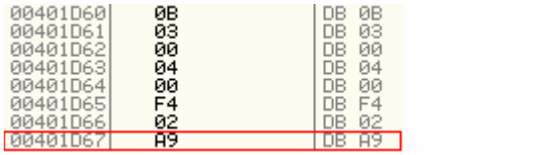
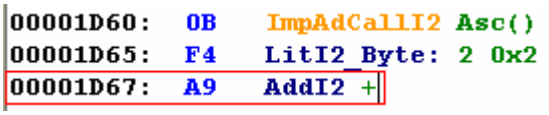
Tutorial Crackme 4 p-code vb Mars

L'introduction qui suit est un morceau du tutorial de karneth sur un crackme en p-code vb
http://kharneth.free.fr/_Tutorials_/CrackMe_Chronos02/index.html
Un programme compilé en P-code utilise une machine virtuelle (ici, *MSVBVM60.DLL*) 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 tel quel:
(listing avec ollydbg)



dans le listing ci-dessus, pour accéder à la fonction AddI2 (entourée en rouge), le moteur lit l'opcode 0xA9 à l'adresse 00401D67. Mais pour cracker un programme compilé en P-code, il serait assez fastidieux de tracer dans la dll en essayant de compendre chaque fonction Heureusement pour nous, certains Reversers ont eu la bonne idée de traduire tous les opcodes par leur nom de fonction plus ou moins compréhensible! On va donc utiliser *P32Dasm* (http://t4c.ic.cz/forum/attachment.php?attachmentid=363) pour "désassembler" le programme et traduire les opcodes.
(listing avec p32Dasm)



C'est bien, mais si on n'est pas adepte de l'analyse statique, on est vite bloqué…
Dans la partie qui suit, on va donc utiliser wktvbde (Whiskey Kon Tekila VB debugger) qui est un un debugger spécialement conçu pour le p-code vb

Installation du debugger

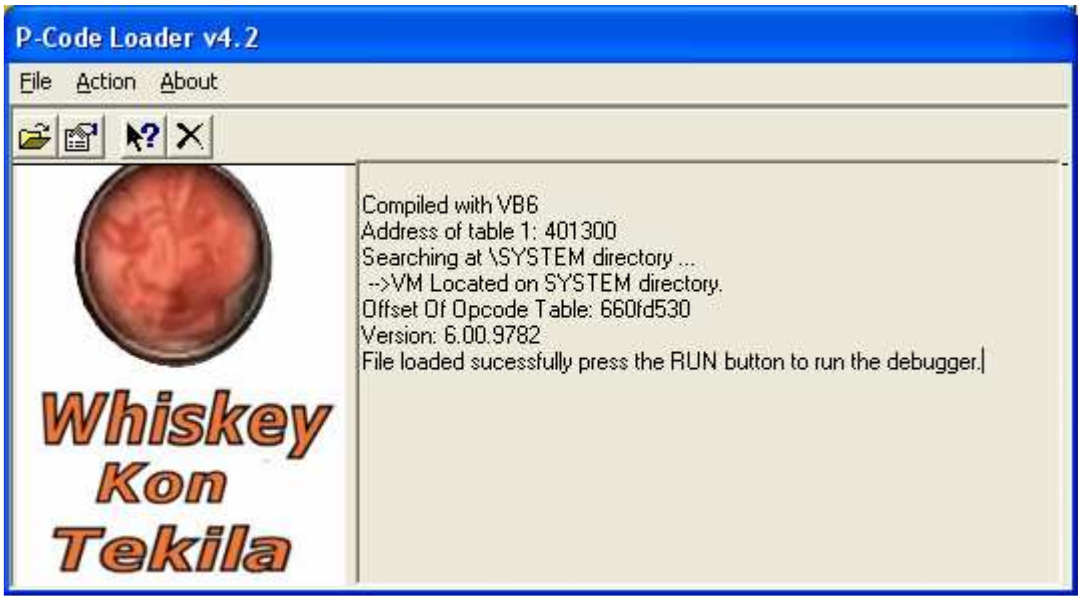
Bon, déjà, télécharger le debugger :
http://programmerstools.org/system/files?file=wkt-vbdebugger14.exe

A l'installation, on a l'erreur suivante :

cliquer sur Ignorer.

