

# Sérial Fishing sur un Soft

*Dans ce cours, nous allons voir comment faire du sérial fishing sur un soft. Qu'est ce que le sérial fishing ?*

*C'est l'art de cracker un programme en trouvant le sérial qui correspond à son nom. Le sérial fishing peut être assez simple lorsque le sérial apparaît en clair dans le programme au cours du debugging, par exemple dans les vieilles versions de Winzip ou PowerArchiver, mais il peut devenir tout de suite plus compliqué lorsque ce sérial correspond à une valeur hexadécimale stockée dans une adresse mémoire...*

*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, et d'avoir de (très) bonnes 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>

Pour les API, j'ai sélectionné un cours qui propose une approche assez fouillée et bien vulgarisée:

Cours de Falcon : <http://xtx.free.fr/liens/tut/api/api.htm>

Normalement vous n'avez besoin de rien de plus.

---

## Outils :

- Eusing Free Registry Cleaner 2.0  
( [http://www.eusing.com/free\\_registry\\_cleaner/registry\\_cleaner.htm](http://www.eusing.com/free_registry_cleaner/registry_cleaner.htm) )
- Peid 0.95 ou RDG Packer Detector 0.6.6 (au choix)
- Un débogueur/désassembleur : OllyDbg (1.10 ou 2.0)
- Le plugin CommandBar pour Olly  
( [www.openrce.org/downloads/details/105/CommandBar](http://www.openrce.org/downloads/details/105/CommandBar) )
- Un cerveau :)

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

---

## Sérial fishing

Eusing Free Registry Cleaner (EFRC) est un freeware assez utile qui sert à nettoyer votre registre. Or, ce logiciel nous balance un nag au début qui nous nous demande de nous enregistrer,

et l'on remarque dans "Help" que l'on a un menu pour s'enregistrer. Je ne sais pas vous, mais moi ça me dérange (même si j'approuve ce concept, au contraire des sharewares...). Regardons donc ce qui se passe si l'on veut s'enregistrer avec pour nom "Horgh" et pour sérial "123456".



Ou



On obtient une messagebox avec un message d'erreur. (Si vous avez une messagebox vide, c'est parce que le fichier pour traduire le logiciel en français est mort. Remettez-le en anglais. J'ai renvoyé le fichier corrigé à l'éditeur du soft :).

Remarquez aussi qu'il est signalé que le nom n'influence pas sur le sérial (je ne sais plus où). Passons maintenant le soft sous Peid pour voir si il est packé :



C'est bon, nous avons de la chance, il n'est pas packé.

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



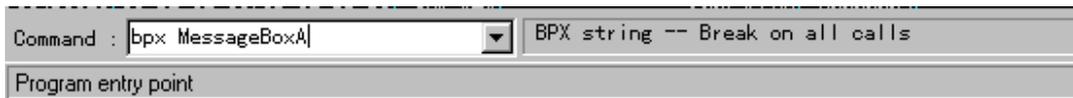
Cette barre permet d'utiliser les commandes de SoftIce dans Olly. Elle va nous servir ici à poser un breakpoint sur l'API MessageBoxA (la fonction qui appelle les MessageBox :).

En effet, vous avez remarqué que lorsque l'on veut s'enregistrer on obtient juste une messagebox vide. Si l'on fait breaker Olly au moment où il appelle celle-ci, on va pouvoir 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.

Comme il n'y a pas la string (bon, il y a, RES\_Registration\_Error, mais on ne l'a pas vue... :), nous allons employer une autre méthode pour retrouver l'endroit où le sérial est vérifié.

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 MessageBoxA.

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 !



On va donc commencer à faire le chemin inverse pour retrouver comment est on arrivé à cet endroit. Remontez les instructions une à une en gardant à l'œil le cadre sous le code pour voir si des jumps nous font arriver à cet endroit. Arrêtez vous en 0048430A et regardez en bas :



