

.....

Guide du cracking pour débutants



Le Guide pour vous apprendre à cracker facilement !

Par Deamon

Juin 2001 – Août 2004

Bonjour et bienvenue dans ce Guide !

Je me nomme Deamon et je suis l'auteur de ce Guide. Je suis également le webmaster du site www.deamoncrack.fr.st. J'ai créé ce site en juin 2001 (à l'époque j'avais presque 15 ans) dans le but de faire partager mes connaissances sur le cracking en écrivant et publiant des cours destinés en priorité aux débutants. Des mois se sont écoulés et j'ai continué à écrire des cours.

C'est l'intégralité de mes 17 cours que je vous propose dans ce Guide avec en complément quelques pages qui pourront vous aider... Ce Guide vous permettra d'imprimer mes cours plus facilement qu'à partir de mon site.

Je n'ai en aucun cas créé ces cours dans un but lucratif, mais uniquement à titre éducatif et informatif. Le cracking peut être illégal suivant l'utilisation que vous en faites. Il est illégal, et par conséquent punissable par la Loi de, par exemple, modifier un programme protégé par des droits copyrights, tout comme il est interdit d'utiliser un serial trouvé en crackant pour s'enregistrer gratuitement à un logiciel si celui-ci est normalement payant. Par contre il est tout à fait possible d'exercer légalement le cracking en utilisant des crackmes par exemple. Et sachez que je ne vous incite pas du tout à transgresser la Loi.

Ces cours sont destinés à apprendre à cracker progressivement, ainsi ceux qui n'ont jamais cracker pourront apprendre facilement. Pour ceux qui connaissent déjà un peu ils pourront passer les premiers cours, la difficulté des cours étant progressive. Seul un cours est un peu à part c'est celui sur « L'Assembleur » qui vous donne certaines bases de ce langage de programmation qui sont utiles de connaître. Je ne vous conseille pas de commencer par lire ce cours, car c'est un peu difficile et pas forcément agréable à lire. Mais par contre c'est mieux (mais pas indispensable) de l'avoir lu, avant le cours n°10.

Après avoir lu et compris tout ce guide, vous devriez avoir les bases pour pouvoir cracker, mais ne vous imaginez pas non plus que vous allez pouvoir cracker n'importe quel logiciel. Il vous faudra beaucoup d'entraînement pour arriver à un très bon niveau. Mais je pense (ou du moins j'espère) que ce Guide vous apportera beaucoup si vous débutez. Et le domaine du cracking est vraiment vaste, il comporte plusieurs spécialités : serial, keygen, keyfile, unpacking, reverse engineering... J'ai essayé de rendre mes cours assez globaux pour avoir une vision générale des différentes méthodes ou protections utilisées par les programmes, mais toutes n'ont bien sur pas été étudiées dans ce Guide.

Et pour finir je m'excuse par avance des fautes ou erreurs qui peuvent s'être glissées dans ce Guide malgré mes relectures et également de la mise en page qui a perdu en qualité à cause de la conversion des cours en .pdf. Mais vous pouvez si vous le souhaitez retrouver les cours avec la mise en page original sur www.deamoncrack.fr.st.

Bonne lecture et bon apprentissage !

Deamon, le 5-6 août 2004

Sommaire de ce Guide

| | |
|---|------|
| ❖ Outils nécessaires pour cracker..... | p.4 |
| ❖ Sommaire détaillé des cours..... | p.5 |
| ○ Cours n°0 : Pour débiter..... | p.7 |
| ○ Cours n°1 : L'hexadécimal et les sauts conditionnels..... | p.10 |
| ○ Cours n°2 : Cracker un crack-me..... | p.12 |
| ○ Cours n°3 : Cracker un crack-me..... | p.14 |
| ○ Cours n°4 : Trouver le bon serial..... | p.16 |
| ○ Cours n°5 : A mort le crack-me !..... | p.18 |
| ○ Cours n°6 : WinRAR 2.71..... | p.22 |
| ○ Cours n°7 : Cracker un crack-me..... | p.25 |
| ○ Cours n°8 : Teleport Pro 1.29.1820..... | p.29 |
| ○ Cours n°9 : Cracker un prog compressé..... | p.33 |
| ○ Cours n°10 : Cracker un crack-me et trouver le serial..... | p.36 |
| ○ Cours n°11 : Le crackme qui se crack lui-même..... | p.40 |
| ○ Cours n°12 : Reverse Engineering sur la calculatrice Windows. | p.46 |
| ○ Cours n°13 : Donne le serial au monsieur..... | p.54 |
| ○ Cours n°14 : Nag d'un crackme en Visual Basic..... | p.61 |
| ○ Cours n°15 : Crackme avec Keyfile..... | p.64 |
| ○ Cours spécial : L'Assembleur..... | p.74 |
| ❖ F.A.Q : questions et problèmes les plus rencontrés..... | p.89 |
| ❖ Liens de sites sur le cracking..... | p.90 |
| ❖ Remerciements..... | p.92 |

Outils nécessaires

Pour cracker il faut des programmes spécifiques, voici les catégories des programmes qu'on peut utiliser plus ou moins souvent en cracking avec des exemples de programmes que vous pourrez être amené à rencontrer :

- Désassembleur :

Description : il permet d'afficher le code source du programme en assembleur qui nous permet en l'étudiant de voir ce qu'il faut cracker et à quel endroit. Un outil quasi indispensable.

Exemples : WinDasm, IDA, ...

- Débugueur :

Description : il permet d'analyser un programme lors de son exécution et de voir « en direct » la valeur des registres et des opérations que le programme effectue. Peut être utile pour trouver un serial.

Exemples : OllyDbg, SoftIce, ...

- Editeur hexadécimal :

Description : l'éditeur affiche le contenu du programme en hexadécimal et ascii. Il est utilisé pour faire des modifications dans un programme pour le cracker.

Exemples : HexDecCharEditor, Hex WorkShop, WinHex, Hiew, ...

- Analyseur :

Description : afin de connaître le langage de programmation dans lequel a été créé le programme ou le logiciel avec lequel il a été compressé ou crypté.

Exemples : StudPe, Peid, ...

- Patcheur :

Description : si vous souhaitez créer un patch pour cracker un programme, vous aurez donc besoin d'un patcheur. Il va comparer le fichier original et le cracké et à partir de là il va créer le patch.

Exemples : Graphical-PatchMaker, CodeFusion, WinPatchEngine, ...

- Unpackeur :

Description : il permet de décompresser ou décrypter automatiquement les programmes qui le sont. Chaque type de compression demande un unpackeur spécifique.

Exemples : UPX, AsPackDie, ...

Sommaire détaillé des cours

- **Cours n° 0 : Pour débiter**
Explication du fonctionnement de deux logiciels indispensables pour cracker : WinDasm et l'éditeur hexadécimal.
- **Cours n° 1 : L'hexadécimal et les sauts conditionnels**
Comprendre ce qu'est le langage hexadécimal et comment il fonctionne. Et explication des sauts et schéma d'une protection basique pour comprendre ce qu'il faut faire pour cracker un programme.
- **Cours n° 2 : Cracker un crack-me**
Mise en application de la théorie en crackant un petit crackme en modifiant un saut.
- **Cours n° 3 : Cracker un crack-me**
Mise en pratique du 2^{ème} cours en crackant un crackme qui utilise un saut conditionnel pour vérifier le serial.
- **Cours n° 4 : Trouver le bon serial**
Utilisation d'une méthode avec le débogueur de WinDasm pour trouver un serial.
- **Cours n° 5 : A mort le crack-me !**
Cracker et trouver le bon serial d'un crackme, puis créer un keygen après avoir trouvé comment est généré le bon serial en fonction du nom.
- **Cours n° 6 : WinRAR 2.71**
Cracker WinRar en enlevant un nag-screen et effaçant les informations qui nous rappellent que nous utilisons une version d'évaluation. Puis création du crack pour pouvoir le diffuser.
- **Cours n° 7 : Cracker un crack-me**
Un cours simple pour cracker un crackme, rien d'original dans ce dernier.
- **Cours n° 8 : Teleport Pro 1.29.1820**
Cracker le logiciel Teleport Pro qui comporte quelques protections comme la vérification du nom du .exe et qui nous empêche de modifier ce programme. Et essayer de s'enregistrer en plus d'empêcher l'expiration du logiciel.
- **Cours n° 9 : Cracker un prog compressé**
Cracker et trouver le bon serial d'un crackme qui n'a pas de Strings Data Reference étant compressé.
- **Cours n° 10 : Cracker un crack-me et trouver le serial**
Trouver le serial en utilisant le débogueur et en analysant les lignes de code en assembleur. Et faire en sorte que le programme affiche le bon message.
- **Cours n° 11 : Le crackme qui se crack lui-même**
Après avoir trouvé ce qu'il fallait modifier pour cracker le crackme, on va rajouter des lignes d'instructions en assembleur dans le programme pour qu'il se patche lui-même. Puis on lui fera afficher le serial à la place du message d'erreur.

- **Cours n° 12 : Reverse Engineering sur la calculatrice Windows**
Pas vraiment du cracking mais analyse intéressante du programme. On va ajouter un menu dans la calculatrice Windows pour qu'il affiche une msgbox à l'aide d'un éditeur de ressource et d'instructions en assembleur qu'on rajoutera.
- **Cours n° 13 : Donne le serial au monsieur**
Rajouter un bouton au crackme pour qu'il affiche le bon serial quand on clique dessus.
- **Cours n° 14 : Nag d'un crackme en Visual Basic**
Comprendre le fonctionnement des fenêtres en Visual Basic pour pouvoir enlever un nag-screen en se servant uniquement d'un éditeur hexadécimal.
- **Cours n° 15 : Crackme avec Keyfile**
Permet d'apprendre à utiliser OllyDbg à la place de WinDasm. Deux méthodes pour cracker ce crackme : une méthode normale en modifiant des sauts et une méthode où il va falloir comprendre ce qu'il faut faire pour rendre notre keyfile valide en affichant notre nom.
- **Cours spécial : L'Assembleur**
Les bases du langage de programmation le plus utile pour les crackeurs. La connaissance de ces bases permet de mieux analyser un programme par la suite. Explication des registres, de la pile, des différents sauts, opérations, et autres instructions...

Cours n°0

Pour débiter

Qu'est-ce qu'un crack ?

Un crack est un programme qui modifie des octets d'un fichier pour, par exemple, jouer sans le CD, pouvoir s'enregistrer sans avoir le vrai serial...

Ce qu'il vous faut pour cracker :

- WinDasm (v8.9 ou supérieure de préférence).
- Un éditeur hexadécimal (celui que vous préférez).
- Un patcher (obligatoire seulement si vous souhaitez diffuser vos cracks sur le Net)
- Un cerveau en état de marche.
- De la patience.

Utilisation des outils :

WinDasm : c'est un désassembleur/debugueur (nous allons au départ nous intéresser uniquement au mode désassemblage). Il sert à voir les instructions d'un exécutable (*.exe en général) ou d'une DLL.

Après l'avoir téléchargé, lancez-le (mais non pas par la fenêtre)... J'entends déjà des personnes s'exclamer : "oh non! C'est en anglais". Ne vous inquiétez pas, n'importe quelle personne ne parlant pas anglais peut facilement l'utiliser.

Bon tout d'abord cliquez sur "Disassembler" >>> "Font..." >>> "Select Font" et choisissez la police que vous voulez (une des mieux étant Courier New 8pt) sinon vous vous retrouverez avec des signes illisibles, cliquez sur "Ok", puis cliquez de nouveau sur "Disassembler" >>> "Font..." >>> "Save Default Font" pour, vous l'aurez compris, garder la police lors d'une prochaine utilisation. Ensuite cliquez sur "Disassembler" >>> "Disassembler Options" et cochez les 3 cases.

Maintenant ouvrez n'importe quel *.exe pas trop lourd, car c'est un peu long à charger. Et là vous allez me dire : "mais qu'est-ce que c'est ce machin? J'comprend rien". Mais c'est normal. Le langage que vous voyez dans WinDasm est de l'Assembleur (appelé également "asm").

Attention : WinDasm refusera d'ouvrir les fichiers qui sont trop enfouis dans de multiples dossiers et sous-dossiers. Dans ce cas il faudra rapprocher les fichiers à désassembler le plus près de la racine de votre disque dur (par exemple dans c:\).

Je vais vous expliquer tout d'abord l'usage des boutons:



- 1° Ouvrir un exécutable pour le désassembler.
- 2° Sauvegarder les pages du programme désassemblé, ce qui permet de le réouvrir beaucoup plus rapidement la fois suivante. Pour l'ouvrir cliquez sur "Project" >>> "Open Project File...".
- 3° Rechercher du texte.
- 4° Permet de copier le texte une ligne, mais il faut d'abord cliquer à gauche de la ligne afin d'obtenir un point rouge. Utiliser la touche MAJ pour sélectionner plusieurs lignes à la fois.

```

:00405BCD 8B4E28          mov ecx, dword ptr [esi+28]
● :00405BD0 8BE8          mov ebp, eax
:00405BD2 8B4624          mov eax, dword ptr [esi+24]

```

- 5° Aller directement au début du code (il y a, au-dessus, des informations qui n'en font pas parti)
- 6° Aller au point d'entrée du programme.
- 7° Aller à la page choisie (le numéro de page étant inscrit tout en bas à gauche).

Line:336694 Pg 4009 of 6181 Code Data @:004B18BB

- 8° Aller à l'offset choisi.
- 9° Permet d'aller là où le saut "atterrit".
- 10° Permet, après avoir utilisé le 9°, de revenir au saut.
- 11° Permet de rentrer dans le CALL.
- 12° Permet, après avoir utilisé le 11°, de sortir du CALL.
- 13° Lister et trouver les fonctions et modules importés (les APIs) du fichier désassemblé.
- 14° Lister et trouver les fonctions exportées.
- 15° Permet de voir les données du fichier en hexa et ascii.
- 16° Permet de voir les données de la section code du fichier affichée.
- 17° Permet de trouver les items des menus (ex : "copier", "coller", "nouveau"...).
- 18° Permet de trouver les dialogs du programme.
- 19° Un des plus importants. Il permet de trouver les Strings Data References c'est à dire les messages des boîtes de dialogues (ex : "Code invalide")
- 20° Permet d'imprimer des pages (fonction très utile pour pouvoir étudier le programme au bord d'une piscine ;-).

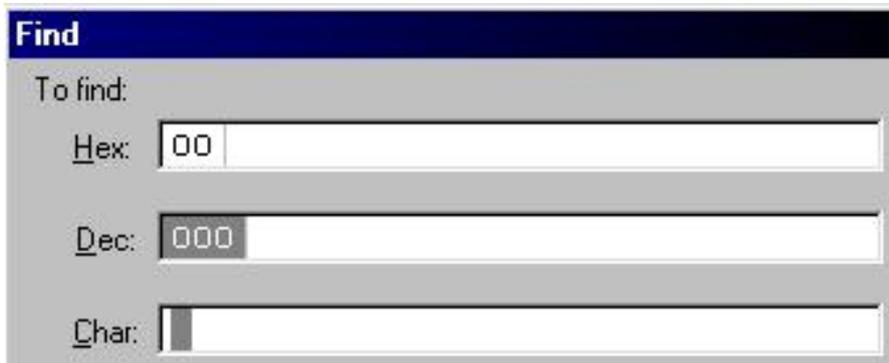
Éditeur hexadécimal ([HexDecCharEditor](#) pour l'exemple, mais ils fonctionnent pratiquement de la même façon) : c'est un outil avec lequel on peut modifier des octets de n'importe quels fichiers. C'est pour ça qu'il vous faut un éditeur hexadécimal et WinDasm si vous voulez cracker. Ce dernier sert à voir ce qu'il faut modifier et où et l'éditeur permet d'effectuer les modifications.

On peut quelques fois cracker avec pour seul compagnon un éditeur hexadécimal (ex : pour enlever les nag-screens, boîtes de dialogue où on peut voir le plus souvent : "Vous n'êtes pas enregistré", ou pour pouvoir utiliser une commande de menu qui ne peut normalement pas être utilisée car grisée) On peut également se servir uniquement de WinDasm mais juste pour trouver les bons serials en utilisant le mode débogueur (je vous expliquerai ça plus tard si vous êtes sage et si vous n'avez pas trop mal à la tête ;-).

Je ne vais pas vous expliquer beaucoup de choses sur cet éditeur hexa car son

fonctionnement est assez basique.

Si vous souhaitez rechercher quelque chose cliquez sur , une fenêtre s'ouvre (c'est beau la technologie non ?) :



The image shows a 'Find' dialog box with a dark blue title bar. Below the title bar, the text 'To find:' is followed by three input fields. The first field is labeled 'Hex:' and contains the text '00'. The second field is labeled 'Dec:' and contains the text '000'. The third field is labeled 'Char:' and is currently empty.

- Taper dans 'Hex' le code hexadécimal que vous voulez modifier.
- 'Dec' ne sert à rechercher du code décimale ce qui est très rarement utilisé.
- 'Char' permet de trouver du texte. Il peut être utile quand on veut cracker uniquement avec l'éditeur.

Voilà dans le cours suivant je vous expliquerai l'hexadécimal et les sauts conditionnels.

Cours n°1

L'hexadécimal et les types de sauts

Comprendre l'hexadécimal ?

Tout d'abord ouvrez un programme avec WinDasm (si vous ne savez pas l'utiliser allez voir le cours précédent). Vous devez apercevoir des lignes semblables à celle-ci :

Exemple :

```
:0040100E      E839000000      Call 0040104C
:00401013      83F807          cmp  eax, 00000007
:00401016      EB1A           jmp  00401032
:00401018      6A40           push 00000040
```

Le numéro en bleu est l'adresse de la ligne pour WinDasm.

Le nombre en noir est l'instructions en hexadécimal correspondant aux instructions rouges. C'est cela qu'on recherchera dans l'éditeur hexadécimal et qu'on modifiera.

Les mots et nombres en rouge sont les instructions en assembleur du programme désassemblé. C'est en fait la "traduction" de l'instruction en hexadécimal en blanc.

En décimal vous comptez comme ça (ou du moins je l'espère...) :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ...

Et bien en hexadécimal on compte de cette manière :

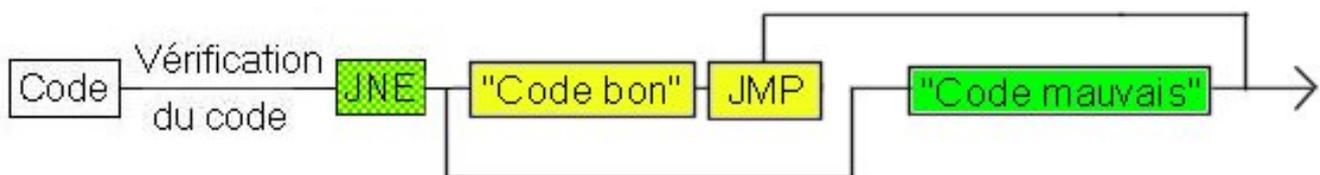
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, ...

Donc par exemple 15 en décimal s'écrira F en hexa, 16 => 10, 17 => 11, 26 => 1A, etc...

Les adresses de WinDasm sont en hexadécimal et dans l'ordre croissant, mais il ne les met pas toutes comme on peut le voir dans l'exemple ci-dessus.

Qu'est-ce qu'un saut ?

Je vais en premier lieu vous montrer le schéma d'une protection utilisée la plupart du temps. Une fois le code rentré il compare ce que vous avez rentré avec le bon code, si c'est pas le même il saute pour arriver vers le mauvais message sinon il affiche le bon message et continue sa route vers de nouvelle aventure en passant au dessus du mauvais message.



JNE est un saut conditionnel. Il saute si le code est faux dans ce cas là (saut si inégal) ou continue sa route si le code est bon.

JMP (= Jump = Saut) est aussi un saut mais inconditionnel puisqu'il saute obligatoirement, même au bord d'un précipice il sautera quand même :-P ...

Et ça donne à peu près ça en assembleur (les adresses sont fictives et je n'ai pas mis l'intégralité de ce passage comme notamment les instructions pour afficher les messages pour ne garder que le nécessaire) :

```
:00401001    #####          Teste si le code est bon
:00401002    75##           jne 00401003
```

Le JNE saute uniquement si le code n'est pas bon.

* Possible StringData Ref from Data Obj ->"Le code est bon"

Windasm indique que "Le code est bon" est le message d'une boîte de dialogue.

```
:00401003    EB##           jmp 00401005
```

Le JMP est un saut. Le numéro à sa droite est l'adresse où il va sauter.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401002(U)

Il indique que la portion de code suivante est appelée par un saut conditionnel (l'adresse de celle-ci est à gauche du "(U)" : 00401002)

```
:00401004    #####          push #####
```

* Possible StringData Ref from Data Obj ->"Le code n'est pas bon"

```
:00401005    #####          #####
```

Après ces schémas, vous avez donc compris (du moins je l'espère) que c'est le `jne` qu'il faut changer. Mais vous allez me dire :

Comment change-t-on un saut conditionnel ?

Regardez le schéma en assembleur et plus précisément la ligne 00401002 avec le `jne`. Dans le code hexa, on voit `75##`. Et bien le `75` c'est notre `jne`. Tout comme le `jmp` qui correspond à `EB`.

Voici une petite liste de sauts conditionnels que vous pourrez trouver le plus souvent :

| Instruction | Valeur hexa | Description |
|------------------|-------------|-----------------|
| <code>jne</code> | 75 ou 0F85 | Saut si inégal |
| <code>je</code> | 74 ou 0F84 | Saut si égal |
| <code>jmp</code> | EB ou E9 | Saut |
| <code>nop</code> | 90 | Pas d'opération |

Pour modifier un saut, il suffira donc de rechercher sa valeur et de la modifier avec l'éditeur hexadécimal. Par exemple pour le schéma ci-dessus on mettra un `je` (il sautera à "Le code est bon" si le code est mauvais) ou `nop` (il ne sautera pas du tout et ira donc directement vers "Le code est bon") à la place de `jne`.

Pour information les 75, 74, EB sont des sauts courts et 0F85, 0F84, E9 des sauts longs. Après les sauts courts on a le nombre (en hexa) d'octets sautés (un octet est un nombre de 2 chiffres qui va donc en hexa de 00 à FF comme peut l'être 75 ou 90) et donc pour les sauts de plus de FF (soit 255 en décimal) octets on utilise des sauts longs à la suite desquelles on met l'adresse de destination. Par exemple EB02 représente un saut (JMP) de 2 octets.

Après la théorie place à la pratique, dans le cours n°2 nous essaierons de cracker un crack-me.

Cours n°2

Cracker un crackme

Objectif :

Cracker un crack-me en modifiant un saut.

Niveau :

Très débutant

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) de 3 Ko (et oui, c'est mieux de l'avoir si vous voulez le cracker ;-)

Allez go !

Le but est donc de le cracker pour arriver sur un bon message quand on clique sur OK. Il faut donc modifier un saut.

Après avoir téléchargé ce crack-me, lancez-le pour voir comment il se présente et relevez le titre du message d'erreur et son contenu.

Désassemblez ensuite ce crack-me avec WinDasm et cliquez sur  pour trouver le message d'erreur. Double-cliquez sur ce message ("Ahhh! Tu n'as pas reussis" ou "FATAL ERROR !!" - titre du message). Vérifiez qu'il n'y en a qu'un seul en double-cliquant encore une fois (dans certains programmes le message peut apparaître plusieurs fois, mais là ce n'est pas le cas).

Remontez un peu et là on aperçoit juste avant le message :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:  
|:00401016(U)
```

Le message est donc appelé par un saut à l'adresse 00401016. On va donc à cette adresse soit en remontant les lignes ou en la recherchant (avec F12 ou "Goto Code Location").

Arrivé à cette ligne, on voit un JMP :

```
:00401016          EB1A          jmp 00401032
```

Pour atteindre la ligne 00401032 (où il saute), on double-clique sur cette ligne pour la sélectionner (elle deviendra verte) et on clique sur  pour arriver à la ligne où il va.

Pour revenir au JMP on clique sur . Il arrive juste avant le mauvais message. Si on regarde un peu en dessous du JMP, on voit un message qu'il saute et qui à l'air

d'être le bon message :

```
* Possible StringData Ref from Data Obj ->"Okay Dokey!"
```

Vous l'avez donc sûrement compris, il faut donc changer ce vilain `jmp` et le remplacer par un `nop` (pas d'opération = pas de saut). On relève donc le code en hexa de la ligne du `jmp` ("`EB1A`"). On ouvre le crack-me avec l'éditeur hexadécimal et il faut rechercher "`EB1A`". Mais il peut, pour certains programmes, y avoir des dizaines de "`EB1A`". Donc on prend le code hexa des instructions qui se retrouvent avant et après le JMP (on recherchera donc : "`83F807EB1A6A40`" et on vérifiera qu'il n'y en a pas plusieurs). Autre méthode plus pratique à mon goût pour retrouver `EB1A` : on peut se repérer avec l'offset, vous trouvez son numéro tout en bas dans la barre d'état quand vous sélectionnez la ligne voulue :

Code Data @:00401016 @Offset 00000616h in Fileetc....

Dans ce cas "`41F5D`" est l'offset, pas besoin des zéros avant le nombre et le "h" indique que c'est en hexadécimal. Pour se rendre à cette adresse dans HexDecCharEditor on fait "Edit" --> "Go To..." ou CTRL+T. Et on tape dans la case "Hex" (car l'offset est en hexadécimal) : `616`, on valide et on se retrouve directement sur le code qu'on souhaite modifier.

Arrivé là où il faut, on remplacera `EB1A` par `9090` ("`nop nop`"). On enregistre, on le lance et là... ça marche !

Attention : si le programme à modifier est déjà ouvert ou désassemblé avec WinDasm vous ne pourrez pas enregistrer le fichier par dessus. Dans ce cas il faut mieux enregistrer sous un autre nom, ce que je vous conseille de faire à chaque fois pour avoir une sauvegarde de l'original.

J'espère que vous avez tout compris. Si vous ne comprenez pas ou si je ne suis pas clair dans mes explications, n'hésitez pas à me contacter à cette adresse : daemon.crack@netcourrier.com

Cours n°3

Cracker un crack-me

Objectif :

Cracker un crack-me pour qu'il nous dise que le code est bon, alors qu'on a tapé un faux serial.

Niveau :

Très débutant

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) de 20 Ko (toujours autant utile :-)

Allez go !

Tout d'abord lancez le logiciel. Il vous demande de taper le sérial. Tapez un nombre et cliquez sur "Check", il vous dira "Incorrect try again!!" ou alors si le crackme vous met que le code est bon courez vite jouer au loto. Notez le message, puis désassemblez le programme avec WinDasm. Regardez les "StringsData References". Vous avez de la chance il y en a que 4. Double-cliquez sur le message que vous avez noté (regardez s'il n'y en a pas plusieurs en double-cliquant une autre fois dessus). Vous devriez arriver là :

```
:0040158D FF1500204000 Call dword ptr [00402000]
:00401593 85C0 test eax, eax << Compare votre sérial avec le vrai
:00401595 7516 jne 004015AD << Saute vers "Incorrect..." s'il est faux sinon
continue à "Correct..."
:00401597 6A40 push 00000040

* Possible StringData Ref from Data Obj ->"CrackMe"
:00401599 6850304000 push 00403050
* Possible StringData Ref from Data Obj ->"Correct way to
go!!" << Message bon
|
:0040159E 6858304000 push 00403058
:004015A3 8B4DE0 mov ecx, dword ptr [ebp-20]

* Reference To: MFC42.MFC42:NoName0077, Ord:1080h
|
:004015A6 E853050000 Call 00401AFE
:004015AB EB14 jmp 004015C1 << Saute au-dessus de "Incorrect..."

