

Tutorial de reverse engineering

du brute-forcing au keygenning

par Valdstyn

Table des matières

1. But de ce papier
2. Outils
3. Brute-forcing
4. Brute-forcing amélioré
5. Sérial fishing
6. *Keygenning (**à venir)*
7. *Programmation du keygen en C++ (**à venir)*
 - a. *Application Console(**à venir)*
 - b. *Application GUI (MSVC6) (**à venir)*
8. Remerciements

1. BUT DE CE PAPIER

J'ai écrit ce tutorial pour plusieurs raisons. La première est que je souhaite faire partager mes connaissances (certes encore maigres) en reverse engineering à la communauté débutante. En effet, les crackers confirmés verront probablement peu d'intérêt, voire pas du tout, à lire ce tutorial, les techniques expliquées étant relativement basiques (tout du moins, pour ceux qui les maîtrisent!). La deuxième raison est que la meilleure façon de vérifier si l'on comprend quelque chose en détail est de l'expliquer à quelqu'un d'autre. Enfin, écrire ce tutorial est un peu la touche finale à l'analyse de mon programme.

Je vais partir du postulat que le lecteur a des vagues notions de cracking (avoir lu 2-3 tutoriaux, même très basique, est recommandé). Les termes essentiels seront expliqués simplement et les instructions seront détaillées de la façon la plus compréhensible pour les débutants.

Je ne prétends pas qu'à la fin de la lecture, vous serez capable de tout cracker (j'en suis moi-même *très* loin) mais votre bagage en cracking sera plus substantiel.

Sur ce, bonne lecture!

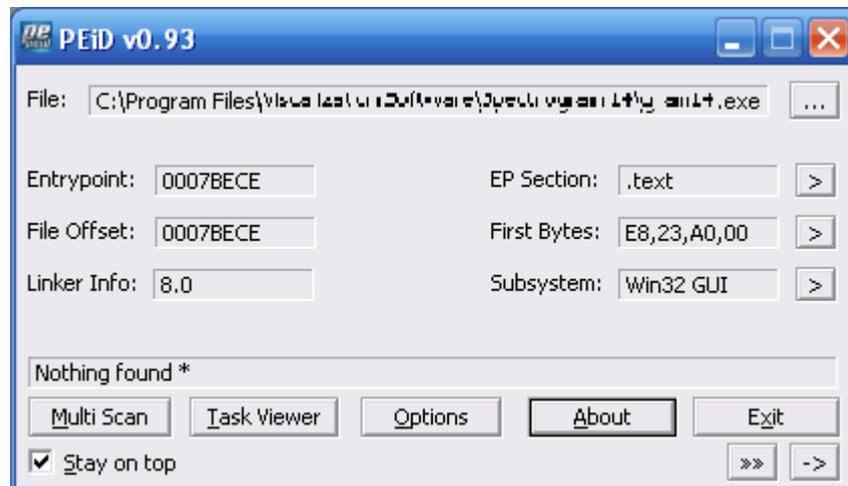
Valdstyn, le 20 mars 2008

2. OUTILS

- OllyDBG
- Calculatrice de Windows (!)
- PEiD
- Un compilateur C++ (Bloodshed DEV-C++ est très performant et gratuit) (7.a)
- Microsoft Visual C++ 6.0 (nécessaire pour 7.b)
- Programme cible : Sp***rog**m
(volontairement masqué, seule la technique importe, pas le logiciel !)

3. BRUTE-FORCING

Première étape, on va regarder le langage utilisé par notre logiciel cible et vérifier s'il utilise une quelconque protection (Aspack, Armadillo,...). Pour ce faire, utilisez l'outil PEiD.

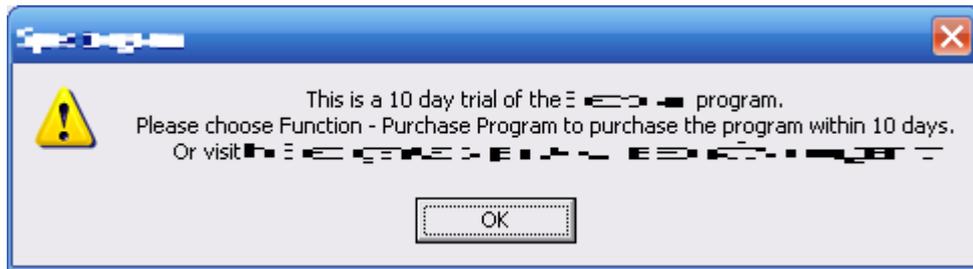


Chance ici :) le logiciel n'est pas protégé. Moralité: je ne m'attaquerai pas à de longues explications sur l'unpacking. D'autres tutoriaux y sont consacrés et de nombreux outils sont téléchargeables sur le net.