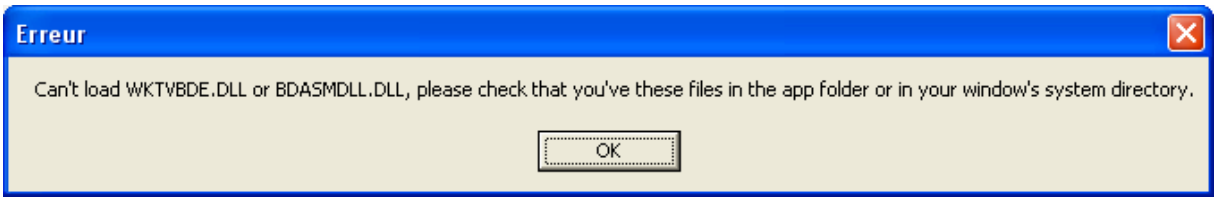
On va dans C:\Program Files\WKTVBDE, et on clique sur Loader.exe

On cliques sur File, Open, et on sélectionne le crackme,



On cliques ensuite sur le menu Action, puis Run

Si on obtient le message d'erreur suivant



il faut copier les fichiers *WKTVBDE.DLL ou BDASMDLL.DLL qui se trouvent dans* C:\Program Files\WKTVBDE, dans le dossier C:\WINDOWS OU C:\WINDOWS\SYSTEM32.

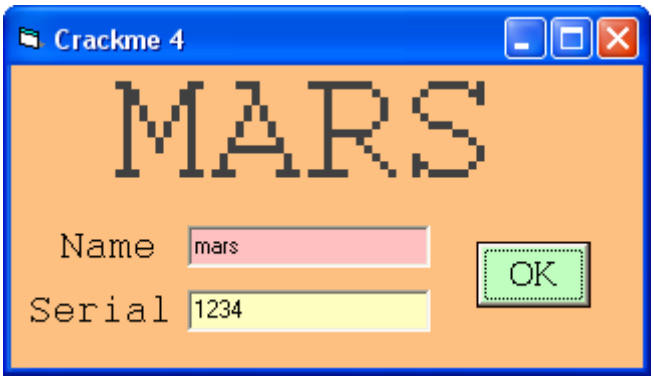
Si on obtient le message d'erreur suivant : "Can NOT locate the VB DLL inside this computer », c'est sans doute que le fichier msvbvm60.dll est en majuscules (il faut le mettre en minuscules)