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401595(C)
|
```

```
:004015AD 6A40 push 00000040
* Possible StringData Ref from Data Obj ->"CrackMe"
:004015AF 686C304000 push 0040306C
* Possible StringData Ref from Data Obj ->"Incorrect try
again!!" << Message d'erreur
|
:004015B4 6874304000 push 00403074
:004015B9 8B4DE0 mov ecx, dword ptr [ebp-20]
```

Donc vous l'aurez compris, du moins si vous avez compris les cours précédents, on peut donc remplacer le saut JNE en JE, mais quand vous taperez le bon code, il vous dira que le code est faux. Il faut donc mieux mettre un NOP (pas d'opération) et donc il pourra continuer vers "Correct...".

On ouvre le crack-me avec l'éditeur hexadécimal, on recherche "85C075166A40", on vérifie que c'est le bon et on remplace 7516 par 7416 (pour un JE) ou par 9090 (pour deux NOP). On peut également se rendre au 7516 avec l'offset (cf. cours précédent). On enregistre, on le lance on tape n'importe quel nombre et là : "Correct way to go!!"

Et voilà !! C'est aussi simple que ça. Dans le prochain cours on utilisera la fonction débogueur de WinDasm pour trouver le vrai sérial sans rien modifier.

Cours n°4

Trouver le bon serial

Objectif :

Trouver le bon sérial d'un crack-me

Niveau :

Débutant

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) de 20 Ko (le même que le cours précédent)

C'est parti !

Avant de commencer je précise que cette méthode ne fonctionne que très rarement quand on crack mais existe néanmoins et de plus les résultats seront différents suivants la version de WinDasm que vous utilisez. Donc il se peut que cela ne marche pas avec votre WinDasm.

Nous allons dans ce cours nous intéresser de près au débogueur de WinDasm, qui nous permettra de trouver le bon mot de passe pour ce crack-me.
Pour commencer désassemblez-le et regardez un peu au-dessus du bon message :

```
* Reference To: KERNEL32.lstrcmpA, Ord:02FCh
|
:0040158D FF1500204000 Call dword ptr [00402000] <<< Intéressant !!!
:00401593 85C0 test eax, eax
:00401595 7516 jne 004015AD
:00401597 6A40 push 00000040

* Possible StringData Ref from Data Obj ->"CrackMe"
|
:00401599 6850304000 push 00403050

* Possible StringData Ref from Data Obj ->"Correct way to go!!"
```

Il va falloir placer un "Break Point" en 0040158D (c'est-à-dire que le programme sera mis en pause quand il arrivera à cette adresse) sur le CALL car il est juste avant la comparaison (test eax, eax). Donc retenez l'adresse précédente (0040158D) et lancez le débogueur en tapant CTRL+L ou en cliquant sur "Load Process" du menu "Debug". Là une boîte de dialogue vous propose de taper une commande (cette fonction ne sert pratiquement jamais) ne tapez rien et cliquez sur "Load". Vous avez maintenant trois fenêtres : l'écran principal de WinDasm (en haut), une pour les DLL, eax, ebx...(qui ne nous servira pas aujourd'hui) et une autre avec les adresses et les APIs (à droite). Vérifiez que dans cette dernière les 5 cases soient cochées.

Dans l'écran principal cliquez sur "Goto Code Location" et tapez l'adresse que vous avez dû retenir en l'écrivant sur un petit bout de papier (0040158D pour ceux qui n'avaient pas de quoi écrire ;-)). Arrivé à cette ligne, posez un "Break Point Toggle" avec F2 ou en cliquant dans le menu "Debug" sur "Break Point Toggle" (la ligne deviendra jaune ou avec un carré jaune à sa gauche si la ligne est déjà sélectionnée ce qui doit être le cas). Dans la 3ème fenêtre, cliquez sur "Run" ou F9 pour lancer le crack-me. Tapez un sérial bidon (ex :123456789) et appuyez sur "Check". Là, normalement, il va breaker et une fenêtre va s'ouvrir. Regardez-là ! (La fenêtre peut ne pas apparaître suivant les versions de WinDasm).

```
API int Arg00 = Istrcmp(Arg01,Arg02)
```

```
API Address=0040158D, API Return Address=00401593
```

```
Arg01 = (LPCTSTR) 0063f810 -> "123456789" <<< Tiens ! Ça ne serait pas le sérial qu'on vient de taper ???
```

```
Arg02 = (LPCTSTR) 0063f800 -> "<BrD-SoB>" <<< Mais qu'est-ce donc ??? ;-)
```

Vous l'aurez tous compris, le "<BrD-SoB>" est le véritable serial (Attention mettez bien les < et >).

Pour vérifier, fermez d'abord le débogueur. Cliquez sur "Close", puis sur "Terminate" (au-dessus de "RUN") et lancez le crack-me. Tapez le code que vous avez trouvé, et là... "Correct way to go!!"

Mission accomplie ! Vous (enfin nous) avez réussi à trouver le bon code. Mais ne vous enflammez pas en disant à vos amis que vous êtes un cracker expert, car ce crack-me est vraiment facile et vous verrez dans les prochains cours qu'il y en a qui sont vraiment plus durs que celui-là. Et vous verrez très peu de bons crackers se vantant de leurs exploits. Alors à la prochaine !!!.

Cours n°5

A mort le crack-me !

Objectif :

- 1° Le cracker.
- 2° Trouver le bon serial du crack-me
- 3° Faire un keygen.

Niveau :

- 1° Débutant
- 2° et 3° Moyen

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) de 163 Ko
- Un [éditeur Basic](#) (870 ko) pour faire le Keygen.

A l'attaque !

Me voici de retour après quelques mois sans nouveaux cours. Dans ce cours il va falloir cracker ce crack-me pour qu'il nous dise que le code est bon même s'il est faux, il va falloir également trouver le serial (comme dans le cours précédent) avec WinDasm et enfin créer un keygen en Basic. Bref on a du boulot ! ;-)

Qu'est-ce qu'un keygen ?

C'est le plus souvent un *.exe qui vous demande votre nom et vous donne le mot de passe du programme correspondant à votre nom. On en trouve peu par rapport aux cracks ou serials, car il faut trouver comment le programme calcule le serial en fonction du nom. Mais dans ce crack-me c'est facile à voir. Le keygen va être programmé en Basic : c'est un langage très simple qui fait des programmes DOS en *.exe.

Commençons par le cracker !

Comme d'habitude on exécute le crack-me pour voir de quoi il a l'air. Il nous demande un nom et un serial, on tape n'importe quoi et il nous dit "Error" - "Wrong Code !". Si vous regardez dans "About" l'auteur nous dit qu'il ne faut pas le patcher mais bon nous on fait ça pour apprendre. Donc on ouvre WinDasm et on désassemble le crack-me. On clique sur "StringsData References", on recherche le message d'erreur ("Wrong Code") et on voit ça :

```

:00441756 8B55F8    mov edx, dword ptr [ebp-08]
:00441759 58              pop eax
:0044175A E8E523FCFF     call 00403B44
:0044175F 7517          jne 00441778 <<< Intéressant !!!
:00441761 6A00          push 00000000
:00441763 668B0DD8174400  mov cx, word ptr [004417D8]
:0044176A B202          mov dl, 02

* Possible StringData Ref from Code Obj -> "Right Code" <<< Message bon
|
:0044176C B8E4174400     mov eax, 004417E4
:00441771 E802FBFFFF     call 00441278
:00441776 EB15          jmp 0044178D <<< Si le code est bon il saute au dessus du message d'erreur.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044175F(C) <<< On arrive au message d'erreur par un saut qui est à cette adresse.
|
:00441778 6A00          push 00000000
:0044177A 668B0DD8174400  mov cx, word ptr [004417D8]
:00441781 B201          mov dl, 01

* Possible StringData Ref from Code Obj -> "Wrong Code" <<< Message d'erreur
|
:00441783 B8F8174400     mov eax, 004417F8
:00441788 E8EBFAFFFF     call 00441278

```

Il y a un **JNE** avant le bon message qui saute, si le code est faux, au message d'erreur, il faut donc l'empêcher de sauter. On peut mettre un JE à la place de JNE mais si le code est juste il dira qu'il est faux (vous me suivez ?) : en clair il fera l'inverse. Donc il vaut mieux mettre un NOP (pas d'opération) et donc il ne sautera pas et passera alors par le bon message.

Bon on ouvre le crack-me avec l'éditeur hexadécimal et on change **7517** qu'on remplace par **9090**. (Si vous ne savez pas utiliser l'éditeur allez voir les premiers cours).

Normalement vous devriez avoir tout compris.... ou alors j'explique mal. Si vous n'avez pas compris relisez le début de ce cours ou les anciens cours avant de lire la suite. Et si vous ne comprenez vraiment pas [écrivez-moi](#).

Trouver le bon serial !

Si vous allez lu et compris le cours précédent vous ne devriez pas avoir de problèmes pour trouver le bon serial.

Désassemblez le programme avec WinDasm, recherchez le message d'erreur et regardez bien :

```

:00441751 E89E23FCFF     call 00403AF4
:00441756 8B55F8    mov edx, dword ptr [ebp-08]
:00441759 58              pop eax
:0044175A E8E523FCFF     call 00403B44 <<<C'est ici qu'il faudra breaker
:0044175F 7517          jne 00441778 <<< Saute vers le bon ou le mauvais message
:00441761 6A00          push 00000000

```

```
:00441763 668B0DD8174400  mov cx, word ptr [004417D8]
:0044176A B202  mov dl, 02
```

* Possible StringData Ref from Code Obj -> "Right Code"

```
|
:0044176C B8E4174400  mov eax, 004417E4
:00441771 E802FBFFFF  call 00441278
:00441776 EB15  jmp 0044178D
```

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

```
|:0044175F(C)
|
:00441778 6A00  push 00000000
:0044177A 668B0DD8174400  mov cx, word ptr [004417D8]
:00441781 B201  mov dl, 01
```

* Possible StringData Ref from Code Obj -> "Wrong Code"

Comme vous avez vu il faudra poser un Break Point sur le Call qui est juste avant le JNE qui va vers le bon ou le mauvais message (le plus souvent le CALL a étudié est juste au-dessus du saut). Donc retenez l'adresse du Call (0044175A). On lance le mode débogueur en cliquant sur "Load Process" du menu "Debug" ou en faisant CTRL +L. Cliquez ensuite sur "Load". Dans la fenêtre de droite vérifiez que toutes les cases soient cochées. Allez à l'adresse du Call en faisant "Goto Code Location" et posez un "Break Point" en double-cliquant sur la ligne voulue (celle du Call) et en appuyant sur F2 ou en cliquant à gauche de la ligne avec le bouton CTRL enfoncé. Il doit y avoir désormais un carré jaune à gauche de la ligne.

Lancez le crack-me en cliquant sur "Run" (à droite) ou en appuyant sur F9. Tapez votre nom (Deamon pour moi) et un serial bidon (ex : 123456789) et cliquez sur "Ok". Il va biper car il s'est arrêté au Call et une fenêtre apparaît où on peut voir le nom qu'on a mis (Deamon) le serial bidon en bas (123456789) et entre les deux le serial : DeamonDeamon625g72 . Cliquez sur "Terminate" pour fermer le débogueur et réessayez la même opération avec un autre nom ("Crack" par exemple). Vous trouverez alors CrackCrack625g72. On comprend donc qu'en fait le crack-me met 2 fois votre nom et rajoute "625g72". Ce sera donc facile de créer un keygen.

Faire un keygen en Basic

Lancez l'éditeur de Basic en double-cliquant sur QB.EXE ou son raccourci QB. Et oui c'est moche mais c'est normal c'est du DOS, c'est sur que ça vous change des beaux programmes sous Windows. ;-)

Voici le code source à taper (en bleu) :

```
1° CLS
2° PRINT "Keygen pour le crack-me n°3"
3° PRINT
4° INPUT "Quel est votre nom"; A$
5° PRINT "Salut "; A$; " ! Ton serial est : "; A$; A$; "625g72 !"
```

Explication :

- 1° Sert à rendre la fenêtre vierge dans laquelle va apparaître le message.
- 2° **PRINT** affiche le message, qui est entre parenthèses, à l'écran.
- 3° Rien à afficher : pour sauter une ligne.
- 4° **INPUT** sert à poser des questions où on peut répondre. Il faut mettre la question entre parenthèses, ne pas mettre de "?" (il le met tout seul) et mettre "; x\$" après le message (on peut remplacer "x" par ce qu'on veut). Le "x\$" est le nom de la réponse, que l'on met, pour l'ordinateur.
- 5° Affiche le message et "A\$" qui est la réponse. Donc si vous répondez "Deamon", A\$=Deamon (ATTENTION : il ne faut pas mettre "x\$" entre parenthèses).

Vous pouvez à tout moment essayer votre programme en faisant MAJ+F5 ou en cliquant sur "Démarrer" dans "Exécution".

Ensuite enregistrez votre code source dans "Enregistrer sous..." du menu "Fichier". Pour le mettre en EXE il faut cliquer sur "Créer un fichier EXE" dans "Exécution" (mettez-le en fichier EXE autonome). Et voilà le keygen est créé dans le dossier de l'éditeur Basic.

J'espère que vous avez compris les commandes, sinon allez voir l'aide de l'éditeur ou regardez les autres programmes qui sont avec (bien que beaucoup ne marchent pas).

Attention : sous Windows XP il semblerait que le keygen créé se ferme automatiquement, la seule méthode que j'ai trouvée est de rajouter STOP à la fin, je ne sais pas si c'est le mieux mais ça a l'air de marcher.

Deamon, août 2001

Cours n°6

WinRAR 2.71

Objectif :

1° Le cracker :

- Virer le Nag Screen
- Remplacer le "40 days trial copy" par votre nom ou mettre "registered version"
- Enlever le "(evaluation copy)" qui est dans la barre de titre.

2° Fabriquer le crack

Niveau :

1° Débutant - Moyen

2° Trop facile

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- [WinRAR 2.71](#) (version us : 570 Ko)

Virer le Nag Screen :

Et voici le 6ème cours que j'écris et cette fois on va s'attaquer à un vrai logiciel plutôt qu'à un crack-me. Et ce logiciel tout le monde doit le connaître : c'est WinRar, le célèbre décompresseur. La version est la 2.71 et celle en anglais (us).

Donc on installe WinRAR on le lance et là au bout de quelques secondes apparaît un nag screen (pour ceux qui savent pas ce que c'est, c'est un message qui apparaît pour nous dire qu'on est pas enregistré ou par exemple que le code n'est pas bon). Celui-ci nous dit "Please note that WinRAR is shareware", ça alors, dans quelques minutes il ne le sera plus. Le titre du message est "Please registrar", très original pour un shareware. Donc à ce moment on va laisser tomber les WinDasm et autres désassembleurs et on va plutôt prendre l'éditeur hexadécimal.

On ouvre winrar.exe avec l'éditeur et on recherche notre message. Donc on fait rechercher texte (faites "rechercher" et taper "Please" avec la majuscule dans la partie recherche de texte 'Char'). Là bizarrement on ne trouve rien, mais c'est normal car les messages comme le notre sont marqués avec des blancs (00 en hexa) entre les lettres.

Ex : **Please** --> **50 6C 65 61 73 65** et avec les blancs (ce que j'appelle des blancs ce sont des vides (00) et surtout pas des espaces (20)) cela fait **P l e a s e** --> **50 00 6C 00 65 00 61 00 73 00 65**

Pour mettre des blancs avec HexDecCharEditor par exemple on tape "Please", on sélectionne le "e" et on appuie sur la touche "Inser" puis on fait pareil pour les autres lettres. J'espère que que vous avez compris jusque là.

Bon, on trouve 3 "Please" le premier c'est le titre de notre message, le 2ème c'est

le message lui-même et le 3ème c'est un autre Please qui ne nous intéresse pas du tout. Alors regardons le 2ème et juste avant le "P" de Please on voit **82 00**. Ce qui nous intéresse c'est le 82, c'est ce qui déclenche le message. On remplace donc le 82 par 7E. Me demander pas pourquoi on met 7E, c'est comme ça. On enregistre et là le Neg screen a disparu. C'est pareil pour la plupart des logiciels qui ont des nags screen, il faut remplacer le **82** qui est un peu avant le début du message par **7E**.

Remplacer le "40 days trial copy" par votre nom ou par "Registered Version" :

Bon alors là c'est facile et ça va donc être vite fait. On garde encore l'éditeur hexadécimal sous la main ou plutôt sous le curseur et on recherche "40 days evaluation copy" (tapez pas toute la phrase, mettez juste le début ou un nom précis). Donc on recherche "days" par exemple. Si vous avez bien suivi ce que j'ai expliqué vous devez trouver le message. Sinon regardez la partie du cours au dessus. Je vous ai dit que pour la plupart des messages il y avait des blancs entre les lettres. Donc on recherche "d a y s" (voir ci-dessus). Il y en a deux dans tout le programme mais il n'y a que le premier avec "40 days". Et donc là vous pouvez mettre ce que vous voulez, en ne mettant surtout pas plus de 18 caractères (nombre de lettres + les espaces de "40 days trial copy") car il ne faut jamais rajouter de bytes. Faites attention aussi à ne pas effacer les blancs (00). Mettez par exemple "Registered Version" qui fait pile 18 lettres (j'aurais bien voulu le mettre en français "version enregistrée" mais ça fait 1 lettre de trop :- (). Et voilà, passons à la suite voulez-vous.

Enlever le vilain "(evaluation copy)" qui est dans la barre de titre :

Passons à la suite qui est un peu plus dur. Vous voyez dans la barre de titre : "WinRAR - WinRAR (evaluation copy)" ce qui fait un peu désordre pour un de vos logiciels. Donc on va d'abord essayer de l'effacer avec l'éditeur hexadécimal. On recherche donc "evaluation copy" avec les blancs bien sûr. Et là on met des espaces (20h) à la place des lettres sans remplacer les blancs (00h) par les espaces (20h). On enregistre (**n'effacez pas la dernière version car on en aura besoin bientôt**) et on lance armstrong (calembour :-P), plus sérieusement on lance le programme, mais là je ne sais pas si c'est à cause de ma vanne foireuse mais il y a un problème ! Les parenthèses restent toujours mais pourtant on ne les a pas vues dans l'éditeur. On va donc (enfin me diront certains) avoir besoin de WinDasm.

On désassemble WinRAR, et on recherche "evaluation copy" dans "search" --> "Find text". Il en trouve un seul. Et on peut voir :

```
:0041BA15 803DCC6C460000    cmp byte ptr [00466CCC], 00
:0041BA1C 752E    jne 0041BA4C <-- JNE qui saute au dessus d'evaluation copy
```

* Possible Reference to String Resource ID=00873: "evaluation copy" <-- Notre "evaluation copy"

```
|
:0041BA1E 6869030000    push 00000369
:0041BA23 E894C6FEFF    call 004080BC
:0041BA28 50    push eax
```

* Possible StringData Ref from Data Obj -> "(%s)"

```
|
:0041BA29 68E95F4600    push 00465FE9
```

```

:0041BA2E 8D8D00FEFFFF    lea ecx, dword ptr [ebp+FFFFFFE0]
:0041BA34 51    push ecx
:0041BA35 E8AACA0300    call 004584E4
:0041BA3A 59    pop ecx
:0041BA3B 8D9500FEFFFF    lea edx, dword ptr [ebp+FFFFFFE0]
:0041BA41 03C2    add eax, edx
:0041BA43 50    push eax
:0041BA44 E84B0B0400    call 0045C594
:0041BA49 83C40C    add esp, 0000000C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041BA1C(C)
|
|:0041BA4C 8D8500FEFFFF    lea eax, dword ptr [ebp+FFFFFFE0] <-- Là où arrive le JNE
:0041BA52 50    push eax

```

Inutile de vous dire donc ce qu'on va faire, hein ? Bon allez je vous le dis quand même pour ceux qui n'aurais pas compris. On va faire un barbecue ! Roh mais que tu es naïf... Il faut tout simplement remplacer le **JNE** par **JE**, donc **75** en **74** avec l'éditeur hexadécimal. Si vous ne savez pas comment faire lisez donc les cours précédents.

Fabriquer le crack :

Alors là c'est vraiment simple. Vous voyez ce que c'est simple ? Et bien encore plus simple ! Vous lancez WinPatchEngine, vous sélectionnez le fichier original et celui cracké, vous tapez le nom du programme, votre nom... Et vous appuyez sur "Compare" et là vous avez un beau crack tout neuf fait par vous (enfin presque vous car je vous ai aidé un peu quand même, non ?) et vous pouvez le diffuser sur le net.

Deamon le 29 décembre 2001

Cours n°7

Cracker un crack-me

Objectif :

Cracker un crackme (original non ? Tout comme le titre du cours d'ailleurs... Manque d'inspiration me diront certains...)

Niveau :

Débutant

Ce qu'il vous faut :

Le kit du parfait petit crackeur comme d'habitude qui comprend :

- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) (10 Ko)

Attention la patience et la réflexion ne sont pas compris dans ce kit, vous devrez les apporter vous-même. Bien que pour ce qui est de la réflexion je vous aiderai un peu. ;-)

C'est parti !

Pour ce 7ème cours nous allons nous intéresser à un crack-me avec un schéma de protection un peu différent des autres mais pas trop dur non plus. Nous allons simplement le cracker pour qu'il nous dise toujours "code bon".

Donc on lance le crack-me, on fait "Help" puis "Register", on tape son nom et un serial bidon et on clique sur Ok. Et là apparaît un vilain message : "No luck" - "No luck there, mate !". On note le message comme d'habitude et on désassemble le crack-me avec notre fameux WinDasm. Ah qu'est-ce qu'on ferait si on ne l'avait pas celui-là ?

On va dans les "Strings Data References" et on trouve facilement notre message d'erreur car il n'y a pas 36 messages fort heureusement. On double-clique dessus pour arriver à l'endroit où il se trouve. On double-clique encore plusieurs fois pour voir s'il n'y en a pas plusieurs et effectivement on en voit deux. Pour savoir lequel il faut prendre on va utiliser le débogueur de WinDasm. Donc on fait CTRL+L pour le lancer et on va poser un BreakPoint juste avant l'un des deux. On va commencer par le deuxième (pourquoi le 2ème d'abord ? aucune idée !). On va au deuxième message d'erreur et on pose un BreakPoint (BP) sur le PUSH juste avant le message (en double-cliquant sur la ligne puis en appuyant sur F2) :

```
:004013AC 5E    pop esi
:004013AD 6A30    push 00000030 <-- C'est ici que l'on va breaker
```

```
* Possible StringData Ref from Data Obj -> "No luck!" <-- 2ème message d'erreur
|
:004013AF 6860214000    push 00402160
```

```
* Possible StringData Ref from Data Obj -> "No luck there, mate!"
|
```

```
:004013B4 6869214000 push 00402169
:004013B9 FF7508 push [ebp+08]
```

Après avoir poser le BreakPoint, on clique sur "Run" ou F9. On entre un nom ("Deamon" pour moi) et un serial bidon (123456) et on fais Ok. Là le message d'erreur apparaît. Donc WinDasm n'a pas breaké, alors ce ne doit pas être le bon message. On va essayer avec le premier message qu'on avait aperçu. On enlève le BreakPoint (en étant sur la ligne du BP et en faisant F2) et on en place un sur le PUSH qui est juste avant le premier message :

```
:00401364 E8AD000000 Call 00401416
:00401369 6A30 push 00000030 <-- C'est ici que l'on va breaker

* Possible StringData Ref from Data Obj ->"No luck!" <-- 1er message d'erreur
|
:0040136B 6860214000 push 00402160

* Possible StringData Ref from Data Obj ->"No luck there, mate!"
|
:00401370 6869214000 push 00402169
:00401375 FF7508 push [ebp+08]
```

Ou plus rapide on pose un BP sur les 2 et on voit où il breake.

On revient dans le crack-me, on entre un nom et un serial, on clique sur Ok et là il break, donc c'est sur ce message qu'il faut travailler (mais vous verrez par la suite que le deuxième message n'est pas là pour rien).

On quitte le débogueur (en cliquant sur "Terminate"), on revient sur le premier message et on regarde un peu au dessus :

```
* Referenced by a CALL at Address: <-- Le message d'erreur est appelé par un CALL
|:00401245 <-- Adresse du CALL
|
:00401362 6A00 push 00000000

* Reference To: USER32.MessageBeep, Ord:0000h
|
:00401364 E8AD000000 Call 00401416
:00401369 6A30 push 00000030

* Possible StringData Ref from Data Obj ->"No luck!" <-- Message d'erreur
|
:0040136B 6860214000 push 00402160

* Possible StringData Ref from Data Obj ->"No luck there, mate!"
|
:00401370 6869214000 push 00402169
:00401375 FF7508 push [ebp+08]

* Reference To: USER32.MessageBoxA, Ord:0000h
|
:00401378 E8BD000000 Call 0040143A
:0040137D C3 ret
```

Ici on ne voit pas de saut conditionnel mais on voit que le message d'erreur est appelé par un CALL. Donc on va à l'adresse du CALL (00401245) :

```
:00401240 58    pop eax
:00401241 3BC3    cmp eax, ebx
:00401243 7407    je 0040124C
:00401245 E818010000    call 00401362 <-- Va au message d'erreur
:0040124A EB9A    jmp 004011E6 <-- Saute au-dessus
```

Donc là on va nopper le CALL et on verra bien si le JMP nous envoie sur le bon message. On ouvre le crack-me avec un éditeur hexadécimal et on remplace E818010000 par 9090909090. On enregistre sous un autre nom par garder l'original et on le lance. Mais là ça ne marche pas. Donc on regarde où on va si on nopper également le JMP. En dessous du JMP il y a :

```
:0040124A EB9A    jmp 004011E6
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401243(C)
|
:0040124C E8FC000000    call 0040134D <-- Va sur le bon message
```

Il faut alors nopper également le JMP en remplaçant le EB9A par 9090. Et là ça marche !

Mais cette méthode n'est pas très fine. En effet il y a mieux à faire en changeant un seul octet (un octet est composé de deux chiffres hexadécimaux, par exemple 74 ou E8 est un octet). Reprenons la partie où se situe le CALL du mauvais message :

```
:00401240 58    pop eax
:00401241 3BC3    cmp eax, ebx <-- Compare eax et ebx (donc sûrement notre serial et le bon)
:00401243 7407    je 0040124C <-- Saute si égal sur le CALL du bon message
:00401245 E818010000    call 00401362 <-- Va au message d'erreur
:0040124A EB9A    jmp 004011E6

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401243(C)
|
:0040124C E8FC000000    call 0040134D <-- Va sur le bon message
```

Il suffit donc de mettre un JMP (EB) à la place du JE (74) pour qu'il saute toujours sur le CALL qui envoie sur le bon message. Donc on remplace 7407 par EB07. On teste et c'est bon.

Voilà donc ce crack-me est cracké, mais on va s'amuser à faire autre chose. Quand vous lancez le crack-me cracké et que vous mettez un nombre à la place du nom là il affiche le message d'erreur puis le bon message. Vous vous souvenez du deuxième message d'erreur ? Et bien c'est à cause de celui qu'il s'affiche (vous pouvez utiliser le débogueur comme on a fait auparavant pour savoir quel message apparaît, mais en mettant bien sûr un numéro au lieu du nom).

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040138B(C)
```

```

/
:004013AC 5E    pop esi
:004013AD 6A30   push 00000030

* Possible StringData Ref from Data Obj -> "No luck!"
/
:004013AF 6860214000   push 00402160

* Possible StringData Ref from Data Obj -> "No luck there, mate!"
/
:004013B4 6869214000   push 00402169
:004013B9 FF7508       push [ebp+08]

```

Comme vous pouvez le constater, le mauvais message n'est pas appelé au téléphone (oui je sais c'est nul) mais par un saut conditionnel. Allons donc voir en 0040138B :

```

:00401389 3C41    cmp al, 41
:0040138B 721F    jb 004013AC <-- Saute juste avant le mauvais message
:0040138D 3C5A    cmp al, 5A

```

Donc là c'est pas trop compliqué on remplace juste le 721F par 9090 pour le nopper. Et voilà. Vous pouvez maintenant vous enregistrer si par exemple vous êtes 007 ou K2000.

