

Crack de Auto Window Manager

Pour mon sixième tuto, nous allons voir comment unpacker manuellement AsPack 2.12 et trouver un sérial valide pour le soft. Ce soft est (encore :) un de la société Eusing, et vous allez voir que le schéma de contrôle du sérial ressemble fortement à celui d'Eusing Free Registry Cleaner.

Vous pouvez donc vous reporter à mon tuto sur ce soft pour essayer de le faire vous même, sinon lisez celui-ci. Ce logiciel est assez facile à cracker, donc ce tuto est à la portée de Newbies.

Pour avoir les bases sur l'unpacking et les packers, je vous conseille de vous reporter à mon premier tuto sur Anti007 2.5.

Je rappelle que je ne peux en AUCUN CAS être tenu pour responsable d'un dommage survenant sur votre PC lors de la mise en pratique de ce tuto.

Prérequis :

Vous aurez besoin de connaître le fonctionnement d'OllyDbg / ImpRec / LordPE (les bases), et d'avoir des connaissances en assembleur, sinon vous n'irez pas très loin.

Voici une introduction à OllyDbg par Crisanar :

<http://daemonftp.free.fr/daemoncrack/Tuts/Crisanar/introOlly.htm>

Pour l'assembleur, j'ai sélectionné deux cours très bien faits, accessibles aux débutants mais proposant tout de même une approche assez complète.

Cours de Deamon : <http://daemonftp.free.fr/daemoncrack/index0.htm>

Cours de Falcon : <http://xtx.free.fr/liens/tut/Assembleur%20par%20Falcon/Assembleur.html>

Normalement vous n'avez besoin de rien de plus.

Outils :

- Le soft Auto Window Manager téléchargeable ici : <http://www.eusing.com/WindowManager/WindowManager.htm>
- Peid 0.95 ou RDG Packer Detector 0.6.6 (au choix)
- Un débogueur/désassembleur : OllyDbg (1.10 ou 2.0)
- LordPE by Yoda
- ImpRec 1.7c
- Le plugin CommandBar pour Olly : www.openrce.org/downloads/details/105/CommandBar
- Un cerveau :)

Les anciennes versions des logiciels proposés (Peid / RDG / ImpRec) marchent également. Pour Olly, je l'ai fait avec la version 1.10. Tout ces logiciels sont trouvables rapidement dans [Google](#).

Introduction

Le cracking va se dérouler en deux grandes parties : on va tout d'abord s'occuper de l'unpacking (recherche de l'OEP, dump du Crackme, reconstruction de l'IAT) et ensuite on passera à la création de sérials valides.

Je ne vais pas développer ici la théorie sur l'unpacking, mais sachez au moins que le logiciel de compression greffe au logiciel cible ce que l'on appelle un loader. Le but de ce loader est de modifier l'entrypoint du programme (EP), c'est à dire le début du code, pour que ce soit le loader qui se lance en premier. Le loader lance ensuite le programme, après l'avoir décompresser en mémoire (dans la RAM).

I) Unpacking d'AsPack 2.12

On va tout d'abord rechercher l'OEP du programme. L'OEP, c'est une abréviation pour Original Entry Point, c'est à dire l'endroit où le programme débute réellement. Souvenez vous que le packer modifie ce point d'entrée par celui de son loader. Vous devinez donc que le but va être de trouver le véritable début du programme pour en quelque sorte "remettre la machine" dans l'ordre.

On va donc tout d'abord passer le fichier à unpacker sous RDG ou Peid, et cela doit devenir systématique à chaque fois que vous débbuger un programme.

Ouvrez PEiD et sélectionnez le programme à analyser en cliquant sur [...] en haut a droite. On s'aperçoit alors que notre packer est détecté, et que sa signature est "ASPack 2.12 -> Alexey Solodovnikov". On remarque aussi que l'Entry Point se situe dans la section .aspack, et que les deux sections caractéristiques d'AsPack sont là : on trouve en fin de code les sections .aspack et .adata. Il y a donc fort peu de chances pour que l'on tombe sur une fausse signature.