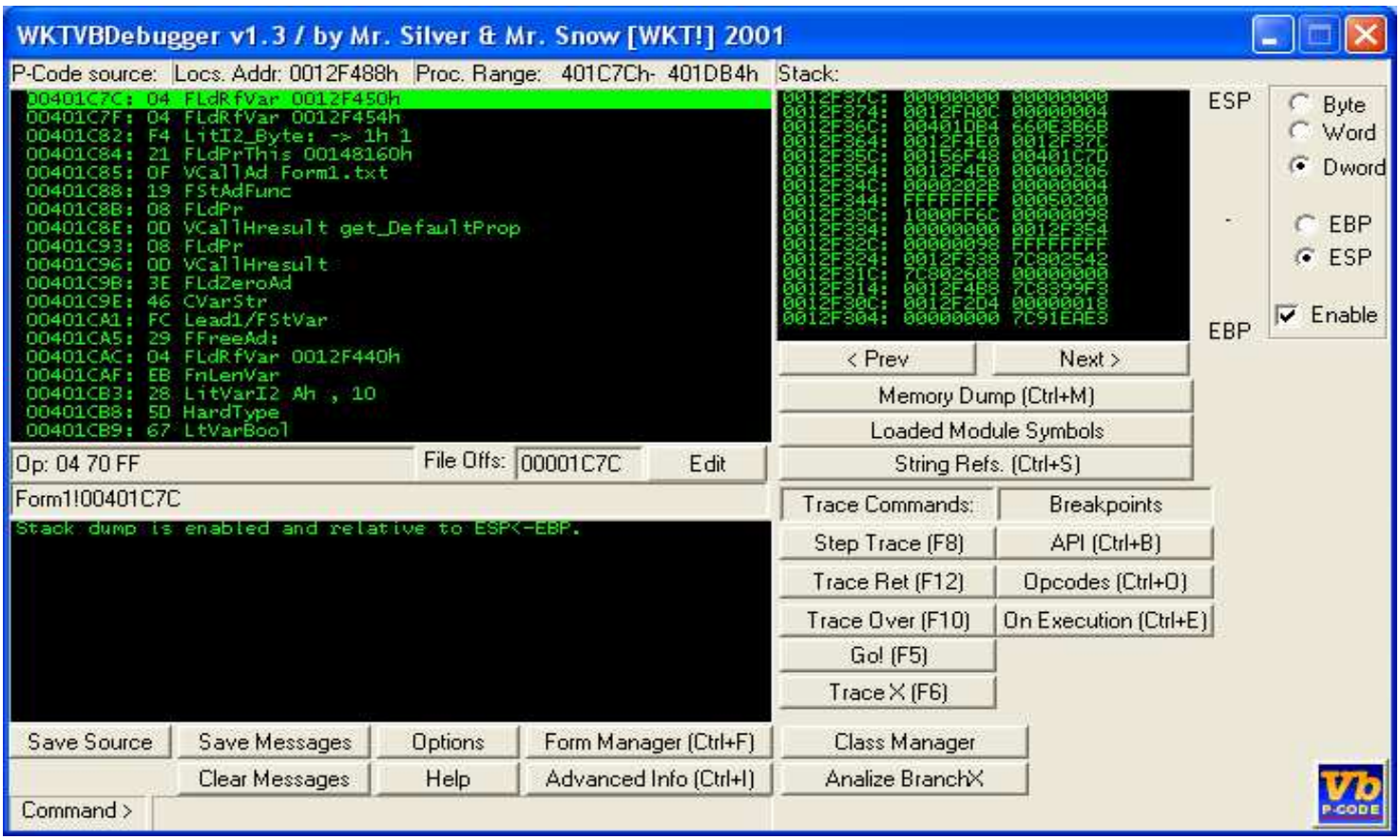
Si on obtient le message d'erreur suivant : « Can NOT find a ENGINE section for the DLL." c'est qu'un programme vb est déja lancé (il faut le fermer avant).

Redémarrer l'ordinateur si wktvbde ne fonctionne toujours pas.

A l'attaque avec Wktvbdbg

On clique sur les boutons OK et le crackme 4 apparaît. On rentre mars comme nom et 1234 comme serial et on valide sur OK, et le debugger wktvbdbg apparaît :





Il y a trois fenêtres dans le debugger :
- la fenêtre en haut à gauche est le code qu'on peut exécuter pas à pas avec F8
- la fenêtre en haut à droite affiche les adresses poussées sur la pille

(cliquer sur ESP, et sur dword : c'est beaucoup plus pratique pour lire les adresses)
- la fenêtre en bas à gauche affiche ce qui est poussé sur la pile quand c'est au format texte

le p-code vb fonctionne comme le vb → tout passe par la pile. Un paramètre est poussé dans la pile, une fonction récupère ce paramètre et renvoie le résultat dans la pile