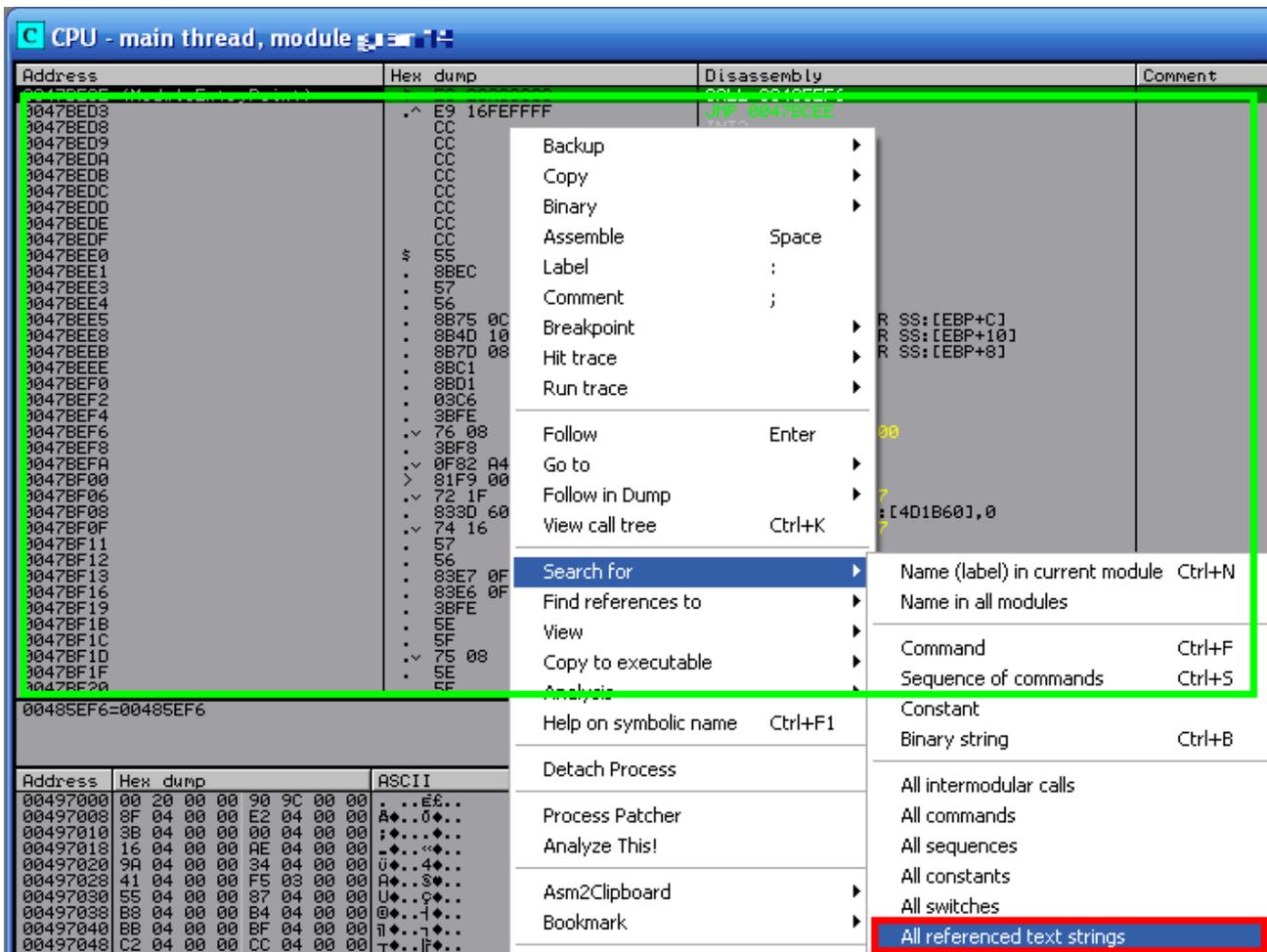
Notre cible étant vierge de protection, on peut la charger dans OllyDBG. Une fois chargée complètement (rien ne doit plus « bouger » dans la fenêtre d'Olly), on lance l'application en appuyant sur « F9 ». Tout de suite, on voit l'infection:



