



# 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 !

---

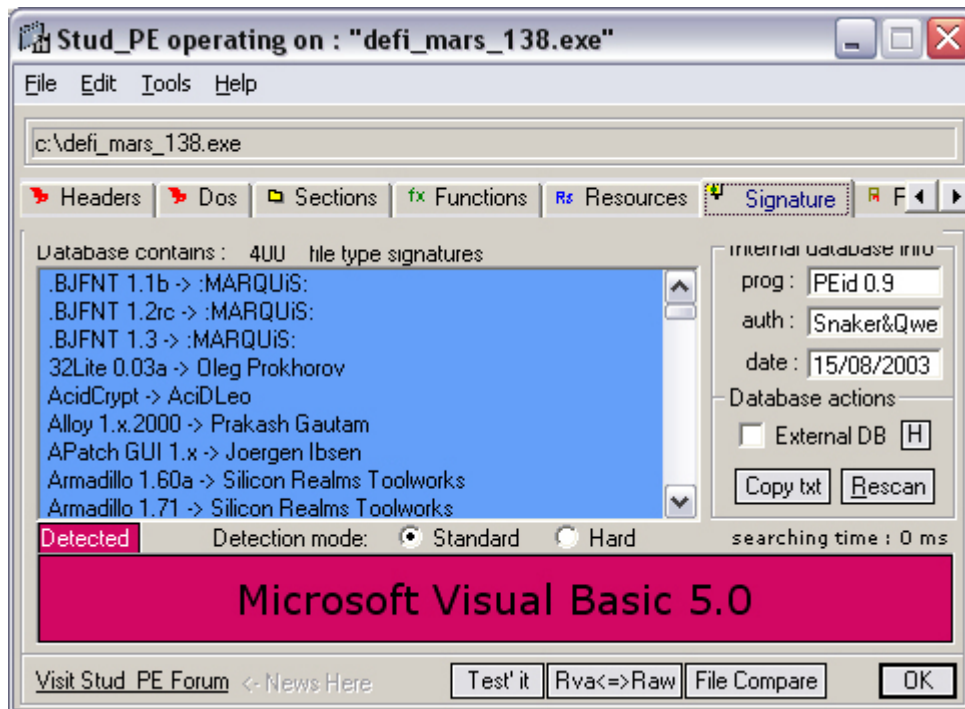
## **Partie 1 : pré-analyse**

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 !

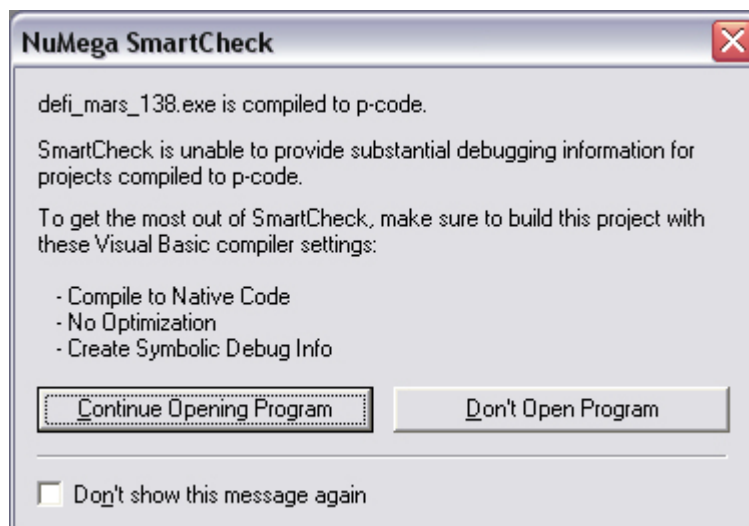
On va déjà voir ce que nous dit Stud\_PE : le langage utilisé est confirmé.



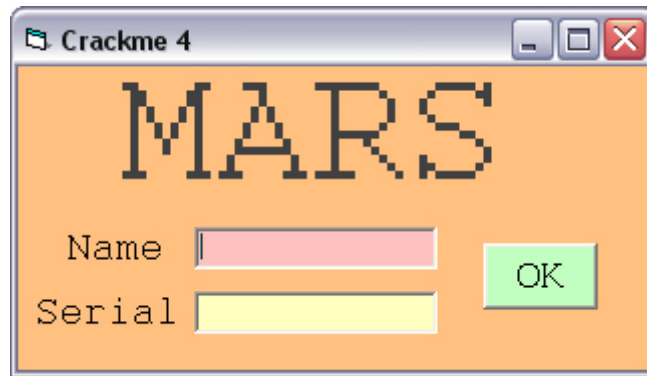
On va donc essayer de debugger avec SmartCheck.