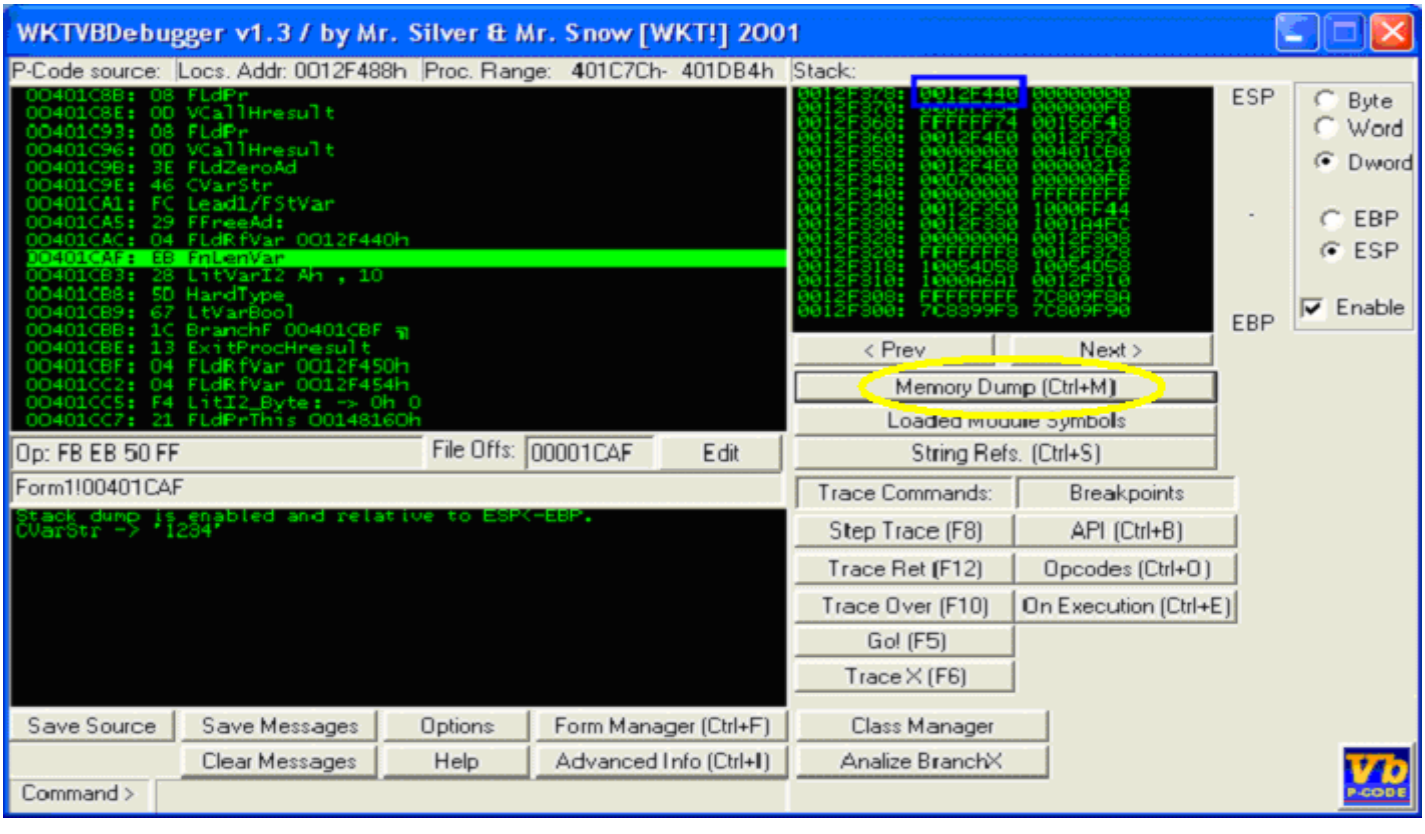
Petit lexique pour s'y retrouver en p-code vb :

Les différents types de variables utilisées :
- **Bool** pour Boolean (Vrai ou Faux)
- **Str** pour String (c'est-à-dire une valeur texte)
- **I2** pour Byte (un nombre)
- **I4** pour Long (un nombre plus grand)
- **Var** pour Variant (Variable qui peut être de n'importe quel type, texte, nombre…)

