Soluce defi_mars_138
par Jericho

le 22/04/2007

Difficulté : facile

Outils Nécessaires :
- Stud_PE : pour l'analyse primaire
- SmartCheck : debugger VB
- P32Dasm : P-code Decompiler

Préambule :

C'est la première fois que je m'attaque à un crackme en p-code. Jusqu'à présent, il était d'usage d'éviter comme la peste les crackme/keygenme en VB surtout quand ils sont compilé en p-code à cause de leur code affreux dans un debugger et leur traçage interminable.

Comme mars nous propose un "petit" crackme de lvl0.5, je me suis dit qu'on pourrait s'y intéresser malgré mes maigres connaissances en cracking.

Après une petite recherche préalable sur le Net, j'ai appris que ce genre de code était traité dans une machine virtuelle; en ce qui concerne le VB, il y a une dll qui s'occupe de ça. Voici un extrait d'un tuto du "CrackMe 02 de Chronos" *par Kharneth*

> Un programme compilé en P-code utilise une machine virtuelle (*MSVBVM60.DLL ou 50 suivant la version de VB*) qui charge les opcodes pour les interpréter à l'exécution. Chaque fonction utilisée dans un programme VB possède un numéro identifiant. Lorsque le programme est compilé, c'est ce numéro qui est écrit à la place des instructions ASM classiques. Puis quand on l'exécute, le moteur est chargé en mémoire et lit la liste d'opcode en exécutant les fonctions correspondantes, au fur et à mesure. Ces opcodes étant propres à VB, le debugger ne sait pas les traduire et les affiche tels quels.

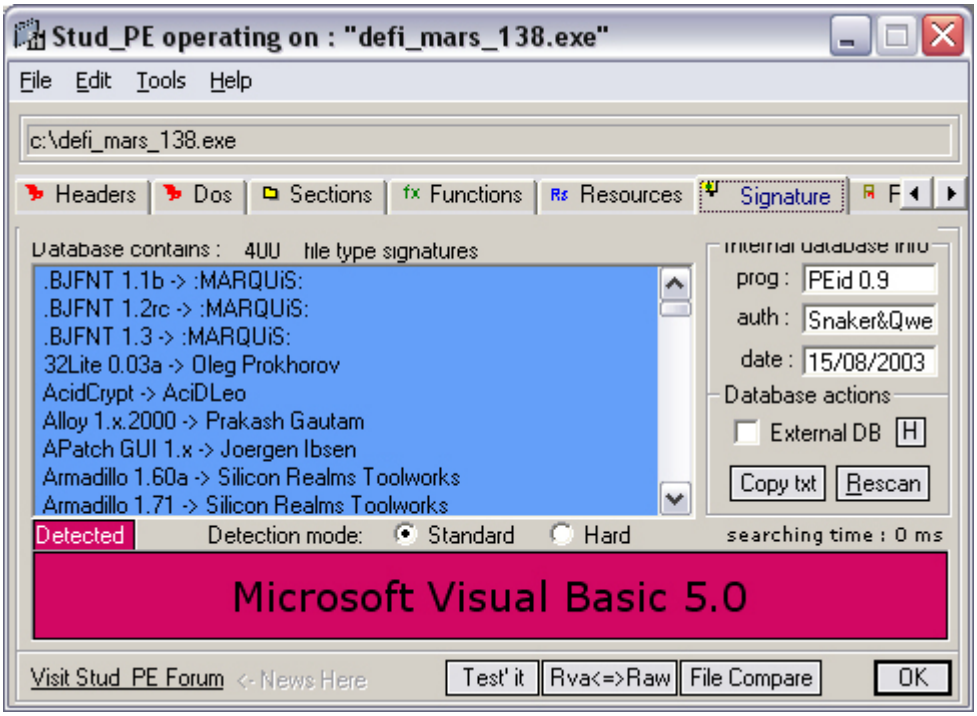
On va donc voir ce que ça peut donner. C'est parti !

Partie 1 : pré-analyse

On lance le soft, et on essaye pour voir ce que le crackme nous propose.

On tape alors n'importe quoi dans les 2 champs qui sont présents : j'ai mis name="abcde" et serial="123456"
On clique sur Ok, et ..... il ne se passe rien. Ben ça alors !

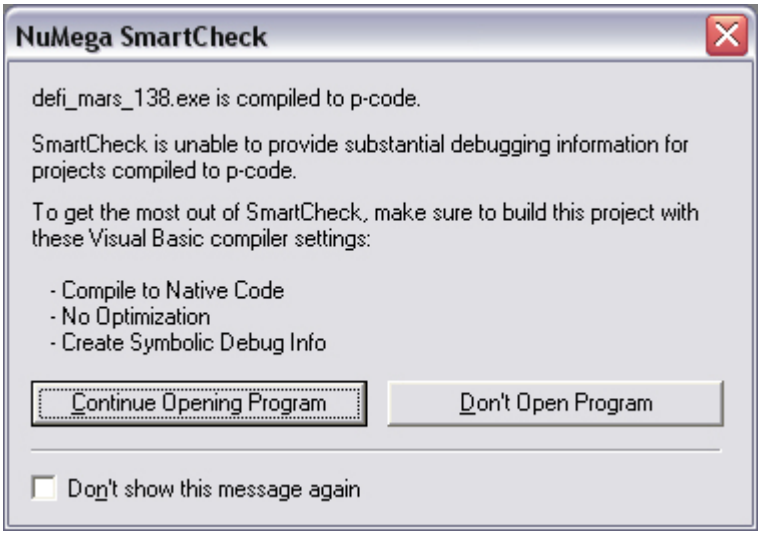
On va déjà voir ce que nous dit Stud_PE : le langage utilisé est confirmé.