## 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 crackme 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 autre 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 :

```
P32Dasm v2.3 - defi_mars_138.exe
File Edit References Tools About
File Edit References Tools About
File: C:\defi_mars_138.exe
P32Dasm v2.3
VB6 Application detected ... PCode
Form1 Events:
1. Command1_Click
Form1 1.1 Command1.Click() ----
00001C7C: 04 FLdRfVar var_90
00001C7F: 04 FLdRfVar var_8C
00001C82: F4 LitI2_Byte: 1 0x1
00001C84: 21 FLdPrThis
00001C85: 0F VCallAd
00001C88: 19 FStAdFunc var_88
00001C8B: 08 FLdPr var_88
00001C8E: 0D VCallHresult .(Index)
Idle Errors: 0 Unknown: 0 Procs: 1/1 Events: 1/1 Calls: 5/5 Prop: 4/4
```

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 :

```

00001C96: 0D  VCallHresult  TextBox.Get_Text()
00001C9B: 3E  FLdZeroAd  var_90
00001C9E: 46  CVarStr  var_B0
00001CA1: FCF6 FStVar  var_A0
00001CA5: 29  FFreeAd:  var_88 var_8C
00001CAC: 04  FLdRfVar  var_A0
00001CAF: FBEB FnLenVar
00001CB3: 28  LitVarI2: 10 0xA var_C0
00001CB8: 5D  HardType
00001CB9: FB67 LtVarBool <
00001CBB: 1C  BranchF  00001CBF
00001CBE: 13  ExitProcHresult
00001CBF: loc_00001CBF
00001CBF: 04  FLdRfVar  var_90
00001CC2: 04  FLdRfVar  var_8C
00001CC5: F4  LitI2_Byte: 0 0x0 (False)
00001CC7: 21  FLdPrThis
00001CC8: 0F  VCallAd

```

Idle Errors: 0 Unknown: 0 Procs: 1/1 Events: 1/1 Calls: 5/5 Prop: 4/4

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...

```

00001CCB: 19  FStAdFunc  var_88
00001CCE: 08  FLdPr  var_88
00001CD1: 0D  VCallHresult  .(Index)
00001CD6: 08  FLdPr  var_8C
00001CD9: 0D  VCallHresult  TextBox.Get_Text()
00001CDE: 3E  FLdZeroAd  var_90
00001CE1: 46  CVarStr  var_B0
00001CE4: FCF6 FStVar  var_E0
00001CE8: 29  FFreeAd:  var_88 var_8C
00001CEF: 04  FLdRfVar  var_E0
00001CF2: FBEB FnLenVar
00001CF6: 28  LitVarI2: 5 0x5 var_C0
00001CFB: 5D  HardType
00001CFC: FB67 LtVarBool <
00001CFE: 1C  BranchF  00001D02
00001D01: 13  ExitProcHresult
00001D02: loc_00001CFE
00001D02: 04  FLdRfVar  var_E0

```

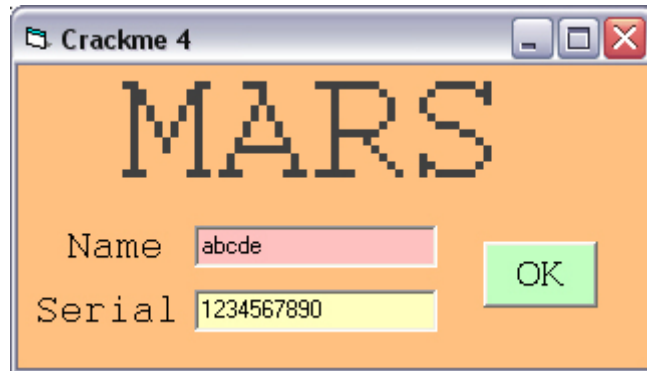
Idle Errors: 0 Unknown: 0 Procs: 1/1 Events: 1/1 Calls: 5/5 Prop: 4/4

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

5 caractères. On penche pour le nom.

