By Horgh

Manual Unpacking d'Anti007 (NSAnti)

Version 1.1
14/04/2010

Table des matières

1. Propos liminaires

a) Généralités

Dans ce cours, nous allons voir la protection Anti007 2.5 sur un UnpackMe, qui est téléchargeable ici :
http://www.tuts4you.com/download.php?view.1580.

Certains antivirus détectent cet UnpackMe comme étant infecté avec le trojan *Packer.Malware.NSAnti.K*. Ceci est parfaitement normal, car Teddy Rogers a pour cet Unpackme recopié la routine du packer Anti007 (aussi appelé NSAnti), qui est le packer de ce trojan.

Voici donc les informations que j'ai recueillies sur le packer [1] :

Ce packer est assez dur à identifier et il gère la possibilité de packer/protéger plusieurs fichiers. Il ne fait pas que compresser le code, il le protège aussi avec différentes méthodes :

Les données sont cryptées dans 3 sections avec des noms aléatoires.
Les imports utilisés par le packer sont résolus d'une façon non standart.

Il détecte également les machines virtuelles et crashe si la détection est positive. Il génère un très grand nombre d'exceptions pour gêner le debugging. Il utilise du code polymorphique.
Le code est rempli d'instructions inutiles (*junk code*), de sauts très longs et inutiles qui ralentissent le debugging. Il reconstitue le code via des opérations de type add/sub/xor, et enfin en insérant des junks calls qui renvoient à des fonctions inexistantes.

Le code polymorphique change très souvent dans le but d'éviter la détection des fichiers packés/protégés par les anti-virus (le code polymorphique est sensé éviter l'émulation/détection, et les antidebugging tricks ne peuvent pas véritablement empêcher le manual debugging/tracing du packer).

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

b) Prérequis

J'ai refait le tuto pour qu'il devienne accessible au plus grand nombre. Il ne vous faudra donc que connaître le fonctionnement d'OllyDbg, et bien évidemment avoir un peu de connaissances en assembleur, sinon vous n'irez pas très loin.

Voici une introduction à OllyDbg par Crisanar :
http://deamonftp.free.fr/deamoncrack/Tuts/Crisanar/introOlly.htm

Pour l'assembleur, j'ai sélectionné deux cours très bien faits, accessibles aux débutants mais proposant tout de même une approche assez complète.
Cours de Deamon : http://deamonftp.free.fr/deamoncrack/index0.htm
Cours de Falcon : http://xtx.free.fr/liens/tut/Assembleur%20par%20Falcon/Assembleur.html
Normalement vous n'avez besoin de rien de plus.

c) Outils utilisés

- L'unpackme de 188 ko
- Peid 0.95 et RDG Packer Detector 0.6.6
- Un débuggeur/désassembleur : OllyDbg (1.10 ou 2.0)
- ImpRec 1.7c
- Un cerveau :)

Les anciennes versions des logiciels proposés marchent également. Pour Olly, je l'ai fait avec la version 1.10. Tout ces logiciels sont trouvables rapidement dans Google.

L'unpacking va se dérouler en deux temps : on va tout d'abord rechercher l'OEP avec Olly, et ensuite on va dumper l'UnpackMe et reconstruire son IAT avec ImportRec.

2) Introduction aux notions du Manual Unpacking

Tout d'abord un peu de théorie sur les packers (désolé pour ceux qui connaissent déjà :) :
Le logiciel de compression greffe au logiciel cible ce que l'on appelle un *loader*. Le but de ce loader est de modifier l'*entrypoint* du programme (EP), c'est à dire le début du code, pour que ce soit le loader qui se lance en premier. Le loader lance ensuite le programme, après l'avoir décompresser en mémoire (dans la RAM).

Le fait de compresser un programme permet de réduire considérablement sa taille, ce qui permettait de les mettre en téléchargement sur Internet (oui, c'est très vieux !). Maintenant ils font beaucoup office de protection contre les méchants crackers qui s'attaquent aux sharewares :). Certes ils découragent les débutants, mais un expert n'en est que seulement ralenti. Et comme nous sommes persévérants, nous allons nous y attaquer dès maintenant…

Comme il faut bien un aspect négatif, et même si celui ci est très minime, je vous précise que le loader va légérement ralentir l'exécution du programme car il se lance avant lui. Cela ne change malgré tout absolument rien car la différence n'est pas perçue par l'utilisateur.