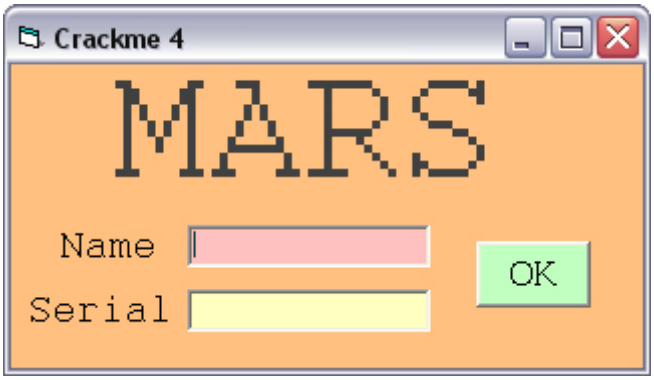
On va donc essayer de debugger avec SmartCheck.

Partie 2 : debuggage et désillusion

On charge le soft dans SmartCheck et on tout de suite celui-ci nous confirme que le crackme est codé en p-code et qu'il ne pourra pas trouver grand chose.



Qu'à cela ne tienne, on continue tout de même. On lance le soft dans Smartcheck, et il apparaît.



On remet les même informations que tout à l'heure, à savoir name="abcde" et serial="123456" puis on appuie sur "OK". Et il ne se passe rien non plus à part l'apparition dans le log d'une entrée "Bouton1_Click" :normal !

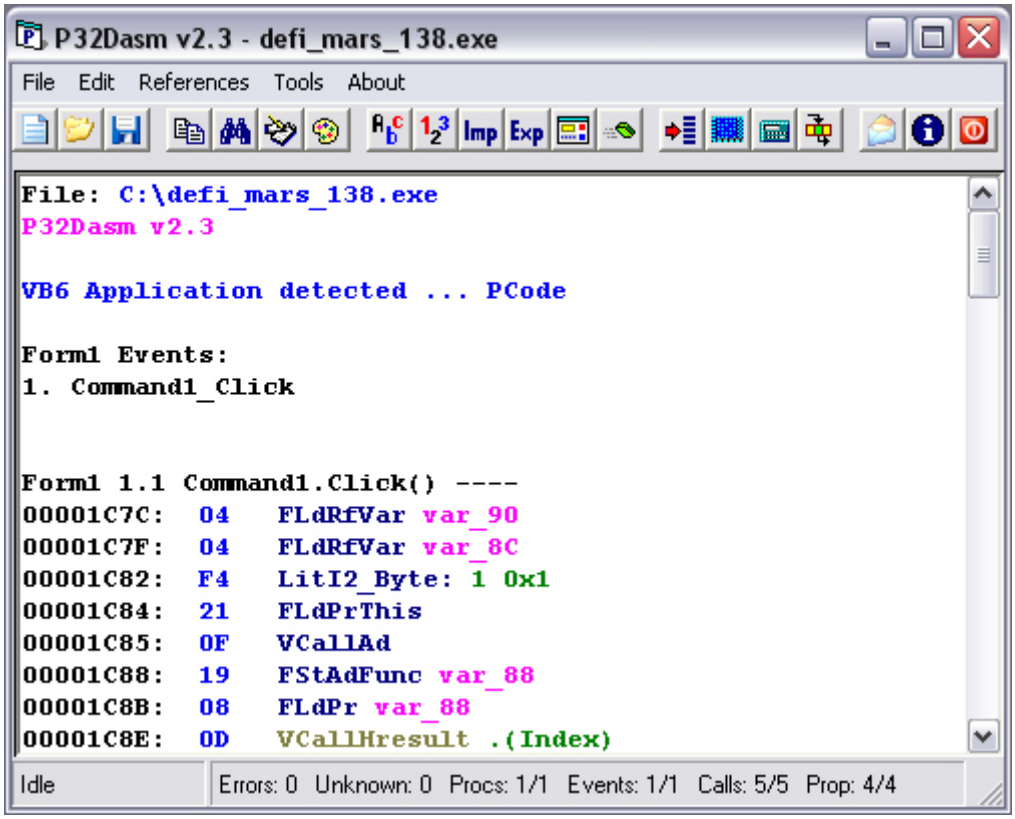
Afin d'analyser tranquillement, les actions faites par le crackme sous SmartCheck, on arrête en pressant le bouton "stop", puis on sélectionne l'entrée "Bouton1_Click". Mais là grande déception : un code imbuvable et aucun renseignement significatif. A croire que le crakme ne fait rien à part attendre une entrée correcte dans les messagebox.