Et voilà un autre crack-me qui ne nous a pas trop résisté. J'espère que vous avez tout compris (sinon il faut relire le cours) et à un prochain cours.

Deamon le 22 avril 2002

Cours n°8

Teleport Pro 1.29.1820

Objectif :

Cracker le logiciel Teleport Pro en version 1.29.1820

Niveau :

Moyen

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Le logiciel [Teleport Pro v.1.29.1820](#) (870 Ko)
- Ne pas être pressé !

C'est parti !

On va s'attaquer dans ce cours à un vrai logiciel. Il s'agit de Teleport Pro (version 1.29.1820). Ce programme est très intéressant à cracker, car il y a plusieurs éléments à modifier.

Tout d'abord on va modifier un "truc" pour éviter que vous vous énerriez (enfin vous je ne sais pas mais moi ça m'énerve :-o), car le logiciel démarre seulement si l'exécutable se nomme « pro.exe » donc si vous voulez l'enregistrer sous un autre nom (ce que je vous conseille pour garder l'original et de toute façon vous ne pouvez pas enregistrer vos modifications sur le fichier qui est déjà ouvert avec WinDasm), vous seriez obligé de modifier le nom de l'original puis du programme cracké pour voir si ça marche. Ce qui serait pénible à faire chaque fois.

On va donc voir ce qu'il nous dit quand le nom du fichier n'est pas bon. On modifie donc « pro.exe » en « pro1.exe » par exemple et on le lance. Là un "beau" message, qu'on va s'empresse d'enlever, apparaît : "This program's name have been changed; please rename...".

On va donc dans WinDasm et on désassemble notre programme. On cherche ensuite notre message dans les Data Strings References et on double-clique dessus. Ce message a une seule référence :

```
:0040BE79 85C0    test eax, eax
:0040BE7B 59      pop ecx
:0040BE7C 7413    je 0040BE91
:0040BE7E 53      push ebx
:0040BE7F 53      push ebx
```

* Possible StringData Ref from Data Obj -> "This program's name has been changed; "

En regardant au dessus du message on peut voir un "TEST" suivi d'un saut, ici "JE". Il faut donc remplacer ce JE en JMP soit 74 par EB.

Rappel : Pour trouver plus facilement l'emplacement de ce 74 dans l'éditeur hexadécimal, double-cliquez dans WinDasm sur la ligne du « JE » et regardez en bas de l'écran dans la barre d'état : "Line:23454 Pg 273 of 2470 Code Data @ :0040BE7C @Offset 0000BE7Ch in File:pro.exe"

L'offset représente l'emplacement du "JE" et donc retenir le numéro qui est entre les « 0 » et le « h » qui signifie que c'est en hexadécimal. Donc ici : **BE7C**.

Allez dans votre éditeur hexadécimal, ouvrez le fichier et pour « HexDecCharEditor » faites « Edit » à « Go to... » ou bien CTRL + T. Tapez dans la case hexadécimal « hex » BE7C et faites « Ok ». Vous arriverez directement sur notre **74** que vous changerez en **EB**.

Une fois ce changement effectué, il faut voir si ça marche. Donc on lance le fichier modifié en lui mettant un autre nom que « pro.exe ». Et là il nous met un nouveau message nous disant que le fichier a été altéré, peut-être par un virus... Ben forcément qu'il a été altéré, on vient de le modifier. Donc on recherche ce message dans WinDasm.

Vous pouvez dans ce dernier, pour chercher une phrase, faire « Search » à « Find Text » et taper un morceau de la phrase comme par exemple « been altered » ou « by a virus ». Faites attention qu'il n'y en ait pas plusieurs. Et évitez de prendre les fins des phrases car elles sont souvent coupées.

```
:0040BECD 741C   je 0040BEEB
:0040BECF A144D74700   mov eax, dword ptr [0047D744]
:0040BED4 3B30   cmp esi, dword ptr [eax]
:0040BED6 7413   je 0040BEEB
:0040BED8 53     push ebx
:0040BED9 53     push ebx
* Possible StringData Ref from Data Obj -> "This program has been altered, "
```

On peut voir deux "JE" avant le message se reportant au même endroit. Vous pouvez donc prendre celui que vous voulez et le remplacez par un JMP (EB en hexadécimal pour ceux qui ne le seraient pas encore).

Donc là on est enfin prêt.

Faites maintenant « Help » à « Register... ». On voit que le logiciel est limité à 40 utilisations. Bon alors on fait comme d'habitude : nom, société et le code (bidon bien sûr). On clique sur « Ok » et là ...[roulements de tambour]... un message d'erreur qui dit que le code n'est pas bon. Ben oui, vous voulez que ça soit quoi d'autre ?

On fait une petite recherche dans WinDasm. Si vous avez du mal à trouver dans les Data String References (dans le haut de la liste) faites donc une recherche de texte (voir plus haut) avec « registration number » par exemple. Faites attention à ne pas faire de fautes en recopiant. Bon et donc on aperçoit notre beau message :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00427CBE(C)
|
:00427CF1 8945F0   mov dword ptr [ebp-10], eax
```

* Possible Reference to String Resource ID=07033: "We're sorry! The registration number you entered appears to"

Ce message est appelé par un saut conditionnel en **00427CBE**, donc on se dirige vers cette adresse et on voit ceci :

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00427C78(C)

```
|  
:00427CB0 57    push edi  
:00427CB1 E80F090000  call 004285C5  
:00427CB6 85C0    test eax, eax  
:00427CB8 A1ECF44700  mov eax, dword ptr [0047F4EC]  
:00427CBD 59    pop ecx  
:00427CBE 7531    jne 00427CF1  
:00427CC0 8945F0    mov dword ptr [ebp-10], eax
```

* Possible Reference to String Resource ID=07032: "You haven't entered a valid username. Your username must be"

On tombe sur un « **JNE** » et on voit qu'il fait un test avant, donc on aurait tendance à dire qu'il faudrait le modifier (cette manie de modifier tous les sauts maintenant...). Mais regardez en dessous : « **You haven't entered a valid name** ». Donc si on modifie ce saut on tombera sur ce message. On regarde alors encore au dessus du saut et on qu'il est appelé par un autre saut conditionnel. On y va :

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00427C33(C)

```
|  
:00427C6F 8BBFD5000000  mov edi, dword ptr [edi+000000D5]  
:00427C75 395FF8    cmp dword ptr [edi-08], ebx  
:00427C78 7536    jne 00427CB0  
:00427C7A A1ECF44700  mov eax, dword ptr [0047F4EC]  
:00427C7F 8945F0    mov dword ptr [ebp-10], eax
```

* Possible Reference to String Resource ID=07031: "You must enter your username in the Name field, exactly as y"

Là on se retrouve dans le même cas. Si on modifie ce saut on tombera sur le message d'erreur qui se trouve en dessous du « **JNE** ». Ce saut est appelé par un autre saut. On n'en finit plus.

On arrive donc en **00427C33** :

```
:00427C2F 3945E8    cmp dword ptr [ebp-18], eax  
:00427C32 59    pop ecx  
:00427C33 753A    jne 00427C6F  
:00427C35 A1ECF44700  mov eax, dword ptr [0047F4EC]  
:00427C3A 8945F0    mov dword ptr [ebp-10], eax
```

* Possible Reference to String Resource ID=07026: "Thank you! Your copy of Teleport Pro is now registered. AI"

Et là que voit-on ? Un beau message nous disant qu'on est enregistré. Youpi, youpi ! On va donc nopper ce saut en remplaçant 753A par 9090 pour qu'il tombe toujours sur ce message. On modifie et bien sûr on essaye. Et là ça marche enfin !

Relancez le programme et regardez dans Help à About. Vous n'êtes plus enregistré ! Il suffit juste de retaper le code à chaque fois que vous lancez le logiciel. Mais bon... On va plutôt s'intéresser à la fenêtre d'About où l'on peut voir « **This copy of Teleport Pro is UNREGISTERED** ». On va donc chercher ce dernier mot dans WinDasm. On tombe ici :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00416D9E(C)
|
:00416DD0 3898CD040000    cmp byte ptr [eax+000004CD], bl
:00416DD6 7430    je 00416E08
:00416DD8 894DF0    mov dword ptr [ebp-10], ecx
* Possible Reference to String Resource ID=07093: "This copy of Teleport Pro is UNREGISTERED."
```

Et un peu plus bas :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00416DD6(C)
|
:00416E08 8B900D050000    mov edx, dword ptr [eax+0000050D]
:00416E0E 395AF8    cmp dword ptr [edx-08], ebx
:00416E11 7537    jne 00416E4A
:00416E13 8BB809050000    mov edi, dword ptr [eax+00000509]
:00416E19 894DF0    mov dword ptr [ebp-10], ecx
* Possible Reference to String Resource ID=07102: "Licensed to %s"
```

Donc on va modifier le "JE" (en 00416DD6) en JMP (EB) pour arriver tout le temps sur « **Licensed to...** ». Mais si on regarde au dessus du message de « **UNREGISTERED** ». Il y a :

```
:00416D9E 7430    je 00416DD0
:00416DA0 894DF0    mov dword ptr [ebp-10], ecx
* Possible Reference to String Resource ID=07076: "The free evaluation period for Teleport Pro has expired."
```

Ça c'est le message qu'il nous mettra passé les 40 utilisations. Donc on met un **JMP** à la place du **JE**, qui arrivera sur le JMP du 00416DD6 qui sautera lui à son tour vers « **Licensed...** ». Et voilà !

Deamon le 11 juillet 2002

Cours n°9

Cracker un prog compressé

Objectif :

Cracker et trouver le serial d'un programme qui a été compressé

Niveau :

Assez facile (ou alors c'est que j'explique mal)

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) (130 Ko)
- Le logiciel pour le décompresser : [UPX](#) (120 Ko)

Bon et bien allons-y gaiement !

Pour ce cours je vais utiliser un WinDasm patché, car avec les autres versions on ne verra pas les Data Strings References. Ne me demandez pas pourquoi, je n'en sais rien. Je vous conseille de garder votre ancienne version de WinDasm + la version patchée. Car ils ne donnent pas les mêmes résultats pour certains programmes. Ce patch vous permettra également de voir les Data String References des logiciels programmés en Visual Basic et il mets des couleurs (comme c'est joli :-).

Bon allez je crois que tout est prêt, alors on peut y aller. Tout d'abord comme d'habitude, en première phase, l'observation. On lance donc ce crack-me qui a été programmé en Delphi (on peut connaître le langage de programmation ou le type de compression avec des logiciels comme StudPe ou Peid par exemple). On voit une fenêtre dans laquelle, il faut rentrer le bon serial. On tape un joli serial bidon, du style 654321 pour varier un peu :-). On valide et là : "Wrong Serial - You are a bad cracker". Pffffff. "You are a bad cracker" : c'est vite dit ça. Attends un peu que je te désassemble, tu feras moins le malin après. Non mais ! Donc une fois cette brève partie "sketch" de ce cours passée on lance notre WinDasm patché et on désassemble ce crack-me. Et qu'est-ce qu'on voit aux toutes premières lignes :

```
Object01: UPX0 RVA: 00001000 Offset: 00000400 Size: 00000000 Flags: E0000080  
Object02: UPX1 RVA: 00036000 Offset: 00000400 Size: 0001FA00 Flags: E0000040  
Object03: .rsrc RVA: 00056000 Offset: 0001FE00 Size: 00001200 Flags: C0000040
```

On peut voir qu'il a été compressé avec UPX. Et comme il est compressé, on ne voit pas de Data Strings References. C'est là qu'intervient à la rescousse notre décompresseur UPX (si vous n'avez pas de décompresseur il est toujours possible de le faire manuellement mais c'est assez difficile). Pour le décompresser il faut copier le crack-me dans le même dossier que le décompresseur et renommez le fichier à décompressé en "1.exe" (ou alors modifier le 'decompress.bat'). Une fois cette modification effectuée, on lance le 'decompress.bat'. Et voilà le prog est décompressé.

On désassemble alors le crack-me décompressé. Et là on voit enfin les Data Strings References. On cherche notre message "You are a bad cracker". Il n'y en a qu'un seul, donc pas de problème :

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00442B8B(C)

|
:00442BA7 6A00 push 00000000

* Possible StringData Ref from Code Obj -> "Wrong Serial"

|
:00442BA9 B9582C4400 mov ecx, 00442C58

* Possible StringData Ref from Code Obj -> "You are a bad cracker!"

On va donc à l'adresse 00442B8B qui est juste au dessus :

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00442B5D(C)

|
:00442B79 8D55FC lea edx, dword ptr [ebp-04]
:00442B7C 8B83C4020000 mov eax, dword ptr [ebx+000002C4]
:00442B82 E835FCFDFF call 004227BC
:00442B87 837DFC00 cmp dword ptr [ebp-04], 00000000
:00442B8B 751A jne 00442BA7
:00442B8D 6A00 push 00000000

* Possible StringData Ref from Code Obj -> "Nothing entered"

|
:00442B8F B92C2C4400 mov ecx, 00442C2C

* Possible StringData Ref from Code Obj -> "You have to enter a serial"

Mais on voit que si on change ce saut, il va nous dire "Nothing entered" (= "rien d'entré" pour ceux qui ne sont pas bilingues). Ce message est appelé par un autre saut conditionnel, donc on y va :

* Possible StringData Ref from Code Obj -> "12011982"

|
:00442B53 BAE82B4400 mov edx, 00442BE8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00442AE7(C)

|
:00442B58 E8E70FFCFF call 00403B44
:00442B5D 751A jne 00442B79
:00442B5F 6A00 push 00000000

* Possible StringData Ref from Code Obj -> "Trial CrackMe Cracked!"

|
:00442B61 B9F42B4400 mov ecx, 00442BF4

* Possible StringData Ref from Code Obj -> "Congrats! You were successful!"

Ah et bien ce message me plait bien ! On va donc forcer le programme à arriver sur ce message. On changera donc le 751A en 741A ou 9090. Donc ouvrez le crack-me avec l'éditeur hexadécimal (prenez bien celui qui est décompressé). Pour trouver ce 751A, je vais répéter comment faire car beaucoup de personnes me le demandent par mail. On double-clique dans WinDasm sur la ligne du JNE. On regarde en bas dans la barre d'état et on voit :

Code Data @:00442B5D @Offset 00041F5Dh in Fileetc....

C'est l'offset dont on a besoin. C'est l'adresse de la ligne du JNE. L'offset ici est 41F5D. Le "h" qui suit indique que l'offset est en hexadécimal et non en décimal. Et nous vous occupez pas des "0" qui sont devant. C'est comme pour 8 ou 08 : c'est la même chose. Une fois qu'on a l'offset, on retourne dans l'éditeur hexa et on fait : "Edit" --> "Go To..." ou CTRL+T. Et on tape dans la case "Hex" (car l'offset est en hexadécimal) : 41F5D on valide. Et là comme par magie on tombe pile sur le 751A. On le change en 741A ou 9090 et on enregistre.

Un petit point sur l'enregistrement car je reçois également beaucoup de mails à ce sujet et j'en ai marre de répéter à chaque fois la même chose. Quand vous enregistrez votre programme modifié dans l'éditeur hexadécimal, si vous voulez enregistrer par dessus l'ancien il faudra fermer le crack-me s'il est déjà lancé et fermer WinDasm s'il est désassemblé avec. Sinon il vous dira "Writing failed". Mais le mieux est de l'enregistrer sous un autre nom. Et n'oubliez pas de mettre le ".exe" à la suite du nom quand vous enregistrez. Voilà pour ces 2 rappels.

Bon le crack-me est désormais cracké. Mais pour trouver le bon serial retournons un peu en arrière :

** Possible StringData Ref from Code Obj ->"12011982" <-- Tiens c'est bizarre ça !!!!*

|
:00442B53 BAE82B4400 mov edx, 00442BE8

** Referenced by a (U)nconditional or (C)onditional Jump at Address:*

|:00442AE7(C)
|
:00442B58 E8E70FFCFF call 00403B44
:00442B5D 751A jne 00442B79
:00442B5F 6A00 push 00000000

** Possible StringData Ref from Code Obj ->"Trial CrackMe Cracked!"*

|
:00442B61 B9F42B4400 mov ecx, 00442BF4

** Possible StringData Ref from Code Obj ->"Congrats! You were successful!"*

Avez-vous remarqué ce nombre : 12011982. Et bien c'est notre serial. Pas trop dur à trouver cette fois-ci, hein ?

Donc voilà comment cracker un programme qui semblait incrackable car il n'y avait pas de Data Strings References.

Deamon le 31 décembre 2002

Cours n°10

Cracker un crackme et trouver le serial

Objectif :

Faire un peu tout et n'importe quoi du moment qu'on arrive à obtenir le bon message et le bon serial. ;-)

Niveau :

Je n'en sais rien ! Ça dépend de vous. ;-) Mais plus compliqué que les précédents.

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) (8 Ko)

Aller on y va !

Je sais pas trop par quoi commencer mais enfin bon, on y va quand même ! Alors déjà après l'avoir téléchargé, curieux comme vous êtes, vous avez dû lancer ce petit crackme. Et vous avez dû voir qu'il n'y a pas de case pour le nom ! Donc soit le serial est unique, soit il est calculé avec par exemple le numéro du disque dur ou soit avec l'âge de votre belle-mère. A première vue ça m'a l'air d'être un serial unique.

Ni une, ni deux, on ouvre WinDasm et on désassemble ce crackme. On recherche dans les Data Strings Reference le vilain message d'erreur : "Bzzz, Essaye encore!!". Et on arrive sur ce message :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:  
|:00401429(C), :00401436(C), :0040143F(C), :00401448(C) --> pour ceux qui n'auraient par encore compris ou  
fais attention le (C) veut dire saut Conditionnel et le (U) saut inconditionnel  
|  
:00401465 FC    cld  
  
* Possible StringData Ref from Data Obj -> "Bzzz, Essaye encore!!"  
|  
:00401466 6838314000    push 00403138  
:0040146B FF3599314000    push dword ptr [00403199]
```

Ce message est donc appelé par 4 sauts conditionnels. Ce qui est bien c'est que tous rassemblés un peu plus haut ce qui évite contrairement à certains programmes de retrouver les sauts éparpillés au quatre coins du code.

```
:00401418 E88B000000    Call 004014A8  
:0040141D A37D314000    mov dword ptr [0040317D], eax  
:00401422 833D7D31400003    cmp dword ptr [0040317D], 00000003  
:00401429 753A    jne 00401465  
:0040142B BBF0314000    mov ebx, 004031F0  
:00401430 0FBE03    movsx eax, byte ptr [ebx]  
:00401433 83F84D    cmp eax, 0000004D  
:00401436 752D    jne 00401465  
:00401438 43    inc ebx  
:00401439 0FBE03    movsx eax, byte ptr [ebx]  
:0040143C 83F84F    cmp eax, 0000004F
```

```

:0040143F 7524   jne 00401465
:00401441 43   inc ebx
:00401442 0FB03   movsx eax, byte ptr [ebx]
:00401445 83F849   cmp eax, 00000049
:00401448 751B   jne 00401465
:0040144A EB00   jmp 0040144C

```

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040144A(U)
|
:0040144C FC cld

* Possible StringData Ref from Data Obj -> "Arf C'est le bon Code!!"

Donc ces 4 sauts nous amenant sur le message d'erreur il faut les remplacer. Des NOP partout doivent très bien marcher (bonjour l'élégance du crack me direz-vous) ! Mais au moins ça a le mérite de marcher ! ;-)

Analyse du code à l'aide du débogueur

Afin de trouver le bon code on va donc utiliser le débogueur de WinDasm. Donc CTRL+L ou Debug --> Load Process et là, tel Sésame, le débogueur s'ouvre. Il faudrait poser un BP (Break Point) avant ces 4 sauts. Donc on met un BP avant l'arrivée du premier saut qui nous amène sur le message d'erreur. Comme le CALL n'est pas intéressant on va poser le BP sur la ligne d'après.

```

:00401418 E88B000000   Call 004014A8
:0040141D A37D314000   mov dword ptr [0040317D], eax --> Break Point sur cette ligne
:00401422 833D7D31400003   cmp dword ptr [0040317D], 00000003
:00401429 753A   jne 00401465

```

Inutile de cocher les 5 cases de la fenêtre de droite, car là on en aura pas besoin. On fait Run (F9) et on tape un code bidon : 123456. On valide et là il break. Là il va falloir s'occuper de la fenêtre de gauche. Mais tout d'abord on analyse un peu le code.