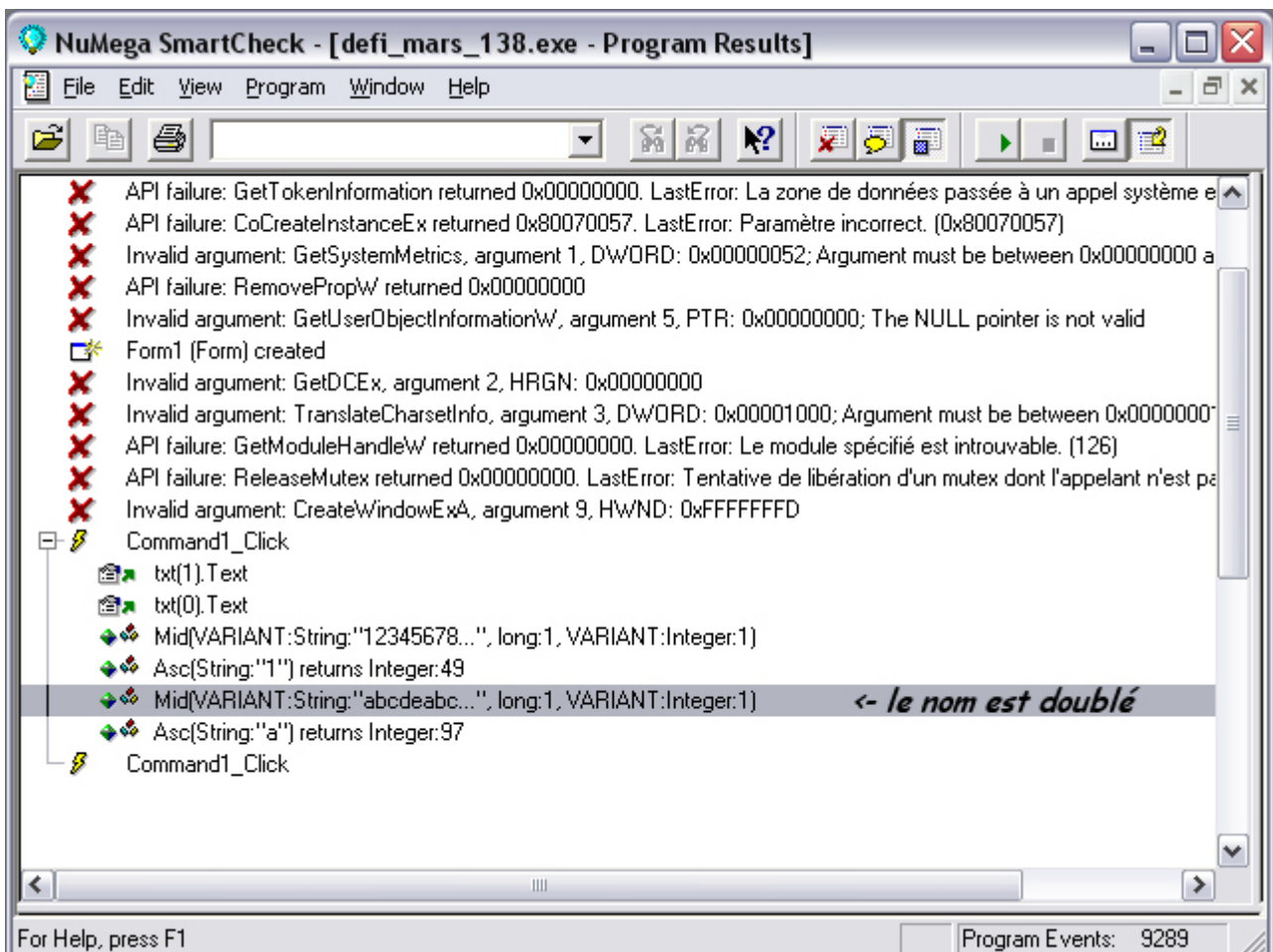
## Partie 5 : Vérification avec SmartCheck.