Partie 3: décompilation

Il nous faut donc voir le code de plus près, et avec un aute outil. C'est parti pour une nouvelle recherche : google est ton ami. Comme on n'y connaît rien et que le hasard fait souvent bien les choses, on choisit "***P32Dasm***" parmi toutes les propositions de google aux termes "*p-code decompiler*".

Il s'agit d'un logiciel qui possède quelques petits outils intégrés et intéressants qui vont nous servir.

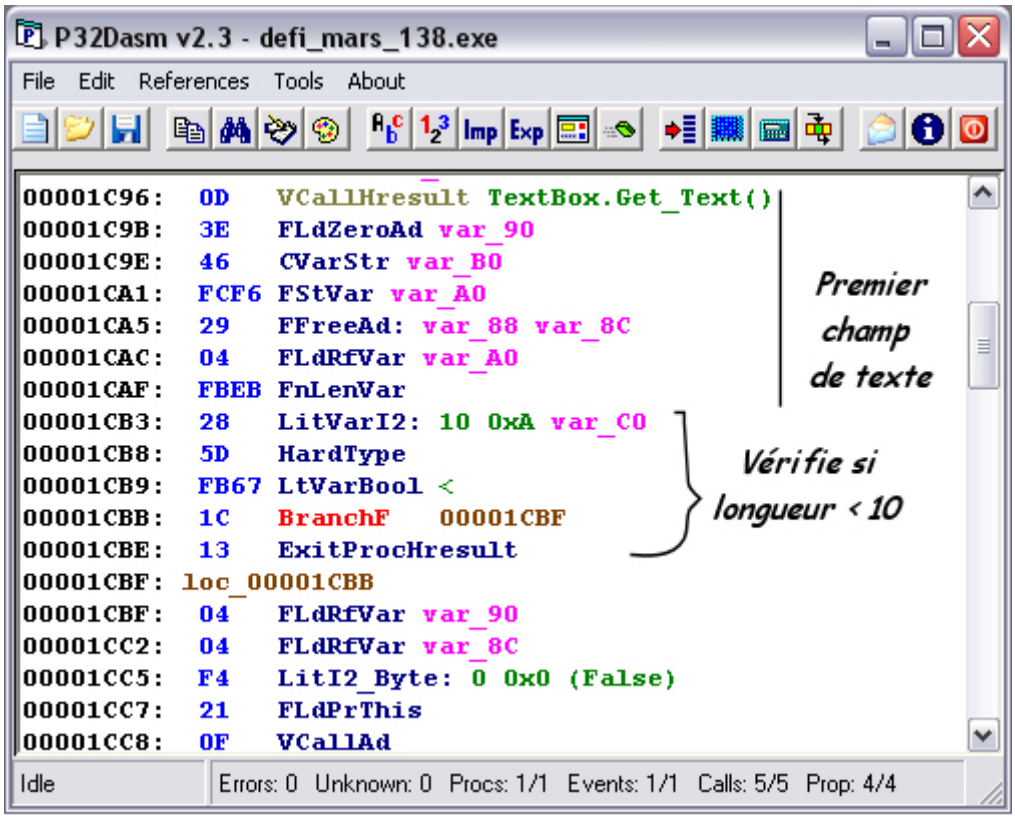
On lance donc le soft, on charge notre crackme et on observe :



Belle coloration syntaxique !
- **les adresses en noir**
- **les opcodes propres au p-code en bleu**
- **les instructions en bleu foncé**
- **les variables utilisées en mauve**
- **les contenus des variables en vert.**

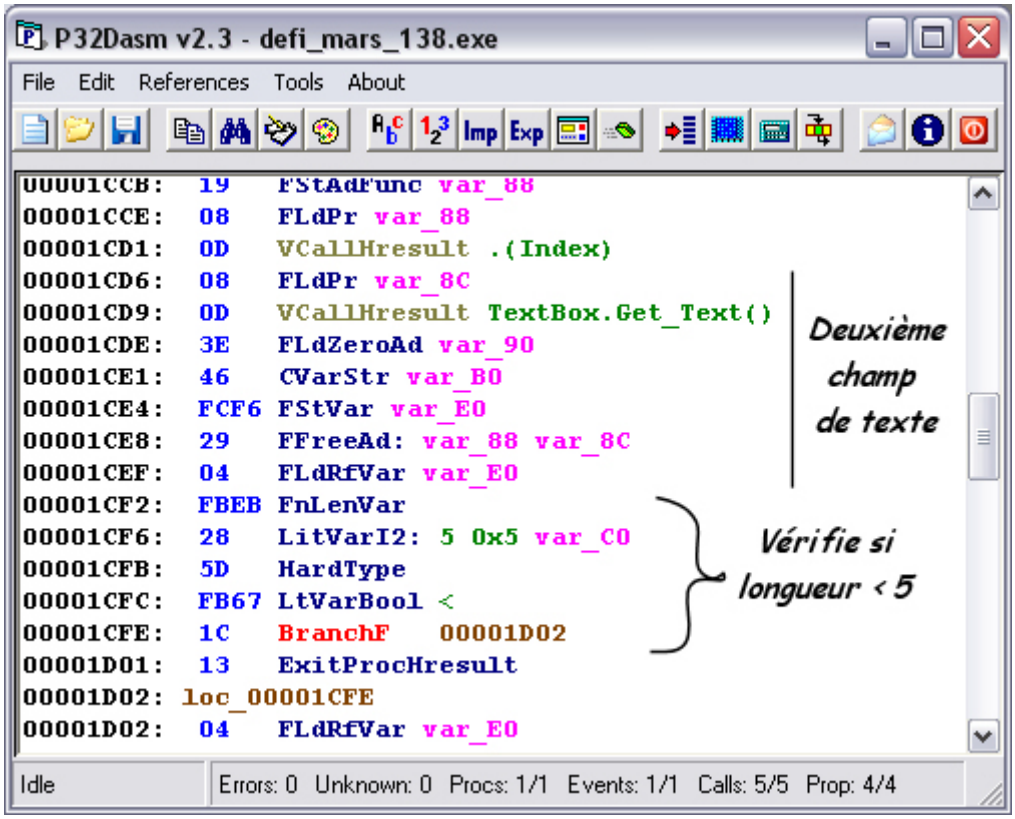
Partie 4 : analyse statique du code décompilé

