Solution du défi KeygenMe#2 de CarosVonMagor
par skirby

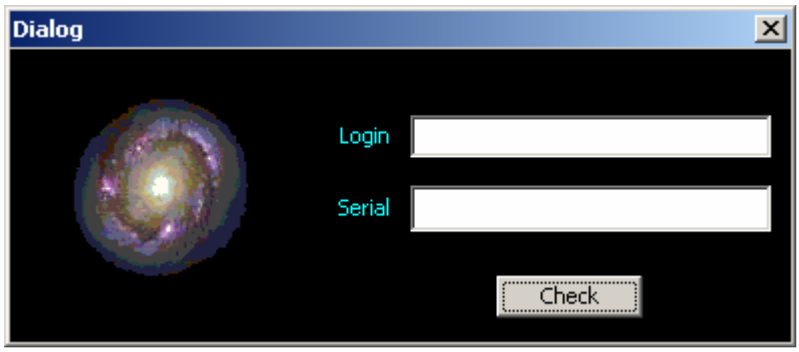
Aperçu du défi

Pour résoudre ce défi vous aurez besoin de OllyDbg.

Premier réflexe avant de débuguer un défi, on le test avec PEiD ou RDG Packer Detector.
Dans le cas présent, aucun packer n'est détecté.
En revanche, on voit ceci : Microsoft Visual C++ 7.0 [Debug]

Debug signifie qu'il va y avoir pas mal de code inutile dans le source une fois désassemblé.

Allez hop, on charge le défi dans OllyDbg et on fait F9.
On obtient l'écran suivant :



A priori, aucun anti-debugger ni aucun trick anti-ollydbg au lancement de l'application.

Analyse du des fonctions de l'API

Un deuxième bon réflexe consiste à jeter un œil rapide sur les fonctions de l'API Windows.
Clic droit sur la fenêtre du code puis « Search For » puis « All Intermodular Calls »

On peut voir 3 appels à CreateThread, 7 appels à DialogBoxParam et 1 appel à GetDlgItemTextA
On ne constate aucun appel à MessageBox.

CreateThread permet d'exécuter une portion de code en parallèle de l'exécution du programme.
On peut s'en servir pour des fonctions de vérification.

On va donc poser un breakpoint sur les 3 fonctions.