Je ferais, dans un prochain cours, un cours sur l'assembleur pour que vous ayez quelques bases car cela vous servira plus d'une fois.

```

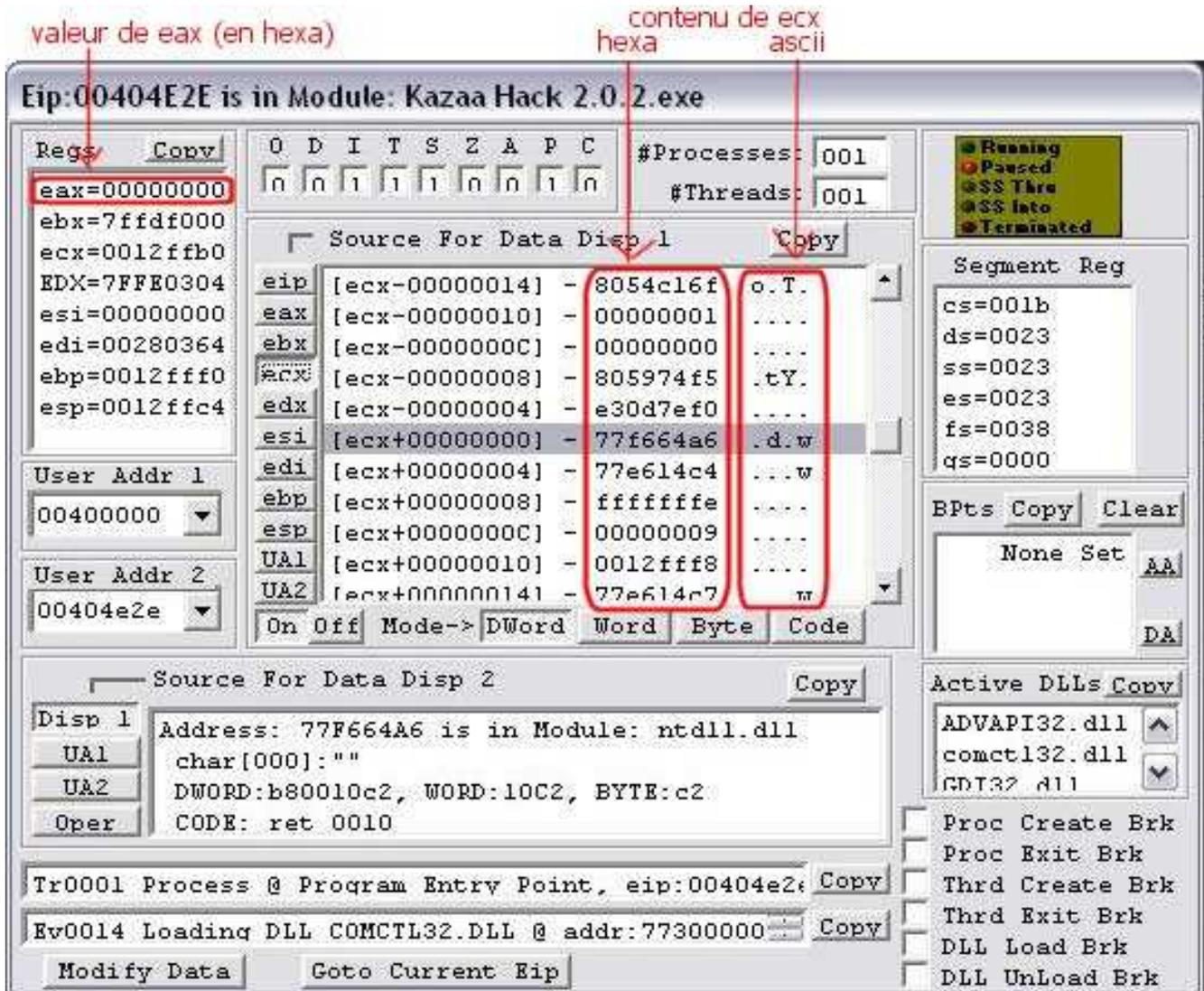
:0040141D A37D314000   mov dword ptr [0040317D], eax --> Vous êtes ici ! :-) La valeur de eax est copié
dans dword ptr [0040317D] et la température extérieur est de 23.6°C ! (oups pardon !)
:00401422 833D7D31400003   cmp dword ptr [0040317D], 00000003 --> la valeur de dword ptr [0040317D]
(donc eax) est comparé à 3 (3 en hexa = 3 en decimal)

```

On va donc regarder ce qu'il y a dans EAX pour voir ce qu'il compare à 3, car si EAX n'est pas égal à 3 (JNE) il saute vers le mauvais message. Pour cela il faut regarder dans le cadre "regs" en haut à gauche de la fenêtre de gauche. On voit EAX = 00000006. Notre code était 123456 donc la valeur de EAX doit être le nombre de lettres. On peut essayer avec d'autres serials bidons pour vérifier. Donc le vrai serial fait 3 caractères. Il faut donc recommencer le débuggage pour rentrer cette fois un serial de 3 caractères. Le BP toujours sur la ligne du "mov", on rentre cette fois 123, on valide. Et là il break. Là on va y aller en Pas à Pas (Step) dans les boutons à coté de Run. Le Step Into (F7) est le pas à pas en rentrant dans les CALL et Step Over (F8) sans rentrer dedans en passant par dessus (enfin le programme rentre bien sur dedans mais vous ne le voyez pas). Là prenez celui que vous voulez car on n'a pas de CALL à traverser.

Les modifications des registres s'effectuent une fois la ligne passée. Si par exemple vous êtes sur une ligne avec 'mov eax, 6', eax sera égale à 6 seulement quand vous serez passé à la ligne suivante.

Donc on avance pas à pas jusqu'à la ligne du premier JNE. Ayant rentré un serial de 3 chiffres, il ne devrait pas sauter. On appuie sur F7 ou F8 pour passer cette ligne : Ouf ! il ne saute pas ! Là on est sur une ligne où il y a : mov ebx, 004031F0 . Il va donc copier une valeur dans ebx. On passe cette ligne pour regarder ce que contient ebx. Voici un exemple avec la valeur de eax et le contenu de ecx.



Donc il faut cliquer sur "ebx" pour voir ce qu'il y a mis. Et là on voit le code '123' que nous avons tapé. Le serial bidon est donc stocké dans ebx.

Ensuite il y a cette ligne : movsx eax, byte ptr [ebx] qui va donc donner la valeur d'un byte (en anglais bit=bit et byte=octet ! Ne pas confondre ! un octet correspond là à un caractère) de ebx (de notre serial) à eax. En effet on voit ensuite que eax = 31. Et 31 (en hexa bien sur) représente le caractère '1' (en ascii). L'ascii c'est le format qui regroupe les caractères que nous voyons à l'écran. Pour connaître les correspondances, allez dans HexDecCharEditor et cliquer sur "characters table" qui est représenté par un rectangle jaune. Regardez à la valeur 31 hexa et vous verrez qu'il correspond à 1 en Window Char (Ascii).

Après avoir mis le premier caractère dans eax, il le compare à 4D (hexa) : cmp eax, 0000004D. On cherche donc dans la liste des valeurs hexa 4D et on voit qu'il correspond à M.

Ensuite il incrémente (augmenter de 1) ebx (inc ebx) pour passer à la lettre suivante et compare les deux lettres suivantes par 4F et 49 (O et I). Après chaque test il y a un JNE qui saute si la lettre n'est pas la même.

Donc petit récapitulatif :

```
:0040141D A37D314000 mov dword ptr [0040317D], eax --> nombre de caractères du serial bidon
:00401422 833D7D31400003 cmp dword ptr [0040317D], 00000003 --> compare ce nombre à 3
:00401429 753A jne 00401465 --> si pas égal saute vers mauvais message
:0040142B BBF0314000 mov ebx, 004031F0 --> mets notre serial bidon dans ebx
:00401430 0FBF03 movsx eax, byte ptr [ebx] --> la valeur de eax est le premier caractère du serial bidon en hexa
:00401433 83F84D cmp eax, 0000004D --> compare la 1ere lettre à 4D (M)
:00401436 752D jne 00401465 --> saute si pas égal
:00401438 43 inc ebx --> ebx + 1 : caractère suivant
:00401439 0FBF03 movsx eax, byte ptr [ebx] --> la valeur de eax est le deuxième caractère du serial bidon en hexa
:0040143C 83F84F cmp eax, 0000004F --> compare le 2eme caractère à 4F (O)
:0040143F 7524 jne 00401465 --> saute si pas égal
:00401441 43 inc ebx --> ebx + 1 : caractère suivant
:00401442 0FBF03 movsx eax, byte ptr [ebx] --> la valeur de eax est le troisième caractère du serial bidon en hexa
:00401445 83F849 cmp eax, 00000049 --> compare le 3ème caractère à 49 (I)
:00401448 751B jne 00401465 --> saute si pas égal
:0040144A EB00 jmp 0040144C --> saute vers bon message
```

Le bon serial est donc **MOI**.

Pour obtenir le bon message avec un serial faux vous pouvez aussi inverser les PUSH :

```
* Possible StringData Ref from Data Obj ->"Arf C'est le bon Code!!"
|
:0040144D 6820314000 push 00403120
:00401452 FF3599314000 push dword ptr [00403199]

* Possible StringData Ref from Data Obj ->"Bzzz, Essaye encore!!"
|
:00401466 6838314000 push 00403138 --> A remplacer par push 00403120
:0040146B FF3599314000 push dword ptr [00403199]
```

Le push après le mauvais message devra être remplacé par celui du bon message. Donc il faut mettre **push 00403120** (6820314000) à la place de **push 00403138** (6838314000). Pour info : PUSH = 68, et ensuite il faut lire à l'envers 2 par 2 : **6820314000** = **push 00403120**

Et voilà encore un crack-me de cracké et avec la manière ! ;-)

Deamon le 13 avril 2003

Cours n°11

Le Crackme qui se cracke tout seul

Objectif :

- 1° Faire en sorte que le crackme se patche lui-même.
- 2° Cracker le crackme pour qu'il nous donne le bon serial.

Niveau :

Avancé. (Non mais bougez pas rester sur votre siège !)

Ce qu'il vous faut :

- Connaître les bases de l'Assembleur (lire mon cours à ce sujet)
- Savoir utiliser WinDasm (si vous avez compris mes cours précédents c'est bon ! ;-)
- WinDasm
- Un éditeur hexadécimal
- Le [crack-me](#) du cours n°3 et 4 (20 Ko)
- [StudPE](#) (409 Ko)

Introduction

Je n'ai pas de définition exacte du Reverse Engineering, donc je ne sais pas si on peut considérer ce cours comme étant un cours de Reverse Engineering. Mais cela n'est pas bien grave. Pour ceux qui ne savent pas ce que c'est, c'est l'art de modifier ou d'ajouter des fonctions d'un programme en rajoutant, modifiant du code de ce dernier. Enfin du moins c'est comme ça que je le définirais.

Dans ce cours tout ce que je vais vous montrer ne vous servira pas forcément plus tard. Je fais cela juste pour m'amuser, pour utiliser mes connaissances en assembleur et pour vous en faire profiter. Donc je vous montrerais des méthodes qui sont beaucoup plus longues que la simple inversion de saut, mais dont le résultat revient au même.

Dans ce cours vous apprendrez également à mieux utiliser le débogueur de WinDasm.

Le Crack-Me qui se patche lui-même !

J'ai pris le crackme du cours n°3 et 4. Donc nous l'avons déjà cracké ensemble de manière brève mais efficace. Le crack-me comporte juste une boîte de dialogue qui demande un serial. Et si c'est pas le bon, comme d'habitude, nous avons le droit à un beau message "Incorrect try again!!".

Maintenant que vous avez téléchargé StudPe, cela va (ou plutôt doit) devenir un réflexe désormais : regarder si le programme est compressé ou voir en quel langage il a été programmé. Donc lancez StudPe, allez directement à Options, ne passez pas par

la case départ et ne recevez pas 20.000 F. Oups pardon je m'égare quelque peu ! Une fois sur l'onglet "Options", cochez la case "Shell Menu Extension". Cela vous permettra, en faisant un clic droit sur un .exe ou .dll de l'analyser avec StudPe grâce à l'ajout d'un raccourci dans le menu "clic-droit". N'oubliez pas de configurer l'éditeur hexa avec la case "HexEditor" en sélectionnant le .exe de votre éditeur préféré.

Une fois tout cela fait, ouvrez notre crackme avec et allez à l'onglet "Signature". Là on peut voir qu'il a été programmé avec Visual C++ 6.0. Donc il n'est pas compressé. Bon allez, vous vous amuserez à voir les autres onglets quand j'aurai fini mon cours.

Bon cela ne nous empêche pas, bien sûr, de désassembler notre cher crackme. On recherche le message d'erreur, on tombe sur :

```
:0040158D FF1500204000 Call dword ptr [00402000]
:00401593 85C0 test eax, eax
:00401595 7516 jne 004015AD
:00401597 6A40 push 00000040

* Possible StringData Ref from Data Obj ->"CrackMe"
|
:00401599 6850304000 push 00403050

* Possible StringData Ref from Data Obj ->"Correct way to go!!"
|
:0040159E 6858304000 push 00403058
:004015A3 8B4DE0 mov ecx, dword ptr [ebp-20]

* Reference To: MFC42.Ordinal: 1080, Ord: 1080h
|
:004015A6 E853050000 Call 00401AFE
:004015AB EB14 jmp 004015C1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401595(C)
|
:004015AD 6A40 push 00000040

* Possible StringData Ref from Data Obj ->"CrackMe"
|
:004015AF 686C304000 push 0040306C

* Possible StringData Ref from Data Obj ->"Incorrect try again!!"
```

Oula que c'est dur à cracker ça ! ;-)) Bon il faut donc modifier le JNE en 00401595 et le remplacer par un JE ou deux NOP. Il va falloir que le crackme modifie lui-même ce saut comme un grand :-). Pour modifier par exemple le JNE en JE on mettra donc :

MOV byte PTR [00401595], 74 : il s'agit donc de mettre 74 (JE) à l'adresse 401595.

Il va falloir insérer cette instruction à un endroit où il y a de la place (là où c'est plein de 00). La plupart du temps vers la fin du programme on trouve de la place. Ouvrez le avec l'éditeur hexadécimal et allez donc vers la fin du programme. On le mettra par exemple à l'offset 4890. Et pour que le programme passe par ici il va falloir l'ammener par un JMP. Donc il faut trouver une instruction qu'on peut remplacer par le JMP. Et il ne faut pas oublier non plus le JMP après le MOV pour pouvoir revenir de là où on vient. J'espère avoir été assez clair.

Petit rappel : pour un saut le premier byte (octet) correspond au type de saut et le

deuxième au nombre (en hexa bien sur) de bytes qu'il va sauter. Ex: 7516 : 75 = JNE et saute de 16 bytes.

Le problème ici c'est que le saut est largement supérieur à FFh (255d). Il prendra environ 5 bytes (1 pour le 74 et 4 pour la taille du saut) ce qui est donc impossible à faire comme cela. Si le saut avait été inférieur à FF on aurait très bien pu faire comme ceci : exemple :

- **JMP 00401234** --> le programme original saute à 00401234
- **xxxxxxx** --> instructions
- [...]
- **xxxxxxx**

Remplacer par :

- **JMP 00404890** --> saut à l'endroit que nous avons créé
- **xxxxxxx** --> instructions
- [...]
- **xxxxxxx**
- **MOV byte PTR [00401595], 74** --> modifie le JNE en JE (75 par 74)
- **JMP 00401234** --> saute à l'endroit prévu par le programme original

Mais là il va falloir remplacer une instruction qui ne sert pas (ou plutôt qui sert mais à rien ;-)) d'une taille minimum de 5 bytes et dont le programme passe forcément dessus.

Donc on regarde au-dessus du test qui amène de notre message d'erreur car il y passe forcément :

** Reference To: KERNEL32.IstrlenA, Ord: 0308h*

```
|  
:00401560 FF1504204000 Call dword ptr [00402004]  
:00401566 8945F0 mov dword ptr [ebp-10], eax --> met le nombre de lettres du serial dans ebp-10  
:00401569 837DF001 cmp dword ptr [ebp-10], 00000001 --> compare à 1  
:0040156D 7316 jnb 00401585 --> si c'est supérieur saute, si pas de lettre ne saute pas  
:0040156F 6A40 push 00000040
```

** Possible StringData Ref from Data Obj -> "CrackMe"*

```
|  
:00401571 682C304000 push 0040302C
```

** Possible StringData Ref from Data Obj ->"Enter Registration Number"*

```
|  
:00401576 6834304000 push 00403034  
:0040157B 8B4DE0 mov ecx, dword ptr [ebp-20]
```

** Reference To: MFC42.Ordinal: 1080, Ord: 1080h*

```
|  
:0040157E E87B050000 Call 00401AFE  
:00401583 EB3C jmp 004015C1
```

** Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040156D(C)*

```
|  
:00401585 8D4DE4 lea ecx, dword ptr [ebp-1C]  
:00401588 51 push ecx  
:00401589 8D55F4 lea edx, dword ptr [ebp-0C]  
:0040158C 52 push edx
```

** Reference To: KERNEL32.lstrcmpA, Ord: 02FCh*

```
|  
:0040158D FF1500204000 Call dword ptr [00402000]  
:00401593 85C0 test eax, eax  
:00401595 7516 jne 004015AD --> saute vers bon ou mauvais message  
:00401597 6A40 push 00000040
```

** Possible StringData Ref from Data Obj ->"CrackMe"*

```
|  
:00401599 6850304000 push 00403050
```

** Possible StringData Ref from Data Obj ->"Correct way to go!!"*

Là où il teste s'il y a des caractères ou non, il y aurait de la place car ce test ne sert à rien pour nous, car il nous force à taper quelque chose. Et faignant comme on est (je sais pas vous mais moi si :-)) et bien pouvoir ne rien taper c'est mieux.

Vous vous demander peut-être comment je sais que c'est le nombre de caractères du serial. Et bien lancez le débogueur, mettez un BreakPoint sur la ligne en 00401566, faites "Run", tapez n'importe quoi comme serial et faites Ok. Là il break ! Regardez dans la fenêtre des registres (celle en bas à gauche) et plus particulièrement le cadre "Regs". Vous pouvez voir la valeur de EAX. En mettant "123456" comme serial vous aurez 6.

Bon et bien pendant qu'on break ici et bien on va installer notre patch ! On clique sur "Patch Mode" de la fenêtre de droite. Et là on va pouvoir taper directement en assembleur nos instructions. Donc on tape : **JMP 00404890**. C'est là qu'on va placer le patch en lui-même. On fait Entrée, si l'instruction est fautive il l'indique et si c'est juste il la met dans le cadre du dessous. L'avantage de le faire ici c'est de pouvoir voir la correspondance en hexa, qu'on pourra éventuellement recopier dans l'éditeur hexadécimal.

Une fois l'instruction validée, on fait "Apply Patch", là une fenêtre de confirmation s'affiche. Si vous ne la voyez pas c'est qu'elle est cachée derrière la fenêtre actuelle. On fait "Oui" et avant de fermer la fenêtre du Patch Mode on clique sur "Copy" et on colle ça quelque part pour s'en souvenir. Les modifications effectuées ne se voient pas dans la fenêtre principale de WinDasm mais dans celle de droite.

Là on va devoir aller à 00404890 donc on fait "F7" - Step Into ou "F8" - Step Over (peu importe). Les deux permettent d'avancer en pas à pas. Mais sachez que le Step Into rentre dans les CALL contrairement au Step Over.

Une fois le F7 (ou F8) fait, on voit : `:00404890 add byte ptr [eax], al` (toujours dans la fenêtre de droite). C'est là où le JMP nous a emmené. On va insérer notre patch ici même, donc on clique sur "Patch Mode" et on met :

- **MOV byte PTR [00401595], 74**
- **JMP 00401585**

Pourquoi sauter à 00401585 ? Rappelez-vous ! On avait : `:0040156D 7316 jnb 00401585 --> si c'est supérieur saute, si pas de lettre ne saute pas.`

Donc il saute à cette adresse s'il y a des caractères. Donc là on pourra mettre rien du tout et il nous dira quand même que c'est bon. C'est-y pas magique ? Ben en fait non ! Car appliquez ce patch là (n'oubliez pas de copier les instructions) ! Pour le tester ensuite, cliquez sur Run. Et là plantage ! AAARRRRGGG ! Cela vient du fait que la section dans laquelle doit être effectuée le patch ne peut pas être modifiée (comme si elle était en lecture seule).

Faites quand même les modifications que l'on vient d'effectuer dans l'éditeur hexadécimal. Si vous avez pas copier les instructions en hexa vous n'avez plus qu'à recommencer ! Et respectez bien les endroits !!!

Ouvrez ensuite le crackme modifié avec StudPe. Allez dans "section", double-cliquez sur la ligne de ".text" et cochez "MEM_WRITE". Faites Save et voilà, vous pouvez maintenant le tester : notre crackme est cracké en s'auto-patchant.

Pour remplacer le JNE par des NOP il aurait fallu inscrire :

```
:00401DE0 66C6059515400090 mov byte ptr [00401595], 90  
:00401DE8 66C6059615400090 mov byte ptr [00401596], 90
```

Petite variante

On aurait pu mettre un CALL à la place d'un jump. Exemple :

Voici ce qu'il y a dans le programme original :

```
:00401566 8945F0 mov dword ptr [ebp-10], eax --> met le nombre de lettres du serial dans ebp-10  
:00401569 837DF001 cmp dword ptr [ebp-10], 00000001 --> compare à 1  
:0040156D 7316 jnb 00401585 --> si c'est supérieur saute, si pas de lettre ne saute pas
```

Si on break sur la ligne en 00401566 on peut voir que EBX est tout le temps égal à 1. Donc on pourrait mettre :

```
:00401566 39D8 cmp eax, ebx --> Compare notre nombre de lettres à EBX (=1), ce qui revient au même que les 2 lignes MOV et CMP.  
:00401568 E823330000 call 00404890 --> Va à l'endroit de notre patch
```

Puis à l'endroit du patch, en 00404890 :

```
:00404890 66C6059515400074 mov byte ptr [00401595], 74 --> Remplace le saut 75 en 74  
:00404898 C3 ret --> Retourne juste après le CALL
```

Par contre si vous ne mettez rien comme serial il vous demandera d'en mettre un.

Le crackme qui nous donne le bon serial !

Voilà comment le crackme vous donne le bon serial !

```
* Possible StringData Ref from Data Obj ->"CrackMe"  
/  
:004015AF 686C304000 push 0040306C  
  
* Possible StringData Ref from Data Obj ->"Incorrect try again!!"  
/  
:004015B4 6874304000 push 00403074  
:004015B9 8B4DE0 mov ecx, dword ptr [ebp-20]
```

Le PUSH envoie en fait le texte situé au dessus que WinDasm nous montre gentiment :-). Pour le vérifier lancez le debuggeur, et tapez le numéro qui suit le PUSH dans la case "User Addr" 1 ou 2 (en dessous des registres) et cliquez en dessous sur UA1 ou UA2 selon le numéro choisi. En mettant 00403074 vous pourrez voir "Incorrect try again". Les autres messages se situent vers le même endroit (dans les 004030xx), on va donc aller dans l'éditeur hexa à l'offset 3074 par exemple. Et là on voit les autres messages dont le serial !! On repère donc son adresse en relevant l'offset du premier caractère ('<') du serial. Ici on a 3020. Donc on va remplacer le PUSH que vous souhaitez par un PUSH 00403020.

Par exemple celui du "Incorrect..." : 6874304000 push 00403074
On mettra à la place : 6820304000 push 00403020

Je rappelle la correspondance du PUSH en hexa : PUSH = 68 et ensuite les octets sont inversés, regardez l'exemple en couleur :

```
6874304000 push 00403074
```

Et donc une fois les modifications effectuées, on teste : on met n'importe quoi, on valide ! Et là un beau message vient nous rappeler le bon serial : <BrD-SoB>. Et n'oubliez pas de mettre les < et > pour le bon serial.

Voilà c'est fini pour aujourd'hui, j'espère vous avoir appris quelque chose de plus avec ce cours ! Et comme toujours si vous avez un problème ou si vous n'avez pas compris quelque chose 2 adresses sont là pour ça : www.forumcrack.fr.st et daemon.crack@netcourrier.com.

Daemon le 24 et 25 juillet 2003

Cours n°12

Reverse Engineering sur la calculatrice Windows

Objectif :

- 1° Rajouter un menu.
- 2° Afficher une boîte de dialogue quand on clique sur notre nouveau menu.

Niveau :

Elevé

Ce qu'il vous faut :

- Connaître les bases de l'Assembleur (lire mon cours à ce sujet)
- WinDasm
- Un éditeur hexadécimal
- Un éditeur de ressources ([Restorator](#) ou [ResHacker](#))
- La calculatrice de Windows XP (dans Windows/System32/Calc.exe)
- Et un brin de logique !

Introduction

Cette fois-ci ce n'est pas vraiment du cracking mais du Reverse Engineering ! Domaine dans lequel je suis encore débutant mais qui est tellement passionnant et intéressant ! Car ce qu'on va faire ne sert pas à grand chose sinon de se faire plaisir ! :-P Et utilisez nos connaissances.

Pourquoi la calculatrice de Windows particulièrement ? Je ne sais pas trop, j'ai pris le premier petit programme avec un menu que j'ai vu ! Et de plus tout le monde doit l'avoir. Par contre je pense que la calculatrice des Windows précédent au XP ne doivent pas être identiques, donc pour faire le cours il faudra vous adapter.

A l'attaque !

Bon alors tout d'abord on va prendre la calculatrice de Windows (dans Windows/System32/Calc.exe) et la copier dans un autre dossier, pour ne pas abîmer l'original car on ne sait jamais !

On va tout d'abord ajouter un menu à coté du menu "?". Pour cela on va utiliser un éditeur de ressources. Ces logiciels sont très utilisés pour la traduction de programmes notamment. Je vais détailler la méthode à utiliser avec 2 éditeurs : Restorator et ResHacker (appelé aussi RessourceHacker ou ResHack) comme ça tout le monde sera content. Personnellement j'utilise Restorator mais le plus utilisé est ResHacker.

Avec Restorator :

Vous lancez donc Restorator, vous faites File --> Open et vous sélectionnez la calculatrice que vous avez copiée dans un dossier. Vous pouvez voir différents dossiers sous "Calc.exe". Voici un rapide descriptif de ces dossiers :

- Menu : comme son nom l'indique ce sont les menus (Fichier, Édition, ...etc)
- Dialog : les boîtes de dialogues du logiciel
- String : les phrases ou mots du logiciel
- Accelerators : principalement les raccourcis clavier
- Icon : les icônes du programme
- Version : les informations sur le programme (numéro de version, auteur...)

Donc ce qui nous intéresse ici c'est 'Menu'. On regarde donc dans 'Menu' puis '106'. Là on voit les menus de la calculatrice (Edition, Affichage et ?). Pour mieux visualiser tout ça cliquer sur le bouton "ab]" (ou View et Edit Mode). Là une fenêtre s'ouvre avec la représentation du menu.

On va donc ajouter un menu que je nommerai "Infos" et un sous-menu "Reverse" (mais prenez les noms que vous voulez ça n'a pas d'importance). Comme vous pouvez le voir un menu s'écrit comme ceci :

```
POPUP "&Edition"  
{  
MENUITEM "Co&mpier\tCtrl+C", 300  
MENUITEM "C&oller\tCtrl+V", 301  
}
```

Le caractère & signifie que la fonction peut être appelée avec le raccourci clavier : ALT+[lettre qui suit le &]. Ici ALT+E pour Edition.

Le \tCtrl+C est le raccourci qui exécute la fonction.

Le 300 ou 301 ici sont les ID des menus. C'est à dire leur numéro pour que le programme puisse les identifier.