En regardant de plus près les instructions propres à ce code, on peut quand-même déduire un certain nombre de choses, même en n'y connaissant pas grand chose. Par exemple :



Ici le crackme charge le premier champ de texte, vérifie si la le nombre de caractères est inférieur à **10** et saute en **00001CBF** si **ce n'est pas** le cas. Sinon on a droit à un joli "**ExitProcHresult**" qui doit correspondre à une sortie de la procédure de vérification et à un premier "badboy" silencieux.

Autrement dit l'un des deux champs de texte doit au moins avoir 10 caractères : par expérience on pencherait plutôt pour le serial...

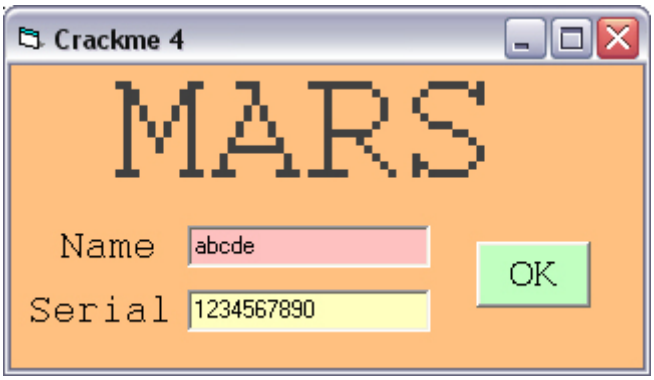


Ici idem que plus haut mais pour le deuxième champ de texte qui doit avoir au moins

5 caractères. On penche pour le nom.

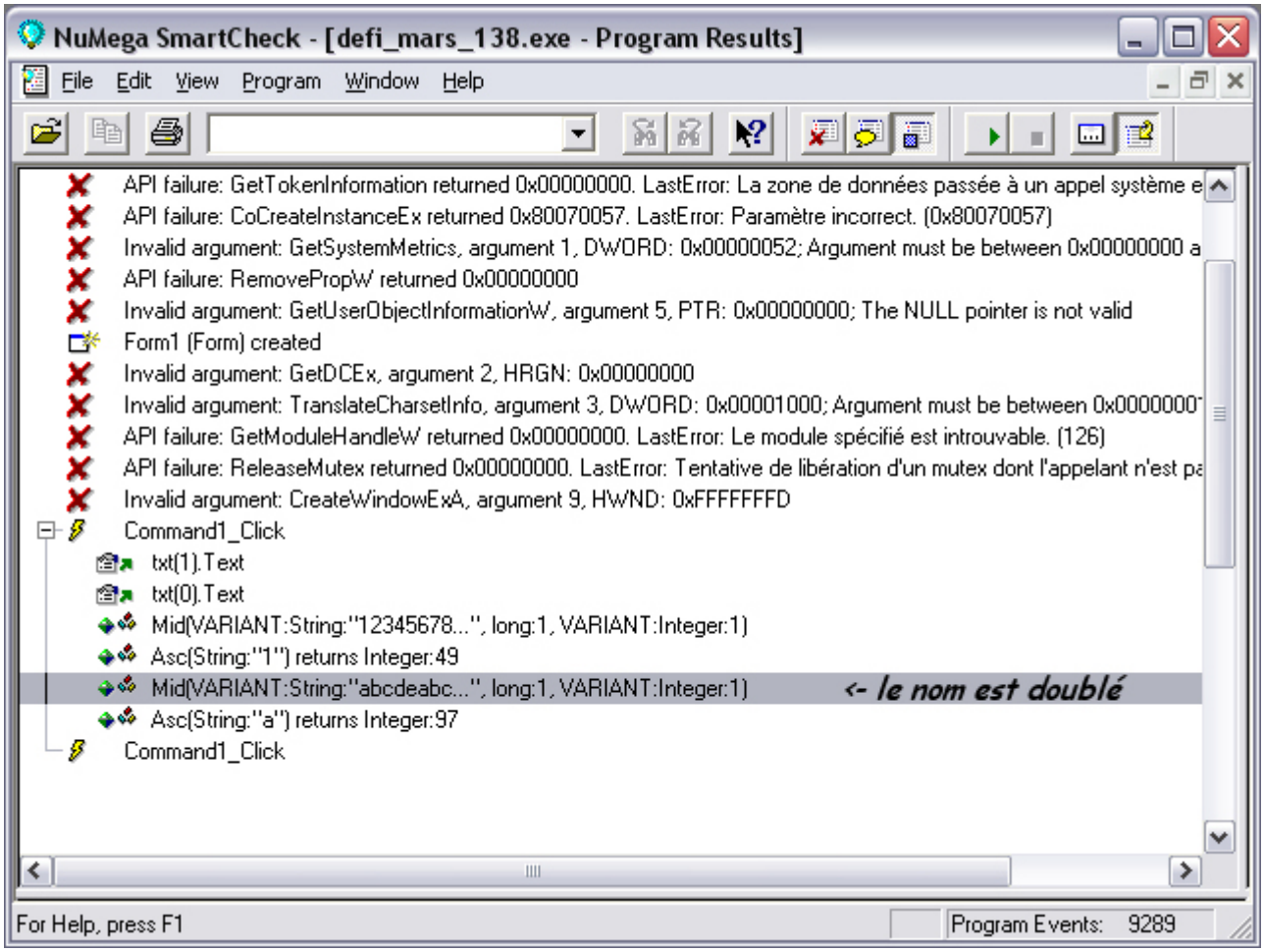
Partie 5 : Vérification avec SmartCheck.