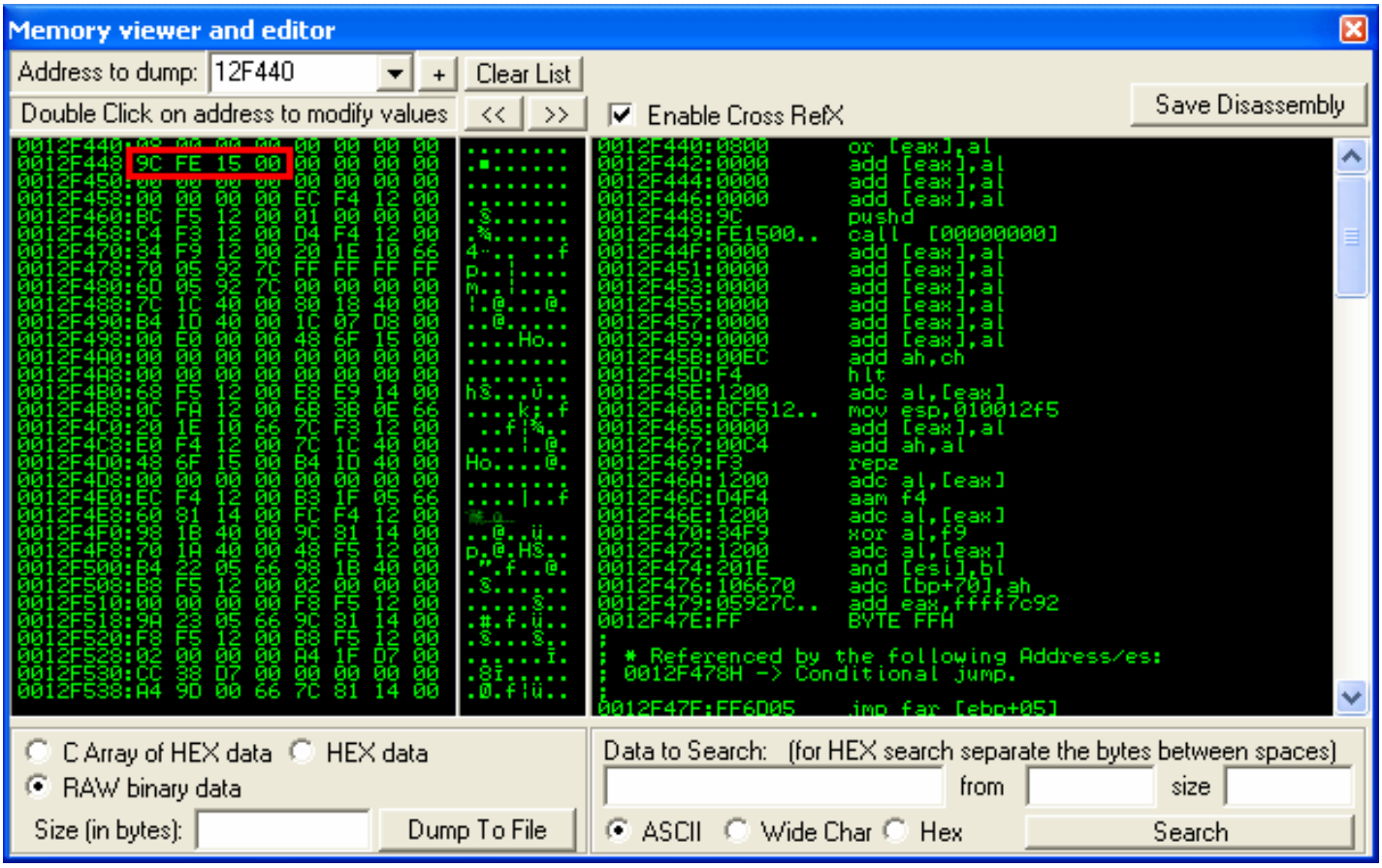
Les différentes instructions :
FLd charge une variable sur la pile (comme un push en assembleur)
Lit pousse une valeur sur la pile (comme un push en assembleur)
C indique une conversion d'un type de variable vers un autre
Vcall correspond à un call (faire attention à la pile avant et après)
FnLen une fonction (ici len qui récupère le nombre de caractères d'une variable)
Branch correspond à un saut. (jmp en assembleur)
BranchF saute si faux (False). (jne en assembleur)
BranchT saute si vrai (True). (je en assembleur)
ExitProc quitte la procédure