Les logiciels qui permettent la compression d'un programme sont très nombreux, et ils ont chacun leurs spécificités. Il va ainsi y avoir les programmes qui ne font que "compresser" le code

du programme, tels UPX ou AsPack. Ceux ci ne sont pas très difficiles à unpacker, car ils ne rajoutent pas de protection, et que la procédure est toujours la même.

Mais il y en a d'autres, beaucoup plus coriaces. Et ceux-là cachent un véritable arsenal : il y a tout d'abord les protections anti-debugging qui peuvent par exemple fausser le calcul d'un sérial, couper le programme si le debugger est detecté, ou interdire la pause de breakpoints. Nous trouvons aussi le cryptage du code, le remplacement des noms de sections par ceux d'autres packers, l'ajout de fausses signatures, des protections anti-dumping ou anti-désassemblage… On n'a que l'embarras du choix face à l'ingéniosité des programmeurs !

Voici un résumé de l'action du packer sur le programme fait par Krom :

Avant la compression :
• Début du code.
• Fin du code.

Après la compression :
• Lancement du Loader à la place du début du code.
• Décompression en RAM du programme.
• Début du code.
• Fin du code.

Voilà, nous arrivons à la fin de l'introduction. Je sais que c'est peut-être un peu long, voire même désagréable, mais c'est un passage obligé pour comprendre ce que vous faites, et pour savoir à quoi vous vous attaquez. Passons maintenant à la pratique.

3) Recherche de l'OEP

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

On va donc tout d'abord passer le fichier à unpacker sous RDG ou Peid (*Fig. 1)*, et cela doit devenir systématique à chaque fois que vous débugger un programme.
Ouvrez Peid et sélectionnez le programme à analyser en cliquant sur [...] en haut a droite.
On s'aperçoit alors que le packer utilisé sur notre fichier n'est détecté par aucun des deux. Ne vous en faites pas, ce n'est pas grave pour la suite :).

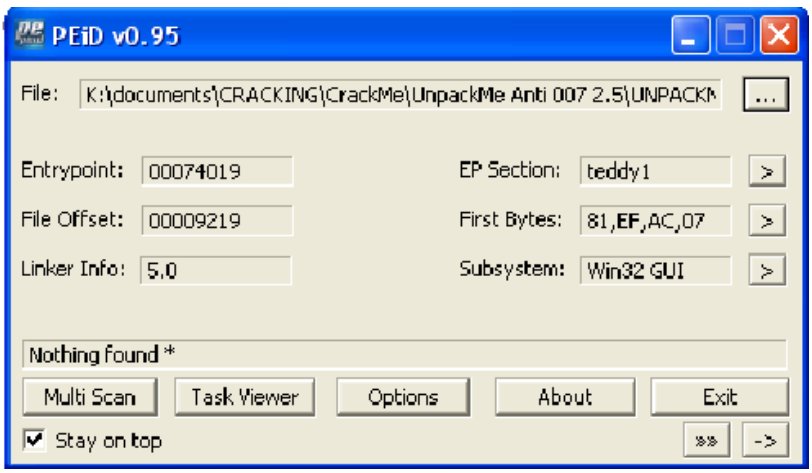

Figure 1 : Scan sous Peid

Petite parenthèse sur l'usage de Peid :

Lorsque le programme est packé et que la signature est connue, on voit le nom du packer en bas. Vous pourrez ainsi avoir " UPX 0.89.6 - 1.02 / 1.05 -1.24 (Delphi) stub -> Markus et Laszlo " ou bien encore "ASPack 2.12 -> Alexey Solodovnikov". Si la signature est contenue dans la base externe (le UserDB.txt), vous aurez une étoile après comme ceci "Anti007 2.5*".

Si aucune protection n'a été détectée, il nous affichera soit "Nothing found*", soit le langage de programmation utilisé comme "Borland Delphi 2.0" ou "Microsoft C++ 6.0".
Nous pouvons aussi avoir plus de détails sur le cryptage en appuyant sur [»»] en bas à droite.
Pour voir les sections d'un programme, cliquez sur la flèche à côté de EP Section.