On va donc mettre après le menu "&?" :

```
POPUP "&Infos"  
{  
MENUITEM "&Reverse", 318 --> Je mets 318 car il ne semble pas être utilisé par une autre  
fonction, mais vous pouvez mettre ce que vous voulez du moment qu'il n'est pas déjà pris.  
}
```

Une fois cela fait on appuie sur F7 afin de mettre à jour la fenêtre de visualisation. Et on voit bien notre nouveau menu. Ensuite on appuie sur 'Commit changes' (ou F8) pour appliquer les changements. Mais comme vous avez pu le voir il y a plusieurs menus (106 ; 107 ; 108 et 109). Car suivant le mode de la calculatrice (standard ou scientifique) vous n'aurez pas les mêmes menus. Il faut donc rajouter notre menu "Info" - "Reverse" dans les 3 premiers (le 4ème est pour l'aide). L'ID de notre menu doit être le même pour les 3.

Pensez à faire F8 après chaque modification.

Voici ce que vous devez obtenir pour le menu 106 :

```
106 MENU
{
POPUP "&Edition"
{
MENUITEM "Co&pier\tCtrl+C", 300
MENUITEM "C&oller\tCtrl+V", 301
}
POPUP "&Affichage"
{
MENUITEM "S&tandard", 305
MENUITEM "S&cientifique", 304
MENUITEM SEPARATOR
MENUITEM "Groupement des ch&iffres", 303
}
POPUP "&?"
{
MENUITEM "&Rubriques d'aide ", 317
MENUITEM SEPARATOR
MENUITEM "À &propos de la Calculatrice", 302
}
POPUP "&Infos"
{
MENUITEM "&Reverse", 318
}
}
```

Une fois tout cela fait, vous pouvez sauvegarder votre calculette avec le nouveau menu, avec File --> Save As.

Avec ResHacker :

C'est quasi-identique ! Vous ouvrez la calc.exe, vous allez dans 'Menu', puis '106' et enfin '1036'. Là il apparaît directement la fenêtre de visualisation du menu. Vous rajoutez :

```
POPUP "&Infos"
{
MENUITEM "&Reverse", 318
}
```

Et enfin vous cliquez sur 'Enregistrer.' Vous faites la même chose pour le menu '107' et '108'. Et pour finir vous enregistrez le .exe.

Note :

Suivant que vous avez modifié avec Restorator ou ResHacker les bytes du programmes ne seront pas identiques, mais je ne pense pas que cela change quelque chose dans la correspondance avec la suite de mon cours. Si toutefois il y a des

différences pour les adresses et offsets par la suite, sachez que je vais utiliser la version modifiée avec Restorator.

Ajout de la MessageBox :

Comme vous avez pu le constater, notre menu est beau mais pour l'instant il ne sert à rien car lorsqu'on clique dessus rien ne se passe ! Il va donc falloir ajouter l'affichage d'une boîte de dialogue. Mais il va falloir que le programme passe par le bout de code qu'on aura rajouté lorsqu'on clique sur le menu. Pour cela on va chercher avec WinDasm l'endroit où le programme analyse quel menu a été cliqué. On va donc désassembler notre calculatrice modifiée. Et tout au début on peut voir :

MenuID_006A

Edition {Popup}

Copier Ctrl+C [ID=012Ch]

Coller Ctrl+V [ID=012Dh]

Affichage {Popup}

Standard [ID=0131h]

Scientifique [ID=0130h]

Groupement des chiffres [ID=012Fh]

? {Popup}

Rubriques d'aide [ID=013Dh]

À propos de la Calculatrice [ID=012Eh]

Infos {Popup}

Reverse [ID=013Eh] --> Notre menu avec son ID.

Comme vous l'avez vu l'ID du menu "Reverse" est 13E alors que l'on avait mis 318 ! Et bien comme le "h" l'indique c'est que l'ID est en hexadécimal ici alors que nous l'avions noté en décimal tout à l'heure.

Pour savoir quel menu a été cliqué le programme va donc comparer la valeur que lui a renvoyé le clic avec les ID des menus. On va donc faire une recherche dans WinDasm de 013D par exemple (l'ID de la rubrique d'aide). On tombe premièrement sur :

```
:010026DA 83FE51    cmp esi, 00000051
:010026DD 7411      je 010026F0
:010026DF 83FE52    cmp esi, 00000052
:010026E2 740C      je 010026F0
:010026E4 81FE3D010000    cmp esi, 0000013D
:010026EA 0F85DF000000    jne 010027CF
```

Mais les autres ESI sont comparés à 51 et 52. et en regardant les autres ID, aucun ne correspond à 51 et 52. On continue donc notre recherche. Et on arrive ici :

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

:01003A9F(C)

```
|
:01004166 81FE8D000000    cmp esi, 0000008D
:0100416C 0F8458010000    je 010042CA
:01004172 81FE2B010000    cmp esi, 0000012B
:01004178 0F8667010000    jbe 010042E5
:0100417E 81FE31010000    cmp esi, 00000131
:01004184 7620    jbe 010041A6
:01004186 81FE35010000    cmp esi, 00000135
:0100418C 0F8653010000    jbe 010042E5
:01004192 81FE3C010000    cmp esi, 0000013C
:01004198 7617    jbe 010041B1
:0100419A 81FE3D010000    cmp esi, 0000013D
:010041A0 0F853F010000    jne 010042E5
```

C'est ici que le programme regarde quel menu a été cliqué. Ce qu'il faudrait qu'on fasse c'est mettre une comparaison de esi avec 13E (notre ID) et faire un saut vers la messagebox si c'est égal. Mais où mettre la comparaison ? Il n'y a pas de place ici ! On voit que cette partie du code est appelée par un saut conditionnel. Et bien c'est parfait ça ! Voici le saut qui mène vers la comparaison des ID :

:01003A9F 0F87C1060000 ja 01004166

On va donc remplacer ce JA 01004166 par un JA qui ira vers un endroit où nous allons mettre :

```
cmp esi, 13E --> regarde si c'est bien notre menu qui a été cliqué
jne 01004166 --> si c'est pas le menu saute en 01004166 pour continuer la comparaison
[afficher la messagebox] --> nous verrons plus tard comment l'écrire
jmp 01004166 --> retourne en 01004166
```

Maintenant il va falloir trouver de la place pour mettre tout ça. On va donc aller voir tout en bas du code dans Windasm. Là où on voit :

```
:010136AE 00000000000000000000000000000000 BYTE 10 DUP(0)
:010136B8 00000000000000000000000000000000 BYTE 10 DUP(0)
:010136C2 00000000000000000000000000000000 BYTE 10 DUP(0)
```

Ce sont des bytes (octets) libres. On aura donc la place de les mettre ici. Regardons cela dans l'éditeur hexadécimal. Effectivement à l'offset 12AAE (adresse 010136AE Windasm) on voit des 00. Pas la peine de trop coller au code qui est au-dessus. On commencera par exemple à écrire en 12AC0 (mais on pourrait commencer n'importe où) donc à l'adresse 010136C0 de WinDasm. On retient cette adresse WinDasm car on va installer notre saut JA. On ouvre donc le débogueur. Et on pose un break point en 01003A9F (adresse du JA qu'on a décidé de changer). On fait Run et on clique sur la Rubrique d'aide et là bien sur ça break.

On clique sur Patch Code pour mettre notre nouveau saut : **ja 010136C0**. On fait entrée et là on clique sur Copy ! Car ce qui nous intéresse c'est le code en hexa. On fait "Apply Patch", on valide et on ferme le Patch Code.

Avant de continuer je vais vous montrer comment marche une MessageBox :

```
MessageBox(  
HWND hWnd, // Handle de la fenêtre (si 00, pas de fenêtre attribuée)  
LPCTSTR lpText, // adresse du texte de la MessageBox  
LPCTSTR lpCaption, // adresse du titre de la MessageBox  
UINT uType // style de la messagebox (bouton Ok, Annuler...etc) ;
```

Donc en assembleur il faut écrire ça à l'envers (car on utilise des PUSH donc à cause de la pile on inverse, cf. cours sur l'assembleur). Ce qui donnera :

```
Push code_du_style_souhaité  
Push adresse_du_titre  
Push adresse_du_texte  
Push 00  
Call MessageBox
```

Pour le code du style vous pouvez mettre 00 si vous n'avez rien de particulier à mettre (mais on verra ça après).

Il va falloir écrire le titre et le texte de la message box dans l'éditeur hexa à une adresse que nous aurons définie. On va donc mettre le titre en 12B00 et le texte en 12B20 (adresse windasm : 01013700 et 01013720).

Le 'Call MessageBox' il faut trouver dans le programme l'adresse de celle ci. Pour cela on va regarder dans les Imports de Windasm. On trouve [USER32.MessageBoxW](#) cela devrait faire l'affaire. On double clique dessus plusieurs fois pour voir les Call qui l'appelle. On voit qu'il s'agit de :
[FF15A8110001 Call dword ptr \[010011A8\]](#)

Donc maintenant nous avons tous les éléments nécessaires. Nous pouvons donc continuer.

Toujours dans le débogueur vous devez être sur le break du JA qui vient d'être modifié. On va donc faire F7 ou F8 (Avance pas à pas). On voit bien dans la fenêtre de droite : `:010136C0 add byte ptr [eax], al`

Ce qui signifie que nous sommes bien sur nos octets vides. On lance Patch Code et on va rentrer ces instructions (faites 'entrée' après chaque instruction tapée) :

```
cmp esi, 0000013E --> Compare pour savoir si c'est le menu 'Reverse' qui a été cliqué  
jne 01004166 --> Retourne à la routine si c'est pas le menu 'Reverse'  
push 00 --> Style de la MessageBox  
push 01013700 --> Adresse du titre de la MessageBox  
push 01013720 --> Adresse du texte de la MessageBox  
push 00 --> Handle de la fenêtre  
call [010011A8] --> Appel l'API MessageBoxW  
jmp 01004166 --> Retourne à la routine
```

N'oubliez surtout pas de faire "copier" et de coller ça dans n'importe quel fichier

pour garder les valeurs hexadécimal à rentrer.

Vous pouvez faire Apply Patch et cliquez sur Run. Là vous cliquez désormais sur "Infos" --> "Reverse"? Il va breaker si vous n'avez pas enlevé le BP et vous recliquez sur Run pour continuer. Et là une boîte de dialogue vide apparaîtra. Bon ça à l'air de marcher notre modification ! Il n'y a plus qu'à rentrer tout ça dans l'éditeur hexa :

```
:01003A9F 0F871BFC0000 ja 010136C0 --> Offset 2E9F
-----
:010136C0 81FE3E010000 cmp esi, 0000013E --> Offstet 12AC0
:010136C6 0F859A0AFFFF jne 01004166
:010136CC 6A00 push 00000000
:010136CE 6800370101 push 01013700
:010136D3 6820370101 push 01013720
:010136D8 6A00 push 00000000
:010136DA FF15A8110001 call dword ptr [010011A8]
:010136E0 E9810AFFFF jmp 01004166
```

Ensuite vous allez pouvoir marquer le titre que vous souhaitez pour votre messagebox à l'offset : 12B00 (01013700). Mais il faut mettre des 00 entre chaque lettre. Par exemple "I n f o r m a t i o n s". Attention à bien laisser 2 octets (2 x "00") avant le 12B20 (adresse du texte). Si votre titre est trop long alors changez l'adresse du texte.

Une fois le titre écrit on fait pareil pour le message en 12B20. Par exemple "R e v e r s e b y D e a m o n !". Pour séparer 2 mots il faut bien mettre un espace (20 en hexa) en n'oubliant pas les deux "00" qui entourent l'espace "20".

On enregistre et on teste ! Et voilà ! C'est-y pas beau ? :-) Pour faire mieux on peut changer le style de la messagebox (avec le push 00). Pas besoin de passer par le Patch Code de WinDasm car le PUSH 00 correspond à 6A 00 donc vous pouvez le remplacer directement dans l'éditeur. Attention c'est le premier PUSH 00 (qui se trouve après le jne) car le 2ème c'est la handle de la fenêtre. Voici par quoi remplacer le 00 d'après mes tests ! :-)

Le premier chiffre correspond au type de message box :

- 0 : message normal
- 1 : message d'erreur (X)
- 2 : question (?)
- 3 : avertissement (/!\)
- 4 : informations (i)

Le second chiffre correspond aux boutons disponibles (Ok, Annuler ...)

- 0 : Ok
- 1 : Ok - Annuler
- 2 : Abandonner - Recommencer - Ignorer
- 3 : Oui - Non - Annuler
- 4 : Oui - Non
- 5 : Recommencer - Annuler
- 6 : Annuler - Recommencer - Continuer

Donc en mettant un PUSH 24 vous obtiendrez une question avec Oui et Non comme choix de boutons. Mais sachez qu'en appuyant sur n'importe quel bouton le résultat est le même car nous avons pas fait de test sur le bouton appuyé.

Voilà ce cours touche à sa fin ! Je viens de vous faire découvrir le reverse engineering, discipline assez peu utilisée. On peut l'utiliser pour par exemple intégrer un keygen à un programme en ajoutant un menu qui nous donnerait le bon serial. Mais il y a pleins d'autres possibilités...

Deamon le 22 et 23 août 2003

Cours n°13

Donne le serial au monsieur

Objectif :

Rajouter un bouton au crackme pour que, lorsque l'on clique dessus, un message nous indique quel est le bon serial !

Niveau :

Elevé

Ce qu'il vous faut :

- WinDasm
- Un éditeur hexadécimal
- Un éditeur de ressources ([Restorator](#) ou [ResHacker](#))
- Le [crack-me](#)

Introduction

Dans la lignée du cours précédent voici le 13ème cours. Vous allez voir à quoi sert ce que vous avez appris dans le dernier cours, car n'ayons pas peur de le dire : l'exercice sur la calculette de Windows ne sert pas à grand chose.

Bon tout d'abord on regarde, avec [Stup_Pe](#) par exemple, en quel langage ce crackme a été programmé. On voit qu'il a été fait en assembleur. Donc aucun signe de compression quelconque.

Donc maintenant on examine ce crackme. Il y a comme la plupart du temps 2 champs à remplir : le nom et le serial. Le bouton "Ok" est grisé, en remplissant les cases il ne s'active toujours pas donc on en conclut que le bouton se dégrisera lorsque le serial sera bon (élémentaire mon cher Watson). Pour savoir s'il faut ou non débloquent le bouton, le crackme est obligé de vérifier le serial à chaque fois qu'on tapera quelque chose. Contrairement à d'autres crackme où la vérification du serial s'effectue uniquement au moment de cliquer sur "Ok". Il va donc falloir qu'on rajoute un bouton dans ce crackme pour ensuite faire afficher une messagebox en cliquant dessus pour nous donner le bon serial par rapport au nom qu'on aura précédemment tapé.

Ajout d'un bouton

On ouvre donc le crackme avec votre éditeur de ressources favori. On va dans 'Dialog' pour rajouter un bouton qu'on appellera "Serial". Pour cela regardons comment sont faits les autres boutons.

Avec Restorator :

Après avoir cliqué sur [abI] ou 'Viewer' --> 'Edit Mode' on voit :

```
DEFPUSHBUTTON "&OK", 1, 37, 70, 50, 16, WS_DISABLED
```

- **DEFPUSHBUTTON** : type de bouton
- **"&OK"** : texte du bouton (le & signifie que la lettre O sera souligné avant de pouvoir permettre le raccourci clavier ALT+O)
- **1** : l'ID du bouton
- **37** : position dans la fenetre en horizontal
- **70** : position dans la fenetre en vertical
- **50** : longueur du bouton
- **16** : largeur du bouton
- **WS_DISABLED** : bouton grisé

Ainsi la syntaxe est la suivante :

```
TYPE Texte , ID , x , y , longueur , largeur , options
```

Donc on va rajouter :

```
DEFPUSHBUTTON "&Serial", 3, 67, 90, 50, 16
```

On mettra 3 comme ID car le 1 est déjà attribué au bouton "OK" et le 2 au bouton "Annuler". On fait F7 pour visualiser ce que ça donne. Le nouveau bouton colle un peu trop au bas de la fenêtre. On va donc augmenter la hauteur de la fenêtre du crackme.

On change : **MYDIALOG DIALOG 0, 0, 186, 106** par **MYDIALOG DIALOG 0, 0, 186, 116**. Et voilà. Vous pouvez organiser comme vous le voulez les boutons.

On appuie sur F8 pour valider et on enregistre.

Avec ResHacker :

Pour ceux qui n'aiment pas taper du code pour créer le bouton un peu "à l'aveugle"

comme on vient de la faire avec Restorator utilisez plutôt ResHacker.

Faites un clic droit sur l'aperçu du crackme et sélectionner "Insérer un contrôle". Cliquez ensuite sur l'icône représentant un bouton "Ok" pour ajouter un bouton (BUTTON in english ;-), sélectionnez "BS_DEFPUSHBUTTON" dans style. Dans "text" tapez ce que vous voulez (pour moi ce sera "Serial") et surtout n'oubliez pas de bien mettre "3" dans l'ID. Et pour finir validez.

Vous pouvez voir votre bouton sur l'aperçu. Déplacez-le avec la souris où vous le souhaitez (vous pouvez également déplacer les autres éléments à votre guise). Une fois vos changements effectués, cliquez sur "Enregistrer" à côté de "Cacher". Puis 'Fichier' --> 'Enregistrer sous...' pour sauvegarder votre crackme modifié.

Pour info : pour dégriser le bouton dans les 2 éditeurs il suffit de supprimer "WS_DISABLED". Le bouton sera ainsi dégrisé. Cela peut vous servir pour vos prochains cracks. Mais si on essaye cette méthode avec ce crackme, cela marchera (il affichera le bon message) si vous ne tapez rien. Mais dès que vous tapez votre nom le bouton redeviendra grisé car il vérifie à chaque frappe de clavier.

Ajout de la msgbox

Bon alors on ouvre le crackme avec WinDasm. Et on va essayer de trouver le moment où il regarde si un bouton a été cliqué ou non. L'ID du bouton "Ok" est 0001. Donc on cherche "0001" en regardant si on voit pas un truc du genre "cmp [registre], 1" (vous pouvez le faire également avec le bouton "Annuler"). Et effectivement on voit un :

```
:00401092 6683F801 cmp ax, 0001 --> regarde si c'est le bouton "Ok" qui a été cliqué  
:00401096 7518 jne 004010B0 --> saute si ce n'est pas le bouton "Ok"  
:00401098 6A30 push 00000030
```

* Possible StringData Ref from Data Obj -> "Enregistrement..."

```
|  
:0040109A 6816304000 push 00403016
```

* Possible StringData Ref from Data Obj -> "Merci de vous " --> le bon message !

Et en 004010B0 :

```
:004010B0 6683F802 cmp ax, 0002 --> regarde si c'est le bouton "Annuler" qui a été cliqué  
:004010B4 0F85C9000000 jne 00401183 --> si ce n'est pas "Annuler" alors saute
```

Donc là il y a pleins de solutions différentes, moi je vais changer le **cmp ax, 0001** pour mettre à la place un JMP qui nous emmènera vers notre bout de code. Dans ce code on mettra :

- **cmp ax, 0001** --> on remet le test du "Ok"

- je 00401098 --> si c'est "Ok" va vers bon message
- cmp ax, 0003 --> regarde si c'est le bouton "Serial" qui a été cliqué
- jne 004010B0 --> si ce n'est pas Serial saute en 004010B0 qui vérifiera si c'est le bouton "Annuler"
- push [style de la msgbox] |
- push [titre de la msgbox] |
- push [message de la msgbox] | Affiche la msgbox
- push 00 |
- call messagebox |
- jmp 00401183 --> retourne au programme

Recherche du serial

Maintenant on va voir où le programme stocke le bon serial afin de l'afficher avec la msgbox.

```
:00401090 753B jne 004010CD --> saute si c'est pas le bon serial
:00401092 6683F801 cmp ax, 0001 --> regarde si c'est le bouton "Ok" qui a été cliqué
:00401096 7518 jne 004010B0 --> saute si c'est pas le bouton "Ok"
:00401098 6A30 push 00000030
```

* Possible StringData Ref from Data Obj -> "Enregistrement..."

Comme le programme vérifie tout le temps le serial (comme expliqué dans l'introduction) on va voir où nous mène le JNE. On tombe sur des API GetDlgItemTextA (pour info le "A" signifie qu'il fonctionne en 32 bits). Voici leur fonctionnement :

La fonction GetDlgItemText retrouve le titre ou le texte associé à un control dans une boîte de dialogue.

```
UINT GetDlgItemText (
HWND hDlg, // handle de la boîte de dialogue.
int nIDDlgItem, // n°ID du control
LPTSTR lpString, // adresse du buffer recevant le texte
int nMaxCount // nombre maximal de caractères formant la chaîne.
);
```

En assembleur il faut inverser, ce qui donne donc :

- push [nombre maximal de caractères formant la chaîne]
- push [adresse du buffer recevant le texte]
- push [n°ID du control]
- push [handle de la boîte de dialogue]
- call GetDlgItemText

Ainsi voici ce que l'on a :

```
:004010D8 6800020000 push 00000200 --> nombre maximal de caractères
:004010DD 6850344000 push 00403450 --> le serial que nous avons tapé est stocké à cette adresse

* Possible Reference to Dialog: MYDIALOG, CONTROL_ID:03E9, ""
|
:004010E2 68E9030000 push 000003E9 --> l'ID du champ pour le serial
:004010E7 FF7508 push [ebp+08] --> handle

* Reference To: USER32.GetDlgItemTextA, Ord:0102h
|
:004010EA E8CD000000 Call 004011BC --> appel de l'API
:004010EF 6800020000 push 00000200 --> nombre maximal de caractères
:004010F4 6850324000 push 00403250 --> le nom que nous avons tapé est stocké à cette adresse

* Possible Reference to Dialog: MYDIALOG, CONTROL_ID:03E8, ""
|
:004010F9 68E8030000 push 000003E8 --> l'ID du champ pour le nom
:004010FE FF7508 push [ebp+08] --> handle

* Reference To: USER32.GetDlgItemTextA, Ord:0102h
|
:00401101 E8B6000000 Call 004011BC --> appel de l'API
:00401106 8D3550324000 lea esi, dword ptr [00403250] --> met notre nom dans ESI
:0040110C 8D3D55364000 lea edi, dword ptr [00403655] --> met ce qui est à l'adresse 00403655
dans EDI
```

Pour savoir ce que contient **00403655** lançons le débogueur. On fait "Run", on tape un nom et un serial bidon. Ensuite dans la fenêtre des registres (celle en bas à gauche) tapez **00403655** dans le champ "User Addr 1". Après il suffit de cliquer en dessous sur "UA1". Et là à droite du bouton vous pourrez apercevoir un numéro ! Tapez ce serial dans le crackme tout en gardant votre nom. Et là ça marche : le bouton "Ok" se dégrise et affiche le bon message.

Pendant qu'on y est on va chercher l'adresse de l'API MessageBoxA. On clique sur les Imports on double clique plusieurs fois pour être sûr sur l'API MessageBoxA. Et on tombe sur "Call **004011C2**"

Bon maintenant qu'on sait où se situe le serial et l'adresse de l'API, il va falloir trouver de l'espace dans le crackme pour insérer notre code.

Mise en place de notre code

On ouvre le crackme avec l'éditeur hexa et on recherche de la place (là où il y a des octets vides c'est à dire avec des 00). Allez hop il y a de la place de l'offset 750 jusqu'à 800. Ça devrait suffire. Bon on va commencer à l'offset 760 pour éviter de trop coller au texte qui est au dessus. Il va falloir trouver la correspondance de l'offset avec l'adresse WinDasm. Pour cela on prend une ligne au hasard dans Windasm, par exemple 004010CD, et on regarde l'offset, ici 4CD. On fait la différence (avec la calculatrice de Windows en mode scientifique et en hexa) : $004010CD - 4CD = 00400C00$.

Donc avec notre offset 760 cela donne : $760 + 00400C00 = 00401360$. Ce qui nous donne l'adresse Windasm.

Bon il n'y a plus qu'à rentrer notre code. On va voir ça avec le débogueur de WinDasm avec le Patch Code. On pose un break point sur la ligne du :

```
:00401092 6683F801 cmp ax, 0001
```

Pour qu'il break cliquez sur "Serial". Et on remplace par un JMP 00401360. N'oubliez pas de copier la correspondance si vous voulez recopier le code hexa dans l'éditeur hexadécimal par la suite. Ce qui nous donne :

```
:00401092 E9C9020000 jmp 00401360
```

Mais il faut remplacer ces 2 lignes :

```
:00401092 6683F801 cmp ax, 0001  
:00401096 7518 jne 004010B0
```

Et là il y a 6 octets (66-83-F8-01-75-18) contre 5 pour notre JMP. Donc l'octet suivant pourra mettre le bazar à la suite du code. Donc on va rajouter un NOP :

```
:00401092 E9C9020000 jmp 00401360  
:00401097 90 nop
```

Donc on applique le changement et on fait un F7 pour avancer dans le code vers nos bytes libres (mais plus pour longtemps ;-)).

Et on va entrer le code suivant (les explications de chaque ligne ont été expliquées plus haut) :

```
:00401360 6683F801 cmp ax, 0001  
:00401364 0F842EFDFFFF je 00401098  
:0040136A 6683F803 cmp ax, 0003  
:0040136E 0F853CFDFFFF jne 004010B0  
:00401374 6A40 push 00000040 --> Cf. cours précédent pour choisir le type de messagebox (ici Informations - Ok)  
:00401376 6850324000 push 00403250 --> Titre : notre nom
```

```
:0040137B 6855364000 push 00403655 --> Message : le bon serial  
:00401380 6A00 push 00000000  
:00401382 E83BFEBFFF call 004011C2 --> Appel l'api MessageBoxA  
:00401387 E9F7FDFFFF jmp 00401183
```

Pour tester, on applique, on enlève le BP et on clique sur "Run". On retape un nom puis on clique sur "Serial" et là apparaît un beau message avec notre nom en titre et le serial en message. On recopie le serial dans le champ prévu à cet effet et là le bouton "Ok" se dégrise et ça marche.

