

Date : 2005-12-08

Type : Tutorial

Auteur : Requiem

Keygenme : Crapouillette by Elooo for Infoshackers



0. Introduction

Nous allons aborder un défi codé par Elooo, c'est un keygenme qui a la particularité de nous proposer de nombreux points à étudier, dont une touche crypto (très très simple ici) et les instructions FPU par exemple.

Avant d'aller plus l'exé est packé mais cela ne doit pas nous poser de problème, nous pourrions nous servir du plugin OllyDump (sans reconstruction d'imports) puis ImpRec pour obtenir un dump parfaitement fonctionnel, sur lequel nous pourrions entamer notre analyse. Nous aborderons chacune des difficultés présentes dans le keygenme dans l'ordre où elles apparaissent dans le code.

1. L'anti-debug

| | | | |
|----------|------------------|-------------------------------|--|
| 004021AE | 60 | PUSHAD | |
| 004021AF | E8 36020000 | CALL Crapouil.004023EA | JMP to kernel32.GetTickCount |
| 004021B4 | 50 | PUSH EAX | |
| 004021B5 | 8B15 217A4000 | MOV EDX,DWORD PTR DS:[407A21] | Valeur précédente d'un GetTickCount |
| 004021B8 | 2BC2 | SUB EAX,EDX | |
| 004021BD | 83F8 10 | CMP EAX,10 | Si l'écart est trop important, on arme un flag |
| 004021C0 | 7C 0C | JL SHORT Crapouil.004021CE | |
| 004021C2 | C605 30764000 00 | MOV BYTE PTR DS:[407630],1 | |
| 004021C9 | E8 08000000 | CALL Crapouil.004021D6 | |
| 004021CE | 8F05 217A4000 | POP DWORD PTR DS:[407A21] | |
| 004021D4 | 61 | POPAD | |
| 004021D5 | C3 | RETN | |

L'API `kernel32.GetTickCount` renvoie le nombre de millisecondes écoulées depuis le démarrage du système, un reverser débugeant le programme va bien entendu augmenter de manière considérable le temps d'exécution, et l'écart entre le premier appel à cette API et les suivant sortira de la limite de 10ms fixée.

La faiblesse de cette protection ici, c'est qu'elle n'est pas soutenue par une fonction chargée de vérifier l'intégrité du code donc nous allons tout simplement la patcher (remplacer `sub eax, edx`, par `sub eax, eax`). Dans l'archive se trouve un exe unpacké et patché (au passage j'ai aussi stoppé la musique pour pouvoir travailler plus tranquillement).

2. Création d'un identifiant unique

Le programme va générer une sorte d'identifiant pour l'utilisateur, celui-ci dépend de caractéristiques intrinsèques de l'ordinateur mais aussi du name entré dans le keygenme.

- Le volume serial number :

```
00401C41 E8 AA070000 CALL <JMP.&kernel32.GetVolumeInformation>
00401C46 8B15 657B4000 MOV EDX,DWORD PTR DS:[407B65]
00401C4C 8915 20764000 MOV DWORD PTR DS:[407620],EDX
00401C52 8BC2 MOV EAX,EDX
00401C54 C1E8 10 SHR EAX,10
00401C57 25 FFFF0000 AND EAX,0FFFF
00401C5C 32C6 XOR AL,DH
00401C5E 32E2 XOR AH,DL
00401C60 C1CA 03 ROR EDX,3
00401C63 81E2 0000FFFF AND EDX,0FFFF
00401C69 03C2 ADD EAX,EDX
00401C6B 68 657B4000 PUSH Crapouil.00407B65
00401C70 50 PUSH EAX
00401C71 E8 96060000 CALL Crapouil.0040230C
```

L'API `Kernel32.GetVolumeInformation` nous permet justement d'accéder au **volume serial number**, c'est une sorte de numéro qui identifie votre disque dur. Le code ci-dessus reprend ce numéro dans EDX et lui impose une série de manipulations, avant de transformer le résultat en string au format décimal (signé).