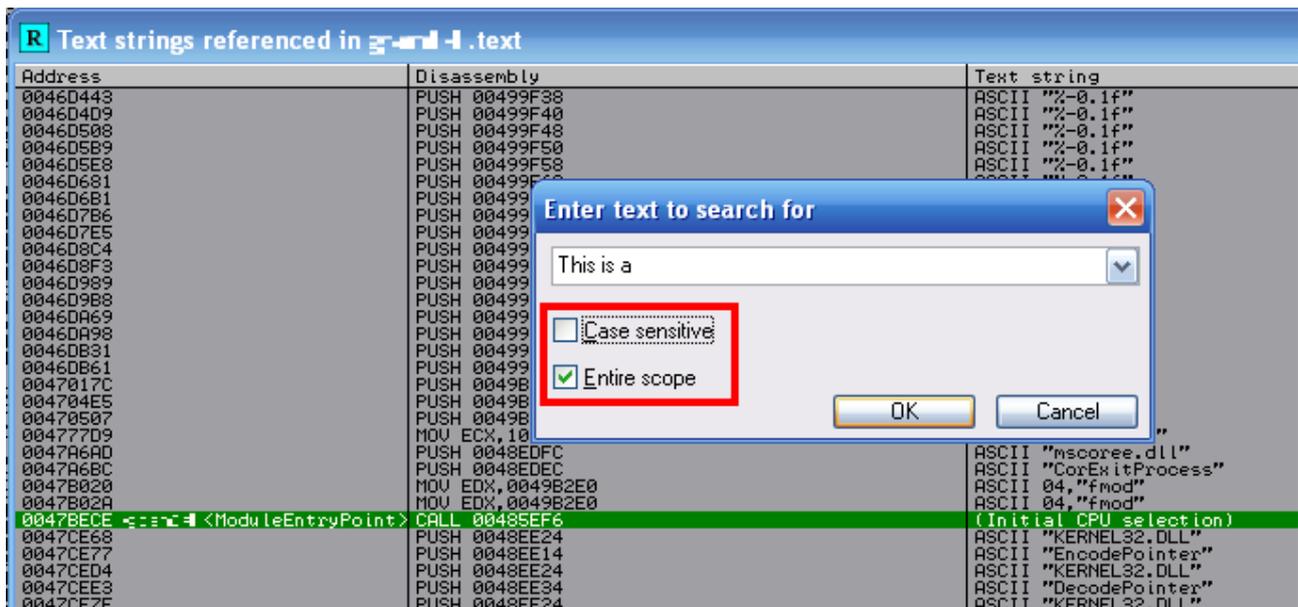
On entre alors un nom et un sérial bidon. Réaction du software sans surprise...



« Ah ben zut alors ! ». Les choses sérieuses vont commencer ;). Dans Olly, on fait clique droit sur le code assembleur (votre programme traduit en un langage très basique) => Search for => All referenced text strings



Vous vous souvenez de notre message après l'insertion du mauvais code? « This is a 10 day... ». On va rechercher cette chaîne de caractères. Cochez bien la case «Entire Scope » (=rechercher dans tout le code, sans considérer de position de recherche) et décochez « Case sensitive» (=ne pas prendre en compte la casse, c'est-à-dire la distinction entre majuscules et minuscules). L'image à la page suivante est assez explicite.