Vous n'avez plus qu'à faire les modifications dans votre éditeur hexa. Je ne vous explique plus comment faire, je vous fais confiance ;-) (sinon voir le dernier cours). Je rappelle qu'il faut commencer à l'offset 760.

Voilà ce cours se termine. J'espère que vous avez compris et que ce cours vous a plu. N'hésitez pas à me demander si vous avez un problème, une question, une suggestion ou même une critique, soit par l'intermédiaire du forum : www.forumcrack.fr.st ou par email : daemon.crack@netcourrier.com

Deamon le 31 août 2003

Cours n°14

Nag d'un crackme en Visual Basic

Objectif :

Enlever le Nag d'un crackme.

Niveau :

Moyen

Ce qu'il vous faut :

- Un éditeur hexadécimal
- Le [crack-me](#) (40 Ko)

Introduction

On va s'attaquer à un crackme en Visual Basic. Mais on va uniquement s'occuper de la partie Nag. Pour info le serial et le 'Word code' sont visibles dans les Text String Referenced avec OllyDbg (accessible en faisant clic droit : Search For --> All Referenced Text Strings par exemple.

En voyant les outils nécessaires pour ce cours vous avez dû être surpris ! Car pour cette fois ci nous n'aurons besoin que d'un éditeur hexadécimal.

Vous avez pu voir qu'un vilain nag avec marqué "NAG" dessus (comme c'est original) apparaît au lancement du programme, avant même que la fenêtre principale ne s'affiche. Le Nag comporte un Timer de 5000ms qui une fois les 5s passées, ferme le nag pour ouvrir la fenêtre principale.

[Éditeur hexadécimal] 1 - 0 [Visual Basic]

On va donc étudier ce crackme avec l'éditeur hexadécimal. Le programme étant fait en Visual Basic 6 on va donc chercher dans l'éditeur : "VB5!" ! Logique non ? Cela marche pour les programmes faits avec Visual Basic 4,5 et 6. Et dans ces 3 cas il faudra chercher "VB5!". Et on tombe ici :

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------------|
| 0E | 4E | 65 | 77 | 73 | 20 | 47 | 6F | 74 | 68 | 69 | 63 | 20 | 4D | 54 | FF | News Gothic MTý |
| 02 | 04 | 00 | 00 | 06 | 00 | 00 | 00 | D8 | 2E | 40 | 00 | 56 | 42 | 35 | 21 | ø.ø VB5! |
| FO | 1F | 2A | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | š * |
| 7E | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 0A | 00 | ~ |
| 09 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 30 | 14 | 40 | 00 | o ø |
| 1E | F8 | 38 | 01 | 00 | FF | FF | FF | 08 | 00 | 00 | 00 | 01 | 00 | 00 | 00 | ø8 ýýý |
| 06 | 00 | 00 | 00 | E9 | 00 | 00 | 00 | 50 | 12 | 40 | 00 | B4 | 11 | 40 | 00 | é P ø ´ ø |
| 78 | 10 | 40 | 00 | 78 | 00 | 00 | 00 | 81 | 00 | 00 | 00 | 8A | 00 | 00 | 00 | x ø x Š |
| 8B | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | < |
| 00 | 00 | 00 | 00 | 63 | 5F | 73 | 69 | 6F | 63 | 6D | 31 | 00 | 43 | 72 | 61 | c siocm1 Cra |
| 63 | 6B | 2D | 4D | 65 | 00 | 00 | 50 | 72 | 6F | 6A | 65 | 63 | 74 | 31 | 00 | ck-Me Project1 |
| 50 | 00 | 00 | 00 | 83 | 8E | D9 | 2B | 6C | 64 | D3 | 11 | 8B | E8 | 9E | BD | P f Û+ldó <è ½ |
| 08 | 6F | D2 | 46 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | oÓF |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 45 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | B0 | 3E | 40 | 00 | 4C | 00 | 00 | 00 | E °>ø L |
| 50 | 00 | 00 | 00 | 87 | 8E | D9 | 2B | 6C | 64 | D3 | 11 | 8B | E8 | 9E | BD | P + Û+ldó <è ½ |
| 08 | 6F | D2 | 46 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | oÓF |
| 00 | 00 | 00 | 00 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | uu |
| 8A | 04 | 00 | 00 | 00 | 00 | 00 | 00 | F8 | 42 | 40 | 00 | 9C | 00 | 00 | 00 | š øBø œ |
| 50 | 00 | 00 | 00 | 8B | 8E | D9 | 2B | 6C | 64 | D3 | 11 | 8B | E8 | 9E | BD | P < Û+ldó <è ½ |
| 08 | 6F | D2 | 46 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | oÓF |
| 00 | 00 | 00 | 00 | 02 | 00 | 00 | 00 | 10 | 01 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| F6 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | BC | 10 | 40 | 00 | EC | 00 | 00 | 00 | ö ø ø i |
| 50 | 00 | 00 | 00 | 8F | 8E | D9 | 2B | 6C | 64 | D3 | 11 | 8B | E8 | 9E | BD | P Û+ldó <è ½ |
| 08 | 6F | D2 | 46 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | oÓF |
| 00 | 00 | 00 | 00 | 03 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| DB | 07 | 00 | 00 | 00 | 00 | 00 | 00 | 84 | 47 | 40 | 00 | 3C | 01 | 00 | 00 | Û øø < |
| 50 | 00 | 00 | 00 | 93 | 8E | D9 | 2B | 6C | 64 | D3 | 11 | 8B | E8 | 9E | BD | P Û+ldó <è ½ |
| 08 | 6F | D2 | 46 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | oÓF |
| 00 | 00 | 00 | 00 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 48 | 04 | 00 | 00 | 00 | 00 | 00 | 00 | 68 | 3A | 40 | 00 | 8C | 01 | 00 | 00 | H h:ø œ |
| 50 | 00 | 00 | 00 | 97 | 8E | D9 | 2B | 6C | 64 | D3 | 11 | 8B | E8 | 9E | BD | P - Û+ldó <è ½ |
| 08 | 6F | D2 | 46 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | oÓF |
| 00 | 00 | 00 | 00 | 05 | 00 | 00 | 00 | 80 | 03 | 00 | 00 | 00 | 00 | 00 | 00 | |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | |
| A0 | EF | 00 | 00 | 00 | 00 | 00 | 00 | 60 | 4F | 40 | 00 | DC | 01 | 00 | 00 | i `øø Û |
| F4 | 01 | 00 | 00 | A0 | 2A | 40 | 00 | 00 | 00 | 00 | 00 | 20 | 4B | 41 | 00 | ô *ø KA |

Voici les explications de tout cela : on a donc notre "VB5!" en bleu clair en haut. En dessous, en vert, c'est l'en-tête de projet VB qui contient l'adresse des en-têtes fenêtres qui est ici en jaune. On va donc aller à l'en-tête des fenêtres. Ici on a 50|12|40|00 ce qui fait en bytes inversés (on lit à de droite à gauche en gardant des "paquets" de deux chiffres) : 00|40|12|50. On va donc aller à l'offset 1250h (on ne s'occupe pas du 0040) qui se trouve juste en dessous. En VB les programmes sont faits de plusieurs fenêtres appelées Form (une pour le menu, une pour la fenêtre de serial, de code... et une pour le NAG !!!).

Pour les en-têtes des Form on a en rouge la signature, c'est à dire le nombre de byte de l'en-tête. Normalement les en-têtes font 50h bytes ce qui fait 80 en décimal (dans l'éditeur une ligne fait 16 octets donc 16*5=80, l'en-tête s'étend sur 5 lignes donc). En Jaune on a l'adresse de la Form, là où sont tout le contenu de la form.

Et en rose/violet, le plus intéressant, c'est l'attribut de la Form. C'est ça que nous allons modifier.

Pour connaître quel Form appartient à quelle fenêtre on peut utiliser des programmes comme VBEditor ou VBReformer. Mais sinon on peut aller voir ce qu'il contient dans l'éditeur hexa.

Pour trouver la bonne adresse de la Form il faut prendre l'offset donné et rajouter 64h. Par exemple pour le premier dont l'adresse est noté B0|3E|40|00 donc à l'offset 3EB0 cela nous donne $3EB0+64=3F14$ (la calculatrice Windows fait les calculs en hexa en mode scientifique).

A l'offset 3F14 on voit "Form1" et "Serial Section". C'est donc la fenêtre du Serial. Donc on fait pareil avec les 5 autres adresses. Et à la fin on trouve que la Form du NAG est la 3 et la fenêtre principal du crackme est la 4.

Form1 - Serial
Form2 - Name serial
Form3 - NAG
Form4 - Fenêtre Principal
Form5 - Word Code
Form6 - About

On va donc inverser les attributs de ces fenêtres pour que ce soit la fenêtre principale qui s'ouvre au démarrage et non le nag.

L'attribut (en violet) du nag est 10|01 soit 0110h et la Form4 est 00|07 soit 0700h. Quand un attribut est **xx10h** c'est que c'est la form qui se lance au démarrage. Les autres étant à **xx00h** principalement.

On va donc mettre **0010h à la place de 0110h (00|01 au lieu de 10|01) et 0710h à la place de 0700h (10|07 au lieu de 00|07)**.

En faisant ça, la fenêtre principale se lancera au démarrage du programme. ce ne sera plus le Nag.

On enregistre, on teste et Bingo! ça marche !

Donc voici comment on peut modifier un programme fait en Visual Basic, sans utiliser désassembleur ou débogueur.

Deamon le 06 décembre 2003

Cours n°15

Crackme avec KeyFile

Objectif :

- 1° Méthode brute : le cracker tout bêtement ! :-)
- 2° Méthode plus "intelligente" je dirais : générer un keyfile valide pour ce crackme.

Niveau :

Moyen - Elevé

Ce qu'il vous faut :

- Un éditeur hexadécimal
- [OllyDbg](#) (qui va remplacer WinDasm ; aucune connaissance de ce logiciel est nécessaire dans ce cours, je vous expliquerai le fonctionnement au fur et à mesure ; la version utilisée ici est la 1.09d)
- La calculatrice Windows (ou tout autre calculette pour les calculs hexadécimaux)
- Le [crackme](#) (9 Ko)

Introduction

Nous allons dans ce cours étudier un type d'enregistrement différent de ceux que nous avons rencontré au cours de mes anciens tutoriaux. Ici pas de serial, ni de nag-screen ou autres mais c'est de KeyFile qu'il est question. Pour ceux qui ne connaissent pas, le fonctionnement est le suivant : pour vérifier si le programme est enregistré celui-ci va regarder dans un fichier (le plus souvent c'est un .ini, .dat, ...) qui contient une clé. Si la clé est valide le programme est enregistré.

1° Cracker ce crackme (pléonasme devenu habituel à mes cours...)

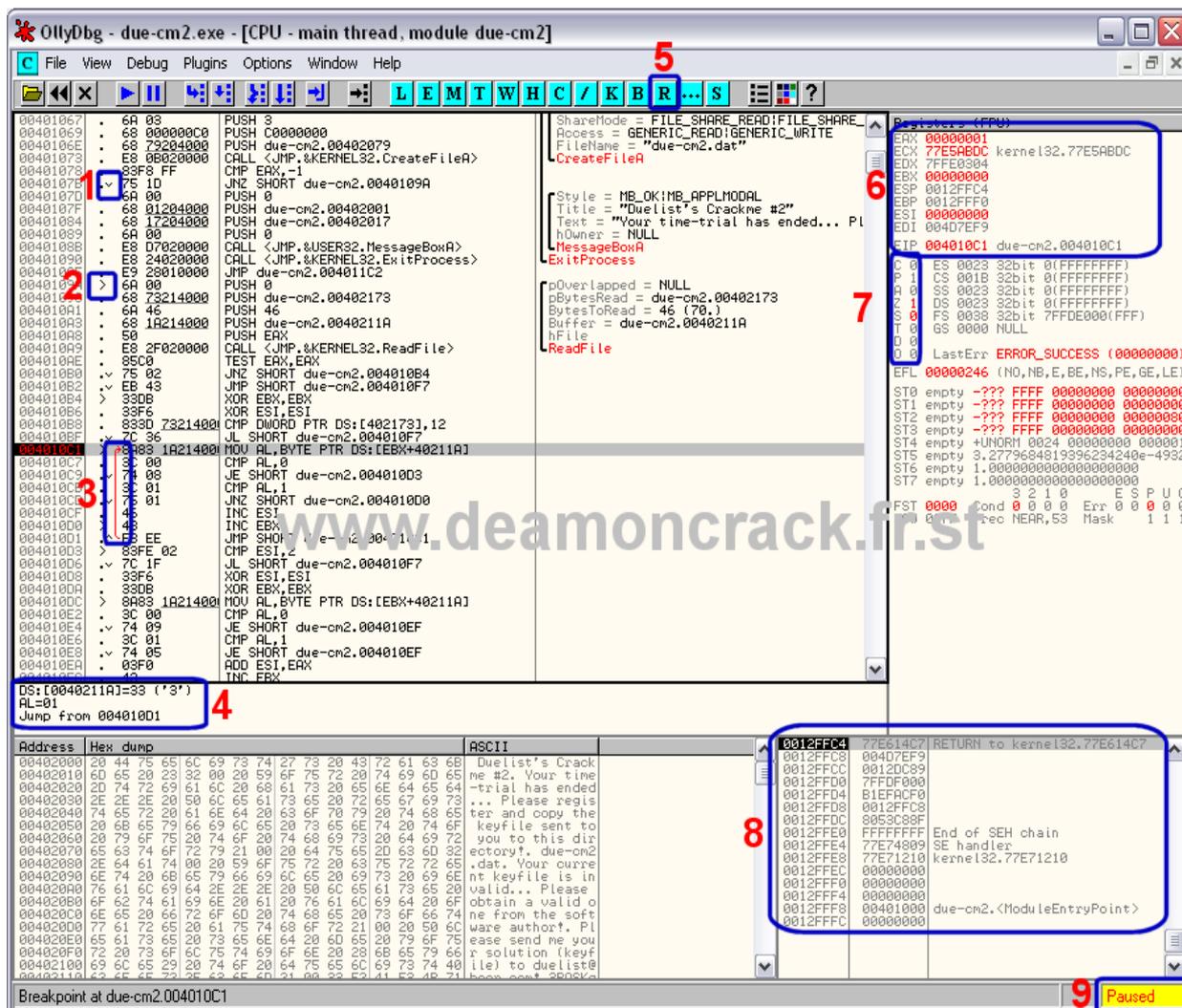
Nous allons donc lancer le crackme pour voir ce qu'il nous demande. Il nous affiche une message box affichant : *"Your time-trial has ended... Please register and copy the keyfile sent to you to this directory"*. Pour les anglophobes voici la traduction : "La période d'essai est terminée... Veuillez vous enregistrer et copier le keyfile que vous avez reçu dans ce dossier". Même pas eu le temps d'utiliser ce crackme et la période d'essai est déjà terminée ! Quelle arnaque ! :-P

Bon alors pour ce cours pas de WinDasm, on va apprendre à utiliser OllyDbg car je trouve qu'il est mieux, plus puissant et de plus en plus utilisé. D'ailleurs c'est moins facile de cracker ce crackme avec WinDasm car il ne trouve aucune String Data References. Donc on lance ce logiciel et on ouvre notre crackme qui tremble déjà de peur, son heure approche ! :-D

Normalement après avoir désassemblé le crackme vous devriez tomber nez à nez avec notre vilain message. Mais on va faire comme si on l'avait pas vu pour ne pas éveiller

les soupçons... On va plutôt le prendre à revers ! :-p Alors discrètement on va faire clic droit dans le cadre où il y a les lignes de code, puis "Search For" --> "All referenced strings text", ce qui correspond au Strings Data References de WinDasm. Autre méthode : cliquer sur l'icône du "R" sur fond bleu (mais il faut obligatoirement avoir préalablement déjà chargé les SDR au moins une fois avec le menu). Une fois la fenêtre des Strings ouverte, double-cliquez sur le mauvais message pour vous y rendre. Bon vous allez tomber au même endroit qu'il y a 30 secondes, mais c'est juste pour vous montrer comment rechercher les messages.

Voici une vue globale d'OllyDbg :



- 1) Cette flèche signifie que l'instruction peut sauter vers un autre endroit, pour voir où le saut ira on peut cliquer sur la ligne et une flèche nous montrera. En mode pas à pas lorsque la flèche est grise cela signifie que le saut n'est pas effectué.
- 2) Cette flèche indique que cette instruction est appelée par un saut, c'est l'arrivée d'un saut.
- 3) La ligne 004010C1 appelé par un saut, étant sélectionnée, une flèche nous montre quel saut nous amène sur cette instruction.
- 4) Très utile car cela nous indique les valeurs que le programme manipule, ainsi pour l'instruction `MOV AL, BYTE PTR DS:[EBX+40211A]`, dans cette fenêtre est indiquée la valeur de `DS:[EBX+40211A]` (ici 33, en hexa bien sur) et celle de AL (AL est une partie de EAX et correspond aux bits de poids faibles, pour simplifier AL est les 2 premiers chiffres

en partant de la droite de EAX donc ici 01). Quand c'est possible on nous donne également le caractère ascii auquel correspond la valeur, ici on nous indique que 33h correspond au caractère "3". Ce qui est utile quand le programme manipule les caractères d'un serial par exemple. C'est ici aussi que vous pouvez voir si un saut s'effectue ou non ("Jump is taken" ou "Jump is not taken").

5) Bouton pour voir les Strings Data References.

6) Les registres et ses valeurs. Peut aussi afficher en ascii pour un nom ou serial par exemple.

7) Les flags. Peut être utile pour inverser un saut. Par exemple si un JZ (=JE) ne saute pas mais vous voulez qu'il saute il suffit de double-cliquez sur la valeur du Flag Z pour l'inverser.

8) Nous indique ce qui est poussé sur la pile (avec PUSH). On peut y voir passer des serials parfois ! :-P

9) Permet de voir si le programme qu'on débogue est en exécution, en pause (s'il break)...

Donc voilà les points importants d'OllyDbg. Maintenant on va regarder comment cracker notre petit programme. On voit à l'endroit du message :

```
00401078 . 83F8 FF    CMP EAX,-1 ; |
0040107B . 75 1D    JNZ SHORT due-cm2.0040109A ; |
0040107D . 6A 00    PUSH 0 ; |/Style = MB_OK|MB_APPLMODAL
0040107F . 68 01204000  PUSH due-cm2.00402001 ; ||Title = "Duelist's Crackme #2"
00401084 . 68 17204000  PUSH due-cm2.00402017 ; ||Text = "Your time-trial has ended... Please
register and copy the keyfile sent to you to this directory!"
00401089 . 6A 00    PUSH 0 ; ||hOwner = NULL
0040108B . E8 D7020000  CALL <JMP.&USER32.MessageBoxA> ; |\MessageBoxA
00401090 . E8 24020000  CALL <JMP.&KERNEL32.ExitProcess> ; \ExitProcess
00401095 . E9 28010000  JMP due-cm2.004011C2
0040109A > 6A 00    PUSH 0 ; /pOverlapped = NULL
```

Il y a donc un test pour savoir si EAX = -1 et si ce n'est pas le cas il saute après le mauvais message. Nous verrons par la suite à quoi cela correspond. On va donc remplacer le JNZ par un JMP (à noter que JZ=JE et JNZ=JNE, ce sont les mêmes sauts, sous OllyDbg ils sont plutôt notés JZ/JNZ et sous WinDasm JE/JNE). Pour tester dans OllyDbg on double-clique sur l'instruction JNZ et on remplace **JNZ SHORT 0040109A** par **JMP SHORT 0040109A**. On valide et on regarde si ça marche. On lance donc le crackme dans OllyDbg avec F9 (les raccourcis claviers des fonctions principales sont identiques à celle du débogueur de windasm : F2 pour poser un BreakPoint, F9 pour Run, F7 : Step Into, F8 : Step Out...).

Alors maintenant le message n'est plus le même puisqu'il nous dit : "Your current keyfile is invalid... Please obtain a valid one from the software author!". Message que nous retrouvons un peu en dessous du précédent. Le programme s'étant fermé, il faut réinitialiser OllyDbg en appuyant sur CTRL+F2.

Bon donc ce saut ne suffit pas, mais il faut tout de même le modifier. Mais après la réinitialisation les modifications sont annulées (de toute façon les modifications dans OllyDbg tout comme le débogueur de WinDasm ne sont que temporaires). Donc il va falloir refaire la modification sur le JNZ. On va pour cela poser un breakpoint dessus. On Run, il breake sur le JNZ. Là on va modifier le flag Z en double-cliquant sur sa valeur qui est ici à 1 pour le faire passer à 0. La flèche du saut devient rouge et le "Jump is NOT taken" se transforme en "Jump is taken". Et maintenant on va avancer en pas à pas avec F8 jusqu'au prochain saut qui pourrait nous emmener sur le

mauvais message. Et on arrive ici :

```
004010AE . 85C0 TEST EAX,EAX  
004010B0 . 75 02 JNZ SHORT due-cm2.004010B4  
004010B2 . EB 43 JMP SHORT due-cm2.004010F7  
004010B4 > 33DB XOR EBX,EBX
```

Donc là si le JNZ saute (ce qu'il ne fera pas si vous avez bien regardé) il passera au dessus du JMP. Ce dernier sautera à l'adresse **004010F7**. Regardons ce qui se trouve à cette adresse : notre mauvais message "Your current keyfile is invalid...". Il faut donc faire sauter ce JNZ. Alors lorsqu'on s'arrête sur cette ligne en avançant en pas à pas, on fait la même chose que tout à l'heure : modifier le flag Z pour qu'il saute. Et on note le changement qu'il faudra effectuer ensuite dans l'éditeur hexa. Puis on tombe sur :

```
004010BF . 7C 36 JL SHORT due-cm2.004010F7
```

Il saute sur l'adresse du mauvais message donc on le noppera ou l'inversera. Pour modifier le flag ce n'est plus le Z car le saut est JL (Jump if Less = Saute si moins/ inférieur) mais le flag S (Sign Flag). Ici même topo :

```
004010D6 . 7C 1F JL SHORT due-cm2.004010F7
```

Et donc juste avant le mauvais message il y en a encore un autre (le dernier ?) à transformer en JMP et qu'on passera avec le flag Z :

```
004010F5 . 74 1D JE SHORT due-cm2.00401114
```

Maintenant on croit en avoir fini, car on a déjà dû modifier 6 sauts pour en arriver là, mais non il en reste encore un plus bas qui saute lui aussi vers 004010F7 :

```
00401155 . 75 A0 JNZ SHORT due-cm2.004010F7
```

Et voilà ça devrait être bon ! :-)) Il ne semble pas y en avoir d'autre donc après avoir modifié le flag Z pour ce dernier on fait F9 pour continuer le programme... Et là : "This program is registered to our dear user :". Bon forcément il n'y a pas notre nom. Si vous voulez afficher votre nom lisez la suite :-)) on va faire ça dans la deuxième partie de ce cours.

Donc pour résumer voici les sauts que nous avons à modifier :

```
0040107B . 75 1D JNZ SHORT due-cm2.0040109A ---> JMP  
004010B0 . 75 02 JNZ SHORT due-cm2.004010B4 ---> JMP  
004010BF . 7C 36 JL SHORT due-cm2.004010F7 ---> JNL ou NOP  
004010D6 . 7C 1F JL SHORT due-cm2.004010F7 ---> JNL ou NOP  
004010F5 . 74 1D JE SHORT due-cm2.00401114 ---> JMP  
00401155 . 75 A0 JNZ SHORT due-cm2.004010F7 ---> NOP ou JZ
```

Mais maintenant que nous avons vu que le bon message commençait ici :
[00401157 . 6A 00 PUSH 0 ; /IPParam = NULL] on peut donc mettre à la place du premier saut (0040107B . 75 1D JNZ SHORT due-cm2.0040109A) un **JMP 00401157** pour ainsi sauter directement vers le bon message. Ça marche et il n'y a qu'un endroit à modifier.

Même si ça empiète sur le premier mauvais message ce n'est pas grave car il ne passera plus jamais par là ! :-P

Ce qui donne à la place de :

```
0040107B /75 1D JNZ SHORT due-cm2.0040109A
0040107D |6A 00 PUSH 0
0040107F |68 01204000 PUSH due-cm2.00402001 ; ASCII "Duelist's Crackme #2"
```

On aura :

```
0040107B /E9 D7000000 JMP due-cm2.00401157
00401080 |90 NOP
00401081 |90 NOP
00401082 |90 NOP
00401083 |90 NOP
```

Petite info : dans OllyDbg lorsque vous modifiez une ligne, le fait de cocher "Fill with NOP's" remplit les octets vides par des NOP comme pour l'exemple au-dessus. Par exemple si vous avez un JMP qui donne [E9D7000000] en hexa et que vous voulez mettre un petit JNZ de type [751D], vous aurez 3 octets de "libre" du JMP. Ceux là seront remplacés par des NOP pour ne pas perturber le programme. Donc le mieux est de laisser cette case cochée.

2° Créer un KeyFile valide

Bon on recommence tout depuis le début. On ouvre avec OllyDbg le crackme (et pas celui déjà modifié ;-). On va vers notre premier message d'erreur ("Your time-trial..."). Et on remarque que juste au dessus l'API CreateFileA, donc l'accès à un fichier. Et dans les paramètres de l'API que voit-on ?

```
0040106E . 68 79204000 PUSH due-cm2.00402079 ; ||FileName = "due-cm2.dat"
00401073 . E8 0B020000 CALL <JMP.&KERNEL32.CreateFileA> ; |\CreateFileA
```

FileName="due-cm2.dat" !! Ça ne serait pas notre keyfile par hasard ? Pour le savoir on va créer un fichier nommé "due-cm2.dat" dans le dossier du crackme. Mais on va pas le laisser vide on va (avec le bloc-notes par exemple) mettre "123456" dedans (sans les guillemets bien sûr !). Vous enregistrez et vous lancez le crackme (en dehors d'OllyDbg). Et là le deuxième message d'erreur apparaît : "Your current keyfile is invalid... Please obtain a valid one from the software author!". Bon on va voir pourquoi le contenu du fichier n'est pas bon. Pour cela direction OllyDbg.