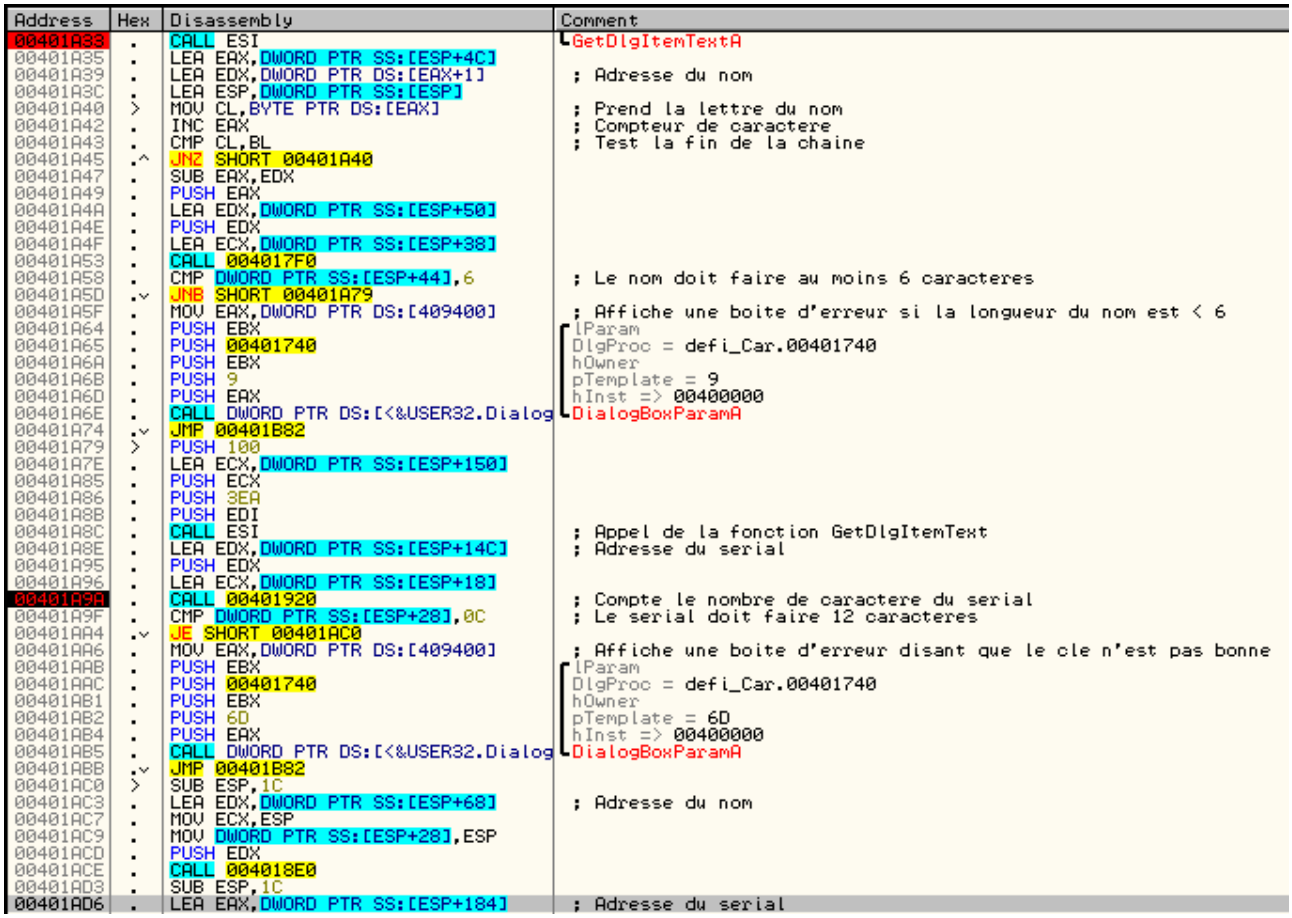
Analyse du des String Data Reference

En règle générale, il faut pouvoir se passer de SDR quand on veut cracker un programme.
J'en veux pour preuve le défi présent qui ne contient aucune SDR intéressante.
On passe donc à autre chose.

Debuggage de l’application

On lance le programme (F9) et la boîte de dialogue nous invite à saisir le nom et le serial.
On entre skirby et 1234567890 puis on clique sur le bouton Check.

On break sur la fonction GetDlgItemTextA. On va tracer à partir de la.



```
Address   Hex   Disassembly                          Comment
00401A33  .     CALL ESI                             GetDlgItemTextA
00401A35  .     LEA EAX,DWORD PTR SS:[ESP+4C]
00401A39  .     LEA EDX,DWORD PTR DS:[EAX+1]         ; Adresse du nom
00401A3C  .     LEA ESP,DWORD PTR SS:[ESP]
00401A40  >     MOV CL,BYTE PTR DS:[EAX]             ; Prend la lettre du nom
00401A42  .     INC EAX                              ; Compteur de caractere
00401A43  .     CMP CL,BL                            ; Test la fin de la chaine
00401A45  .^    JNZ SHORT 00401A40
00401A47  .     SUB EAX,EDX
00401A49  .     PUSH EAX
00401A4A  .     LEA EDX,DWORD PTR SS:[ESP+50]
00401A4E  .     PUSH EDX
00401A4F  .     LEA ECX,DWORD PTR SS:[ESP+38]
00401A53  .     CALL 004017F0
00401A58  .     CMP DWORD PTR SS:[ESP+44],6          ; Le nom doit faire au moins 6 caracteres
00401A5D  .v    JNB SHORT 00401A79
00401A5F  .     MOV EAX,DWORD PTR DS:[409400]        ; Affiche une boite d'erreur si la longueur du nom est < 6
00401A64  .     PUSH EBX                             lParam
00401A65  .     PUSH 00401740                        DlgProc = defi_Car.00401740
00401A6A  .     PUSH EBX                             hOwner
00401A6B  .     PUSH 9                               pTemplate = 9