Ici, nous sommes vraiment chanceux, car ce packer est un des plus simples de sa catégorie. En effet, il ne fait que compresser le code, et sa routine est toujours la même : l'EP du programme est un PUSHAD, situé au début du code, et la fin de la routine du loader se compose ainsi : un POPAD suivi d'un JNZ vers le PUSH 0. Ce PUSH va recevoir l'adresse de l'OEP. Le RETN fait ici office de JMP vers l'EIP, c'est à dire vers l'OEP.

Le PUSHAD, c'est l'instruction qui sauvegarde tout les registres, et le POPAD l'instruction qui les restaure.

Petit récapitulatif :

PUSHAD

[.....]

```

POPAD
JNZ Adresse PUSH 0
MOV EAX,1          // instruction non utilisée
RETN 0C           // idem
PUSH 0 (puis PUSH OEP)
RETN

```

Quand on essaie de l'ouvrir avec OllyDBG, on obtient un message d'erreur qui nous avertit que l'EP du soft se situe en dehors du code. Un message nous prévient ensuite que le code est très probablement compressé (si votre version d'Olly n'a pas été patchée). Nous voyons aussi qu'il ne trouve aucune référence dans :

• ->> Search for ->> All Referenced text strings.

Donc là même si vous ne l'aviez pas analysé avec Peid, aucun doute qu'il est bien packé.

```

00570001  CALL     AUTOWIN.00570004
00570002  JNZ     45B404F7
00570003  PUSH    EBP
00570004  RETN
00570005  CALL     AUTOWIN.00570014
00570006  JNZ     SHORT AUTOWIN.00570072
00570007  MOV     EBX,-13
00570008  ADD     EBX,EBP
00570009  SUB     EBX,170000
0057000A  CMP     DWORD PTR SS:[EBP+422],0
0057000B  MOV     DWORD PTR SS:[EBP+422],EBX
0057000C  JNZ     AUTOWIN.00570039
0057000D  LEA    EAX,DWORD PTR SS:[EBP+42E]
0057000E  PUSH    EAX
0057000F  CALL   DWORD PTR SS:[EBP+F4D]

```

Vous avez maintenant le choix entre plusieurs méthodes :

- La méthode feignasse : on utilise AsPackDie.
- La méthode avec l'ESP Trick : pour la voir, allez lire mon tuto sur l'unpacking d'Anti 007. Méthode à maîtriser absolument.
- La méthode du scrolling : on descend à la main pour trouver la routine (ça peut être très long)
- La méthode recherche : on fait une recherche d'instructions dans Olly. C'est cette méthode que je vais développer ici car je trouve que c'est la plus rapide.

Cette méthode va se baser sur le fait que la routine du loader d'AsPack est invariable. Nous allons rechercher la suite d'instructions qui caractérise la fin du loader pour tomber directement dessus.

Vous vous souvenez (normalement :) que la fin du loader d'AsPack est toujours la même.

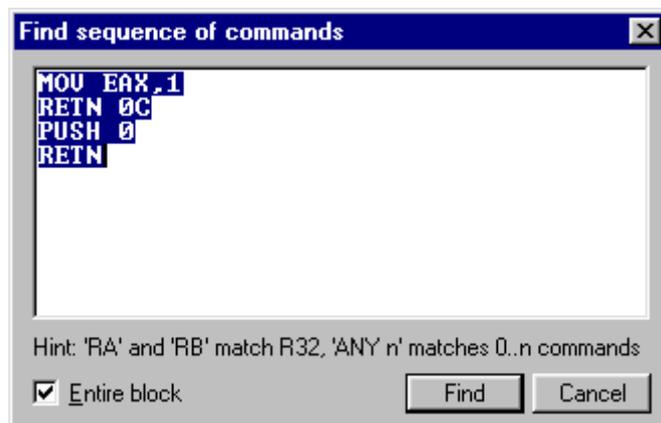
Petite piqûre de rappel :

```

POPAD
JNZ Adresse du Push 0
MOV EAX,1
RETN 0C
PUSH 0 // qui deviendra PUSH Adresse de l'OEP
RETN