- Le nom de l'ordinateur :

```
00401BD1 E8 08080000 CALL <JMP.&kernel32.GetComputerNameA>
```

Cette API porte un nom assez explicite, elle retrouve tout simplement le nom attribué à l'ordinateur, nous ne nous attarderons pas dessus.

- Le name fournit par l'utilisateur :

Le troisième élément pris en compte est le name, il devra répondre à certains critères, en particulier sa taille qui devra être comprise entre 2 et 19d caractères. Sa récupération se fait au moyen de l'API `GetDlgItemText`.

Une fois nos trois éléments obtenus nous allons les concaténer (chacun étant sous forme de string), puis hashé la chaîne pour obtenir l'identifiant :

```
00402015 68 DC704000 PUSH Crapouil.0040700C
0040201A 68 257A4000 PUSH Crapouil.00407A25
0040201F E8 F6030000 CALL <JMP.&kernel32.lstrcpy>
00402024 68 657B4000 PUSH Crapouil.00407B65
00402029 68 257A4000 PUSH Crapouil.00407A25
0040202E E8 E1030000 CALL <JMP.&kernel32.lstrcat>
00402033 68 ED7A4000 PUSH Crapouil.00407AED
00402038 68 257A4000 PUSH Crapouil.00407A25
0040203D E8 D2030000 CALL <JMP.&kernel32.lstrcat>
00402042 68 257A4000 PUSH Crapouil.00407A25
00402047 E8 D4030000 CALL <JMP.&kernel32.lstrlen>
0040204C 68 157B4000 PUSH Crapouil.00407B15
00402051 50 PUSH EAX
00402052 68 257A4000 PUSH Crapouil.00407A25
00402057 E8 A4EFFFFF CALL Crapouil.00401000
ASCII "Requiem"
ASCII "Requiem-468957388BODDINGTONS"
ASCII "-468957388"
ASCII "Requiem-468957388BODDINGTONS"
ASCII "BODDINGTONS"
ASCII "Requiem-468957388BODDINGTONS"
ASCII "Requiem-468957388BODDINGTONS"
ASCII "Requiem-468957388BODDINGTONS"
le hash MD5
```

Les trois éléments sont clairement visibles, maintenant il nous faut déterminer la fonction en 401000.

```

MOV DWORD PTR DS:[EDI-8],EAX
MOV DWORD PTR DS:[EDI-4],EDX
MOV EDX,DWORD PTR SS:[EBP+C]
MOV EDI,DWORD PTR SS:[EBP+8]
MOV ESI,DWORD PTR SS:[EBP+10]
MOV DWORD PTR DS:[ESI],76534201
MOV DWORD PTR DS:[ESI+4],ABDCEF89
MOV DWORD PTR DS:[ESI+8],98FECDBA
MOV DWORD PTR DS:[ESI+C],10243567
MOV EAX,DWORD PTR DS:[ESI]
MOV DWORD PTR SS:[EBP-4],EAX
MOV EAX,DWORD PTR DS:[ESI+4]
MOV DWORD PTR SS:[EBP-8],EAX
MOV EAX,DWORD PTR DS:[ESI+8]
MOV DWORD PTR SS:[EBP-C],EAX
MOV EAX,DWORD PTR DS:[ESI+C]
MOV DWORD PTR SS:[EBP-10],EAX
PUSH D76AA478
PUSH 7
PUSH DWORD PTR DS:[EDI]
PUSH DWORD PTR SS:[EBP-10]
PUSH DWORD PTR SS:[EBP-C]
PUSH DWORD PTR SS:[EBP-8]
PUSH DWORD PTR SS:[EBP-4]
CALL Crapouil.00401862
MOV DWORD PTR SS:[EBP-4],EAX
PUSH E8C7B756
PUSH 0C
PUSH DWORD PTR DS:[EDI+4]
PUSH DWORD PTR SS:[EBP-C]
PUSH DWORD PTR SS:[EBP-8]
PUSH DWORD PTR SS:[EBP-4]
PUSH DWORD PTR SS:[EBP-10]
CALL Crapouil.00401862
MOV DWORD PTR SS:[EBP-10],EAX
PUSH 242070DB
PUSH 11
PUSH DWORD PTR DS:[EDI+8]
PUSH DWORD PTR SS:[EBP-8]
PUSH DWORD PTR SS:[EBP-4]
PUSH DWORD PTR SS:[EBP-10]
PUSH DWORD PTR SS:[EBP-C]
CALL Crapouil.00401862