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

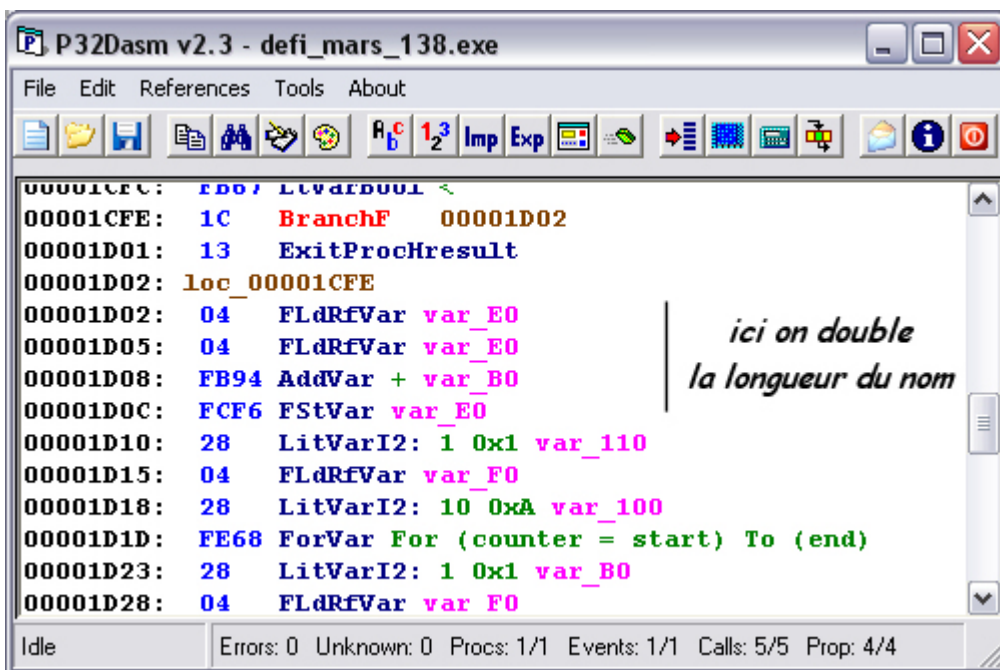
sans doute à "**abcdeabcde**".

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.



```
P32Dasm v2.3 - defi_mars_138.exe
File Edit References Tools About
[Icons]
00001CFE: fdb7 LitVarI2: 1 0x1 var_B0
00001CFE: 1C BranchF 00001D02
00001D01: 13 ExitProcHresult
00001D02: loc_00001CFE
00001D02: 04 FLdRfVar var_E0
00001D05: 04 FLdRfVar var_E0
00001D08: FB94 AddVar + var_B0
00001D0C: FCF6 FStVar var_E0
00001D10: 28 LitVarI2: 1 0x1 var_110
00001D15: 04 FLdRfVar var_F0
00001D18: 28 LitVarI2: 10 0xA var_100
00001D1D: FE68 ForVar For (counter = start) To (end)
00001D23: 28 LitVarI2: 1 0x1 var_B0
00001D28: 04 FLdRfVar var_F0
Idle Errors: 0 Unknown: 0 Procs: 1/1 Events: 1/1 Calls: 5/5 Prop: 4/4
```

*ici on double  
la longueur du nom*

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**".



```

P32Dasm v2.3 - defi_mars_138.exe
File Edit References Tools About
00001D18: 28 LitVarI2: 10 0xA var_100
00001D1D: FE68 ForVar For (counter = start) To (end)
00001D23: 28 LitVarI2: 1 0x1 var_B0
00001D28: 04 FLdRfVar var_F0
00001D2B: FC22 CI4Var
00001D2D: 04 FLdRfVar var_A0
00001D30: 04 FLdRfVar var_D0
00001D33: 0A ImpAdCallFPR4 Mid()
00001D38: 04 FLdRfVar var_D0
00001D3B: FDDE CStrVarVal var_90
00001D3F: 0B ImpAdCallI2 Asc()
00001D44: 28 LitVarI2: 1 0x1 var_140
00001D49: 04 FLdRfVar var_F0
00001D4C: FC22 CI4Var
00001D4E: 04 FLdRfVar var_E0
00001D51: 04 FLdRfVar var_150
00001D54: 0A ImpAdCallFPR4 Mid()
00001D59: 04 FLdRfVar var_150
00001D5C: FDDE CStrVarVal var_154
00001D60: 0B ImpAdCallI2 Asc()
00001D65: F4 LitI2_Byte: 2 0x2
00001D67: A9 AddI2 +
00001D68: CB NeI2 <>
00001D69: 32 FFreeStr var_90 var_154
00001D70: 36 FFreeVar var_B0 var_00 var_140 var_150
00001D7B: 1C BranchF 00001D7F
00001D7E: 13 ExitProcHresult
00001D7F: loc_00001D7B
00001D7F: 04 FLdRfVar var_F0
00001D82: FE7E NextStepVar Next (element)
Idle Errors: 0 Unknown: 0 Procs: 1/1 Events: 1/1 Calls: 5/5 Prop: 4/4