```

Sur cette portion de code, on remarque qu'il y a une suite d'instructions qui ne changera JAMAIS, car elle ne dépend pas du programme. Nous allons l'utiliser pour tomber directement sur le code qui nous intéresse. Cette portion part du MOV EAX,1 et va jusqu'au RETN. En effet, le JNZ ne peut pas être inclut dans la recherche de notre séquence d'instructions, car l'adresse vers laquelle il pointe est propre au programme. On va donc faire Click droit => Search For => Sequence of Commands (utilisez à l'avenir Ctrl+S).



Entrez la séquence à rechercher comme sur l'image et faites "Find". C'est OK, Olly tombe directement sur la portion du code que nous recherchions.

A noter : Avec => Search For => Command (Ctrl + F), en recherchant le POPAD, nous devons faire trois fois Ctrl + L pour arriver à la quatrième occurrence qui débouche sur la bonne routine. Cette méthode est donc un peu moins rapide.

Vous arrivez normalement ici :

```

005703A7 0BC9          OR  ECX,ECX
005703A9 8985 A8030000 MOV  DWORD PTR SS:[EBP+3A8],EAX
005703AF 61           POPAD
005703B0 75 08        JNZ  SHORT AUTOWIN.005703BA
005703B2 B8 01000000  MOV  EAX,1
005703B7 C2 0C00     RETN 0C
005703BA 68 00000000  PUSH 0
005703BF C3           RETN
005703C0 8B85 26040000 MOV  EAX,DWORD PTR SS:[EBP+426]
005703C6 8080 3B040000 LEA  ECX,DWORD PTR SS:[EBP+43B]

```

Placez maintenant un breakpoint sur le RETN en 005703BF en faisant F2, ce qui aura pour effet d'arrêter le programme dès qu'il arrivera à cette ligne de code. Le programme sera ainsi entièrement décompressé dans la RAM et nous n'aurons plus qu'à le dumper :).

Pressez maintenant F9 pour lancer le programme.

```

005703A9 8985 A8030000 MOV  DWORD PTR SS:[EBP+3A8],EAX
005703AF 61           POPAD
005703B0 75 08        JNZ  SHORT AUTOWIN.005703BA
005703B2 B8 01000000  MOV  EAX,1
005703B7 C2 0C00     RETN 0C
005703BA 68 90A04C00  PUSH AUTOWIN.004CA090
005703BF C3           RETN
005703C0 8B85 26040000 MOV  EAX,DWORD PTR SS:[EBP+426]

```

Que remarque t'on ?

Tout d'abord le PUSH en 005703BA a changé : il pointe vers l'adresse 004CA090.

Regardez maintenant les Virtual Offsets des sections dans Peid :

Name	V. Offset	V. Size	R. Offset	R. Size	Flags
CODE	00001000	000CA000	00000400	0004E200	C0000040
DATA	000CB000	00003000	0004E600	00001800	C0000040
BSS	000CE000	00001000	0004FE00	00000000	C0000040

Calcul de l'OEP : Adresse de l'OEP - Image Base

$$004CA090 - 400000 = CA090$$

Vous avez $1000 < CA090 < CB000$.

L'adresse que l'on a trouvé est donc dans la section CODE. Or, cette section contient le code du programme, et c'est tout le temps elle qui contient l'OEP. Nous avons donc le bon.

Nous remarquons ensuite cela dans la petite barre sous le code de la fenêtre CPU :

Return to 004CA090 (AUTOWIN.004CA090)

Le RETN fait donc ici office de JMP vers la dernière adresse qui a été pushée dans la pile.