```

Nous voyons une initialisation de quatre dword suivi de 'round', nous reconnaissons donc un hash 128bits, en revanche les valeurs d'initialisation ne sont pas « traditionnelles ». Le plus courant des hash est le MD5, et en effet c'est bien cet algorithme qui est présent.

En temps normal nous pouvons trouver pour l'initialisation :

067452301h
0efcdab89h
098badcfeh
010325476h

Il ne faudra pas oublier de corriger ces valeurs dans notre keygen et bien reprendre les même que celles qu'a utilisé **Elooo**.

Au passage je signale que l'auteur de cette source est **Kush**, je l'utilise aussi dans mon keygen, merci à lui.

Le résultat du hash forme notre identifiant, reste maintenant à voir comment se forme notre serial.

3. Vérification du serial

Notre identifiant subit plusieurs transformation dont une à l'aide d'une table de dword présente en mémoire. Cette table va être dans un premier temps 'cryptée' grâce à la longueur de la chaîne de caractère que nous avons passée au hash :

| | | | |
|----------|-----------------|-------------------------------------|--|
| 0040206F | 33F6 | XOR ESI,ESI | initialement EAX = longueur de la chaîne hashée La table comprendra 184 dword pointe sur notre table simple Xor On crée une seconde table dword suivant |
| 00402071 | B9 B8000000 | MOV ECX,0B8 | |
| 00402076 | C1C8 02 | ROR EAX,2 | |
| 00402079 | 8B14B5 4D714000 | MOV EDX,DWORD PTR DS:[ESI*4+40714D] | |
| 00402080 | 33D0 | XOR EDX,EAX | |
| 00402082 | 8914B5 43764000 | MOV DWORD PTR DS:[ESI*4+407643],EDX | |
| 00402089 | 46 | INC ESI | |
| 0040208A | ^F2 FA | LOOPE SHORT Crapouil.00402076 | |

Chaque dword de la première table est *XOR*é avec une valeur qui découle de la valeur de la chaîne qui nous avons hashé (à chaque itération nous réalisons un ror dessus), cela permet la formation d'une seconde table.

Notre hash est stocké sous forme hexa et comprend donc 16 bytes, la prochaine fonction va manipuler ce hash avec des routines assez simples.