On va donc retourner sous SmartCheck pour vérifier nos conclusions et voir si celui-ci est plus bavard que tout à l'heure en donnant les bons paramètres au crackme...



On change donc nos entrées avec les valeurs ci-dessus, on appuie sur "OK" et on arrête le SmartCheck en appuyant sur "stop".

Là encore est apparue la procédure "Bouton1_Click". On déroule en appuyant sur le petit bouton "+" à côté de la procédure.



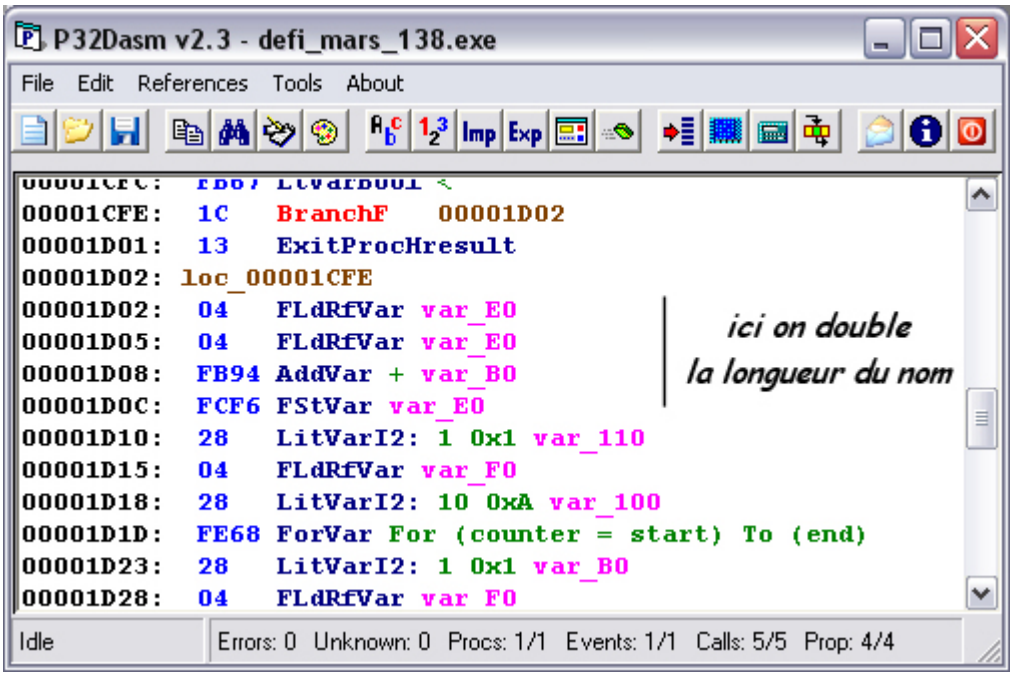
Il n'était pas beaucoup plus bavard que tout à l'heure, mais quand même... On remarque notre nom a été doublé : "**abcde**" est devenu "**abcdeabc..**" qui correspond

sans doute à "**abcdeabcde**".

Une petite réflexion nous dit que le crackme a sûrement besoin d'une longueur de nom d'au moins **10** afin de calculer/comparer terme à terme avec le serial qui lui est probablement de longueur **10**. Afin d'être sûr que la longueur soit suffisante, l'auteur a probablement simplement copié deux fois la séquence du nom bout à bout.