Quand on essaie de l'ouvrir avec OllyDBG, un message d'erreur apparaît (Fig. 2) nous disant que l'EP est placé en dehors du code, c'est à dire hors de la section qui contient le code du programme. En général, la section "Code" est la première, hors ici l'EP est dans la deuxième section (regardez l'adresse de l'EP en haut à gauche), ce qui nous laisse supposer que c'est la section ou se situe le loader. Nous voyons aussi qu'il ne trouve aucune référence dans : Search for ->> All Referenced text strings. Donc là même si vous ne l'aviez pas analysé avec Peid, aucun doute qu'il est bien packé.

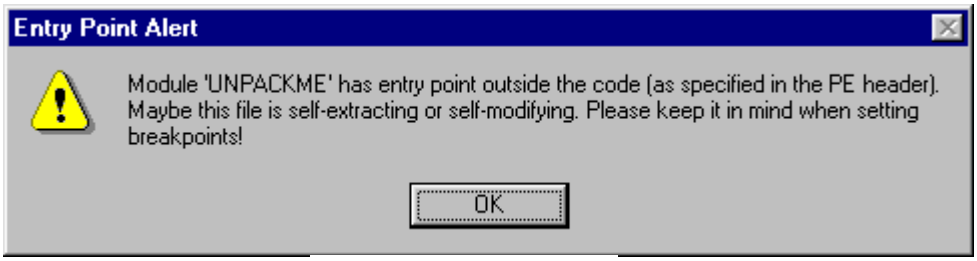

Figure 2 : Message d'alerte

Nous allons utiliser ici la méthode dite de *l'ESP Trick*[2], qui est très utilisée en Manual Unpacking. Gardez la en mémoire, car elle fonctionne sur énormément de packers (en particulier ceux qui commençent par un PUSHAD).
Bon alors, maintenant que Olly est lancé gardez un œil sur le registre ESP (en haut à droite). Vous faites F7, jusqu'à ce que la valeur de ESP change (Fig. 3). F7 veut dire "Step into", c'est à dire "rentre dedans". Olly va alors rentrer dans chaque instruction si il le peut. On utilise en général cette commande pour rentrer dans les calls, mais ici c'est par mesure de sécurité, pour ne pas manquer une instruction par un debugging trop rapide. La valeur de ESP est modifiée au PUSH en 0047402C (ESP change de couleur). Vous stoppez donc ici.

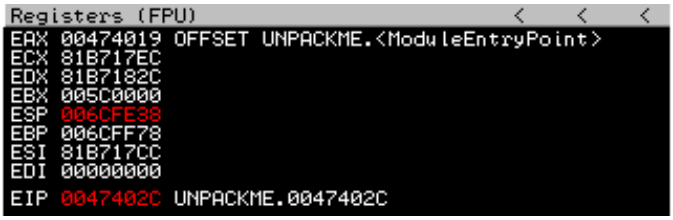

Figure 3 : Changement de la valeur d'ESP

Ensuite, faites click droit sur ESP, Follow in dump. Vous regardez alors dans le cadre en bas à gauche. Click droit sur le premier dword (les quatre premiers 00), Breakpoint => harware on access => dword (Fig. 4). Vous venez de placer un breakpoint matériel sur l'accès à ce dword.

Le programme va ainsi s'arrêter sur l'instruction qui aura tenté d'accéder à cette valeur. A noter que les BP matériels ne sont disponibles qu'à partir de Win2K, et qu'ils ne ralentissent pas la vitesse d'exécution du programme.