| | | | |
|----------|-----------------|-------------------------------------|--|
| 0040209F | B9 05000000 | MOV ECX,5 | |
| 004020A4 | 33F6 | XOR ESI,ESI | |
| 004020A6 | B8 20000000 | MOV EAX,20 | |
| 004020A8 | 99 | CDQ | |
| 004020AC | F7F9 | IDIV ECX | |
| 004020AE | 88FA | MOV EDI,EDX | |
| 004020B0 | 8A86 157B4000 | MOV AL,BYTE PTR DS:[ESI+407B15] | |
| 004020B6 | 3C 00 | CMP AL,0 | |
| 004020B8 | 74 4F | JE SHORT Crapouil.00402109 | |
| 004020BA | 83FE 04 | CMP ESI,4 | |
| 004020BD | 7D 09 | JGE SHORT Crapouil.004020C8 | |
| 004020BF | 3204BD 4D764000 | XOR AL,BYTE PTR DS:[EDI*4+40764D] | Ces adresses pointent sur une partie de la table cryptée |
| 004020C6 | EB 38 | JMP SHORT Crapouil.00402100 | |
| 004020C8 | 83FE 08 | CMP ESI,8 | |
| 004020CB | 7D 12 | JGE SHORT Crapouil.004020DF | |
| 004020CD | 8A14BD A8764000 | MOV DL,BYTE PTR DS:[EDI*4+4076A8] | Ces adresses pointent sur une partie de la table cryptée |
| 004020D4 | COCA 43 | ROR DL,43 | bytes 4 à 7 du hash |
| 004020D7 | B3 4D | MOV BL,4D | |
| 004020D9 | 32D3 | XOR DL,BL | |
| 004020DB | 32C2 | XOR AL,DL | |
| 004020DD | EB 21 | JMP SHORT Crapouil.00402100 | |
| 004020DF | 83FE 0C | CMP ESI,0C | |
| 004020E2 | 7D 0F | JGE SHORT Crapouil.004020F3 | |
| 004020E4 | 8A14BD 71764000 | MOV DL,BYTE PTR DS:[EDI*4+407671] | Ces adresses pointent sur une partie de la table cryptée |
| 004020EB | B3 0D | MOV BL,0D | |
| 004020ED | 2AD3 | SUB DL,BL | |
| 004020EF | 32C2 | XOR AL,DL | caractères 7 à 10 |
| 004020F1 | EB 0D | JMP SHORT Crapouil.00402100 | |
| 004020F3 | 8A14BD 87764000 | MOV DL,BYTE PTR DS:[EDI*4+407687] | Ces adresses pointent sur une partie de la table cryptée |
| 004020FA | B3 50 | MOV BL,50 | |
| 004020FC | 02D3 | ADD DL,BL | |
| 004020FE | 32C2 | XOR AL,DL | |
| 00402100 | 8886 3D7B4000 | MOV BYTE PTR DS:[ESI+407B3D],AL | |
| 00402106 | 46 | INC ESI | |
| 00402107 | EB A2 | JMP SHORT Crapouil.004020A8 | |
| 00402109 | 33FF | XOR EDI,EDI | |
| 0040210B | 33D2 | XOR EDX,EDX | |
| 0040210D | 0314BD 3D7B4000 | ADD EDX,DWORD PTR DS:[EDI*4+407B3D] | On additionne les 4 dword du hash |
| 00402114 | 47 | INC EDI | |
| 00402115 | 83FF 04 | CMP EDI,4 | |
| 00402118 | 7C F3 | JL SHORT Crapouil.0040210D | |
| 0040211A | 52 | PUSH EDX | |

Encore une fois il ne faut pas se fier aux apparences, il n'y a rien de très dur là dedans, nous avons une routine spécifique pour les bytes 0 à 3 puis une autre pour 4 à 7 et ainsi de suite, au total cela nous fait 4 routines différentes mais elles ne sont composées chacune que de quelques instructions. Nous pouvons aussi remarquer que chacune d'elle fait intervenir la table cryptée que nous avons vu précédemment. Au final nous obtenons un hash crypté duquel on déduit un simple dword par additions des quatre dword du hash crypté.

4. Fort Alamo

Notre connaissance de ce keygenme est maintenant bien avancée, il ne nous reste plus qu'à faire tomber les dernières barrières pour être en mesure de produire le keygen. Une dernière charge s'impose :