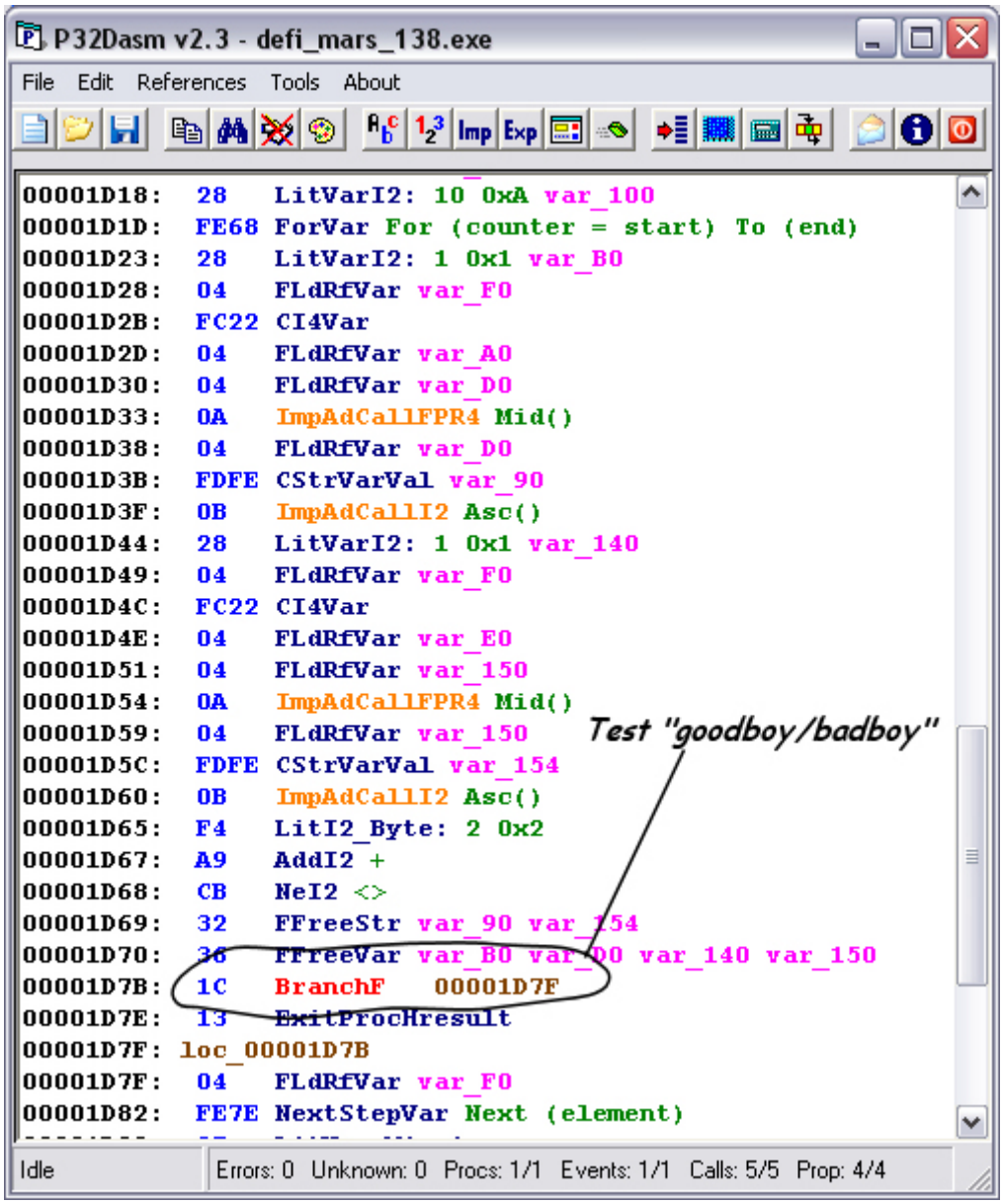
Partie 6 : La suite de l'analyse statique.

Sous ***P32Dasm*** on va regarder la suite du code et on retrouve effectivement l'endroit où le nom est doublé.



Regardons plus loin :

On remarque une boucle qui va vérifier chaque caractère du serial. A la sortie de chaque caractère, et avant de passer au suivant, on remarque un test qui va nous éjecter si ce n 'est pas bon. En fait c'est plus tordu : le test nous fait passer à l'itération suivante si *ce n'est pas différent de ce qu'on attend* (subtilité !). C'est pour cela qu'il y a le "**BranchF**" qui correspond à un "**branch if false**".

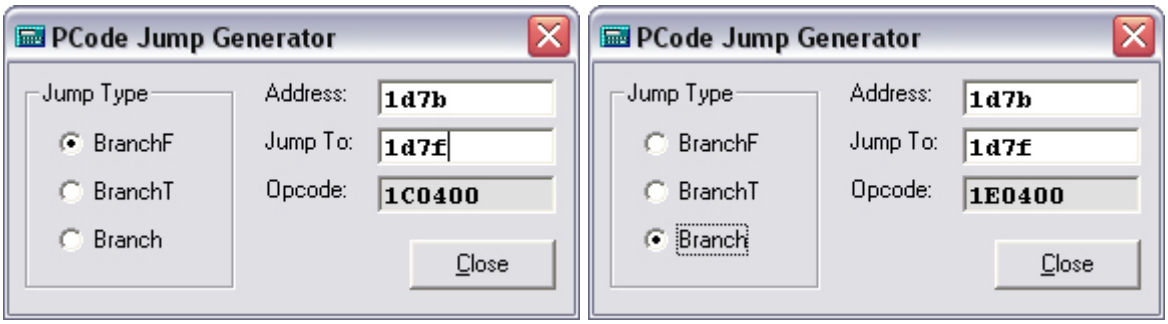


Partie 7 : Mode "patching".