Passer les conditions sur la longueur du nom et du serial

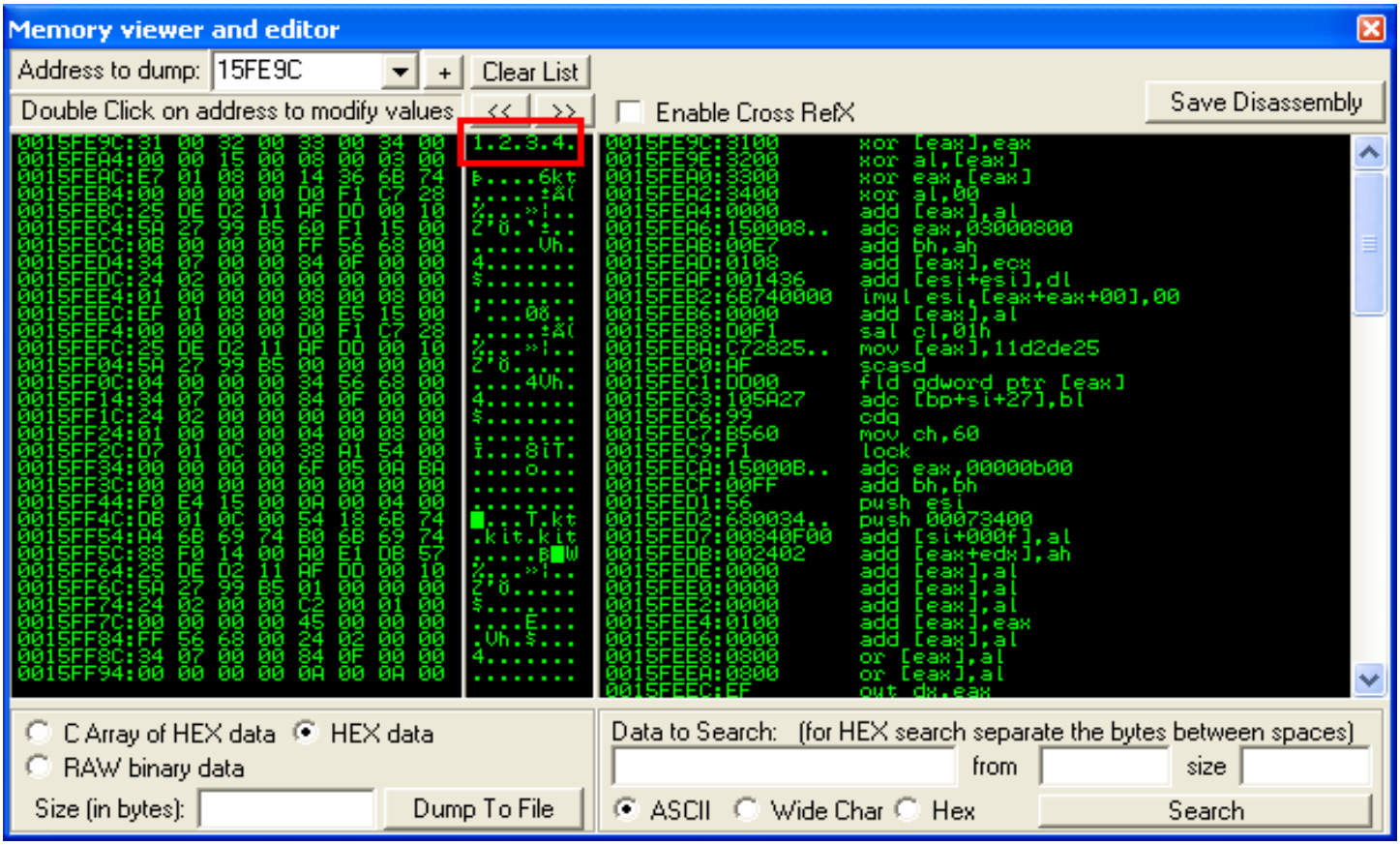
Les premières instructions ne sont pas très explicites mais servent à récupérer le serial. On appuie sur F8 jusqu'à la ligne 00401CAF FnLenVar. Cette instruction récupère le nombre de caractère d'une variable. Pour savoir de quelle variable il s'agit, on regarde la pile :