00401A6D  .     PUSH EAX                             hInst => 00400000
00401A6E  .     CALL DWORD PTR DS:[<&USER32.Dialog    DialogBoxParamA
00401A74  .v    JMP 00401B82
00401A79  >     PUSH 100
00401A7E  .     LEA ECX,DWORD PTR SS:[ESP+150]
00401A85  .     PUSH ECX
00401A86  .     PUSH 3EA
00401A8B  .     PUSH EDI
00401A8C  .     CALL ESI                             ; Appel de la fonction GetDlgItemText
00401A8E  .     LEA EDX,DWORD PTR SS:[ESP+14C]       ; Adresse du serial
00401A95  .     PUSH EDX
00401A96  .     LEA ECX,DWORD PTR SS:[ESP+18]
00401A9A  .     CALL 00401920                        ; Compte le nombre de caractere du serial
00401A9F  .     CMP DWORD PTR SS:[ESP+28],0C         ; Le serial doit faire 12 caracteres
00401AA4  .v    JE SHORT 00401AC0
00401AA6  .     MOV EAX,DWORD PTR DS:[409400]        ; Affiche une boite d'erreur disant que le cle n'est pas bonne
00401AAB  .     PUSH EBX                             lParam
00401AAC  .     PUSH 00401740                        DlgProc = defi_Car.00401740
00401AB1  .     PUSH EBX                             hOwner
00401AB2  .     PUSH 6D                              pTemplate = 6D
00401AB4  .     PUSH EAX                             hInst => 00400000
00401AB5  .     CALL DWORD PTR DS:[<&USER32.Dialog    DialogBoxParamA
00401ABB  .v    JMP 00401B82
00401AC0  >     SUB ESP,1C
00401AC3  .     LEA EDX,DWORD PTR SS:[ESP+68]        ; Adresse du nom
00401AC7  .     MOV ECX,ESP
00401AC9  .     MOV DWORD PTR SS:[ESP+28],ESP
00401ACD  .     PUSH EDX
00401ACE  .     CALL 004018E0
00401AD3  .     SUB ESP,1C
00401AD6  .     LEA EAX,DWORD PTR SS:[ESP+184]       ; Adresse du serial
```

Les commentaires devraient permettre de comprendre facilement ce que fait le code.
Il n’y a aucune difficulté particulière, on continue plus bas pour voir ce qu’il se passe.