Donc là on comprend mieux à quoi sert le saut situé entre le CreateFileA et le 1er message :

```
00401078 . 83F8 FF CMP EAX,-1
0040107B 75 1D JNZ SHORT due-cm2.0040109A --> si le fichier n'existe pas, alors ne saute pas et passe par le 1er message.
```

Ce saut nous emmène sur l'API ReadFile (tout le monde aura deviné à quoi cette API sert :-)) :

```
0040109A > 6A 00 PUSH 0 ; /pOverlapped = NULL
0040109C . 68 73214000 PUSH due-cm2.00402173 ; |pBytesRead = due-cm2.00402173
004010A1 . 6A 46 PUSH 46 ; |BytesToRead = 46 (70.)
004010A3 . 68 1A214000 PUSH due-cm2.0040211A ; |Buffer = due-cm2.0040211A
004010A8 . 50 PUSH EAX ; |hFile
004010A9 . E8 2F020000 CALL JMP.&KERNEL32.ReadFile ; \ReadFile
004010AE . 85C0 TEST EAX,EAX
```

On va donc breaker juste après l'API sur le TEST EAX, EAX. On sélectionne la ligne est on fais F2 (le bout de la ligne devient rouge). On Run (F9) et ensuite on réfléchit. On peut voir (dans la fenêtre numérotée 4 ou 6 sur le schéma au-dessus) que EAX est à 1. On va avancer en pas à pas avec F8 (ou F7 mais ce dernier va vous faire rentrer dans les CALL et là on n'en aura sans doute pas besoin). Pour mieux vous repérer vous pouvez mettre des commentaires sur les lignes en faisant clic-droit et "Comment" (;) sur la ligne que vous souhaitez.

Le saut JNZ de la ligne suivante saute au-dessus du JMP qui nous mène au bad message. Ce qui est bien donc. Ensuite on trouve deux XOR sur EBX lui-même et ESI également ce qui a pour effet de mettre ces registres à 0. Les XOR sont souvent utilisés pour réinitialiser les registres. On arrive après les XOR ici :

```
004010B8 . 833D 73214000 CMP DWORD PTR DS:[402173],12 --> compare DS:[402173] à 12h
004010BF . 7C 36 JL SHORT due-cm2.004010F7 --> saute vers le mauvais message si DS:
[402173] est inférieur à 12h
```

A ce moment DS:[402173] vaut 6, si on se souvient bien on à mis le nombre 123456 dans le keyfile. 6 correspond donc à son nombre de caractères. Il faut donc que le serial du keyfile fasse 12h lettres minimum (12 en hexa correspond à 18 en décimal, donc 18 caractères minimum). On note donc cette condition et on continue en changeant le flag S pour éviter le saut du JL. Et là on arrive sur une petite routine :

```
004010C1 MOV AL,BYTE PTR DS:[EBX+40211A] --> met le caractère dans AL
004010C7 CMP AL,0 --> compare le caractère à 00h
004010C9 JE SHORT due-cm2.004010D3 --> sort de la routine si le caractère = 00h
004010CB CMP AL,1 --> compare le caractère à 01h
004010CD JNZ SHORT due-cm2.004010D0 --> saute au dessus de INC ESI si le caractère est
différent de 01h
004010CF INC ESI --> incrémente ESI
004010D0 INC EBX --> incrémente EBX = caractère suivant
004010D1 JMP SHORT due-cm2.004010C1 --> boucle au début de la routine
004010D3 CMP ESI,2 --> compare ESI à 2
004010D6 JL SHORT due-cm2.004010F7 --> Saute vers le Bad Boy si ESI < 2
```

J'espère qu'avec les couleurs c'est plus compréhensible ! :-)) J'ai fait du mieux que j'ai pu.

Donc pour résumer en rose/rouge c'est la routine qui va d'abord mettre à chaque passage un caractère (le 1er puis le 2ème...etc) dans AL. Ensuite si le caractère est 00 (en hexa) il sort de la routine. Il trouvera un 00 quand il n'y aura plus de caractères. Après cela si le caractère = 01h il ne saute pas et incrémente ESI, sinon pour tout autre caractère il n'incrémente pas ESI car il saute par dessus. Et enfin il incrémente EBX pour passer au caractère suivant.

Quand vous êtes sur la ligne du MOV, vous pouvez voir la valeur du caractère

étudié, le premier sera le "1", mais attention "1" est un caractère ascii et non une valeur. Sa valeur est 31h.

Donc pour éviter le mauvais message il faut que ESI soit égal ou supérieur à 2. C'est à dire qu'il y ait au moins deux caractères 01h dans le serial. On va donc mettre un serial valide. Pour mettre 01h il faut donc utiliser l'éditeur hexadécimal. Vous ne pourrez pas modifier le .dat à cet instant car le crackme est en exécution dans OllyDbg, on va donc noter à quel ligne on repartira (004010D6) et on réinitialise OllyDbg avec CTRL+F2. Si un message s'affiche faites "Oui", une erreur peut survenir mais c'est normal dans ce cas les breakpoint et les modifications seront annulées mais ce n'est pas grave. Une fois initialisé on peut modifier le .dat. On va donc mettre 18 caractères dont deux 01h. Dans HexDecCharEditor pour ajouter des bytes il faut faire "Edit" --> "Insert Bytes" ou la touche Ins.

On va donc mettre par exemple (en ascii) : 123456 789ABC DEFG (les 01h, ne pouvant pas être représentés en ascii, correspondent ici aux espaces).
Ce qui donne en hexa : 31|32|33|34|35|36|01|37|38|39|41|42|43|01|44|45|46|47
Je vous déconseille de mettre des caractères identiques dans le serial car après vous aurez des difficultés à savoir sur quel caractère travaille le crackme dans ses routines et ses tests. Bon donc on enregistre et on va regarder ce qu'il nous veut encore, ce crackme.

On pose un BreakPoint en 004010D6 comme on l'avait noté, car les tests d'avant doivent être corrects. On Run, ça break normalement et sur le JL on voit : "Jump is NOT taken" (enfin si vous avez bien rempli correctement le .dat). Le crackme a donc bien compté les deux 01h. On avance en pas à pas toujours : XOR EBX, EBX et XOR ESI, ESI : mise à 0 de ces deux registres. Et ensuite juste avant le deuxième mauvais message on rencontre une routine quasi-semblable à la précédente :

```
004010DC MOV AL,BYTE PTR DS:[EBX+40211A] --> met le caractère dans AL
004010E2 CMP AL,0 --> compare le caractère à 00h
004010E4 JE SHORT due-cm2.004010EF --> sort de la routine si le caractère = 00h
004010E6 CMP AL,1 --> compare le caractère à 01h
004010E8 JE SHORT due-cm2.004010EF --> sort de la routine si le caractère = 01h
004010EA ADD ESI,EAX --> ajoute la valeur du caractère à ESI
004010EC INC EBX --> caractère suivant
004010ED JMP SHORT due-cm2.004010DC --> boucle au début de la routine : et c'est reparti pour un tour :-p !
004010EF CMP ESI,1D5 --> compare ESI à 1D5h
004010F5 JE SHORT due-cm2.00401114 --> saute au dessus du bad boy si ESI = 1D5h
```

Explication pour ceux qui ne comprennent toujours pas mes belles routines colorées :-): chaque caractère est placé tour à tour dans AL (donc dans EAX car je répète que AL est une partie de EAX). Si le caractère est 00h ou 01h alors il sort de la routine. Sinon il ajoute la valeur hexa du caractère dans ESI. Ainsi on aura dans ESI, à la fin de la routine, la somme des caractères situés avant le premier 01h ou 00h qui joue le rôle de séparateur (pour notre .dat ce sera 01h qui arrivera avant le 00h). Après la routine il vérifie que ESI (donc la somme des caractères) soit égale à 1D5h. Et si ce n'est pas le cas PAF un "Your keyfile is invalid..." te saute à la figure ! Pour résumer : il faut donc que la somme des caractères situés avant le premier 01h soit égale à 1D5h.

Bon on note où on en est (004010F5) si on ne veut pas se perdre, on réinitialise tout ça et on va faire les modifications nécessaires. Pour notre exemple nous avons

mis 6 caractères avant le 01h. On va donc prendre la calculatrice pour se débrouiller à avoir une somme de 1D5h (et oui car de tête c'est pas pratique le calculs hexadécimaux :-/). Pour la calculatrice mettez vous en Mode Scientifique et en "Hex". On va faire 1D5/6 ce qui donne 4E. Mais en hexa il n'y a pas de décimal donc 4E correspond à la partie entière. Pour trouver le reste vous devez savoir faire je pense ! :-) Bon je vais vous aider quand même ! ;-) On fait 4E*6 on trouve 1D4. Et 1D5-1D4=1 (cette soustraction à été calculée de tête là ! Balèze hein ? :-P) donc on aura cinq 4E et un 4F (car 4E+1=4F). Ce qui nous fait un total de 1D5 ! Le compte est bon ! :-) On a donc dans l'éditeur hexa : 4E|4E|4E|4E|4E|4F|01|37|38|39|41|42|43|01|44|45|46|47.

On retourne à OllyDbg en posant un breakpoint sur 004010F5, on Run, ça break et on voit sur le JE : "Jump is taken" ! Ouais ça marche ! Et là on se dit que c'est fini car on saute après le deuxième mauvais message, mais comme on l'a vu en première partie il reste encore des vérifications ! Donc on avance prudemment en faisant attention aux sauts qui, tels des mines, peuvent vous faire sauter n'importe où ! :-P On passe par dessus le bad boy, on tombe sur une mise à 0 de ESI et une incrémentation de EBX. Pourquoi changer EBX ? Car EBX correspond toujours au numéro de position des caractères et donc il passe au caractère suivant. Rappelez vous on s'était arrêté au 01h à la routine précédente ! Et là à partir du INC EBX on tombe encore une routine, c'est donc reparti pour une routine colorée par mes soins :-) :

```
00401116 INC EBX --> caractère suivant
00401117 MOV AL, BYTE PTR DS:[EBX+40211A] --> met le caractère dans AL
0040111D CMP AL, 0 --> compare le caractère à 00h
0040111F JE SHORT due-cm2.00401139 --> sort de la routine si le caractère = 00h
00401121 CMP AL, 1 --> compare le caractère à 01h
00401123 JE SHORT due-cm2.00401139 --> sort de la routine si le caractère = 01h
00401125 CMP ESI, 0F --> compare ESI à 0Fh
00401128 JNB SHORT due-cm2.00401139 --> sort de la routine si ESI > ou = 0Fh
0040112A XOR AL, BYTE PTR DS:[ESI+40211A] --> XOR entre ESI+40211A et le caractère (cf.
explication plus bas)
00401130 MOV DWORD PTR DS:[ESI+402160], EAX --> met le résultat dans ESI+402160
00401136 INC ESI --> incrémente ESI = caractère suivant (cf. explication plus bas)
00401137 JMP SHORT due-cm2.00401116 --> boucle au début de la routine
00401139
```

Bon donc on commence avec le caractère qui est situé après le premier 01h. Si le caractère est 00h ou 01h (qui servent encore de séparateur) on sort de la routine. On la quitte également si ESI >= 0Fh.

Avant d'expliquer ce qu'il fait avec le XOR, voyons ce qui correspond à quoi. 40211A est l'adresse où est situé le serial. Donc EBX+40211A correspond au caractère placé en "EBX position" (donc au départ la position juste après le 01h, donc, ici, en 7ème position). Pour ESI+40211A, il faut savoir que juste avant la routine, ESI a été mis à 0. Donc ESI+40211A correspond au 1er caractère. Ainsi pour le `XOR AL, BYTE PTR DS:[ESI+40211A]` cela signifie qu'il fait une opération XOR entre le N^{ème} caractère après le 01h (donc AL) et le N^{ème} caractère du serial (ESI+40211A). Par exemple lors de la première boucle on aura un XOR entre le 1er caractère et le 7ème. Puis à la deuxième boucle : 2ème caractère XOR 8ème caractère. C'est compliqué à expliquer donc voici pour vous aider la correspondance en couleur : 4E|4E|4E|4E|4E|4F|01|37|38|39|41|42|43|01|44|45|46|47. Les XOR se font entre les valeurs de même couleur.

Le résultat de calcul est mis à l'adresse : ESI+402160. Si vous voulez voir ce que

cela donne vous pouvez regarder dans la fenêtre en bas à l'adresse 402160 (+ESI mais comme celui-ci a varié de 0 à 6 les résultats commencent en 402160+0 donc bien 402160 et finissent en 402166) ce que cela donne (ici "yvw "). Nous verrons par la suite à quoi cela sert.

On continue notre pas à pas (pour éviter de faire en pas à pas les 6 boucles de la routine vous pouvez poser un BP juste à la sortie de la routine en 00401139 et appuyez sur Run pour s'y rendre immédiatement). Par la suite on trouve un INC EBX (pour passer au caractère suivant, donc caractère suivant le deuxième 01h) et une mise à 0 de ESI. Et ensuite encore une routine qui est strictement identique à la deuxième rencontrée à une différence près : **CMP ESI,1B2** et non 1D5. Il faut donc que la somme des caractères après le deuxième 01h soit égale à 1B2h. On a 4 caractères après le deuxième séparateur. On fait donc $1B2/4 = 6C$ et il reste 2. Donc on aura trois fois 6C et une fois 6C+2 soit 6E : **4E|4E|4E|4E|4E|4F|01|37|38|39|41|42|43|01|6C|6C|6C|6E**. On fait donc les modifications dans l'éditeur hexa (pensez à réinitialiser OllyDbg et noter à quelle ligne on est). On teste en lançant le .exe. Et là : BINGO ! Ça marche ! Mais il y a un petit détail qui ne va pas ! Notre nom ne ressemble à rien ! :- (On va donc voir comment il trouve le nom. On va donc breaker là où en était (00401155) et continuer à analyser le code. Une fois le programme breaké, on va regarder un peu en dessous pour avoir une vue globale de la suite. Et on remarque :

```
004011F4 > \68 60214000 PUSH due-cm2.00402160 ; /Text = "yvw "
004011F9 . 6A 01 PUSH 1 ; |ControlID = 1
004011FB . FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hWnd
004011FE . E8 5E010000 CALL <JMP.&USER32.SetDlgItemTextA> ; \SetDlgItemTextA
```

Le texte qu'il va afficher : yvw ressemble étrangement à ce qu'on avait aperçu en 402160, là où on avait le résultat du XOR. Cette routine servait donc à trouver le nom. Maintenant il n'y a plus qu'à calculer pour arriver à afficher notre nom. On va essayer d'afficher "Deamon" (le nom fera forcément 6 caractères donc 6 lettres au maximum). On va donc décomposer mon nom "Deamon" en valeur hexa (pour cela soit vous prenez une table ascii qui est disponible dans HexDecCharEditor (l'icône du carré jaune :-)) ou soit vous tapez dans la partie ascii de l'éditeur et sa correspondance en hexa s'affiche).

D|e|a|m|o|n donne donc **44|65|61|6D|6F|6E**. Mais pour trouver les lettres le crackme calcule en faisant un XOR entre les caractères après le premier 01h et ceux avant. On a : **4E|4E|4E|4E|4E|4F|01|X1|X2|X3|X4|X5|X6|01|6C|6C|6C|6E**. Pour avoir 44 en pour la première lettre ("D") il faut que $4E \text{ XOR } X1 = 44$ (la variable X1 correspondant au nombre recherché pour la première lettre, X2 pour la seconde...) donc $X1 = 4E \text{ XOR } 44 = 0A$. Faites le calcul du XOR avec la calculatrice de windows. Donc voici ce que ça donne :

```
X1 = 4E XOR 44 = 0A
X2 = 4E XOR 65 = 2B
X3 = 4E XOR 61 = 2F
X4 = 4E XOR 6D = 23
X5 = 4E XOR 6F = 21
X6 = 4F XOR 6E = 21
```

Ce qui nous donne : **4E|4E|4E|4E|4E|4F|01|0A|2B|2F|23|21|21|01|6C|6C|6C|6E**

Si votre nom est plus court ou plus long vous pouvez déplacer les séparateurs et

augmenter le nombre total de caractères. Mais le nom ne dépassera pas les 15 caractères à cause du **CMP ESI,0F**. Et veillez à mettre le même nombre de lettre avant et après le 1er séparateur. Mais respectez la valeur des sommes des caractères du début et de la fin.

Attention : Si par exemple vous souhaitiez afficher un "N" dans votre nom, cela n'aurait pas marché car "N" = 4E et comme $4E \text{ XOR } 4E = 00h$ cela aurait donc posé un problème (c'est la même chose pour 00h ou 01h). Il aurait donc fallu changer le 4E du départ par exemple en mettant 40 (ce qui fait $4E - 40 = E$) et donc rajouter Eh à un autre pour que la somme soit toujours égal à 1D5h. Exemple : **40|5C|4E|4E|4E|4F|01|0E|39|2F|23|21|21|01|6C|6C|6C|6E**. Ce qui revient au même car $4D + 5C = 4E + 4E$. Et ainsi le "N" en première place aurait donné $X1 = 40 \text{ XOR } 4E = 0E$. Et donc la deuxième lettre si on garde le "e" (65h) donnerait : $X2 = 5C \text{ XOR } 65 = 39$. Bon j'espère avec ça que vous aurez compris.

Récapitulatif :

- Nom du keyfile : due-cm2.dat
 - 18 caractères minimum
 - 2 caractères 01h minimum qui servent de séparateur
 - 00h ou un vide pour indiquer la fin du keyfile
 - La somme des caractères avant le premier 01h fait 1D5h
 - La somme des caractères après le deuxième 01h fait 1B2h
 - Les caractères du nom sont donnés par l'opération XOR entre les caractères d'avant et d'après le premier 01h
- Exemple valide avec le nom "Deamon" : **4E|4E|4E|4E|4E|4F|01|0A|2B|2F|23|21|21|01|6C|6C|6C|6E**

Et avec ça, ça sera tout ! :-)) Le crackme est cracké est avec la manière ! Et en plus vous savez désormais utiliser OllyDbg.

Deamon le 02 et 03 mars 2004

L'assembleur

Introduction

Ce cours vous semblera sans doute long et pénible (ça l'est également pour moi pour l'écrire ;-)) car ce n'est pas forcément facile de comprendre tout ça, mais efforcez vous à bien vous concentrer dessus car c'est important si vous voulez devenir un bon cracker. Attention, il ne s'agit pas non plus d'apprendre tout cela par coeur. Je vous conseille de prendre des notes et de les garder à côté de vous quand vous crackez ou sinon, pour les faignants ;-) , d'imprimer ce cours.

Dans ce cours vous allez apprendre les bases de l'assembleur. L'assembleur c'est le langage de programmation que vous voyez quand vous désassemblez un fichier dans WinDasm. C'est un langage assez proche du langage machine.

Sommaire :

1° Le processeur et les registres

- A) Les registres généraux ou de travail
- B) Les registres d'offset ou pointeur
- C) Les registres de segment
- D) Le registre Flag

2° La Pile et ses instructions

- A) Fonctionnement de la pile
- B) L'instruction PUSH
- C) L'instruction POP

3° Les sauts conditionnels et le CMP

- A) Les sauts conditionnels
- B) L'instruction CMP

4° Les opérations mathématiques

- A) Addition et Soustraction : ADD et SUB
- B) Multiplication : MUL / IMUL
- C) Division : DIV / IDIV
- D) Autres divisions et multiplication : SHR et SHL
- E) L'opposé d'un nombre : NEG

5° Les nombres à virgules et les nombres négatifs

- A) Les nombres à virgules
- B) Les nombres négatifs

6° Les instructions logiques

- A) Et logique : AND
- B) Ou logique inclusif : OR
- C) Ou logique exclusif : XOR
- D) Non logique : NOT
- E) TEST

7° La mémoire et ses instructions

- A) LEA
- B) MOVsx
- C) STOSx

8° Fin du cours !

1° Le processeur et les registres

Les données dont le processeur a besoin sont stockées dans des "cases" qu'on appelle des registres. Il existe plusieurs types de registres et chacun a son utilité.

- - **registres généraux ou de travail**
Ils servent à manipuler des données, à transférer des paramètres lors de l'appel de fonction DOS et à stocker des résultats intermédiaires.
- - **registres d'offset ou pointeur**
Ils contiennent une valeur représentant un offset à combiner avec une adresse de segment
- - **registres de segment**
Ils sont utilisés pour stocker l'adresse de début d'un segment. Il peut s'agir de l'adresse du début des instructions du programme, du début des données ou du début de la pile.
- - **registre de flag**
Il contient des bits qui ont chacun un rôle indicateur.

A) Les registres généraux ou de travail

Les 4 registres généraux les plus utilisés sont : AX , BX , CX et DX.

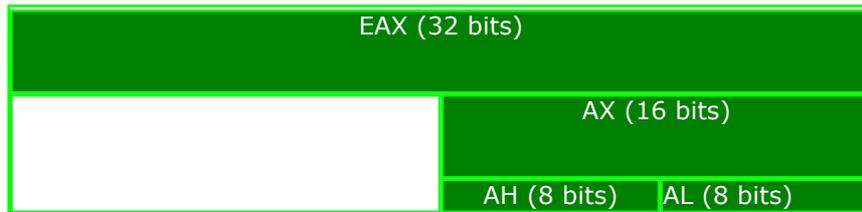
- **AX** -- **accumulateur** -- sert à effectuer des calculs arithmétiques ou à envoyer un paramètre à une interruption
- **BX** -- **registre auxiliaire de base** -- sert à effectuer des calculs arithmétiques ou bien des calculs sur les adresses
- **CX** -- **registre auxiliaire (compteur)** -- sert généralement comme compteur dans des boucles
- **DX** -- **registre auxiliaire de données** -- sert à stocker des données destinées à des fonctions

Ceci est leur utilisation théorique, mais dans la pratique ils peuvent être utilisés à d'autres usages.

Ces registres sont de 16 bits (petit rappel : 8 bits = 1 octet et 1 bit = 0 ou 1) et chacun de ses registres est divisé en 2 parties : L (comme Low pour désigner les bits de poids faible) et H (pour High afin de désigner les bits de poids forts). Par exemple : AX (AL et AH), BX (BL et BH), CX (CL et CH), DX (DL et DH).

Pour obtenir un registre de 32 bits il faut rajouter un "E" (pour Extended, en français "étendu") devant le registre. Par exemple EAX (32 bits) pour AX (16 bits), (EBX pour BX, ECX pour CX, EDX pour DX). Il existe également des EAH ou EAL, mais la partie haute d'un registre 32 bits ne sera pas directement accessible.

Les bits correspondent en fait à leur capacité : 8 bits = 255 (en décimal) au maximum, 16 bits = 65535 au maximum, 32 bits = 4 294 967 295 au maximum.



B) Les registres d'offset ou pointeur

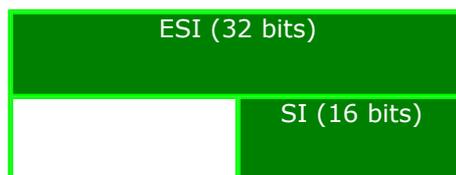
Voici les différents registres :

- **IP -- Pointeur d'instruction** -- est associé au registre de segment CS (CS : IP) pour indiquer la prochaine instruction à exécuter. Ce registre ne pourra jamais être modifié directement ; il sera modifié indirectement par les instructions de saut, par les sous-programmes et par les interruptions
- **SI -- Index de source** -- est principalement utilisé lors d'opérations sur des chaînes de caractères ; il est associé au registre de segment DS.
- **DI -- Index de destination** -- est principalement utilisé lors d'opérations sur des chaînes de caractères ; il est normalement associé au registre de segment DS ; dans le cas de manipulation de chaînes de caractères, il sera associé à ES
- **SP -- Pointeur de pile** -- est associé au registre de segment SS (SS :SP) pour indiquer le dernier élément de la pile.
- **BP -- Pointeur de base** -- est associé au registre de segment SS (SS :BP) pour accéder aux données de la pile lors d'appels de sous-programmes (CALL).

Comme les registres généraux, ces registres peuvent être étendus à 32 bits en ajoutant le "E" : (EIP, ESI, EDI, ESP, EBP). Par contre ils ne possèdent pas de partie 8 bits (il n'y aura pas de EIH ou ESL par exemple).

Les registres SI et DI sont le plus souvent employés pour les instructions de chaîne et pour accéder à des blocs en mémoire.

Ces registres (sauf IP) peuvent apparaître comme des opérandes dans toutes les opérations arithmétiques et logiques sur 16 bits



C) Les registres de segment

Le processeur utilise des adresses pour stocker ou récupérer des données. L'adresse est composée de 2 parties de 32 bits (ou 16 bits selon le mode utilisé). La première est le segment et la deuxième est l'offset. L'adresse est présentée comme cela : segment:offset (par exemple : 0A000h:00000h). Voici les différents segments :

- **CS -- Registre segment de code** -- indique l'adresse du début des instructions d'un programme ou d'une sous-routine
- **DS -- Registre segment de données** -- contient l'adresse du début des données de vos programmes. Si votre programme utilise plusieurs segments de données, cette valeur devra être modifiée durant son exécution
- **SS -- Registre segment de pile** -- pointe sur une zone appelée la pile
- **ES -- Registre segment supplémentaire (Extra segment)** -- est utilisé, par défaut, par certaines instructions de copie de bloc. En dehors de ces

- instructions, le programmeur est libre de l'utiliser comme il l'entend
- **FS** -- **Registre segment supplémentaire** -- Rôle semblable à ES
- **GS** -- **Registre segment supplémentaire** -- Rôle semblable à ES

D) Le registre Flag

Ce registre est un ensemble de 16 bits. Mais cet ensemble ne signifie rien, car les bits sont modifiés un par un. Et la modification d'un seul bit peut changer le comportement du programme. Les bits de cet ensemble sont appelés "indicateurs".