On voit un "Jump from 00484302", mais ce n'est pas le bon, en effet, c'est celui qui est deux lignes plus haut... Continuez donc à monter jusqu'au XOR EAX,EAX en 004842E6. On remarque alors un deuxième saut. Pour y aller, click droit => Go to => JE FROM 00484284. Bon, il n'y a rien d'intéressant non plus ici, alors continuons à monter.

```

00440000 00000000 MOV EAX,EAX
00440001 00000000 MOV EBP,ESP
00440002 00000000 ADD EBP,-54
00440003 00000000 PUSH AC
00440004 00000000 PUSH ESI
00440005 00000000 PUSH EDI
00440006 00000000 MOV EDI,ECX
00440007 00000000 MOV ESI,EDX
00440008 00000000 MOV DWORD PTR SS:[EBP-4],EAX
00440009 00000000 MOV DWORD PTR SS:[EBP+8],EAX
0044000A 00000000 CALL <JMP.&user32.GetActiveWindow>
0044000B 00000000 MOV DWORD PTR SS:[EBP-C],EAX
0044000C 00000000 PUSH 2
0044000D 00000000 MOV EAX,DWORD PTR SS:[EBP-C]
0044000E 00000000 MOV DWORD PTR DS:[4D9C9C],EAX
0044000F 00000000 MOV DWORD PTR DS:[EAX],EAX
00440010 00000000 CALL EAX
00440011 00000000 MOV DWORD PTR SS:[EBP-14],EAX
00440012 00000000 PUSH 2
00440013 00000000 MOV EAX,DWORD PTR SS:[EBP-4]
00440014 00000000 MOV DWORD PTR DS:[EAX+30],EAX
00440015 00000000 PUSH EAX
00440016 00000000 MOV DWORD PTR DS:[4D9C9C],EAX
00440017 00000000 MOV DWORD PTR DS:[EAX],EAX
00440018 00000000 CALL EAX
00440019 00000000 MOV DWORD PTR SS:[EBP-18],EAX
0044001A 00000000 MOV DWORD PTR SS:[EBP-14],EAX
0044001B 00000000 CMP EAX,DWORD PTR SS:[EBP-18]
0044001C 00000000 JNZ SHORT REGCLEAN.004842E6
0044001D 00000000 MOV DWORD PTR SS:[EBP-44],28
0044001E 00000000 LEA EAX,DWORD PTR SS:[EBP-44]
0044001F 00000000 PUSH EAX
00440020 00000000 MOV DWORD PTR SS:[EBP-14],EAX
00440021 00000000 PUSH EAX
00440022 00000000 MOV DWORD PTR DS:[4D9B78],EAX
00440023 00000000 MOV DWORD PTR DS:[EAX],EAX
00440024 00000000 CALL EAX
00440025 00000000 LEA EAX,DWORD PTR SS:[EBP-54],EAX
00440026 00000000 PUSH EAX

```

Local calls from 00484439, 004C560D, 004C688A, 004C886B, 004CE204, 004D15F5, 004D2E22, 004D30CA,

Et là, regardez en bas : on arrive ici par 8 calls ! La solution ici, c'est de placer un BP sur les huit calls, et de voir sur lequel le programme break. Mais je suis gentil, je vais vous dire lequel nous intéresse : c'est le deuxième, le CALL 004C560D. Faites click droit => Go To => Call from 004C560D. Arrivés là, vous continuer à remonter. Or, que voit-on en 004C55DB ? Une string "RES\_Registration\_Error". IL semblerait donc que nous soyons sur la bonne voie.

Au dessus, on a aussi quelque chose qui ressemble à une routine de vérification de sérial (CALL / TEST/ JNZ). Placez donc un Bp sur le CALL.

```

004C55A0 00000000 PUSH EBP
004C55A1 00000000 PUSH REGCLEAN.004C568A
004C55A2 00000000 PUSH DWORD PTR FS:[EAX]
004C55A3 00000000 MOV DWORD PTR FS:[EAX],ESP
004C55A4 00000000 LEA EDX,DWORD PTR SS:[EBP-4]
004C55A5 00000000 MOV EAX,DWORD PTR DS:[EBX+3088]
004C55A6 00000000 CALL REGCLEAN.00463978
004C55A7 00000000 MOV EAX,DWORD PTR SS:[EBP-4]
004C55A8 00000000 CALL REGCLEAN.004D5328
004C55A9 00000000 TEST AL,AL
004C55AA 00000000 JNZ SHORT REGCLEAN.004C5614
004C55AB 00000000 PUSH 10
004C55AC 00000000 LEA EDX,DWORD PTR SS:[EBP-C]
004C55AD 00000000 MOV EAX,DWORD PTR DS:[4DAF1C]
004C55AE 00000000 CALL REGCLEAN.0040671C
004C55AF 00000000 MOV EDX,DWORD PTR SS:[EBP-C]
004C55B0 00000000 LEA ECX,DWORD PTR SS:[EBP-8]
004C55B1 00000000 MOV EAX,REGCLEAN.004C55B0
004C55B2 00000000 CALL REGCLEAN.004D70A0
004C55B3 00000000 MOV EAX,DWORD PTR SS:[EBP-8]
004C55B4 00000000 CALL REGCLEAN.00404BC8
004C55B5 00000000 PUSH EAX
004C55B6 00000000 LEA EDX,DWORD PTR SS:[EBP-10]
004C55B7 00000000 MOV EAX,DWORD PTR DS:[4DAF90]
004C55B8 00000000 CALL REGCLEAN.0040671C
004C55B9 00000000 MOV EAX,DWORD PTR SS:[EBP-10]
004C55BA 00000000 CALL REGCLEAN.00404BC8
004C55BB 00000000 MOV ECX,EAX
004C55BC 00000000 MOV DWORD PTR DS:[4D9B9C],EAX