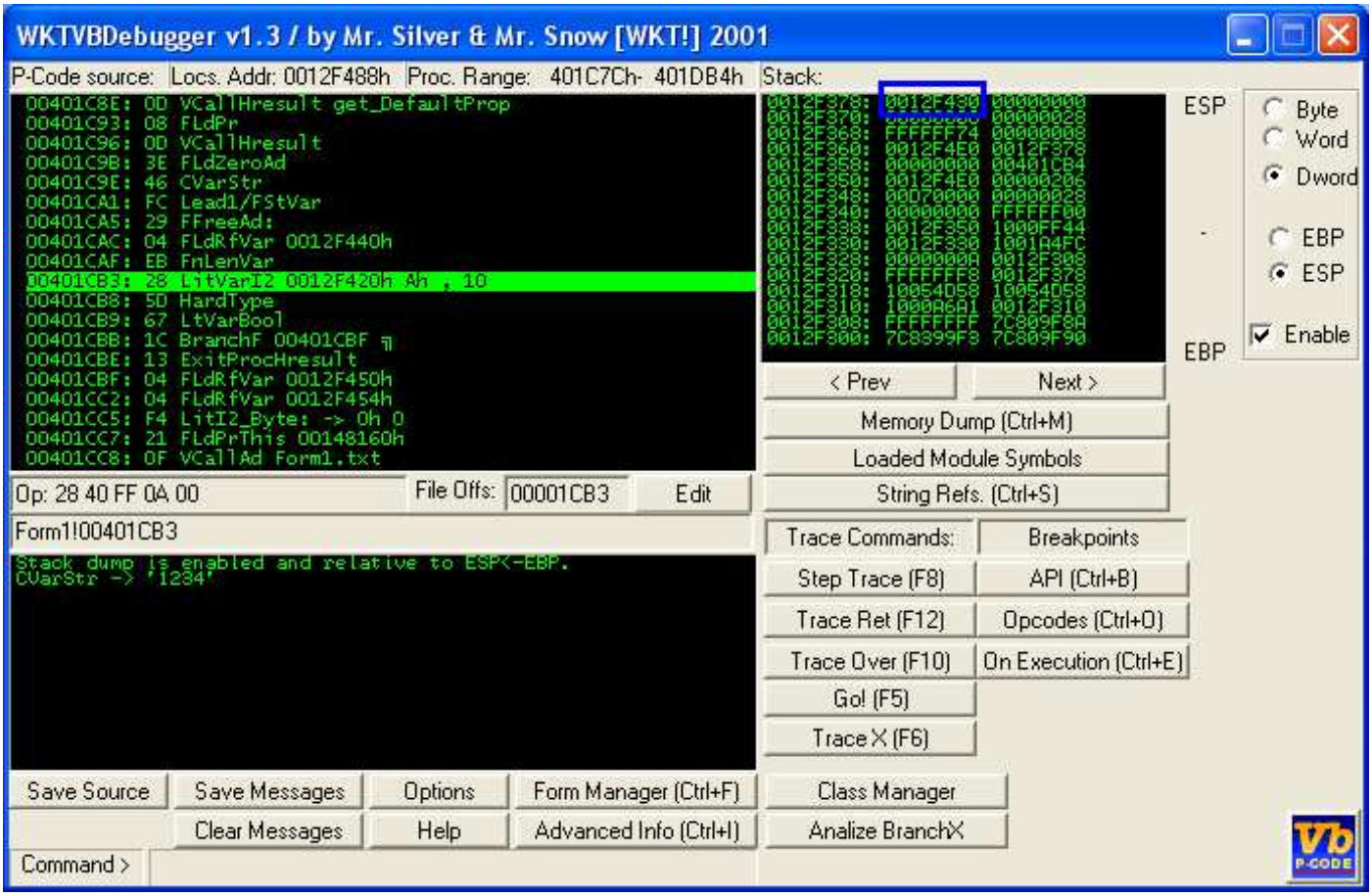
Une variable de type variant (entourée en bleu) a été poussée sur la pile donc pour connaître le contenu de cette variable, on clique sur le bouton « Memory Dump » (entouré en jaune) et on saisi 12F440 dans le champ Adress to dump. On obtient ceci :