On peut donc à ce moment de l'analyse, patcher le crackme très facilement. Il suffit de toujours faire sauter le programme en **00001D7F** en remplaçant le **BranchF** (saut conditionnel) par un **Branch** (saut inconditionnel).

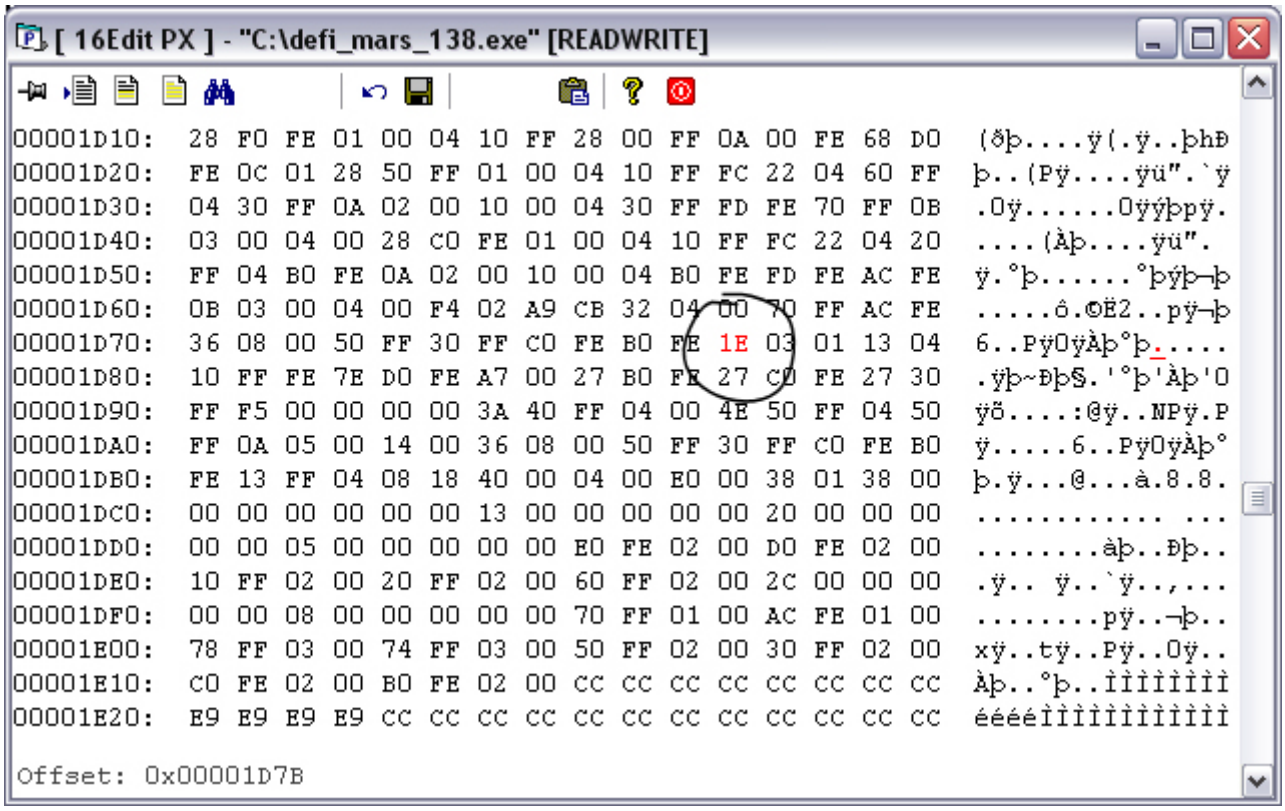
Mais comment le faire rapidement et surtout quel est le code pour le **Branch** ? Pour savoir cela on va utiliser des outils fournis dans ***P32Dasm***.

Il existe un *calculateur de saut* dans le menu "**tools/jump calculator**" qui nous donne cela lorsqu'on rempli les champs avec nos adresses :



On voit donc qu'il nous suffit de remplacer l'opcode **1C** par un **1E** pour que le soft soit patché.

Là encore pour le patch on va utiliser l'outil de dump hexadécimal fourni avec ***P32Dasm*** : le menu "**Edit/Internal Hex Editor**" nous donne un petit éditeur hexadécimal qui nous permet de mofifier le programme directement comme indiqué ci-dessous :