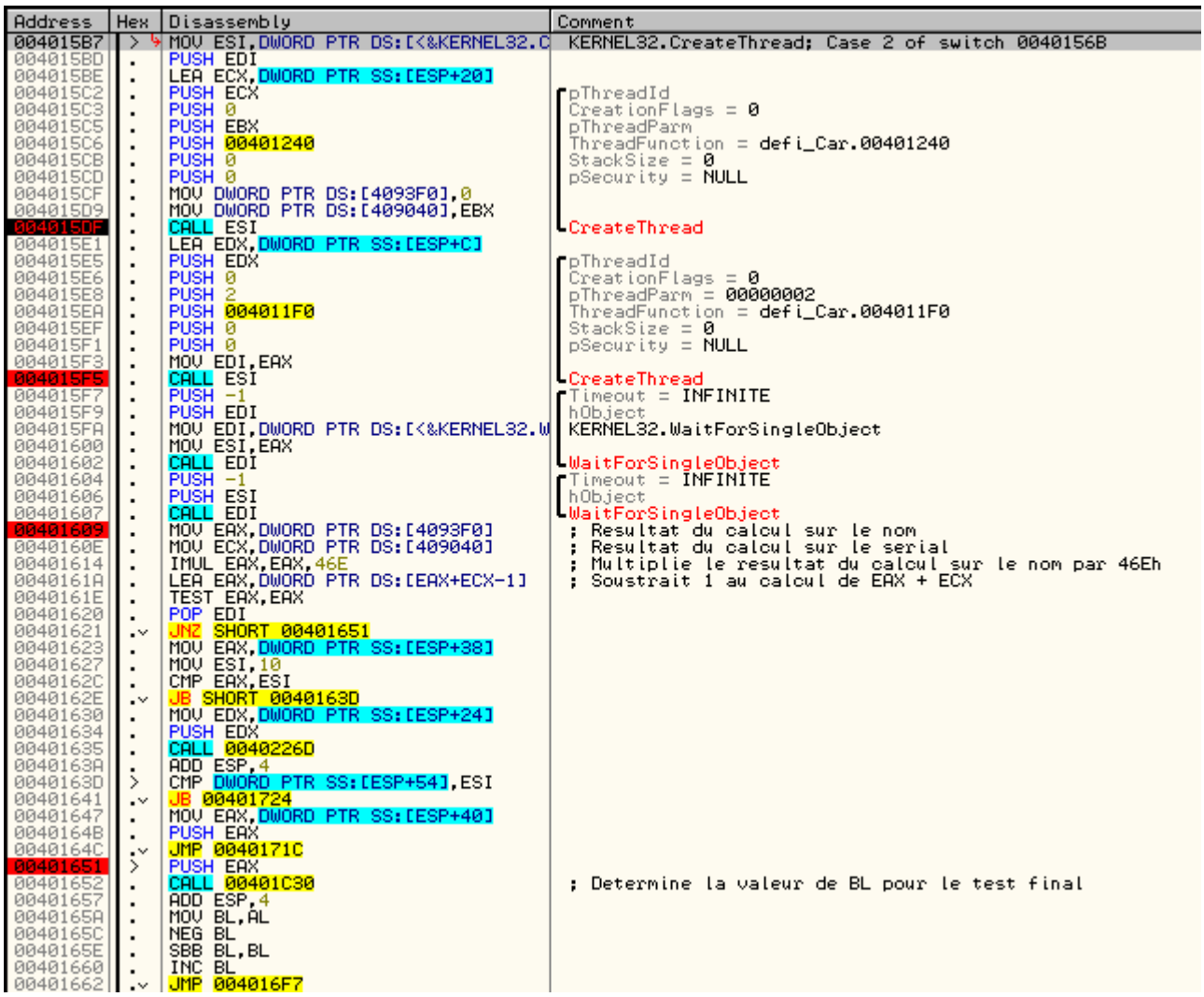
```
Address  Hex  Disassembly                          Comment
00401AD6  .   LEA EAX,DWORD PTR SS:[ESP+184]       ; Adresse du serial
00401ADD  .   MOV ECX,ESP
00401ADF  .   MOV DWORD PTR SS:[ESP+48],ESP
00401AE3  .   PUSH EAX
00401AE4  .   MOV BYTE PTR SS:[ESP+294],2
00401AEC  .   CALL 004018E0
00401AF1  .   PUSH 1                               ; passe 1 en parametre
00401AF3  .   MOV BYTE PTR SS:[ESP+294],1
00401AFB  .   CALL 00401520                        ; Fonction d'anti debug
00401B00  .   ADD ESP,3C
00401B03  .   CMP AL,BL                            ; Test de verification de l'antidebug
00401B05  .v  JNZ SHORT 00401B7B                   ; (ne pas tracer le call pour ne pas tomber dans l'anti-debug)
00401B07  .   SUB ESP,1C
00401B0A  .   LEA EDX,DWORD PTR SS:[ESP+68]        ; Adresse du nom
00401B0E  .   MOV ECX,ESP
00401B10  .   MOV DWORD PTR SS:[ESP+2C],ESP
00401B14  .   PUSH EDX
00401B15  .   CALL 004018E0
00401B1A  .   SUB ESP,1C
00401B1D  .   LEA EAX,DWORD PTR SS:[ESP+184]       ; Adresse du serial
00401B24  .   MOV ECX,ESP
00401B26  .   MOV DWORD PTR SS:[ESP+44],ESP
00401B2A  .   PUSH EAX
00401B2B  .   MOV BYTE PTR SS:[ESP+294],3
00401B33  .   CALL 004018E0
00401B38  .   PUSH 2                               ; passe 2 en parametre
00401B3A  .   MOV BYTE PTR SS:[ESP+294],1
00401B42  .   CALL 00401520                        ; Fonction de verification (meme fonction que l'antidebug)
00401B47  .   ADD ESP,3C
00401B4A  .   CMP AL,BL
00401B4C  .   PUSH EBX                             lParam
00401B4D  .v  JE SHORT 00401B66                    ; Comparaison finale ! ! !
00401B4F  .   MOV ECX,DWORD PTR DS:[409400]        defi_Car.00400000
00401B55  .   PUSH 00401740                        DlgProc = defi_Car.00401740
00401B5A  .   PUSH EBX                             hOwner
00401B5B  .   PUSH 6D                              pTemplate = 6D
00401B5D  .   PUSH ECX                             hInst => 00400000
00401B5E  .   CALL DWORD PTR DS:[<&USER32.Dialog   DialogBoxParamA
00401B64  .v  JMP SHORT 00401B82
00401B66  >   MOV EDX,DWORD PTR DS:[409400]        defi_Car.00400000
00401B6C  .   PUSH 00401740                        DlgProc = defi_Car.00401740
00401B71  .   PUSH EBX                             hOwner
00401B72  .   PUSH 6E                              pTemplate = 6E
00401B74  .   PUSH EDX                             hInst => 00400000
00401B75  .   CALL DWORD PTR DS:[<&USER32.Dialog   DialogBoxParamA
00401B7B  >   PUSH EDI                             hWnd
00401B7C  .   CALL DWORD PTR DS:[<&USER32.Destro   DestroyWindow
```