On tape le début de notre message d'erreur (ou en entier, si vous êtes motivés...) et on clique sur OK. La première chaîne correspondante trouvée (pas toujours la bonne) se met en surbrillance. On fait alors double clique gauche dessus. Voilà où on arrive:



Rapide analyse de ce qu'on peut voir:

La chaîne en surbrillance se trouve dans une accolade avec la mention « text = ». Cette accolade se termine, comme vous le constatez, sur [MessageBoxA]. De quoi s'agit-il? MessageBox est une fonction permettant d'afficher une boîte de dialogue contenant un message (en l'occurrence, ici, notre « This is a 10... »). Ce que vous voyez en jaune (JE, JNE ou JNZ) sont des SAUTS CONDITIONNELS. Exprimés en français, ils signifient: «SI ceci est, alors saute (JE)» ou «SI ceci n'est pas, alors SAUTE» (JNE). Or, que voit-on? Juste avant notre message d'erreur, 2 sauts conditionnels! Et si vous regardez de près, les 2 sautent vers le même endroit, c'est-à-dire AU-DELA du message *badboy* (message d'erreur). Premier essai, on va transformer le premier JE en JMP (SAUT sans condition), de sorte que le programme passe *toujours* au-dessus du message d'erreur. Cliquez gauche sur [JE SHORT 00446C52] pour mettre en surbrillance. On appuie sur

Espace. Une boîte apparaît alors avec ladite instruction. Remplacez JE par JMP et cliquez sur Assemble. Cliquez droit sur la zone de code (voir première image) => Copy to executable => all modifications => copy all. Une nouvelle boîte apparaît avec le code modifié. Cliquez droit dedans => Save file. Enregistrez alors sous monprog_crack.EXE. Lancez-le. On tape un nom, un sérial...le programme n'affiche rien et se lance. On a évité l'enregistrement! :)

Première victoire! Mais il va falloir aller plus loin, autrement dit: forcer un faux enregistrement. Il s'agit de l'étape suivante.

4. BRUTE-FORCING AMELIORE

On reprend notre code assembleur et en faisant défiler, on remarque très vite quelque chose de fort intéressant. (environ 20-30 lignes après le *badboy*).

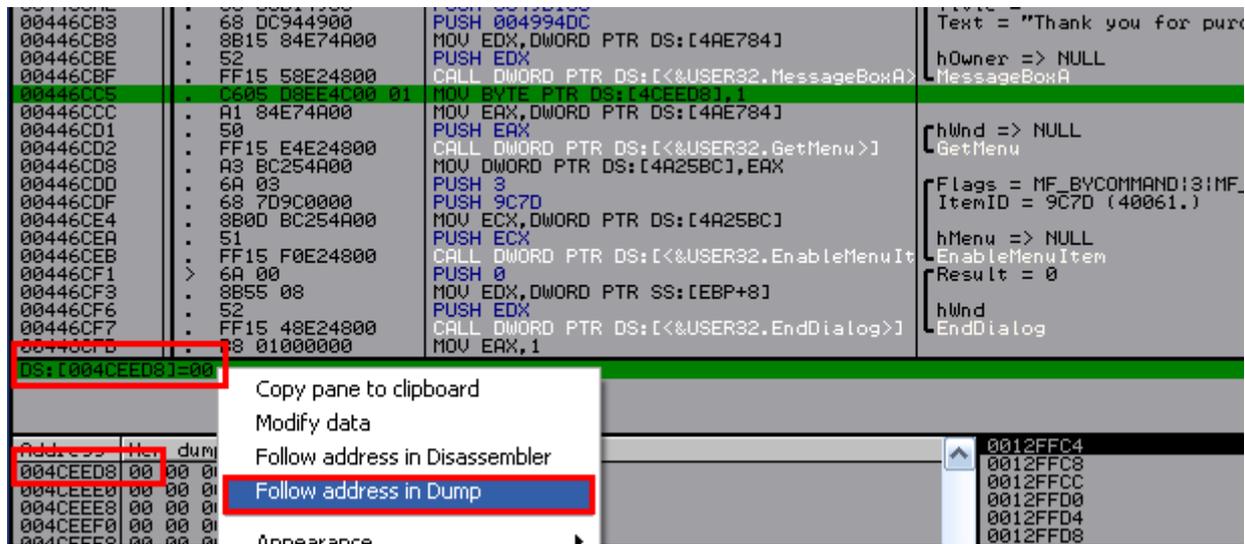
<pre> MOV ECX,DWORD PTR DS:[49C8C4] CMP ECX,DWORD PTR DS:[49CE0C] JNZ SHORT 00446CF1 PUSH 30 PUSH 0049D10C PUSH 0049D40C MOV EDX,DWORD PTR DS:[4AE784] PUSH EDX CALL DWORD PTR DS:[<&USER32.MessageBoxA] MOV BYTE PTR DS:[4CEED8],1 MOV EAX,DWORD PTR DS:[4AE784] u for purchasing [REDACTED].") </pre>	<pre> [Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL Title = "" Text = "Thank you for purchasing [REDACTED]" hOwner => NULL MessageBoxA </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------