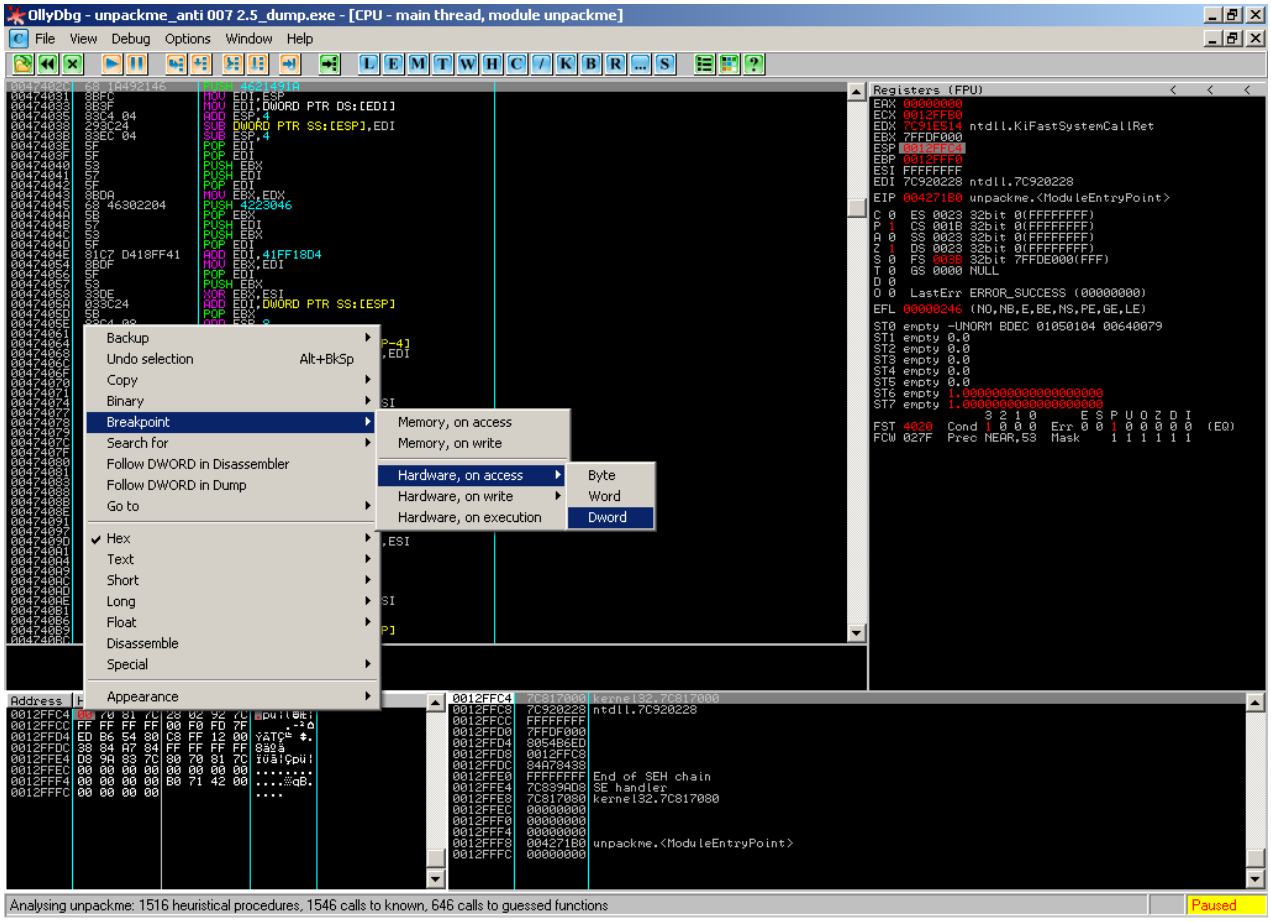

Figure 4 : Placement du Hardware Breakpoint

Bien, maintenant que vous avez posé votre breakpoint faites F9. F9 permet de lancer le programme depuis OllyDbg. Vous devrez le faire plusieurs fois car Olly va breaker sur des lignes de code qui ne nous intéressent pas. Vous arrêterez de faire F9 lorsque vous arrivez au PUSH EBP en 004271B0 (Fig. 5). Notez l'adresse de ce push car c'est l'OEP.