En 401AFB, la fonction prend 1 en paramètre.
Si vous tracer le code de la fonction, vous verrez qu'il y a un anti-debug basé sur la fonction CreateThread et Sleep.
Cette fonction permet de savoir que l'application est en cours de débogage.
Le fonctionnement est basé sur un timeout du à la lenteur de l'exécution du code depuis le debugger (ce qui n'arrive pas en exécution normale)
Pour ne pas tomber dans le piège, exécutez la fonction avec F8.

Sinon vous aurez le droit à ceci :

En 401B42, on a un appel à la même fonction que ci-dessus mais avec la valeur 2 en paramètre.
Cette valeur 2 va lancer le « vrai » code de vérification.

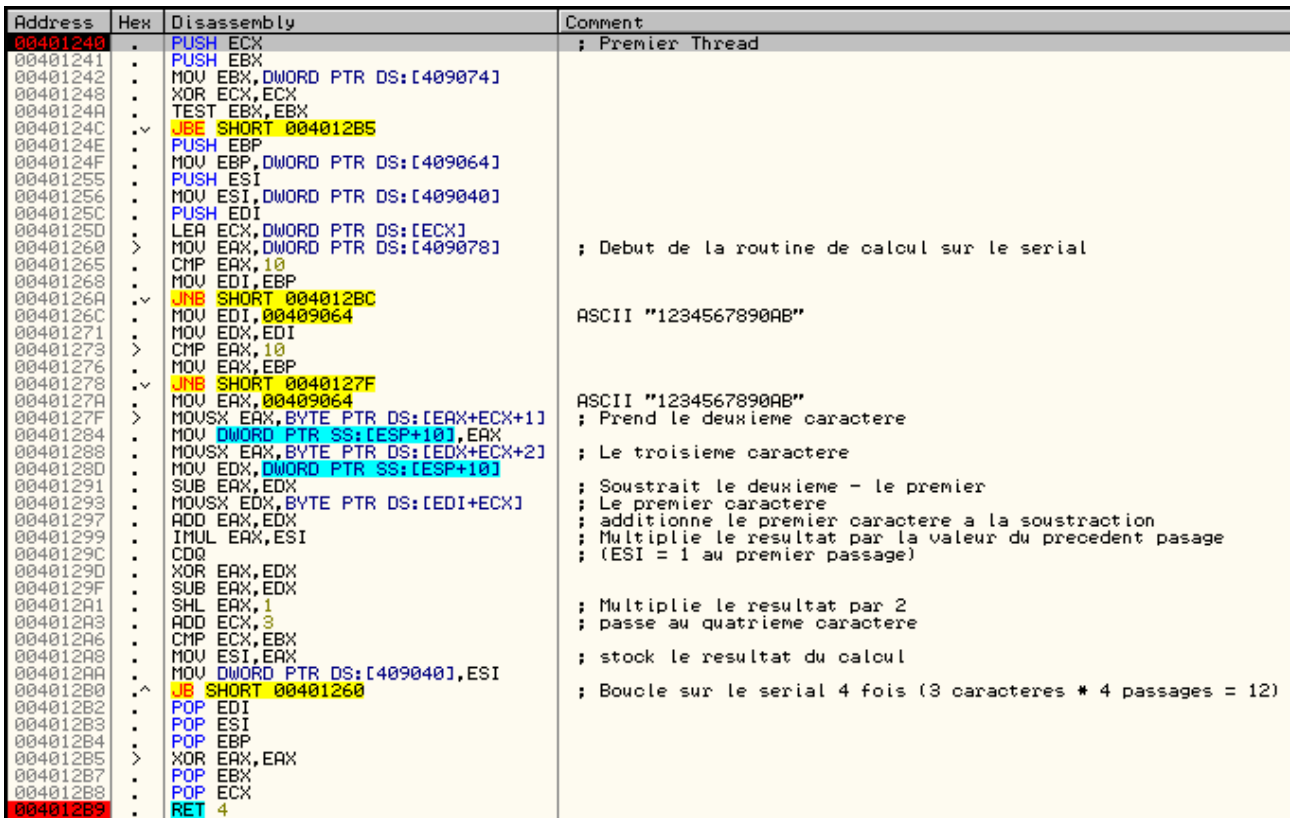
Allons voir ce qui ce passe dans cette fonction.



```
Address   Hex  Disassembly                          Comment
004015B7  > ↳  MOV ESI,DWORD PTR DS:[<&KERNEL32.C   KERNEL32.CreateThread; Case 2 of switch 0040156B
004015BD  .    PUSH EDI
004015BE  .    LEA ECX,DWORD PTR SS:[ESP+20]
004015C2  .    PUSH ECX                             pThreadId
004015C3  .    PUSH 0                               CreationFlags = 0
004015C5  .    PUSH EBX                             pThreadParm
004015C6  .    PUSH 00401240                        ThreadFunction = defi_Car.00401240
004015CB  .    PUSH 0                               StackSize = 0
004015CD  .    PUSH 0                               pSecurity = NULL
004015CF  .    MOV DWORD PTR DS:[4093F0],0
004015D9  .    MOV DWORD PTR DS:[409040],EBX
004015DF  .    CALL ESI                             CreateThread
004015E1  .    LEA EDX,DWORD PTR SS:[ESP+C]
004015E5  .    PUSH EDX                             pThreadId
004015E6  .    PUSH 0                               CreationFlags = 0
004015E8  .    PUSH 2                               pThreadParm = 00000002
004015EA  .    PUSH 004011F0                        ThreadFunction = defi_Car.004011F0
004015EF  .    PUSH 0                               StackSize = 0
004015F1  .    PUSH 0                               pSecurity = NULL
004015F3  .    MOV EDI,EAX
004015F5  .    CALL ESI                             CreateThread
004015F7  .    PUSH -1                              Timeout = INFINITE
004015F9  .    PUSH EDI                             hObject
004015FA  .    MOV EDI,DWORD PTR DS:[<&KERNEL32.W   KERNEL32.WaitForSingleObject
00401600  .    MOV ESI,EAX
00401602  .    CALL EDI                             WaitForSingleObject
00401604  .    PUSH -1                              Timeout = INFINITE
00401606  .    PUSH ESI                             hObject
00401607  .    CALL EDI                             WaitForSingleObject
00401609  .    MOV EAX,DWORD PTR DS:[4093F0]        ; Resultat du calcul sur le nom
0040160E  .    MOV ECX,DWORD PTR DS:[409040]        ; Resultat du calcul sur le serial
00401614  .    IMUL EAX,EAX,46E                     ; Multiplie le resultat du calcul sur le nom par 46Eh
0040161A  .    LEA EAX,DWORD PTR DS:[EAX+ECX-1]     ; Soustrait 1 au calcul de EAX + ECX
0040161E  .    TEST EAX,EAX
00401620  .    POP EDI
00401621  .v   JNZ SHORT 00401651
00401623  .    MOV EAX,DWORD PTR SS:[ESP+38]
00401627  .    MOV ESI,10
0040162C  .    CMP EAX,ESI
0040162E  .v   JB SHORT 0040163D
00401630  .    MOV EDX,DWORD PTR SS:[ESP+24]
00401634  .    PUSH EDX
00401635  .    CALL 0040226D
0040163A  .    ADD ESP,4
0040163D  >    CMP DWORD PTR SS:[ESP+54],ESI
00401641  .v   JB 00401724
00401647  .    MOV EAX,DWORD PTR SS:[ESP+40]
0040164B  .    PUSH EAX
0040164C  .v   JMP 0040171C
00401651  >    PUSH EAX
00401652  .    CALL 00401C30                        ; Determine la valeur de BL pour le test final
00401657  .    ADD ESP,4
0040165A  .    MOV BL,AL
0040165C  .    NEG BL
0040165E  .    SBB BL,BL
00401660  .    INC BL
00401662  .v   JMP 004016F7
```

On voit que deux threads vont être crées. On va donc mettre un point d’arrêt sur le début de la fonction des 2 threads à savoir 401240 et 4011F0.