```

004C55DB:REGCLEAN.004C55DB (ASCII "RES\_Registration\_Error")

Faites F9, la messagebox s'affiche, refaites ok, et Olly break sur notre CALL en 004C55BD. Il est donc fort probable que nous ayons retrouver l'endroit où le sérial est vérifié :).

Rentrez dedans avec F7. On remarque tout d'abord que EAX contient la string "123456". En debuggant avec F8 ligne par ligne, nous arrivons à un premier CMP (il y en a 6 bout à bout) :



```

MOV EDX,DWORD PTR SS:[EBP-4] // met la valeur EBP-4 (le sérial) de la stack dans EDX
MOVZX EDX,BYTE PTR DS:[EDX+1] // met la valeur du 2e caractère du sérial dans EDX
ADD EAX,EDX // ajoute la valeur du 1er caractère et du 2e
ADD EAX,ESI // ajoute le résultat à la valeur du 5e caractère
SUB EAX,90 // soustrait 90h au résultat
MOV ECX,0A // met 10d dans ECX
CDQ // convertit le double mot signé (dword) de EAX en un
// quadruple mot (qword) dans <EDX:EAX>, ce qui met edx
// à 0, pour qu'il puisse stocker le reste de la division
IDIV ECX // divise AX par CX. AX contient le résultat, et EDX le
// reste de la division
MOV EAX,DWORD PTR SS:[EBP-4] // met la valeur EBP-4 (le sérial) de la stack dans EAX
MOVZX EAX,BYTE PTR DS:[EAX+5] // met la valeur du 5e caractère dans EAX
SUB EAX,30 // soustrait 30h à EAX
CMP EDX,EAX // compare EAX et EDX, c'est à dire la valeur du 5e
caractère-30h, et le reste de la division de la somme des valeurs des 1er, 2e, 5e caractères
auxquelles on a soustrait 90h par 10d
JNZ 004D53B7 : Saute si ils ne sont pas égaux vers le XOR.

```

#### 4) Quatrième contrôle

```

CMP BYTE PTR DS:[EAX+2],35 : Compare le 3e caractère du sérial à 35, c'est à dire à la valeur
ASCII du caractère 5.
JBE 004D53B7 : Saute si inférieur ou égal à 5 vers le XOR.

```

#### 5) Cinquième contrôle

```

CMP BYTE PTR DS:[EAX+3],32 : Compare le 4e caractère du sérial à 32, c'est à dire à la valeur
ASCII du caractère 2.
JA 004D53B7 : Saute si supérieur à 2 vers le XOR.

```

#### 6) Sixième contrôle

```

CMP BYTE PTR DS:[EAX+6],38 : Compare le 7e caractère du sérial à 38, c'est à dire à la valeur
ASCII du caractère 8.
JNZ 004D53B7 : Saute si différent de 8 vers le XOR.

```

#### Conclusions :

- 1) Le sérial doit faire 10 caractères.
- 2) La somme des valeurs ASCII du 1<sup>er</sup> et du 5<sup>e</sup> caractère doit être égale ou supérieure à 69.
- 3) La valeur du 5<sup>e</sup> caractère-30h doit être égale au reste de la division de la somme des valeurs des 1<sup>er</sup>, 2<sup>e</sup>, 5<sup>e</sup> caractères auxquelles on a soustrait 90h par 10d, c'est à dire :  
 $(5C - 30h) = \text{Reste de } [(1C + 2C + 5C - 90h) / 10d]$  (C = caractère)
- 4) Le troisième caractère du sérial doit être supérieur à 5.
- 5) Le quatrième caractère du sérial doit être inférieur ou égal à 2.
- 6) Le septième caractère du sérial doit être égal à 8.

Voilà, maintenant vous savez comment fabriquer votre sérial.

Par exemple : 9261458889, 9262458789 ...

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 15/02/2010

Merci aux auteurs de ce soft :)