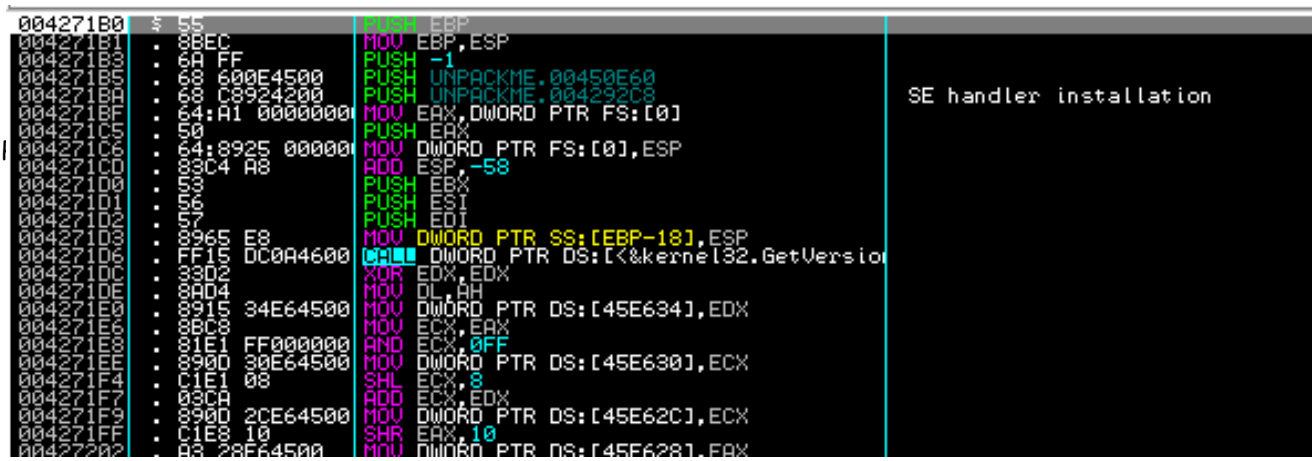

Figure 5 : OEP de l'UnpackMe

Comment sait on que c'est ici l'OEP, me direz vous ? Et bien voilà : en fait le PUSH EBP est la signature de plusieurs compilateurs très répandus. Ils ont chacun leurs propres signatures, et si on en reconnait une on sait ainsi qu'on est à l'OEP.

Le PUSH EBP est commun au Borland Delphi 4/5, 6/7 et au Visual C++ 5.0 / 6.0 (peut-être aussi à quelques autres).

Hors ici, on cherche à savoir ce compilateur pour connaître la signature de son OEP. On lance alors notre UnpackMe, puis RDG Packer Detector. Cliquez sur le "5" en bas, "Compiler detector", sélectionnez l'Unpackme, et faites "Detectar". Et voilà le résultat : Microsoft Visual C++ 5.0 :).

Hors, la signature de Visual C++ 5.0 est celle ci :

```
PUSH EBP
MOV EBP,ESP
PUSH -1
PUSH XXXUNPAC.00450E60 // Les adresses changent selon l'exe
PUSH XXXUNPAC.004292C8 // Idem
MOV EAX,DWORD PTR FS:[0]
PUSH EAX
MOV DWORD PTR FS:[0],ESP
ADD ESP,-58
PUSH EBX
PUSH ESI
PUSH EDI
MOV DWORD PTR SS:[EBP-18],ESP
```

Ce qui correspond parfaitement à ce que nous avons trouvé :). Voilà, vous savez maintenant pourquoi l'OEP correspond à notre PUSH EBP.

Je vous conseille fortement de vous procurer un fichier répertoriant ces signatures, et d'apprendre à repérer les compilateurs quand on ne peut pas les détecter (ex: en regardant les DLL utilisées).

4) Reconstruction de l'IAT et dump du programme

Le *dump*, c'est un enregistrement de tout ou partie des adresses mémoires contenues dans la RAM. Ainsi, quand le loader atteint l'OEP, le programme est entièrement décompressé dans la RAM. Nous allons donc le copier pour l'avoir décompressé :).
A noter qu'un dump est en général plus volumineux que le fichier packé. Vous verrez, bien souvent la différence de taille est conséquente.

L'*IAT*, c'est la table des imports d'un programme, c'est à dire une table qui répertorie les DLL que celui-ci utilise, avec le détail des fonctions importées et leurs adresses.
En réalité, l'IAT compte deux tableaux, et non pas un seul. Ces deux tableaux contiennent le nom des fonctions importées, mais ils ont chacun leur utilité.

Lorsque le programme s'exécute, il va regarder les noms des fonctions importées dans le 1er tableau et en déduire (grâce à certaines API) leurs adresses réelles. Il va alors stocker ces fonctions dans le 2ème tableau, en écrasant les données qui s'y trouvait précedemment. En général lorsqu'un fichier est packé, le packer supprime le premier tableau pour gagner de la place.