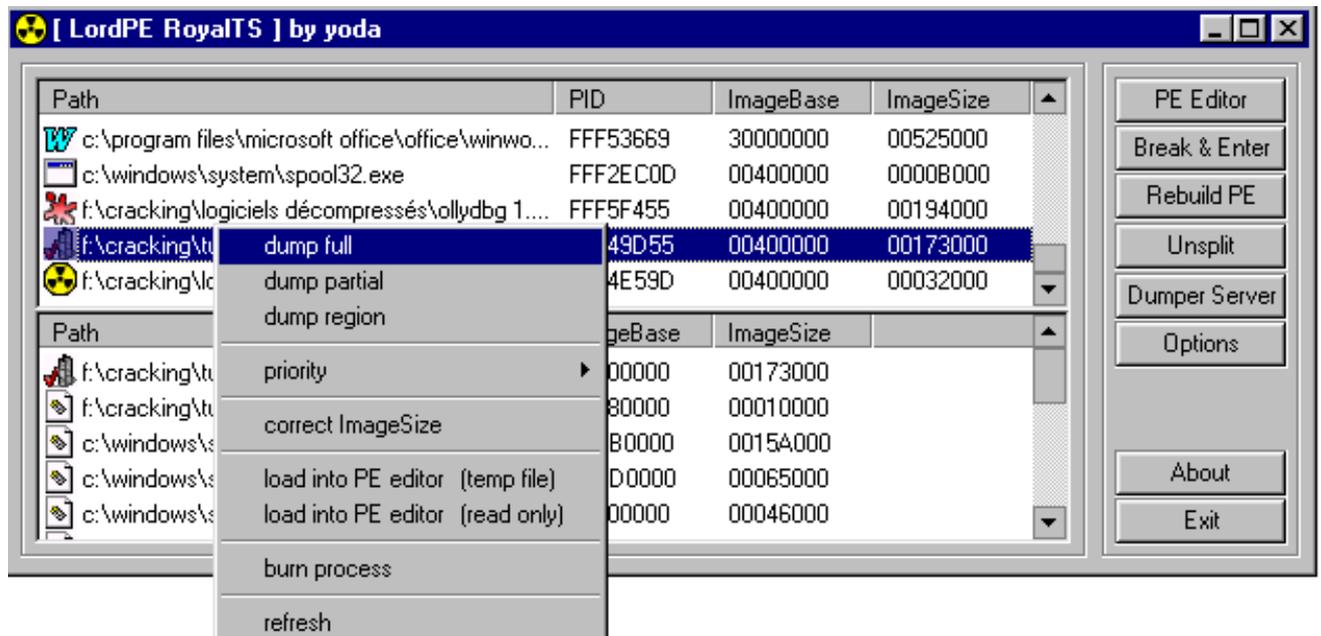
Pourquoi ? Voici l'explication par Kef :

Lorsque ton programme est sur un RETN, il prend la valeur posée sur la pile et la stocke dans EIP, ce qui a pour effet de le faire sauter à l'adresse qui a été pushée.,

Il ne nous reste plus qu'à le dumper.

Le dump, c'est un enregistrement de tout ou partie des adresses mémoires contenues dans la RAM. Ainsi, quand le loader atteint notre RETN, le programme est entièrement décompressé dans la RAM grâce au POPAD. Le but du dump est donc de récupérer le programme décompressé.

Ouvrez pour cela LordPE, faites un click droit sur le programme, Dump Full.



Enregistrez sous autowin_dump.exe, et une messagebox nous avertit que le dump est réussi.

Lancez maintenant le soft, puis ensuite ImpRec.

Faites Attach to an active process puis sélectionnez le fichier packé.

L'IAT, c'est la table des imports d'un programme, c'est à dire une table qui répertorie les DLL que celui-ci utilise, avec le détail des fonctions importées et leurs adresses.

L'IAT compte deux tableaux, et non pas un seul. Ces deux tableaux contiennent le nom des fonctions importées, mais ils ont chacun leur utilité. Retenez que le premier tableau va contenir les noms des fonctions, et le deuxième leurs adresses, après l'exécution du programme.