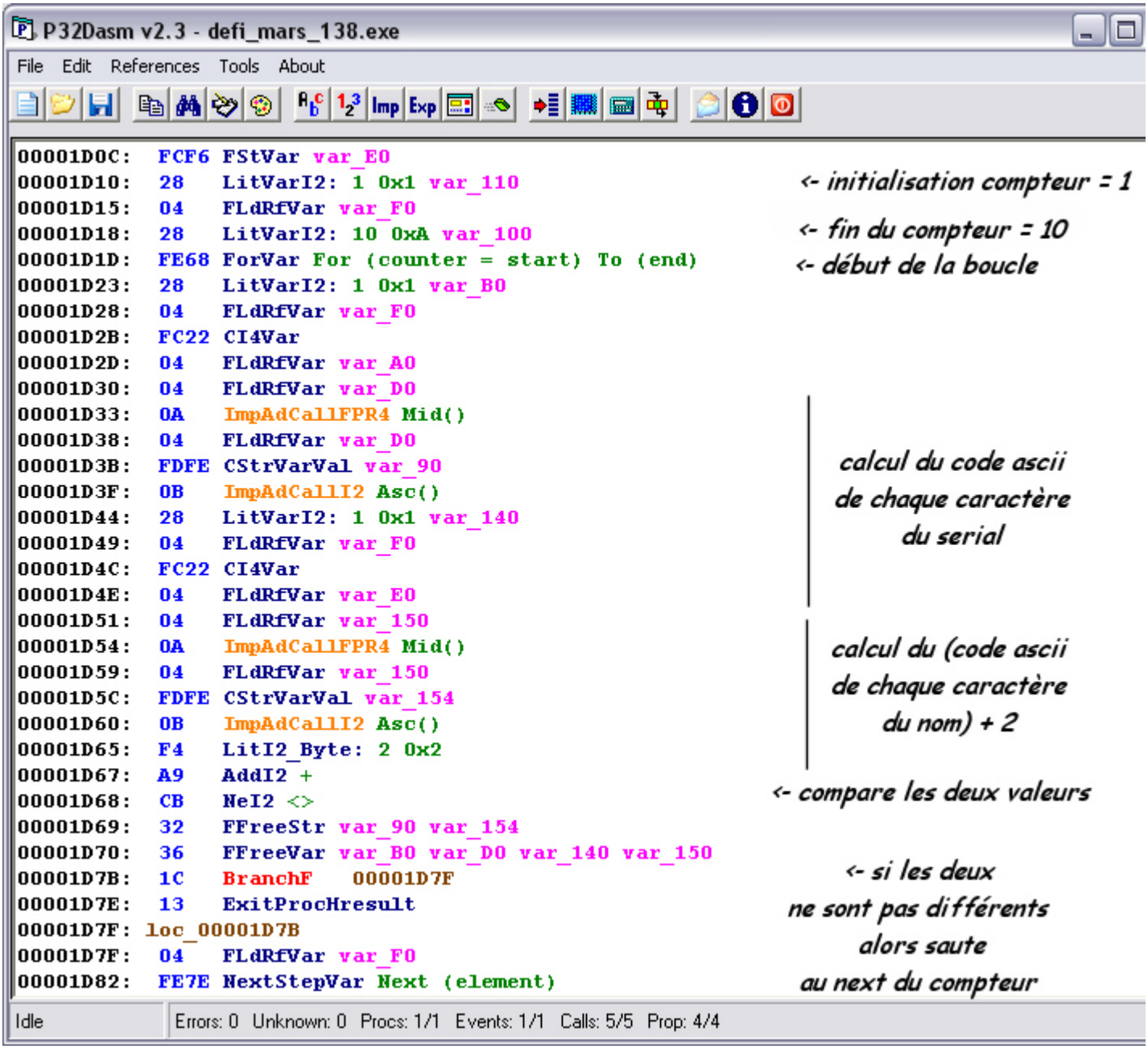


On patche et on enregistre... Vous pourrez vérifier, c'est ok, à condition d'entrer un nom de plus de **4** lettres et un serial quelconque de plus de **9** lettres.

Partie 8 : à la recherche du serial.

Mais si on est déjà en si bon chemin, on peut aussi rechercher le bon serial.

Analysons un peu cette routine trouvée ci dessus :

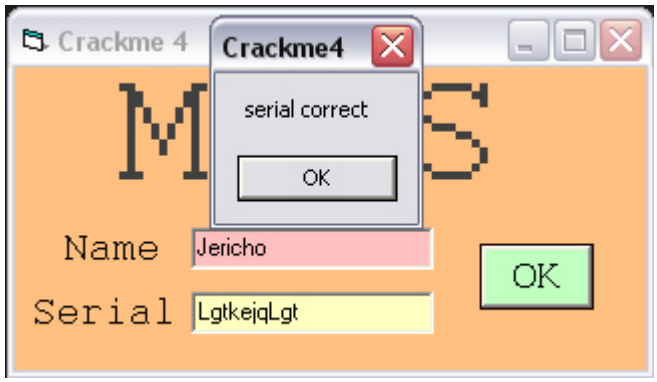


Ben voilà qui est plus clair...

Partie 9 : synthèse.

- Pour name = "**Jericho**"
- on double : name = "**JerichoJericho**"
- on ajoute **2** à chaque code ascii des caractères du nom
- on retransforme en caractères : serial = "**LgtkejqLgtkejq**"
- on ne garde que les **10** premiers caractères : serial = "**LgtkejqLgt**" (on pourrait laisser le tout, le crackme ne vérifie pas les caractères après le dixième).

Essai et ...



BINGO !

Conclusion :

Voilà, ce fut mon premier tuto après une longue période d'inactivité dans ce domaine.

Merci à **mars** de me faire apprendre encore de nouvelles choses par ce genre de petits défis.

J'espère avoir été explicite et clair ; j'attends vos remarques, critiques, suggestions, corrections...

Créé par Jericho
avril 2007