Allons en 401240



```
Address  Hex  Disassembly                          Comment
00401240  .   PUSH ECX                             ; Premier Thread
00401241  .   PUSH EBX
00401242  .   MOV EBX,DWORD PTR DS:[409074]
00401248  .   XOR ECX,ECX
0040124A  .   TEST EBX,EBX
0040124C  .v  JBE SHORT 004012B5
0040124E  .   PUSH EBP
0040124F  .   MOV EBP,DWORD PTR DS:[409064]
00401255  .   PUSH ESI
00401256  .   MOV ESI,DWORD PTR DS:[409040]
0040125C  .   PUSH EDI
0040125D  .   LEA ECX,DWORD PTR DS:[ECX]
00401260  >   MOV EAX,DWORD PTR DS:[409078]      ; Debut de la routine de calcul sur le serial
00401265  .   CMP EAX,10
00401268  .   MOV EDI,EBP
0040126A  .v  JNB SHORT 004012BC
0040126C  .   MOV EDI,00409064                   ASCII "1234567890AB"
00401271  .   MOV EDX,EDI
00401273  >   CMP EAX,10
00401276  .   MOV EAX,EBP
00401278  .v  JNB SHORT 0040127F
0040127A  .   MOV EAX,00409064                   ASCII "1234567890AB"
0040127F  >   MOVSX EAX,BYTE PTR DS:[EAX+ECX+1]  ; Prend le deuxieme caractere
00401284  .   MOV DWORD PTR SS:[ESP+10],EAX
00401288  .   MOVSX EAX,BYTE PTR DS:[EDX+ECX+2]  ; Le troisieme caractere
0040128D  .   MOV EDX,DWORD PTR SS:[ESP+10]
00401291  .   SUB EAX,EDX                        ; Soustrait le deuxieme - le premier
00401293  .   MOVSX EDX,BYTE PTR DS:[EDI+ECX]    ; Le premier caractere
00401297  .   ADD EAX,EDX                        ; additionne le premier caractere a la soustraction
00401299  .   IMUL EAX,ESI                       ; Multiplie le resultat par la valeur du precedent pasage
0040129C  .   CDQ                                ; (ESI = 1 au premier passage)
0040129D  .   XOR EAX,EDX
0040129F  .   SUB EAX,EDX
004012A1  .   SHL EAX,1                          ; Multiplie le resultat par 2
004012A3  .   ADD ECX,3                          ; passe au quatrieme caractere
004012A6  .   CMP ECX,EBX
004012A8  .   MOV ESI,EAX                        ; stock le resultat du calcul
004012AA  .   MOV DWORD PTR DS:[409040],ESI
004012B0  .^  JB SHORT 00401260                  ; Boucle sur le serial 4 fois (3 caracteres * 4 passages = 12)
004012B2  .   POP EDI
004012B3  .   POP ESI
004012B4  .   POP EBP
004012B5  >   XOR EAX,EAX
004012B7  .   POP EBX
004012B8  .   POP ECX
004012B9  .   RET 4
```

On est dans le premier thread.
C’est ici que l’on va faire le calcul sur le serial.

L’algorithme est expliqué dans les commentaires.

Sinon, voici ce que cela donne :
1234567890AB

On prend le troisième caractère : 3
On le soustrait au deuxième : 2
On ajoute le premier : 1
On multiplie le tout par le résultat précédent (1 au départ)
Et on multiplie le tout pas 2.

Comme les caractères sont des caractères ASCII, cela donne :
(33 – 32) + 31 = 32 * 1 = 32 * 2 = 64

Et on boucle sur le serial jusqu'au dernier caractère.

A l’adresse 409040, on a le résultat final du calcul.

Allons voir le deuxième thread.

```
Address   Hex   Disassembly                          Comment
004011F0  .     MOV EDX,DWORD PTR DS:[409058]        ; Deuxieme thread
004011F6  .     XOR EAX,EAX
004011F8  .     TEST EDX,EDX
004011FA  .v    JBE SHORT 00401236
004011FC  .     PUSH EBX
004011FD  .     PUSH ESI
004011FE  .     MOV ESI,DWORD PTR DS:[409048]        ; Le dword sur le nom
00401204  .     PUSH EDI
00401205  .     MOV EDI,DWORD PTR DS:[40905C]
0040120B  .v    JMP SHORT 00401210
0040120D        LEA ECX,DWORD PTR DS:[ECX]
00401210  >     CMP EDI,10
00401213  .     MOV ECX,ESI
00401215  .v    JNB SHORT 0040121C
00401217  .     MOV ECX,00409048                     ASCII "skirby"
0040121C  >     MOVSX ECX,BYTE PTR DS:[ECX+EAX]      ; Prend un caractere
00401220  .     MOV EBX,DWORD PTR DS:[4093F0]        ; Valeur de sauvegarde
00401226  .     ADD EBX,ECX                          ; Fait la somme des caracteres du nom
00401228  .     INC EAX
00401229  .     CMP EAX,EDX                          ; Test la fin de chaine de caractere
0040122B  .     MOV DWORD PTR DS:[4093F0],EBX        ; Resultat final du calcul sur le nom
00401231  .^    JB SHORT 00401210
00401233  .     POP EDI
00401234  .     POP ESI
00401235  .     POP EBX
00401236  >     XOR EAX,EAX
00401238  .     RET 4
```

Ici, rien de plus simple. On fait tout simplement la somme des caractères du nom.
L'adresse 4093F0 contient le résultat du calcul final.

skirby donne 294h soit 660 en décimal.
s + k + i + r + b + y = 73 + 6B + 69 + 72 + 62 + 79 = 294