| | | | |
|----------|-----------------|-------------------------------|--|
| 0040212E | 9B | WAIT | |
| 0040212F | D8E2 | FCLEX | |
| 00402131 | D8E3 | FINIT | |
| 00402133 | D80424 | FILD DWORD PTR SS:[ESP] | Nous mettons le dword issu du hash sur la pile FPU |
| 00402136 | 58 | POP EAX | |
| 00402137 | DA25 31764000 | FISUB DWORD PTR DS:[407631] | soustraction d'une constante |
| 0040213D | D9EB | FLDPI | met le nombre pi sur le haut de la pile FPU |
| 0040213F | DEC9 | FMULP ST(1),ST | |
| 00402141 | DA35 35764000 | FIDIV DWORD PTR DS:[407635] | division par une autre constante |
| 00402147 | D9E1 | FABS | valeur absolue |
| 00402149 | D9FC | FRNDINT | on ne garde que la partie entière |
| 0040214B | E8 5E000000 | CALL Crapouil.004021AE | nous appellerons cette dernière valeur la magic_value |
| 00402150 | 803D 30764000 0 | CMP BYTE PTR DS:[407630],0 | |
| 00402157 | 74 05 | JE SHORT Crapouil.0040215E | |
| 00402159 | E8 61010000 | CALL Crapouil.004022BF | |
| 0040215E | 68 0E714000 | PUSH Crapouil.0040710E | ASCII "1231321321321" |
| 00402163 | E8 6C010000 | CALL Crapouil.004022D4 | Transformation du serial en un dword |
| 00402168 | 8B15 20764000 | MOV EDX,DWORD PTR DS:[407620] | |
| 0040216E | 03D0 | ADD EDX,EAX | On additionne le volume serial number |
| 00402170 | 81F2 FECA0100 | XOR EDX,1CAFE | |
| 00402176 | 52 | PUSH EDX | |
| 00402177 | D80424 | FILD DWORD PTR SS:[ESP] | |
| 0040217A | D9E1 | FABS | |
| 0040217C | DED9 | FCOMPP | Comparaison entre le dword du serial et la magic_value |
| 0040217E | 9B | WAIT | |
| 0040217F | DFE0 | FSTSW AX | |
| 00402181 | 9B | WAIT | |
| 00402182 | 9E | SAHF | |
| 00402183 | 5A | POP EDX | |
| 00402184 | 75 14 | JNZ SHORT Crapouil.0040219A | |

Nous sommes face à des instructions FPU qui manipulent le dword issu des différents cryptages, cela nous permet de déterminer la magic value. Ne nous affolons pas, d'ailleurs nous n'aurons même pas à les reverser. Notre serial (qui doit se composer de chiffre décimal et pas de 0 en tête) est transformé en un dword mis sur la pile FPU et enfin comparé à la magic_value. Nous pourrions nous dire, que c'est la fin mais non, reste un dernier petit piège que nous tend Elooo.

Le serial doit faire un minimum de 10 caractères, si l'on étudie la routine de transformation du serial en dword, on se rend compte qu'il va nous falloir plus que la magic value exprimée string décimal pour atteindre 10 caractères. Alors comment réaliser cela sans modifier la valeur 'portée' par la chaîne ?

C'est très simple : il suffit de voir que nos registres sont modulo 32 bits, par exemples la valeur 300000000h une fois transformée en dword par la fonction du keygenme vaut 0 mais son expression sous forme de string comprend plus de 10 caractères.

C'est parfait notre serial sera donc l'expression sous forme de string, en décimal de : 300000000h + magic value => keygenme cracked!!

5. Conclusion

Ce keygenme ne nous oppose pas de difficultés majeures mais plutôt une suite de petits défis intéressants à relever. Certains points seront peut-être plus clairs en regardant le code par vous-même en debugant ou en regardant la source du keygen. Pour réaliser le keygen, il nous suffit de reprendre point à point tous les éléments que j'ai évoqué dans ce papier, toute la routine de génération de la magic_value est assez frustrante à coder car nous sommes à la limite du rip, pour la dernière partie (l'addition de 300000000h) j'ai choisi d'utiliser la BigLib de Roy, l'opération se code en quelques lignes.

6. Greetz

Un grand merci à **Elooo** pour ce défi, une grosse papouille aux FRETiens : **Beatrix**, **DarkPhoenix**, **Eedy31**, **Kryshaam**, **Mattwood**, **Neitsa**, **TWG**, **Sefo** et **Whiskas**, et un coucou aux membres de FC et #fret.

Requiem^FRET