Il faut reconstruire l'IAT d'un programme car notre dump est une copie de la mémoire après l'exécution de celui-ci, ce qui fait que le 2ème tableau contient les adresses réelles des fonctions et non pas leur nom comme à l'origine. Ainsi, lorsque l'on va vouloir lancer notre dump, le programme va chercher le nom des fonctions, mais il ne trouvera que des adresses de fonctions, et un plantage du programme s'en suivra inévitablement. C'est pour cela qu'on doit remettre le nom des fonctions importées dans le deuxième tableau.

Lancez d'abord le fichier à unpacker, puis ensuite ImpRec.
Faites Attach to an active process puis sélectionnez le fichier packé.

Petite parenthèse pour le calcul de l'OEP :
Adresse de l'OEP - Image Base.
Ici, on a donc Adresse du Push EBP - Image Base, c'est à dire 4271B0 – 400000 = **271B0**.
Pour changer l'OEP dans ImpRec, regardez en bas à gauche. Vous voyez normalement un encart avec comme adresse 00074019. Remplacez cette adresse par celle que nous avons précedemment trouvée, c'est à dire 000271B0.

On clique sur IAT Autosearch et nous obtenons une messagebox qui nous avertit qu'ImpRec a trouvé quelque chose. On clique alors sur Get Imports, et nous obtenons une liste de fonctions, qui répertorie celles utilisées par le programme. Elles ont toutes l'air valides mais par précaution on clique quand même sur Show Invalid (Fig. 6).
Il n'y a rien, on peut donc continuer l'esprit tranquille :)

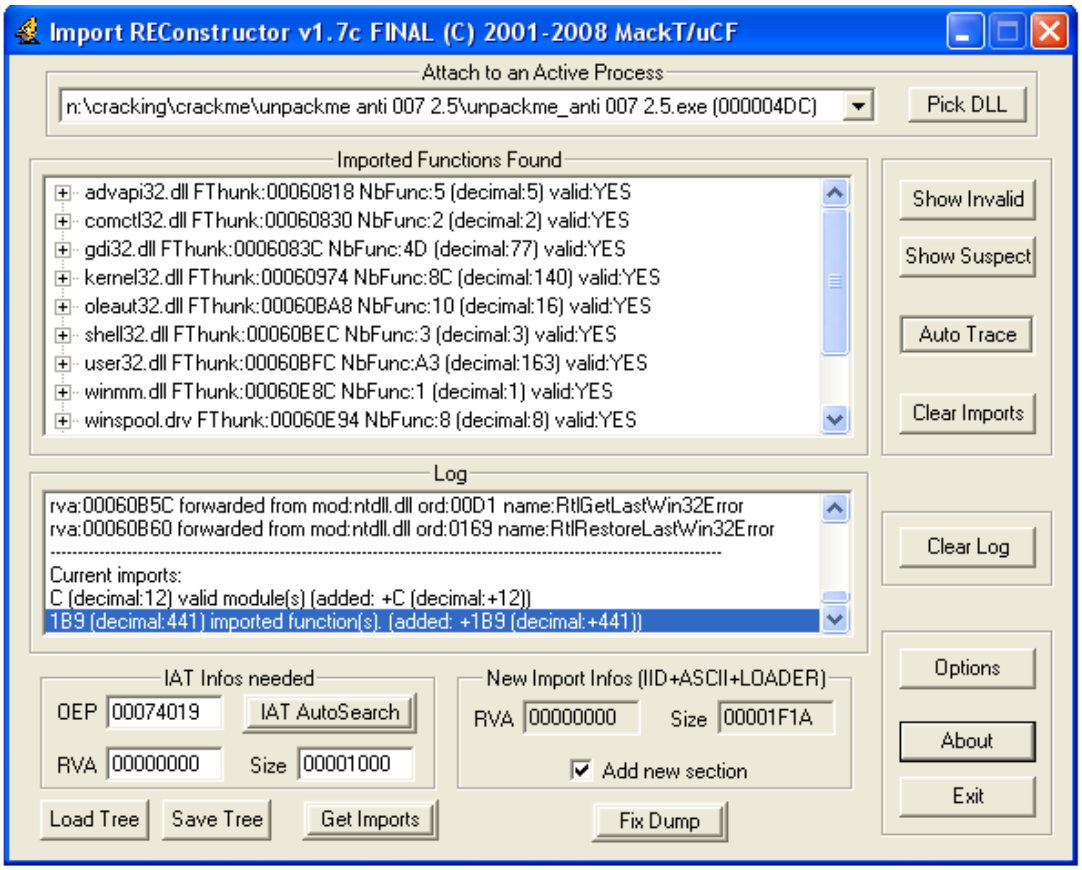

Figure 6 : Reconstruction des imports

Faites ensuite un click droit là où les fonctions ont été importées, Advanced Commands, Select Code Section(s) (Fig.7).