On retourne dans la fonction qui a créée les deux threads.

```
Address   Hex   Disassembly                          Comment
004015B7  > ?   MOV ESI,DWORD PTR DS:[<&KERNEL32.C   KERNEL32.CreateThread; Case 2 of switch 0040156B
004015BD  .     PUSH EDI
004015BE  .     LEA ECX,DWORD PTR SS:[ESP+20]
004015C2  .     PUSH ECX                             pThreadId
004015C3  .     PUSH 0                               CreationFlags = 0
004015C5  .     PUSH EBX                             pThreadParm
004015C6  .     PUSH 00401240                        ThreadFunction = defi_Car.00401240
004015CB  .     PUSH 0                               StackSize = 0
004015CD  .     PUSH 0                               pSecurity = NULL
004015CF  .     MOV DWORD PTR DS:[4093F0],0
004015D9  .     MOV DWORD PTR DS:[409040],EBX
004015DF  .     CALL ESI                             CreateThread
004015E1  .     LEA EDX,DWORD PTR SS:[ESP+C]
004015E5  .     PUSH EDX                             pThreadId
004015E6  .     PUSH 0                               CreationFlags = 0
004015E8  .     PUSH 2                               pThreadParm = 00000002
004015EA  .     PUSH 004011F0                        ThreadFunction = defi_Car.004011F0
004015EF  .     PUSH 0                               StackSize = 0
004015F1  .     PUSH 0                               pSecurity = NULL
004015F3  .     MOV EDI,EAX
004015F5  .     CALL ESI                             CreateThread
004015F7  .     PUSH -1                              Timeout = INFINITE
004015F9  .     PUSH EDI                             hObject
004015FA  .     MOV EDI,DWORD PTR DS:[<&KERNEL32.W   KERNEL32.WaitForSingleObject
00401600  .     MOV ESI,EAX
00401602  .     CALL EDI                             WaitForSingleObject
00401604  .     PUSH -1                              Timeout = INFINITE
00401606  .     PUSH ESI                             hObject
00401607  .     CALL EDI                             WaitForSingleObject
00401609  .     MOV EAX,DWORD PTR DS:[4093F0]        ; Resultat du calcul sur le nom
0040160E  .     MOV ECX,DWORD PTR DS:[409040]        ; Resultat du calcul sur le serial
00401614  .     IMUL EAX,EAX,46E                     ; Multiplie le resultat du calcul sur le nom par 46Eh
0040161A  .     LEA EAX,DWORD PTR DS:[EAX+ECX-1]     ; Soustrait 1 au calcul de EAX + ECX
0040161E  .     TEST EAX,EAX
00401620  .     POP EDI
00401621  .v    JNZ SHORT 00401651
00401623  .     MOV EAX,DWORD PTR SS:[ESP+38]
00401627  .     MOV ESI,10
0040162C  .     CMP EAX,ESI
0040162E  .v    JB SHORT 0040163D
00401630  .     MOV EDX,DWORD PTR SS:[ESP+24]
00401634  .     PUSH EDX
00401635  .     CALL 0040226D
0040163A  .     ADD ESP,4
0040163D  >     CMP DWORD PTR SS:[ESP+54],ESI
00401641  .v    JB 00401724
00401647  .     MOV EAX,DWORD PTR SS:[ESP+40]
0040164B  .     PUSH EAX
0040164C  .v    JMP 0040171C
00401651  >     PUSH EAX
00401652  .     CALL 00401C30                        ; Determine la valeur de BL pour le test final
00401657  .     ADD ESP,4
0040165A  .     MOV BL,AL
0040165C  .     NEG BL
0040165E  .     SBB BL,BL
00401660  .     INC BL
00401662  .v    JMP 004016F7
```

En 4016E7, le code attend la fin de l'exécution des deux threads.

En 401609, le programme reprend le cours normal de son exécution.

On a donc le calcul suivant :
(Le résultat du calcul sur le nom * 46Eh) + Le résultat du calcul sur le serial - 1

Le – 1 à son importance.
En effet, le calcul sur le serial est forcément pair car multiplié par 2.
Le calcul sur le nom est aussi pair car multiplié par un nombre pair.

Le – 1 donne forcément un nombre impair.

En 401652, le programme lance la vérification sur la valeur précédemment calculée.

Allons voir ce qui ce passe dans cette fonction.

```
Address   Hex   Disassembly                    Comment
00401C30  r$    PUSH EDI
00401C31  .     MOV EDI,DWORD PTR SS:[ESP+8]
00401C35  .     CMP EDI,2                      ; Si la valeur du parametre vaut 2 on sort avec AL = 1 (OK)
00401C38  .v    JNZ SHORT 00401C3E
00401C3A  .     MOV AL,1
00401C3C  .     POP EDI
00401C3D  .     RET
00401C3E  >     MOV EAX,EDI
00401C40  .     AND EAX,80000001               ; Test si la valeur est signee
00401C45  .v    JNS SHORT 00401C4C             ; On sort avec AL = 0 si c'est la cas (Erreur)
00401C47  .     DEC EAX
00401C48  .     OR EAX,FFFFFFFE
00401C4B  .     INC EAX
00401C4C  >v    JNZ SHORT 00401C52
00401C4E  .     XOR AL,AL
00401C50  .     POP EDI
00401C51  .     RET
00401C52  >     PUSH EBX
00401C53  .     MOV ECX,EDI                    ; ECX = valeur passee en parametre
00401C55  .     PUSH ESI
00401C56  .     MOV ESI,3                      ; ESI initialise a 3
00401C5B  .     XOR ECX,1                      ; Rend ECX pair
00401C5E  .     CMP ECX,ESI                    ; Test si ECX = 3 (si oui, on sort avec BL = 1)
00401C60  .     MOV BL,1                       ; BL doit valoir 1 pour sortir sans erreur de la fonction
00401C62  .v    JL SHORT 00401C76
00401C64  >     MOV EAX,EDI                    ; EAX = la valeur passee en parametre
00401C66  .     CDQ
00401C67  .     IDIV ESI                       ; Divise la valeur par 3
00401C69  .     TEST EDX,EDX                   ; Test le reste de la division
00401C6B  .v    JNZ SHORT 00401C6F
00401C6D  .     XOR BL,BL                      ; Si le reste est egal a 0 on sort en erreur
00401C6F  >     ADD ESI,2                      ; ESI = ESI + 2 (on va diviser qu'avec des nombres impairs)
00401C72  .     CMP ESI,ECX                    ; Test si ESI est <= a ECX
00401C74  .^    JLE SHORT 00401C64             ; On boucle tant que ESI <= ECX
00401C76  >     POP ESI
00401C77  .     MOV AL,BL
00401C79  .     POP EBX
00401C7A  .     POP EDI
00401C7B  L.    RET
```