Bit 1 : CF
 Bit 2 : 1
 Bit 3 : PF
 Bit 4 : 0
 Bit 5 : AF
 Bit 6 : 0
 Bit 7 : ZF
 Bit 8 : SF
 Bit 9 : TF
 Bit 10 : IF
 Bit 11 : DF
 Bit 12 : OF
 Bit 13 : IOPL
 Bit 14 : NT
 Bit 15 : 0
 Bit 16 : RF
 Bit 17 : VM

Les bits 1, 5, 12, 13, 14, 15 de ce registre Flag de 16 bits ne sont pas utilisés.

Les instructions arithmétiques, logiques et de comparaison modifient la valeur des indicateurs. Les instructions conditionnelles testent la valeur des indicateurs et agissent en fonction du résultat.

Voici à quoi les principaux indicateurs servent :

CF : *Carry Flag* - **Indicateur de retenue**

A la suite d'une opération, si le résultat est codé sur un nombre supérieur de bits, le bit de trop sera mis dans CF. Plusieurs instructions peuvent modifier son état : CLC pour le mettre à 0, STC à 1 et CMC inverse sa valeur.

PF : *Parity Flag* - **Indicateur de parité**

Egal à 1 si le nombre de bits d'une opérande (paramètre d'une instruction) est pair.

AF : *Auxiliary carry Flag* - **Indicateur de retenue auxiliaire**

Cet indicateur ressemble à CF.

ZF : *Zero Flag* - **Indicateur zéro**

Si le résultat d'une opération est nul (égal à 0) ZF passera à 1. Cet indicateur est souvent utilisé

pour savoir si deux valeurs sont égales (en les soustrayant).

SF : Sign Flag - Indicateur de signe

SF passe à 1 quand le résultat est signé (négatif ou positif).

IF : Interruption Flag - Indicateur d'interruption

Il permet d'enlever la possibilité au processeur de contrôler les interruptions. Si IF = 1 les interruptions sont prises en compte et si IF = 0 elles sont ignorées. L'instruction STI met IF à 1 et CLI le met à 0.

DF : Direction Flag - Indicateur de direction

C'est ce Flag qui donne l'indication sur la manière de déplacer les pointeurs (références) lors des instructions de chaînes (soit positivement, soit négativement). STD le met à 1 et CLD à 0.

OF : Overflow Flag - Indicateur de dépassement

Il permet de trouver et de corriger certaines erreurs produites par des instructions mathématiques. Très utile pour éviter les plantages. Si OF=1 alors nous avons affaire à un Overflow.

Les autres indicateurs ne sont pas importants nous n'allons donc pas nous attarder dessus.

Petit exemple sur ces indicateurs avec une opération en binaire :

| | |
|--|--|
| 0 1 1 0 + 0 1 0 1 ----- 1 0 1 1 | <ul style="list-style-type: none">• ZF=0 (le résultat 1011 n'est pas nul)• SF=1 (le signe du résultat est négatif)• CF=0 (il n'y a pas de retenue) |
|--|--|

2° La Pile et ses instructions

A) Fonctionnement de la pile

La pile ("stack" en anglais) est un emplacement qui sert à stocker de petites données. Cette mémoire temporaire fonctionne, comme dirait Falcon, comme une pile d'assiette : c'est-à-dire que la dernière assiette que vous empilez sera la première à être prise et la première assiette empilée (celle qui est tout en dessous) sera la dernière prise. J'espère avoir été clair dans cette métaphore. Pour prendre un vrai exemple, si je mets la valeur "A" dans la pile puis je mets "B" et après "C". Pour récupérer ces valeurs le processeur va d'abord prendre "C", puis "B" et ensuite "A". Là si vous ne comprenez toujours pas je ne peux plus rien pour vous. :-)

Pour l'API GetDlgItemText, Windows a besoin de ces informations :

```
HWND hDlg, // (1) handle de la boîte de dialogue  
int nIDDlgItem, // (2) identifiant de la boîte de dialogue  
LPTSTR lpString, // (3) pointe vers le buffer pour le texte
```

int nMaxCount // (4) taille maximum de la chaîne

On peut donc mettre avant d'appeler cette fonction :

| | |
|-------------------------|-----------------------------------|
| MOV EDI, [ESP+00000220] | place l'handle de la boîte de |
| dialogue dans EDI | |
| PUSH 00000100 | push (4) taille maxi de la chaîne |
| PUSH 00406130 | push (3) adresse du buffer pour |
| le texte | |
| PUSH 00000405 | push (2) identificateur du |
| contrôle | |
| PUSH EDI | push (1) handle de la boîte de |
| dialogue | |
| CALL GetDlgItemText | call la fonction |

B) L'instruction PUSH

Vous avez pu voir cette instruction au dessus. Push signifie "pousser" en français. C'est avec cette instruction qu'on place une valeur sur la pile. La valeur "poussé" doit être de 16 ou 32 bits (souvent un registre) ou une valeur immédiate.

Par exemple :

```
PUSH AX
PUSH BX
PUSH 560
```

AX est placé en haut de la pile en premier mais ce sera 560 qui en ressortira en premier.

C) L'instruction POP

Mettre des données sur la pile c'est bien, mais le but c'est de les récupérer par la suite. C'est donc à ça que sert l'instruction POP : récupérer les données placées sur la pile.

Par exemple (suite à l'exemple au dessus) :

```
POP CX
POP BX
POP AX
```

Ces instructions mettrons la première instruction du haut de la pile dans CX, c'est à dire 560. Et ensuite BX reviendra dans BX et AX dans AX.

La pile est très utilisée pour retrouver les valeurs intactes de registres. Car le nombre de registres est limité, donc on utilise la pile lorsque l'on n'a plus de place. Mais la pile est plus lente que les registres.

3° Les sauts conditionnels et le CMP

A) Les sauts conditionnels

Dans les différents sauts il y a les inconditionnels (qui sautent sans conditions) comme le JMP (Jump) et ceux qui sont conditionnels (qui sautent donc si la condition est vérifiée). Dans cette dernière "famille" il y a du monde. ;-) Les valeurs des indicateurs sont nécessaires pour la vérification de la condition.

Voici les principaux sauts, sachez que J = Jump if (Saut si) :

| Indicateur | Valeur | Saut | Signification |
|------------|--------|------|--|
| CF | 1 | JB | <i>Below</i> - Inférieur |
| | | JBE | <i>Below or Equal</i> - Inférieur ou égal |
| | | JC | <i>Carry</i> - Retenue |
| | | JNAE | <i>Not Above or Equal</i> - Non supérieur ou égale |
| | 0 | JA | <i>Above</i> - Supérieur |
| | | JAE | <i>Above or Equal</i> - Supérieur ou égal |
| | | JNB | <i>Not Below</i> - Non inférieur |
| | | JNC | <i>Not Carry</i> - Pas de retenue |
| ZF | 1 | JE | <i>Equal</i> - Egal |
| | | JNA | <i>Not Above</i> - Non supérieur |
| | | JZ | <i>Zero</i> - Zero (Saut en cas d'égalité) |
| | 0 | JNBE | <i>Not Below or Equal</i> - Non supérieur ou égal |
| | | JNE | <i>Not Equal</i> - Non égal |
| | | JNZ | <i>Not Zero</i> - Non zéro (saut si différent) |
| PF | 1 | JP | <i>Parity</i> - Pair |
| | | JPE | <i>Parity Even</i> - Pair |
| | 0 | JNP | <i>Not Parity</i> - Non pair |
| | | JPO | <i>Parity Odd</i> - Non pair |
| OF | 1 | JO | <i>Overflow</i> - Dépassement |
| | 0 | JNO | <i>Not Overflow</i> - Pas de dépassement |
| SF | 1 | JS | <i>Signed</i> - Signé |
| | 0 | JNS | <i>Not Signed</i> - Non signé |

Et quelques autres :

JG - JNLE
 Greater (arithmétiquement supérieur) - Not Less or Equal
 (arithmétiquement non inférieur ou égale)
 ZF=0 et SF=OF

JGE - JNL
 Greater or Equal (arithmétiquement supérieur ou égal) - Not Less
 (non inférieur arithmétiquement)
 SF=OF

JL - JNGE
 Less (arithmétiquement inférieur) - Not Greater or Equal
 (arithmétiquement non supérieur ou égal)
 SF (signé)=OF

JLE - JNG
Less or Equal (arithmétiquement inférieur ou égal) - Not Greater (non supérieur arithmétiquement)
ZF=1 ou SF (signé)=OF

B) L'instruction CMP

Cette instruction permet d'effectuer des tests entre des valeurs et de modifier les flags suivant le résultat. Le fonctionnement du CMP est semblable au SUB (que l'on verra plus loin et qui sert pour la soustraction). Il s'utilise de cette façon : CMP AX, BX. Cela permet de vérifier si ces 2 valeurs sont égales car il fait AX-BX, si c'est égal à 0 alors ZF=1 sinon ZF=0.

Le plus souvent il est suivi de ces sauts et leur inverse (en rajoutant un "N" après le "J", par exemple JNE pour JE...) :

JA : plus grand que (nombres non-signés)
JB : plus petit que (nombres non-signés)
JG : plus grand que (nombres signés)
JL : plus petit que (nombres signés)
JE ou JZ : égal à (signé et non-signé)

4° Les opérations mathématiques

A) Addition et Soustraction : ADD et SUB

Ces 2 instructions nécessitent 2 opérands : la source et la destination. Ce qui donnera :
Destination = Source + Destination. Et en assembleur : ADD Destination, Source.
Exemple avec AX = 4 et BX = 6 :

```
ADD AX, BX
```

Après cette opération : AX = 4+6 = 10 et BX = 6

Le fonctionnement de SUB est identique.

Il est impossible d'ajouter ou soustraire un 16 bits avec un 8 bits par exemple. Donc ADD BX, CL sera impossible !

B) Multiplication : MUL / IMUL

L'instruction MUL est utilisé pour les nombres non signés et IMUL pour les signés (petit rappel : les nombres signés peuvent être négatifs contrairement aux non signés, par exemple en 16 bits en non signé les nombres vont de 0 à 65535 et en signé ils vont de -32768 à 32767). Contrairement aux opérations précédentes, la multiplication n'a besoin que d'une seule opérande : la source. La destination et donc le nombre qui devra être multiplié se trouvera toujours et obligatoirement dans AX. La source est un registre.

Et vous pouvez vérifier : $5 \times 384 = 1920$! ;-)

Mais vous me direz : "pourquoi ne pas faire simplement un MUL ?". Et bien car SHL est beaucoup plus rapide à exécuter. Mais pour des nombres qui ont une décomposition trop longue, mieux vaut bien sur utiliser un MUL.

E) L'opposé d'un nombre : NEG

L'instruction la plus simple à mon avis : NEG sert à rendre un nombre positif en négatif et inversement. Il ne demande qu'une seule opérande : la destination. Si $AX = 5$, alors un NEG AX mettra $AX = -5$. Ou si $BX = -14$ alors après le NEG BX, $BX = 14$.

5° Les nombres à virgules et les nombres négatifs

A) Les nombres à virgules

Le problème en assembleur c'est qu'on ne peut pas utilisé directement les nombres à virgules. Nous utiliserons les nombres à virgule fixe. Il s'agit simplement d'effectuer des calculs avec des valeurs assez grandes qui seront ensuite, redivisées pour retrouver un résultat dans un certain intervalle.

Par exemple nous voulons multiplier 20 par 0,25. Comme on ne peut pas mettre 0,25 dans un registre on va le mettre en nombre entier en le multipliant par 256 par exemple. $0.25 \times 256 = 64$. On le multipliera à 20 et ensuite ce résultat étant 256 fois trop grand on le redivisera par 256 :

- MOV AX,64 >> On place $0,25 \times 256$ dans AX >> $AX = 64$
- MOV BX,20 >> On place 20 dans BX >> $BX = 20$
- MUL BX >> On multiplie AX par BX >> $AX = AX \times BX = 64 \times 20 = 1280$
- SHR AX,8 >> On divise AX par 256 (2^8) >> $AX = AX / 256 = 1280 / 256 = 5$

Le résultat de $20 \times 0,25$ est donc bien égal à 5.

Faites bien attention à ne pas effectuer de dépassement, surtout avec les nombres signés ou les grands nombres. En utilisant SHR ou SHL, plus l'opérande est grande et plus le résultat sera précis, car il y aura plus de décimales disponibles.

B) Les nombres négatifs

Un autre problème, c'est qu'en décimal pour un nombre négatif on place un "-" devant le nombre, mais en binaire on ne peut mettre que des 0 ou des 1. Pour pouvoir mettre un nombre en négatif on utilise la méthode du complément à 2. Voici en quoi cela consiste :

- Convertir le nombre en binaire s'il ne l'est pas déjà
- Inverser les bits du nombre (inverser les 0 en 1 et 1 en 0)
- Ajouter 1 au nombre obtenu

Voici un exemple avec le nombre décimal 5 au format 8 bits (octet) :

- On le convertit en binaire : $5(d) = 00000101(b)$
- On inverse : $00000101 \rightarrow 11111010$ (ici le format, 8 bits, est important car en 4 bits par exemple cela donnera : 1010)
- On ajoute 1 : $11111010 + 00000001 = 11111011$

-5 est donc égal à 11111011 en binaire.

Pour que le nombre soit négatif il faut qu'il soit signé. En 8 bits en non signé les nombres vont de 0 à 255 et en signé ils vont de -128 à 127.

Quand le nombre est signé et en 8 bits, on peut savoir s'il est négatif en regardant le 7ème bit (en partant bien sûr de la droite). S'il est égal à 1 alors le nombre est négatif. Pour les nombres à 4 bits c'est le 4ème qui compte.

Pour transformer -5 en hexadécimal, on décompose le nombre binaire en demi-octet (11111011 en 1111 et 1011). $1111b = 15d = Fh$ et $1011b = 11d = Ah$. Donc $-5d = FAh$. (Petit rappel : d = décimal, b = binaire, h = hexadécimal, pour ceux qui n'aurait pas encore compris).

6° Les instructions logiques

Les opérations de ces instructions se font bit à bit. Et les opérandes et emplacements mémoires qu'elles utilisent sont en 8 ou 16 bits.

Petit rappel : pour convertir de tête un nombre binaire en décimal : chaque rang d'un nombre binaire est un multiple de 2.

| | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Ainsi par exemple 10110110 donnera $128+32+16+4+2 = 182$:

| | | | | | | | |
|-----|----|----|----|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

A) Et logique : AND

AND (ET) effectue un ET LOGIQUE sur 2 opérandes et le résultat se retrouvera dans la première opérande. Le ET LOGIQUE donne le résultat 1 uniquement quand les 2 opérandes sont 1, sinon il met 0.

| opérande 1 | opérande 2 | AND |
|------------|------------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Exemple avec AX = 15 et BX = 26 :

AND AX,BX

Résultat :

```

AND   0000 1111 (AX=15)
      0001 1010 (BX=26)
      -----
      0000 1010 (AX = 10)
  
```

B) Ou logique inclusif : OR

OR effectue un OU LOGIQUE INCLUSIF sur 2 opérandes et le résultat se retrouvera dans la première opérande. Le OU LOGIQUE INCLUSIF donne le résultat 0 uniquement quand les 2 opérandes sont 0, sinon il met 1.

| opérande 1 | opérande 2 | OR |
|------------|------------|----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Exemple avec AX = 15 et BX = 26 :

OR AX,BX

Résultat :

```
OR      0000 1111 (AX=15)
        0001 1010 (BX=26)
        -----
        0001 1111 (AX = 31)
```

C) Ou logique exclusif : XOR

XOR effectue un OU LOGIQUE EXCLUSIF sur 2 opérandes et le résultat se retrouvera dans la première opérande. Le OU LOGIQUE EXCLUSIF donne le résultat 1 quand les 2 opérandes sont différentes, sinon il met 0.

| opérande 1 | opérande 2 | XOR |
|------------|------------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Exemple avec AX = 15 et BX = 26 :

XOR AX,BX

Résultat :

```
XOR      0000 1111 (AX=15)
        0001 1010 (BX=26)
        -----
        0001 0101 (AX = 21)
```

D) Non logique : NOT

NOT effectue un NON LOGIQUE sur une opérande et le résultat se retrouvera dans l'opérande. Le NON LOGIQUE inverse la valeur de chaque bit.

| opérande | NOT |
|----------|-----|
| 0 | 1 |
| 1 | 0 |

Exemple avec AX = 15 :

NOT AX

Résultat :

```
NOT    0000 1111 (AX=15)
       -----
       1111 0000 (AX = -16 en signé et 240 en non signé)
```

E) TEST

Cette instruction teste la valeur d'un ou plusieurs bits en effectuant un ET LOGIQUE sur les opérandes. Le résultat ne modifie pas les opérandes mais les indicateurs.

Exemple :

TEST AX,1 >> Effectue un ET LOGIQUE sur le premier bit de AX avec 1, si il est égal à 1, ZF passera à 1 et inversement (si 0 alors ZF=0).

7° La mémoire et ses instructions

A) LEA

Pour placer par exemple la valeur de l'offset de MESSAGE dans SI il suffit de mettre : MOV SI, Offset-Message. Mais on ne peut pas procéder de cette façon pour les registres DS, ES, FS, GS car ceux ci n'acceptent pas les valeurs numériques, ils n'acceptent que les registres. Il faudra donc passer par : MOV AX,Seg-Message puis MOV DS,AX.

Il existe une autre instruction, LEA, qui permet de placer l'offset dans un registre mais en plus court. Par exemple : LEA SI,MESSAGE placera dans SI l'offset de Message. L'utilisation de LEA à la place de MOV rend le code plus clair et plus compact.

B) MOVsx

Pour déplacer des blocs entiers de mémoires il faut utiliser les instructions : MOVSB, MOVSW ou MOVSD selon le nombre de bits à déplacer.

- MOVSB : (B) = Byte (Octet : 8 bits) (Attention !! En anglais "byte"= un octet et "bit"= un bit)
- MOVSW : (W) = Word (Mot : 16 bits)
- MOVSD : (D) = DWord (Double Mot : 32 bits)

Si on veut déplacer 1000 octets (bytes) en utilisant la commande MOVSB il faudra qu'elle se répète 1000 fois. Donc l'instruction REP est utilisée comme une boucle. Il faut placer auparavant le nombre de boucles à effectuer dans le registre de compteur (CX) :

```
MOV CX,1000  >> Nombre de boucles à effectuer dans le compteur CX
REP MOVSB    >> Déplace un octet
```

Pour aller plus vite on peut les déplacer par Mot (Word) :

```
MOV CX,500   >> 1000 bytes = 500 words
REP MOVSW    >> Déplace un mot
```

Ou alors en Double Mot (DWord) :

```
MOV CX,250    >> 1000 bytes = 500 words = 250 dwords
REP MOVSD     >> Déplace un Double Mot
```

A chaque MOV_Sx, DI augmente de 1,2 ou 4 bytes

C) STOS_x

Pour garder des données ou les placer dans un emplacement de la mémoire on utilise les instructions STOSB, STOSW ou STOSD.

Pareil que les MOV_Sx, ces instructions servent respectivement à stocker un byte, un word et un dword. Par contre ils utilisent AL, AX (EAX 32bits) comme valeur à stocker. La valeur de AL,AX,EAX sera stockée dans ES:DI. A chaque STOS_x, DI est augmenté de 1,2 ou 4 bytes.

Exemple :

```
MOV CX,10    >> nombre de répétitions placés dans CX, ici 10
MOV AX,0     >> valeur à déplacer, ici 0
REP STOSB   >> stocker AX dans ES:DI
```

Nous aurons donc 10 bytes de 0 dans ES:DI.

8° Fin du cours !

Voilà c'est la fin du cours ! "Ouf" diront certains, "Oh quel dommage" diront les hypocrites ! ;-)
Je félicite ceux qui ont réussi à lire tout ça ! ;-)
J'espère que cela vous servira à mieux comprendre en crackant. Avec ça vous devriez vous sentir plus à l'aise (enfin si vous avez compris mon cours).

J'espère avoir été le plus compréhensible possible. Mais si vous ne comprenez pas quelque chose à propos de ce cours demandez [sur le forum](#) plutôt que de [m'envoyer un mail](#) car je lis de toute façon tous les posts du forum et d'autres personnes que moi pourront vous aider.

Je voulais féliciter le très bon cours de Falcon sur l'assembleur qui m'a permis de découvrir l'assembleur. D'ailleurs j'ai repris un peu son plan, car je n'ai pas réussi à en faire un bien, et copier seulement 2-3 phrases mais c'est tout. Mais c'est bien moi qui ai écrit tout ça, en 6-8 heures (et oui c'est très long à écrire tout ça) !

Pendant que je suis là ;-), je remercie également tout ceux qui viennent sur mon site, ceux qui m'envoient des mails de sympathie et ceux qui participent plus ou moins activement [sur le forum](#).

Deamon le 15-16 juillet 2003

F.A.Q.

Voici les questions que les gens me posent le plus souvent, alors avant de demander de l'aide pour un problème regardez si la réponse que vous cherchez n'est pas ci-dessous :

Q : Quand je désassemble mon fichier avec WinDasm je vois plein de signes bizarres à la place des lignes de code.

R : Il faut changer la police de WinDasm. Pour cela cliquez sur "Disassembler" -> "Font..." -> "Select Font" et choisissez Courier New, validez, puis cliquez de nouveau sur "Disassembler" -> "Font..." -> "Save Default Font" afin de garder cette police lors d'une prochaine utilisation.

Q : Pourquoi ne se passe-t-il rien dans WinDasm lorsque j'ouvre un fichier ?

R : Le fichier à désassembler doit être trop loin de la racine du disque dur, si votre programme est enfoui dans 6 ou 7 dossiers et sous-dossiers WinDasm n'arrivera pas à l'ouvrir alors rapprochez le plus possible de la racine (c:/ par exemple).

Q : Lorsque je veux enregistrer les modifications effectuées dans mon éditeur hexadécimal il met « Writing Failed ! » ou un message de ce genre. Que faire ?

R : C'est parce que le programme à modifier est soit déjà ouvert ou soit désassemblé dans WinDasm, dans ce cas enregistrez le programme modifié sous un autre nom.

Q : Dans un programme désassemblé je ne trouve aucun Strings Data Reference, comment faire pour le cracker ?

R : Le programme est sans doute crypté ou compressé. Regardez si c'est le cas avec StudPe ou Peid par exemple.

Q : J'ai lu tous tes cours mais je n'arrive pas à cracker le dernier Universalis.

R : Ce n'est pas en ayant lu tous mes cours que vous pouvez cracker tout et n'importe quoi, ce n'est pas la peine de s'attaquer aux gros programmes ou jeux car vous allez sûrement vous casser les dents dessus. Commencez plutôt avec des programmes plus modestes.

Q : Où puis-je trouver un crack pour tel ou tel logiciel ?

R : Vous pouvez chercher sur des moteurs de recherches de cracks, regardez la page Liens de ce guide.

Liens utiles

Tout d'abord deux sites incontournables ;-):

- <http://www.demoncrack.fr.st/> : vous pourrez y retrouver les cours de ce Guide sur ce site mais également d'autres tuts.
- <http://www.forumcrack.fr.st/> : le forum du site, où règne une excellente ambiance, qui vous aidera lorsque vous rencontrerez des problèmes, vous pourrez également y demander des conseils, recherchez un crack...etc.

Cours de cracking :

- <http://www.k-509.host.sk/fprod.php> : site impressionnant recueillant près de 1500 cours !!!
- <http://www.rif.fr.fm/> : des cours classés par catégorie et difficulté, et quelques crackmes.

Cracks et serials :

- <http://astalavista.box.sk> : moteur de recherche.
- <http://www.astalavista.com/> : regroupement de moteur de recherche.
- <http://www.keygen.us/> : des keygens comme son nom l'indique.
- <http://www.serials.ws/> : des serials pour applications ou jeux.

Cracks, no-cd pour les jeux :

- <http://www.megagames.com>
- <http://www.gamecopyworld.com>
- <http://www.gjeux.com/cracks-accept.xhtml>

Crackmes, Keygenmes, ReverseMes... :

- <http://www.reversemes.de> : pour s'entraîner au Reverse Engineering avec des ReverseMe avec leur solution.
- <http://www.crackmes.de/> : des centaines de CrackMes avec solutions.
- <http://www.cryptokg.cjb.net/> : quelques KeygenMes.
- <http://opencrackmes.crackmes.de/> : un peu de tout.

Assembleur :

- <http://perso.magic.fr/sleon/prog/DIBUltra/asm.htm> : les instructions utilisées en Assembleur.
- <http://assembly.ifrance.com/assembly/> : de bons cours sur l'Assembleur et le Reverse Engineering.

Reverse Engineering :

- <http://community.reverse-engineering.net/>
ou <http://board.anticrack.de/> : forum anglais sur le RE.
- <http://www.fravia.anticrack.de/> : site en anglais
- <http://membres.lycos.fr/w32assembly/> : des tuts sur des fonctions ASM.

Des outils pour bien cracker :

- <http://www.exetools.com/> : de tout : unpackers, debuggers, désassembleurs...
- <http://protools.cjb.net/> : beaucoup de programmes pour le cracking.
- <http://bestcracks.fateback.com/toolz.htm> : des outils...

Voilà c'est ici que se termine ce Guide. Merci de l'avoir lu et j'espère qu'il vous aura appris quelque chose et qu'il vous aura plu.

Si vous souhaitez faire des commentaires, compliments ou critiques n'hésitez pas à en faire part soit en m'envoyant un mail à daemon.crack@netcourrier.com ou alors sur le forum www.forumcrack.fr.st.

Je tenais à remercier tout d'abord les nombreux gens qui m'envoient des mails pour faire des compliments à propos de mon site ou mes cours, puis je remercie également les membres du forum pour leur gentillesse et leur sympathie qui font du forum un endroit où règne bonne humeur et bonne ambiance et enfin tous ceux qui m'ont aidé, soutenu et encouragé à continuer.

Merci à vous tous !

Daemon, le 5-6 août 2004