```

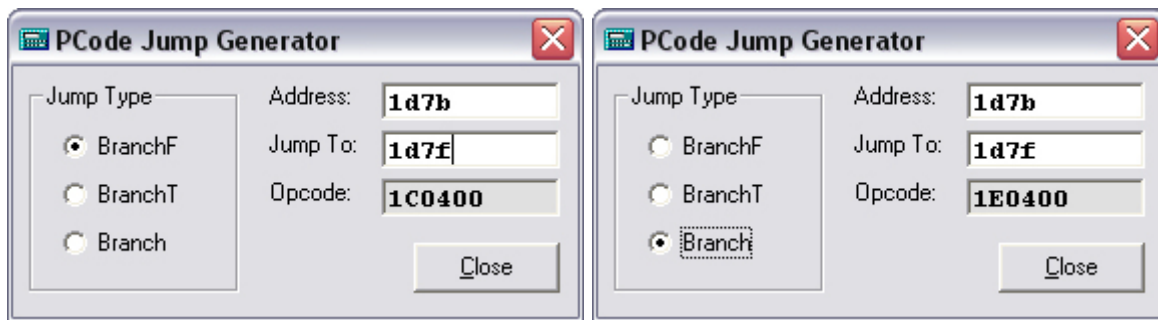
## Partie 7 : Mode "patching".

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**.

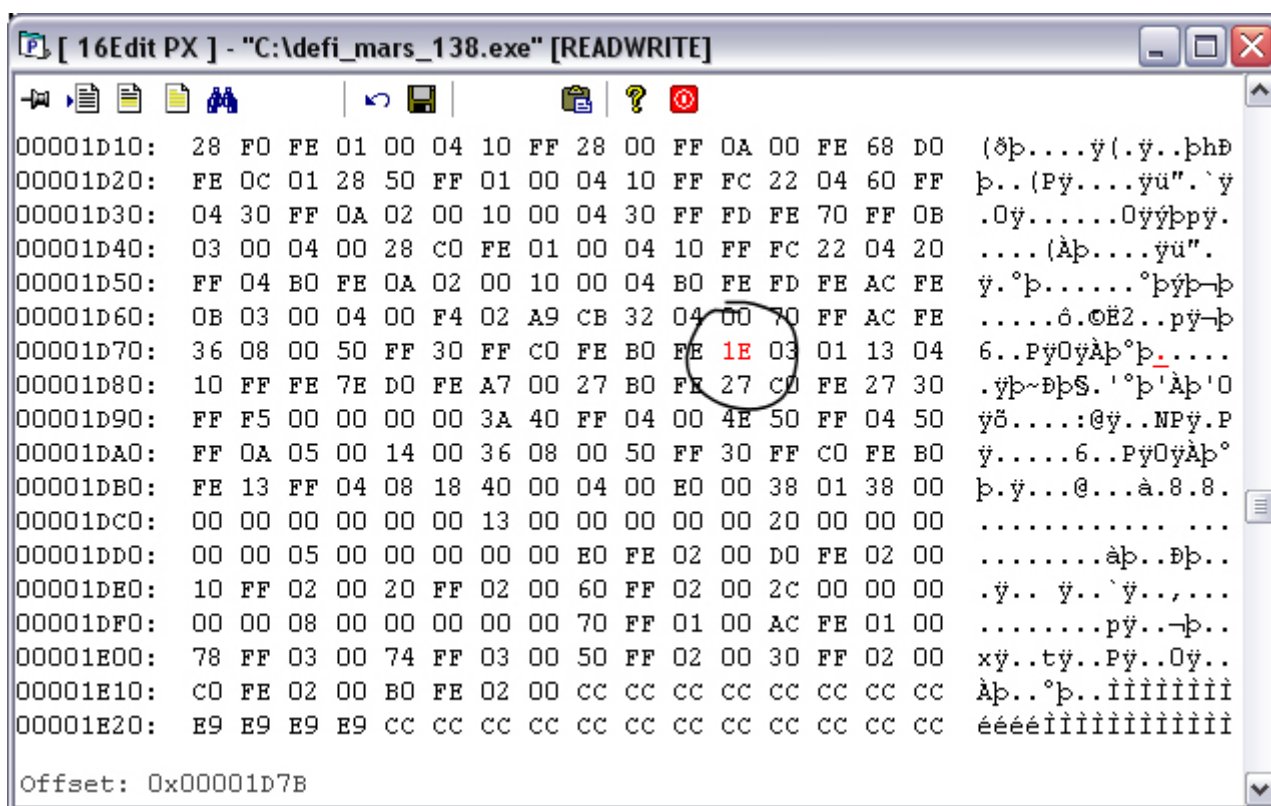
Il existe un *calculateur de saut* dans le menu "**tools/jump calculator**" qui nous donne cela lorsqu'on remplit les champs avec nos adresses :