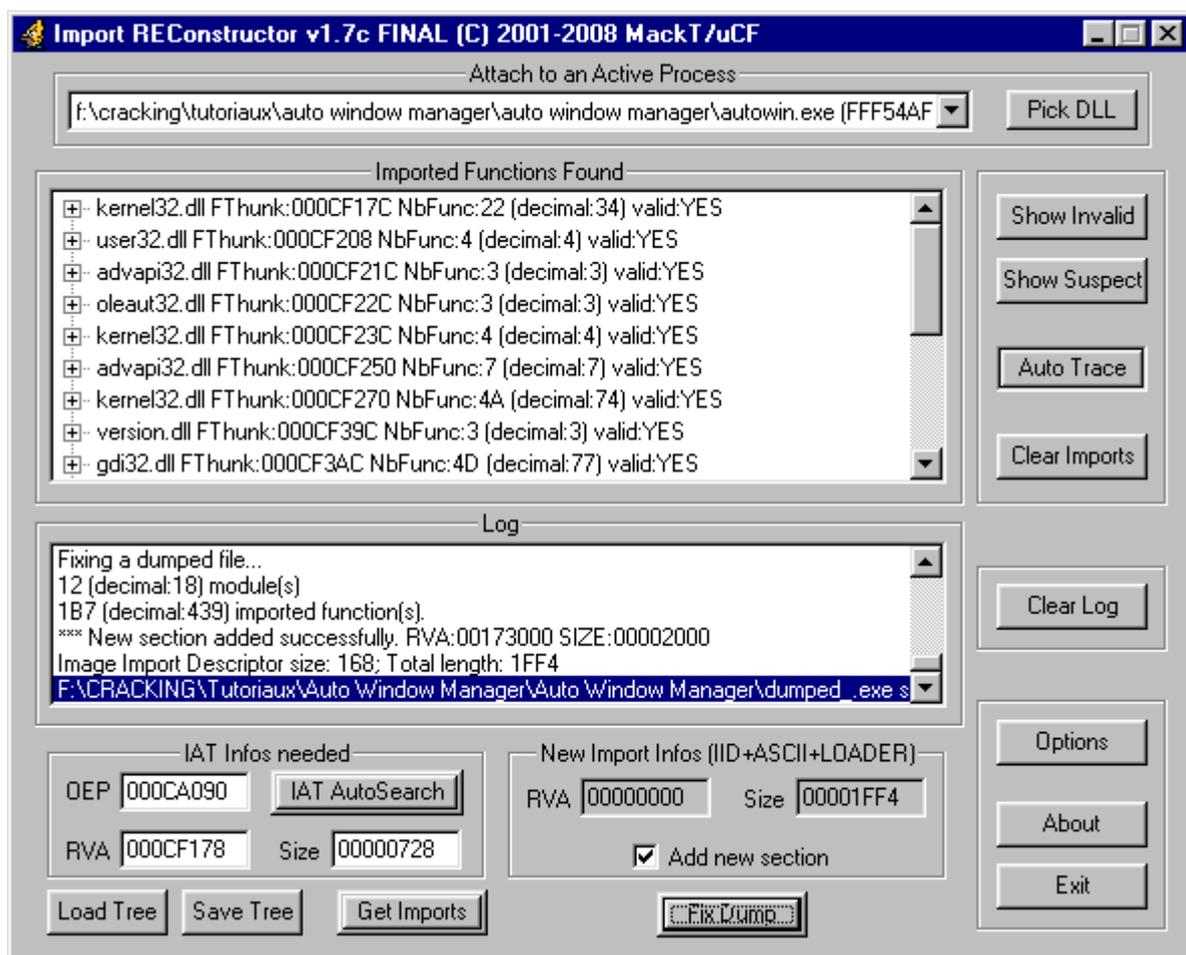
Ainsi, lorsque nous copions le programme avec notre dump, le deuxième tableau contient les adresses réelles des fonctions et non plus leur nom. Il faut le reconstruire car si l'on lance le dump, le programme va chercher le nom des fonctions, mais il ne trouvera que des adresses, et un plantage du programme s'en suivra inévitablement.