Une MessageBox, encore, mais cette fois qui dit «Thank you for purchasing». Et au-dessus, un JNE. Ce saut est pris si le sérial est faux. Donc nous avons 2 possibilités pour le neutraliser: d'une part, on inverse le saut (JNE => JE) soit on le noppe (JNE => NOP), c'est-à-dire qu'on indique de ne faire aucune opération. Résultat, le programme arrive à l'instruction et il continuera gentiment son chemin vers le *goodboy*. On modifie l'instruction comme indiqué au chapitre précédent. On lance le programme; on entre un nom, un faux sérial.

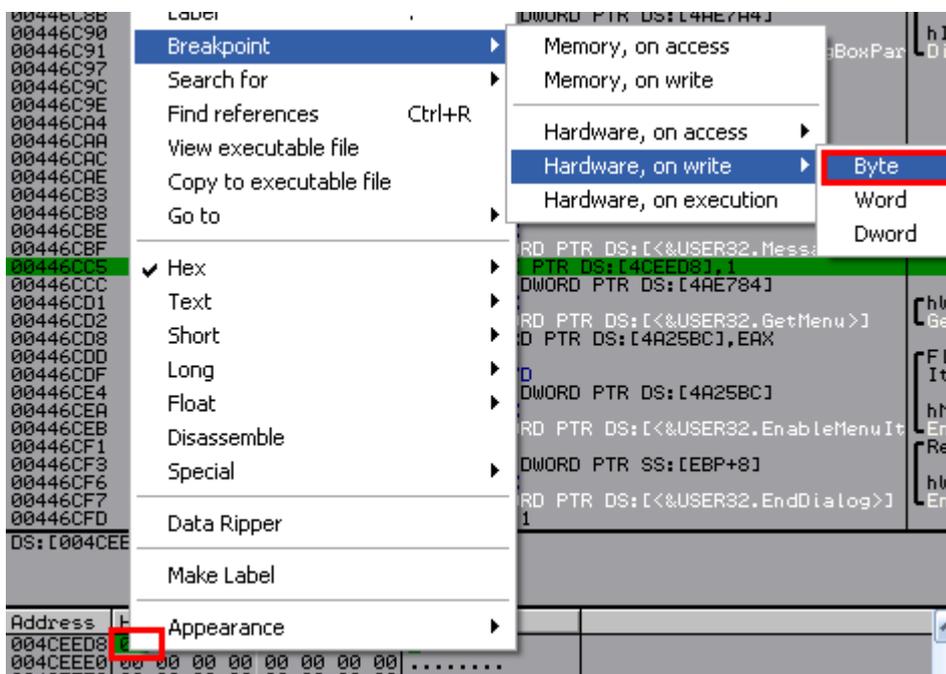


En voilà une bonne nouvelle! Mais (toujours important), on quitte le programme et on le relance pour tester la validité du crack. Argh! Nouvelle boîte de dialogue. L'enregistrement n'a pas été enregistré (sic). En fait, le programme vérifie à chaque lancement le nom et le sérial. Il va donc falloir lui faire croire qu'on a le bon sérial.

Regardons de nouveau l'image correspondant au message *goodboy*. Juste après l'affichage de la MessageBox, on voit: MOV BYTE PTR DS:[4CEED8],1
 Le programme -une fois l'utilisateur enregistré (message *goodboy*)- met une booléenne (une variable qui n'a que 2 valeurs possibles : 1 ou 0) à 1.



Dans le panneau en dessous des instructions, vous voyez: DS:[004CEED8]=00. On fait clique droit => Follow address in Dump. Et dans le panneau tout en bas, vous arrivez à l'adresse où est stockée la valeur de la variable. Si cette variable est mise à 1 après l'enregistrement, on peut supposer que le programme la vérifie à chaque lancement. Si elle est à 1, on sera enregistré, si elle est 0, la boîte d'enregistrement apparaîtra. On va boser un *Breakpoint* sur l'accès à cette variable et se débrouiller pour qu'elle soit constamment à 1.