Pour faire simple, si vous avez compris le code avec les commentaires, on va vérifier que le nombre passé en paramètre est un nombre premier.

Donc pour résumer, il faut que :
La somme des caractères du nom*46Eh + le résultat du calcul sur le serial – 1 = Nombre premier.

Comment programmer ça ?
Bah en fait je ne sais pas trop. J'ai beaucoup réfléchie, écris des équations, je me suis torturer le cerveau mais je n'ai rien trouvé d'intelligent.

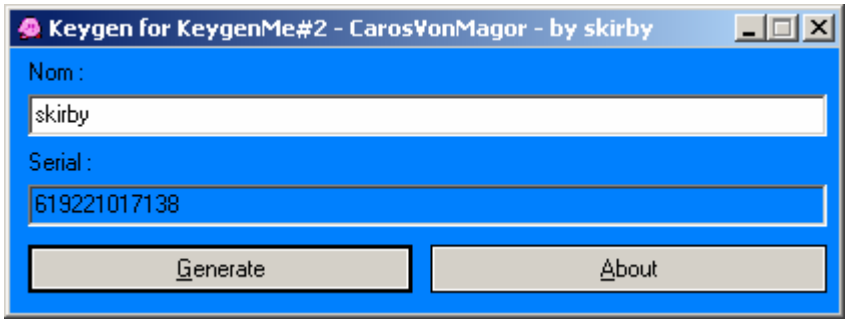
Pour finir, je suis parti sur une méthode de brute force qui je dois le dire donne des résultats tout à fait satisfaisant.
On trouve un serial en moins de 2 secondes.

Voici le code source expliqué :

```
#include <windows.h>
#include <stdio.h>

void Keygen(char* sStrName, char* sStrSerial) {
  char sSerial[20];

  int i;
  unsigned long lName = 0, x = 1, lFinal;

  // Fait la somme des caractères du nom
  for (i = 0; i < (int)strlen(sStrName); i++) {
    lName += sStrName[i];
  }
  lName *= 0x46E; // Multiplie le résultat par 46E

  srand(GetTickCount());  // Initialise le moteur de générateur de nombre aléatoire

  // On va boucle jusqu'a ce que le nombre soit premier
  do {
    // Génération aléatoire du serial
    for (i = 0; i < 12; i++) {
      // Génère un caractère aléatoire compris entre 0 et 9
      sSerial[i] = 0x30 + (rand() % 10);
    }
    sSerial[i] = 0;   // Met le zéro terminale à la fin de la chaine

    // Fait le calcul sur le serial
    unsigned long lSerial = 1, lTmp = 1;
    unsigned char c1, c2, c3;
    for (i = 0; i < (int)strlen(sSerial); i += 3) {
      c1 = sSerial[i];
      c2 = sSerial[i+1];
      c3 = sSerial[i+2];
      lTmp = (((c3 - c2) + c1));
      lSerial *= lTmp;
      lSerial *= 2;
    }

    // Ici on a donc le résultat final
    lFinal = lName + lSerial - 1;

    // Test si le résultat est un nombre premier
    for (x = 2; x < lFinal; x++) {
      if (lFinal % x == 0) {
        break;
      }
    }
  }
  while (x != lFinal);  // On boucle tant que le résultat n'est pas un nombre premier

  strcpy(sStrSerial, sSerial);

  return;
}

int main(void) {
  char szName[MAX_PATH], szSerial[15];

        printf("Entrez votre nom (6 caracteres minimum) : ");
        gets(szName);   // Récupère le nom

        if (strlen(szName) < 6) {
               printf("Le nom doit faire au moins 6 caracteres ! ! !\n");
        }
        else {
    Keygen(szName, szSerial);
        }

  printf("Le serial est : %s\n\n", szSerial);  // Affiche un serial

  system("PAUSE");
  return 0;
}
```

Vous pouvez compiler ce code source avec n'importe quel compilateur C.
Je vous conseille cette excellente IDE : Code::Blocks
Elle est fournie avec le compilateur GCC et MinGW pour faire des programmes Windows.

Voici une liste de serials fonctionnels :



BeatriX : 155539655876
baboon : 801252505514
Guetta : 241424748255
sKiller : 330458769681

Les autres ne soyez pas jaloux.
Utiliser le keygen et faites vous pleins de clés valides ! ! !

J’espère avoir été le plus clair possible.
Si vous avez des questions, n’hésitez pas à venir les poser sur FORUMCRACK.

skirby