Pour changer l'OEP dans ImpRec, regardez en bas à gauche. Vous voyez normalement un encart avec comme adresse 00170001. Remplacez cette adresse par celle que nous avons précédemment trouvée, c'est à dire 000CA090.

On clique sur IAT Autosearch et nous obtenons une messagebox qui nous avertit qu'ImpRec a trouvé quelque chose. On clique alors sur Get Imports, et nous obtenons une liste de fonctions, qui répertorie celles utilisées par le programme. Elles ont toutes l'air valides mais par précaution on clique quand même sur Show Invalid.

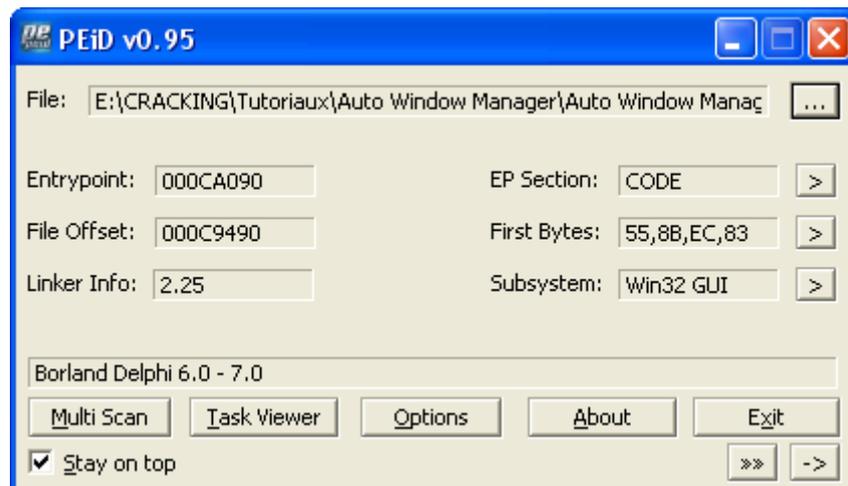
Il n'y a rien, on peut donc continuer l'esprit tranquille :)

Vous faites ensuite Fix Dump sur le dump précédent pour finaliser la reconstruction de son IAT. ImpRec va en effet remplacer les adresses réelles des fonctions par leur nom, et le deuxième tableau sera ainsi valide. C'est bon, vous pouvez maintenant tout fermer, l'unpacking est fini.



(La screen est faite ici lorsque la reconstruction de l'IAT est terminée)

On vérifie dans PEiD :



II) Création de sérials valides

Ouvrez maintenant le dump finalisé dans Olly. On remarque déjà qu'il n'y a plus un message d'erreur, et que l'OEP est typique du Borland Delphi 6.0 - 7.0. Faites un click droit, Search for, All referenced text strings pour partir à la recherche du message d'erreur. Pas de chance, il n'est pas dans les strings.

Nous allons donc exploiter le fait que le message d'erreur est placé dans une messagebox. Celle ci est toujours appelée par l'API MessageBoxA.

Si vous avez lu mon tuto sur Eusing Free Registry Cleaner, essayez de le faire vous même, sinon lisez ce qui va suivre.

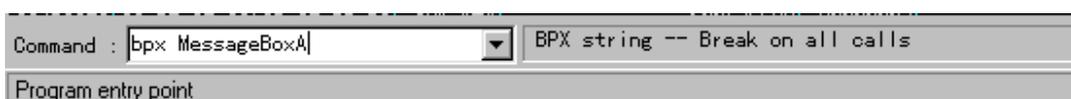
Placez le plugin CommandBar dans le dossier où se trouve Olly, puis lancez notre debugger préféré. Vous apercevez la barre en bas ?



Cette barre permet d'utiliser les commandes de SoftIce dans Olly. Elle va nous servir ici à poser un breakpoint sur l'API MessageBoxA.

Nous allons faire breaker Olly au moment où il appelle celle-ci, et on va pouvoir ainsi remonter jusqu'à l'instruction qui a décidé si notre sérial était bon ou pas. Je rappelle que l'on recherche une routine de ce genre : CALL + TEST / CMP + SAUT CONDITIONNEL.