Appuyez sur CTRL+F2 (ou cliquez sur la double flèche en haut à gauche dans Olly) pour redémarrer le programme. Il bloque alors sur une instruction.

00401284	. 83C4 08	ADD ESP,8	
0040128D	. 68 74DF4900	PUSH 0049DF74	ASCII "fe"
00401292	. 68 A0B34A00	PUSH 004AB3A0	ASCII "fe"
00401297	. E8 64900700	CALL 0047A300	
0040129C	. 83C4 08	ADD ESP,8	
0040129F	. C605 D8EE4C00 00	MOV BYTE PTR DS:[4CEED8],0	
004012A6	. 68 A0B34A00	PUSH 004AB3A0	ASCII "fe"
004012AB	. E8 48910700	CALL 0047A3F8	

«fe» et «fe» correspondent au nom et au sérial que j'avais utilisé pour le test (Désolé, j'aurais dû choisir deux strings plus parlantes^^) Or, on voit que la variable désirée est mise à 0. Conclusion: au lancement du programme, ce dernier reprend le nom et sérial précédemment entrés (ils sont en fait stockés dans un fichier *.ini) et refait le test d'authenticité. Mais vu que le nom et le sérial ne vont pas ensemble, il met la variable à 0. Forçons le destin (sic). On édite en appuyant sur Espace et on change la ligne en MOV BYTE PTR DS:[4CEED8], 1.

On enregistre les modifications (*confer supra dixit*). (On suppose que l'utilisateur n'a pas encore essayé d'entrer un couple nom/sérial) On lance le programme, on tape son nom et un sérial. OK. GoodBoy. On quitte le soft. On relance. Plus aucune boîte de dialogue d'enregistrement et notre nom est toujours dans le titre du programme. Mission accomplie avec succès!

5. SERIAL FISHING

On est heureux, on peut désormais utiliser notre programme autant qu'on veut. Mais...imaginons que nous ayons besoin de mise à jour. Et que pour ce faire, l'éditeur demande de se connecter sur le site et de donner son sérial. On serait alors ennuyé :-p. Qu'à cela ne tienne, on va faire de la pêche au sérial! Sortez la canne et l'hameçon!

On reprend le fichier EXE original (non modifié). On retourne à notre message *badboy*. Et on on fait défiler en arrière. Si le programme nous dit que notre clef est fausse, il a du la comparer avec la vraie et, dans un premier temps, la stocker dans une variable.

FF15 4CE24800	CALL DWORD PTR DS:[&USER32.GetWindowTex	GetWindowTextA
68 98CE4900	PUSH 0049CE98	ASCII "Valdstyn"
68 40F14C00	PUSH 004CF140	ASCII "Valdstyn"
E8 76370300	CALL 0047A300	
83C4 08	ADD ESP,8	
6A 13	PUSH 13	
68 98CE4900	PUSH 0049CE98	Arg2 = 00000013
E8 D74EFDFF	CALL 0041BA70	Arg1 = 0049CE98 ASCII "Valdstyn"
83C4 08	ADD ESP,8	gram14.0041BA70
A3 C4CB4900	MOV DWORD PTR DS:[49CBC4],EAX	
6A 50	PUSH 50	
68 98CE4900	PUSH 0049CE98	Count = 50 (80.)
8B15 20CB4900	MOV EDX,DWORD PTR DS:[49CB20]	Buffer = ██████████.0049CE98
52	PUSH EDX	
FF15 4CE24800	CALL DWORD PTR DS:[&USER32.GetWindowTex	hWnd => 00320510
68 98CE4900	PUSH 0049CE98	GetWindowTextA
68 A0B34A00	PUSH 004AB3A0	ASCII "Valdstyn"
E8 3C370300	CALL 0047A300	
83C4 08	ADD ESP,8	
68 98CE4900	PUSH 0049CE98	ASCII "Valdstyn"
E8 27380300	CALL 0047A3F8	
83C4 04	ADD ESP,4	
A3 0CCE4900	MOV DWORD PTR DS:[49CE0C],EAX	
A1 C4CB4900	MOV EAX,DWORD PTR DS:[49CBC4]	
3B05 0CCE4900	CMP EAX,DWORD PTR DS:[49CE0C]	
74 3A	JE SHORT 00446C28	
6A 50	PUSH 50	
68 98CE4900	PUSH 0049CE98	Count = 50 (80.)
8B0D 40CD4900	MOV ECX,DWORD PTR DS:[49CD40]	Buffer = ██████████.0049CE98
51	PUSH ECX	
FF15 4CE24800	CALL DWORD PTR DS:[&USER32.GetWindowTex	hWnd => 00240206
68 98CE4900	PUSH 0049CE98	GetWindowTextA
68 40F14C00	PUSH 004CF140	ASCII "Valdstyn"
E8 F7360300	CALL 0047A300	ASCII "Valdstyn"
83C4 08	ADD ESP,8	
6A 13	PUSH 13	
68 98CE4900	PUSH 0049CE98	Arg2 = 00000013
E8 584FFDFF	CALL 0041BB70	Arg1 = 0049CE98 ASCII "Valdstyn"
83C4 08	ADD ESP,8	██████████.0041BB70
A3 C4CB4900	MOV DWORD PTR DS:[49CBC4],EAX	
8B15 C4CB4900	MOV EDX,DWORD PTR DS:[49CBC4]	
3B15 0CCE4900	CMP EDX,DWORD PTR DS:[49CE0C]	
74 24	JE SHORT 00446C22	
0FB605 18B54A00	MOVZX EAX,BYTE PTR DS:[4AB518]	
85C0	TEST EAX,EAX	
75 19	JNZ SHORT 00446C22	
6A 30	PUSH 30	Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL

