ECX = 3).
- Ligne 00431AE6, cette ligne a pour but de mettre dans AL le contenu de [EDX+45480C]. Pour voir ce qu'il y a à cette adresse faites CTRL + G dans la fenêtre de Dump en bas à gauche et entrez-y 0045480C.



On voit que 45480C vaut 34 mais comme EDX vaut 5 on regarde en fait sur l'adresse 00454811 (0045480C + 5 = 00454811), on a donc AL qui vaut 24.

- Ligne 00431AEC, met [EBP-4] dans EDX, [EBP-4] est l'emplacement mémoire où est stocké le nom entré "Krom".
- Ligne 00431AEF, met dans DL, [EDX+EBX-1], EDX vaut "Krom", EBX vaut 4 -1 donc met 6D (m) dans DL (à chaque boucle, EBX vaut 1 de plus pour prendre une lettre différente à chaque passages).
- Ligne 00431AF3, voici sûrement une des lignes les plus importantes, car elle compare AL et DL en binaire, et met dans AL (EAX) 1 si les 2 opérandes sont différentes sinon met 0.

Registre	Hexadécimal	Décimal	Binaire
AL	00000024	36	0100100
DL	0000006D	109	1101101
EAX	00000049	73	1001001

- Ligne 00431AF5, on a un AND EAX,0FF qui compare EAX (49) à 0FF et met 1 uniquement quand les 2 opérandes sont 1 sinon met 0.

Registre	Hexadécimal	Décimal	Binaire
EAX	00000049	73	01001001
	000000FF	255	11111111
EAX	00000049	73	01001001

- Ligne 00431AFA, à cette ligne, on multiplie EAX (49) par CL (3) en puissance de ^2.
Voici le calcul : $49 \times 2^3 = 248$. donc on a un EAX = 248.
- Ligne 00431AFC, c'est à cette ligne que le Sérial est stocké, on met EAX (248) dans [EBP-8]. ([EBP-8] contient déjà 3E2, il nous suffira de l'additionner à notre 248 pour avoir [EBP-8] = 62A), 62A enfin ! car ce fameux 62A donne 1578 en décimal, c'est donc notre Sérial !
- Ligne 00431AFF, incrémente EBX de 1, EBX vaut alors 5.
- Ligne 00431B00, décrémente ESI de 1, ESI vaut alors 0.
- Ligne 00431B01, comme ESI vaut 0, le JNZ SHORT HexDecCh.00431AC9 en 00431B01 ne saute pas en 00431AC9 car ESI vaut 0 et continue simplement en 00431B03.

Ca a été long mais on y est arrivé, mais l'algorithme qui est dans ces boucle n'est pas vraiment lisible donc en voici un petit résumé :

1) Met à chaque boucle :

D5, 00, F7, 24, 5D, 5C, 0A, 8C, 10, D6, 9E, B5, 9C, 66, 00, 24, 6F,
 20, 44, 4D, D4, 63, 4D, A2, F0,
 10, 27, 55, 20, E7, F7, E8, A3, 47, F9, 14, AC, E8, 44, 98, 2C, 53,
 58, EB, FE, 25, B0, FC, 4E, 77, CC,
 66, 49, DA, 7C, C6, 8E, D1, 5C, 22, D9, C5, C8, 90, E4, BF, B4, 79,
 87, 7E, 3E, 3D, 76, A3, D6, 08, 43, 1A
 dans AL.

2) Met le nom entré dans EDX.

3) Met a chaque passage une lettre du nom entré dans DL, 1. 4B (K)
 2. 72 (r) 3. 67 (o) 4. 6D (m).

4) XOR AL,DL compare AL et DL en binaire, met 1 si différent et 0 si égale et place le résultat dans AL.

Registre	Boucle 1	Boucle 2	Boucle 3	Boucle 4
AL	D5	00	F7	24
DL	4B	70	6F	6D

5) Multiplie EAX par CL en puissance de 2^0 (2^0 , 2^1 , 2^2 , 2^3 ...).

6) Met le résultat dans un emplacement mémoire (dans ce cas c'est [EBP-8]).

7) Et on recommence la boucle.

La création du Keygen pour Hexa.exe en DOS
se trouve dans le [Cours N° 5](#).

- <http://www.KromCrack.com/cours5.php>

Et la création du Keygen pour Hexa.exe en GUI
se trouve dans le [Cours N° 6](#).

- <http://www.KromCrack.com/cours6.php>

J'espère que ce cours a été clair ;)

Si vous avez rencontré une erreur ou que quelque chose ne marche pas,
vous pouvez [m'envoyer un mail](#) à **Admin@KromCrack.com** ou en parler
sur [le forum](#) :

- <http://www.KromCrack.com/forum/>