Allez donc dans la CommandBar et mettez bpx MessageBoxA. Pourquoi bpx ? Parce que c'est la commande qui permet de placer un breakpoint sur tous les calls qui font appel à l'API que nous recherchons. Vous avez donc ceci :



Faites entrée, coupez la fenêtre "Intermodular Calls" qui s'ouvre, retournez dans la fenêtre CPU, et faites F9 pour lancez le programme.

Mettez "Horgh" comme nom, et "123456" comme sérial. Faites "OK".

Et là que se passe t'il? Olly breake au moment où il appelle la messagebox !

```

004A70B0  E8 EF6AFFFF  CALL DUMPED_.0049DBA4
004A70B5  84C0         TEST AL,AL
004A70B7  75 3B        JNZ SHORT DUMPED_.004A70F4
004A70B9  6A 10        PUSH 10
004A70BB  0055 F8      LEA EDX,DWORD PTR SS:[EBP-8]
004A70BD  91 00E4C00  MOV EAX,DWORD PTR DS:[4CDEB0]
004A70C3  E8 48EFF5FF  CALL DUMPED_.00406010
004A70C8  0045 F8      MOV EAX,DWORD PTR SS:[EBP-8]
004A70CB  E8 0C05F5FF  CALL DUMPED_.004045DC
004A70D0  50         PUSH EAX
004A70D1  0055 F4      LEA EDX,DWORD PTR SS:[EBP-C]
004A70D4  91 3CDE4C00 MOV EAX,DWORD PTR DS:[4CDE3C]
004A70D9  E8 32EFF5FF  CALL DUMPED_.00406010
004A70DE  0045 F4      MOV EAX,DWORD PTR SS:[EBP-C]
004A70E1  E8 F604F5FF  CALL DUMPED_.004045DC
004A70E6  50         PUSH EAX
004A70E7  78FAF5FF    CALL <JMP.&user32.GetActiveWindow>
004A70EC  50         PUSH EAX
004A70ED  E8 B2FCF5FF  CALL <JMP.&user32.MessageBoxA>
004A70F2  EB 4D        JMP SHORT DUMPED_.004A7141
  
```

On va donc remonter les instructions une à une en cherchant notre routine de vérification du sérial. Or que voit on juste au dessus en 004A70B0 ? Une routine semblable à celle que l'o cherche :). Mettez un BP sur ce CALL, faites F9, revalidez notre sérial bidon et Olly breake sur notre CALL. Rentrez dedans avec F7. Vous obtenez ceci :

```

0049DBA4  F7         PUSH EBP
0049DBA5  8BEC        MOV EBP,ESP
0049DBA7  6A 00        PUSH 0
0049DBA9  53         PUSH EBX
0049DBAA  56         PUSH ESI
0049DBAB  0BF0        MOV ESI,EAX
0049DBAD  3C00        XOR EAX,EAX
0049DBAF  55         PUSH EBP
0049DBB0  68 07DC4900 PUSH DUMPED_.0049DC07
0049DBB5  64 FF30     PUSH DWORD PTR FS:[EAX]
0049DBB8  64 8920     MOV DWORD PTR FS:[EAX],ESP
0049DBBB  3DB        XOR EBX,EBX
0049DBBD  0045 FC     LEA EAX,DWORD PTR SS:[EBP-4]
0049DBC0  00B0        MOV EDX,ESI
0049DBC2  E8 ED65F6FF CALL DUMPED_.004041B4
0049DBC7  0045 FC     MOV EAX,DWORD PTR SS:[EBP-4]
0049DBCA  E8 0D63F6FF CALL DUMPED_.004043DC
0049DBCF  3F 08       CMP EAX,8
0049DBD2  75 1D       JNZ SHORT DUMPED_.0049DBF1
0049DBD4  0045 FC     MOV EAX,DWORD PTR SS:[EBP-4]
0049DBD7  0078 03 33  CMP BYTE PTR DS:[EAX+3],33
0049DBDB  75 14       JNZ SHORT DUMPED_.0049DBF1
0049DBDD  0045 FC     MOV EAX,DWORD PTR SS:[EBP-4]
0049DBE0  0078 06 38  CMP BYTE PTR DS:[EAX+6],38
0049DBE4  75 0B       JNZ SHORT DUMPED_.0049DBF1
0049DBE6  0045 FC     MOV EAX,DWORD PTR SS:[EBP-4]
0049DBE9  0078 07 31  CMP BYTE PTR DS:[EAX+7],31
0049DBED  75 02       JNZ SHORT DUMPED_.0049DBF1
0049DBEF  B3 01       MOV BL,1
0049DBF1  3C00        XOR EAX,EAX
0049DBF3  58         POP EAX
0049DBF4  59         POP ECX
0049DBF5  59         POP ECX
0049DBF6  64 8910     MOV DWORD PTR FS:[EAX],EDX
0049DBF9  68 00E4C900 PUSH DUMPED_.0049DC0E
0049DBFE  0045 FC     LEA EAX,DWORD PTR SS:[EBP-4]
0049DC01  E8 1665F6FF CALL DUMPED_.0040411C
0049DC06  C3         RETN
  