Les 3 traits rouges horizontaux correspondent aux fonctions récupérant le contenu d'une case d'édition (là où on écrit son nom/sérial). On pose un bp sur chacune des lignes (mettre en surbrillance et appuyer sur F2). On lance le programme et on essaye de s'enregistrer. Le programme break. Rien d'intéressant. On appuie sur F9 pour continuer. On voit alors apparaître un chiffre «bizarre» dans la fenêtre des registres en EAX (je sais, au début, tous les chiffres semblent bizarres, mais il faut y aller un peu à l'aveuglette).

EAX = 36635F41 = 912482113 (decimal)

Ce qui est intéressant c'est que ce chiffre va être utilisé plusieurs fois dans des tests de comparaison (traits verticaux) juste avant le message *badboy*. On s'essaye de s'enregistrer avec le nom précédent et ce sérial. Ca ne marche pas. Convertissez ce chiffre en décimal. (avec la calculatrice windows par exemple, mais vous pouvez aussi voir la valeur décimal directement dans Olly). On réessaye et paf! (le chien) l'enregistrement fonctionne!

Cette partie est très certainement une des plus difficiles pour un novice. Vous devrez, selon le programme, fouiller et refouiller, tenter et retenter. Gardez ces règles en tête pour trouver le vrai sérial : (règles de base; si c'était chaque fois aussi simple, il n'y aurait aucun intérêt :])

- Toujours après la récupération du nom (GetWindowTextA ou GetDlgItemTextA, par ex)
- Souvent à reconvertir décimal (puisque dans Olly, les valeurs sont données en hexa)
- Si vous avez peu de chance, crypté puis comparé...bonne chance! :)

Notez que parfois (souvent, dsl), le sérial que vous entrez est comparé par morceau, ou de façon assez complexe. Mais pour les softs basiques/amateurs, l'auteur génère la vraie clef et ne fait que comparer. (Je rappelle que ce n'est qu'un tuto pour débutant^^)

L'essentiel c'est que maintenant, on a un couple nom/clef valide. On va également pouvoir orienter nos recherches pour trouver la routine de génération.

6. KEYGENNING

Etant donné que nous connaissons notre sérial, nous pouvons trouver le call qui le génère. La technique est de tracer le PREMIER call qui précède l'apparition de notre sérial aux alentours des GetWindowText et des tests de comparaisons. [...]

7. PROGRAMMATION DU KEYGEN EN C++

A. APPLICATION CONSOLE

B. APPLICATION GUI (Graphical User Interface)

8. REMERCIEMENTS

Je souhaiterais remercier :

Daemon, pour son super guide; la communauté forumcrack,
Ceux qui ont répondu à mon premier message sur le forum et m'ont forcé à bosser :)

«J'y suis arrivé tout seul !» ^^