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 modifier 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 :

```

P32Dasm v2.3 - defi_mars_138.exe
File Edit References Tools About
[Icons] R c 1,3 Imp Exp [Icons] [Icons] [Icons] [Icons] [Icons] [Icons] [Icons] [Icons]
00001D0C: FCF6 FStVar var_E0
00001D10: 28 LitVarI2: 1 0x1 var_110 <- initialisation compteur = 1
00001D15: 04 FLdRfVar var_F0
00001D18: 28 LitVarI2: 10 0xA var_100 <- fin du compteur = 10
00001D1D: FE68 ForVar For (counter = start) To (end) <- début de la boucle
00001D23: 28 LitVarI2: 1 0x1 var_B0
00001D28: 04 FLdRfVar var_F0
00001D2B: FC22 CI4Var
00001D2D: 04 FLdRfVar var_A0
00001D30: 04 FLdRfVar var_D0
00001D33: 0A ImpAdCallFPR4 Mid()
00001D38: 04 FLdRfVar var_D0
00001D3B: FDFF CStrVarVal var_90
00001D3F: 0B ImpAdCallI2 Asc()
00001D44: 28 LitVarI2: 1 0x1 var_140
00001D49: 04 FLdRfVar var_F0
00001D4C: FC22 CI4Var
00001D4E: 04 FLdRfVar var_E0
00001D51: 04 FLdRfVar var_150
00001D54: 0A ImpAdCallFPR4 Mid()
00001D59: 04 FLdRfVar var_150
00001D5C: FDFF CStrVarVal var_154
00001D60: 0B ImpAdCallI2 Asc()
00001D65: F4 LitI2_Byte: 2 0x2
00001D67: A9 AddI2 +
00001D68: CB NeI2 <> <- compare les deux valeurs
00001D69: 32 FFreeStr var_90 var_154
00001D70: 36 FFreeVar var_B0 var_D0 var_140 var_150
00001D7B: 1C BranchF 00001D7F
00001D7E: 13 ExitProcHresult
00001D7F: loc_00001D7B
00001D7F: 04 FLdRfVar var_F0
00001D82: FE7E NextStepVar Next (element) <- si les deux
<- si les deux
ne sont pas différents
alors saute
au next du compteur
Idle Errors: 0 Unknown: 0 Procs: 1/1 Events: 1/1 Calls: 5/5 Prop: 4/4

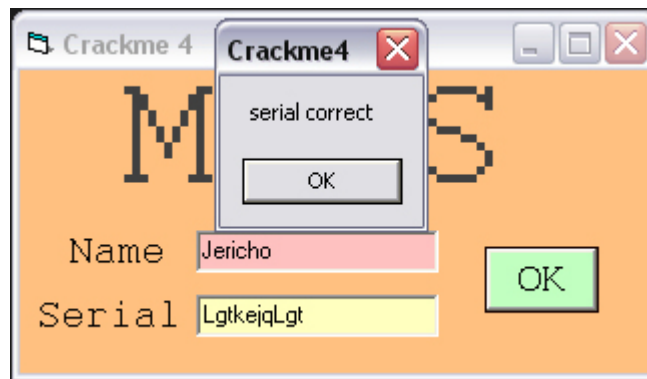
```

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