Vous obtenez alors ça :

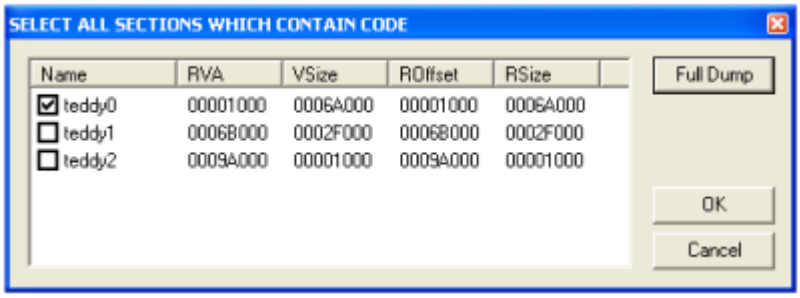

Figure 7 : Dump du programme

Laissez teddy0, qui est la section qui contient l'OEP (1000 < 271B0 < 6B000), et faites Full Dump. Enregistrez le dump, et une messagebox nous prévient que ça a marché.

Vous faites ensuite Fix Dump sur le dump précédent pour finaliser la reconstruction de son IAT. ImpRec va en rajouter au programme une nouvelle section (MackT) qui contiendra notre nouvel IAT valide. C'est bon, vous pouvez maintenant tout fermer, l'unpacking est fini.

On vérifie dans Peid (Fig.8) :

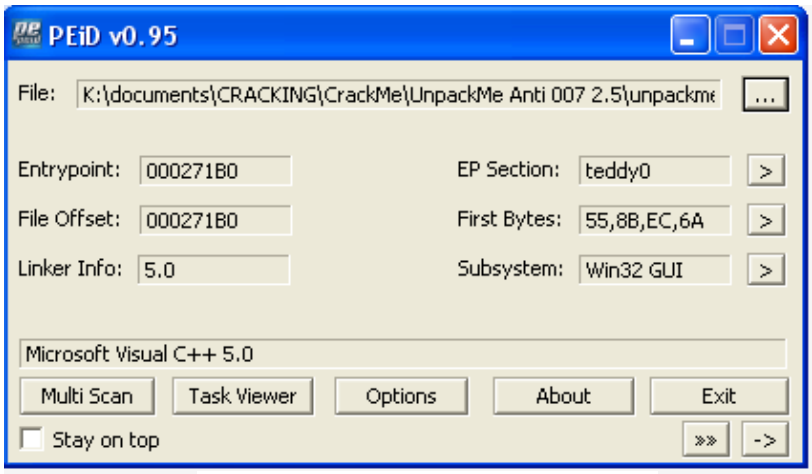

Figure 8 : Vérification dans Peid

5) Conclusion et remerciements

J'espère que ce tuto vous a plu, voire appris des choses. Nous y avons revu les bases du Manual Unpacking, en utilisant la technique de l'ESP Trick pour unpacker ce packer de malware.

Je tiens à remercier :

Shub-Nigurrath / ARTeam, pour le template Word
Teddy Rogers / SND, pour son Unpackme, et pour le site Tuts4you
Joker_Italy, grâce à qui j'ai découvert cette méthode
Krom, grâce à qui j'ai pu débuter
Et toi, pour avoir pris le temps de me lire !

6) Références

[1] : http://www.bitdefender.com/VIRUS-1000214-en--Packer.Malware.NSAnti.J.html
[2] : Pour plus de détails sur le fonctionnement de l'ESP Trick, allez voir le tuto #3 de Gabri3l / ARTeam "Beginner Tutorial: Unpacking and Patching".

7) Historique

- Version 1.0 first public release
- Version 1.1 : - Nouvelle mise en page
  - Description du packer
  - Compléments sur l'ESP Trick
  - Ajout d'un complément d'informations sur la recherche de l'OEP