```

On remarque tout d'abord que EAX contient la string "123456". Il y a aussi 4 CMP qui travaillent sur EAX. Ces CMP sont tout le temps suivis de sauts conditionnels. Il semblerait donc que le soft effectue des tests à différents endroits de notre sérial. Analysons le code :

1) Premier contrôle

CMP EAX,8 : Compare notre sérial à 8.

JNZ 0049DBF1 : Saute si ce n'est pas égal. Notre sérial doit donc faire 8 caractères.

2) Deuxième contrôle

MOV EAX,DWORD PTR SS:[EBP-4] : Met dans EAX notre sérial (adresse EBP-4 dans la pile)

CMP BYTE PTR DS:[EAX+3],33 : Compare le 4^e caractère de notre sérial à 33h, c'est à dire à la valeur ASCII du caractère 3.

JNB 0049DBF1 : Saute si ce n'est pas inférieur. Notre 4^e caractère devra donc être égal à 1 ou à 2.

3) Troisième contrôle

MOV EAX,DWORD PTR SS:[EBP-4] : Met dans EAX notre sérial (adresse EBP-4 dans la pile)

CMP BYTE PTR DS:[EAX+6],38 : Compare le 7^e caractère de notre sérial à 38h, c'est à dire à la valeur ASCII du caractère 8.

JNZ 0049DBF1 : Saute si ce n'est pas égal. Notre 7^e caractère devra donc être égal à 8.

4) Quatrième contrôle

MOV EAX,DWORD PTR SS:[EBP-4] : Met dans EAX notre sérial (adresse EBP-4 dans la pile)

CMP BYTE PTR DS:[EAX+7],31 : Compare le 8^e caractère de notre sérial à 31h, c'est à dire à la valeur ASCII du caractère 1.

JNZ 0049DBF1 : Saute si ce n'est pas égal. Notre 8^e caractère devra donc être égal à 1.

Conclusions :

- 1) Le sérial doit faire 8 caractères.
- 2) Le quatrième caractère du sérial doit être inférieur à 3.
- 3) Le septième caractère du sérial doit être égal à 8.
- 4) Le huitième caractère du sérial doit être égal à 1.

Voilà, maintenant vous savez comment fabriquer votre sérial. Le sérial est stocké dans le fichier autowin.dat.

Par exemple : 12325681

Voilà, ce tutoriel est fini. J'espère qu'il a été clair, et que vous n'avez pas rencontré de difficultés. Si vous codez un keygen pour ce soft, pourriez vous m'envoyer la source, pour que cette fois ça soit moi qui apprenne comment faire :).

Tuto finalisé par Horgh le 14/03/2010

Merci aux auteurs de ce soft :)