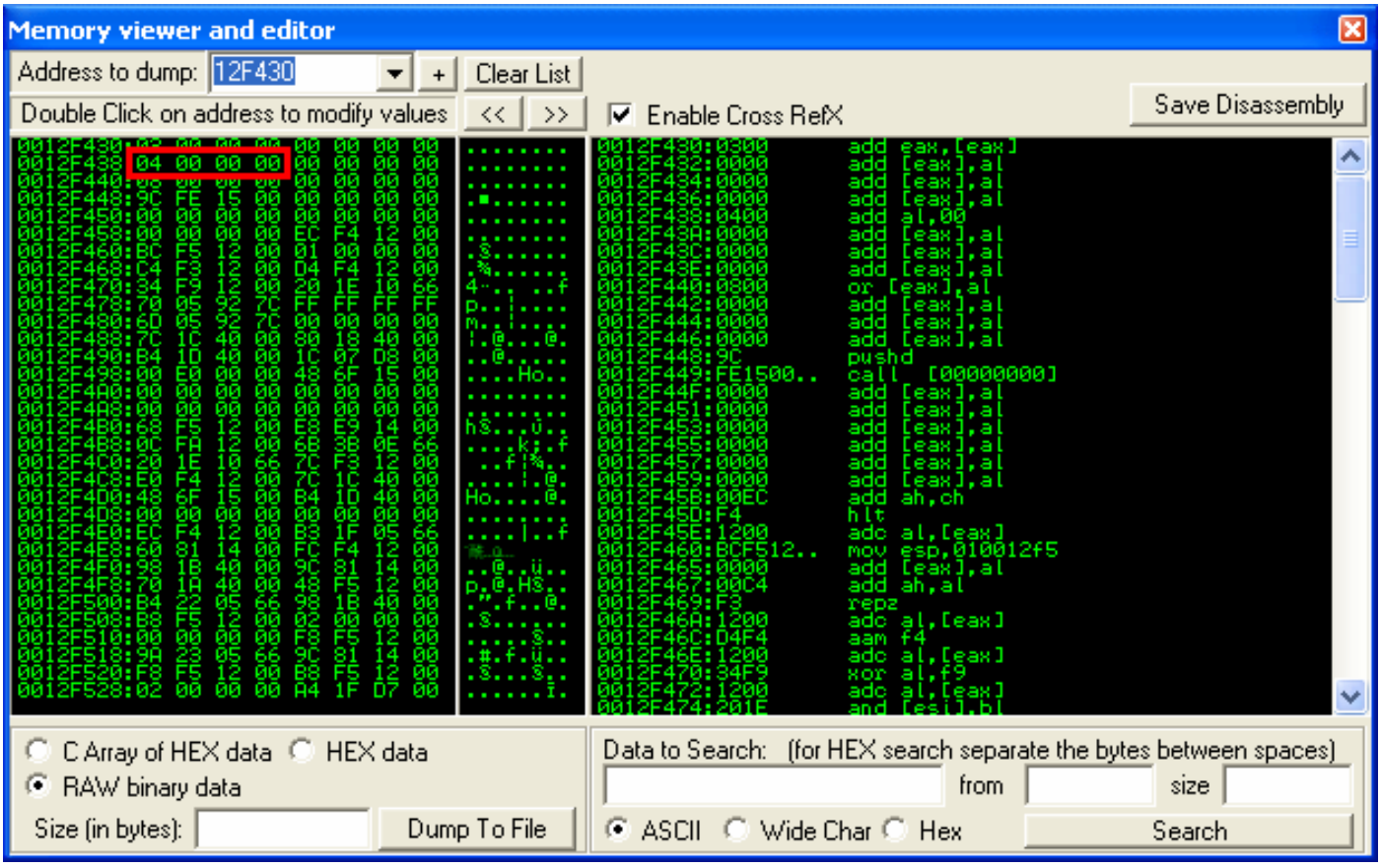
Ce qui nous intéresse se trouve une ligne en dessous. C'est l'adresse 0015FE9C entourée en rouge (il faut prendre les chiffres deux par deux en commençant par la droite). On saisit 0015FE9C dans le champ Address to dump :



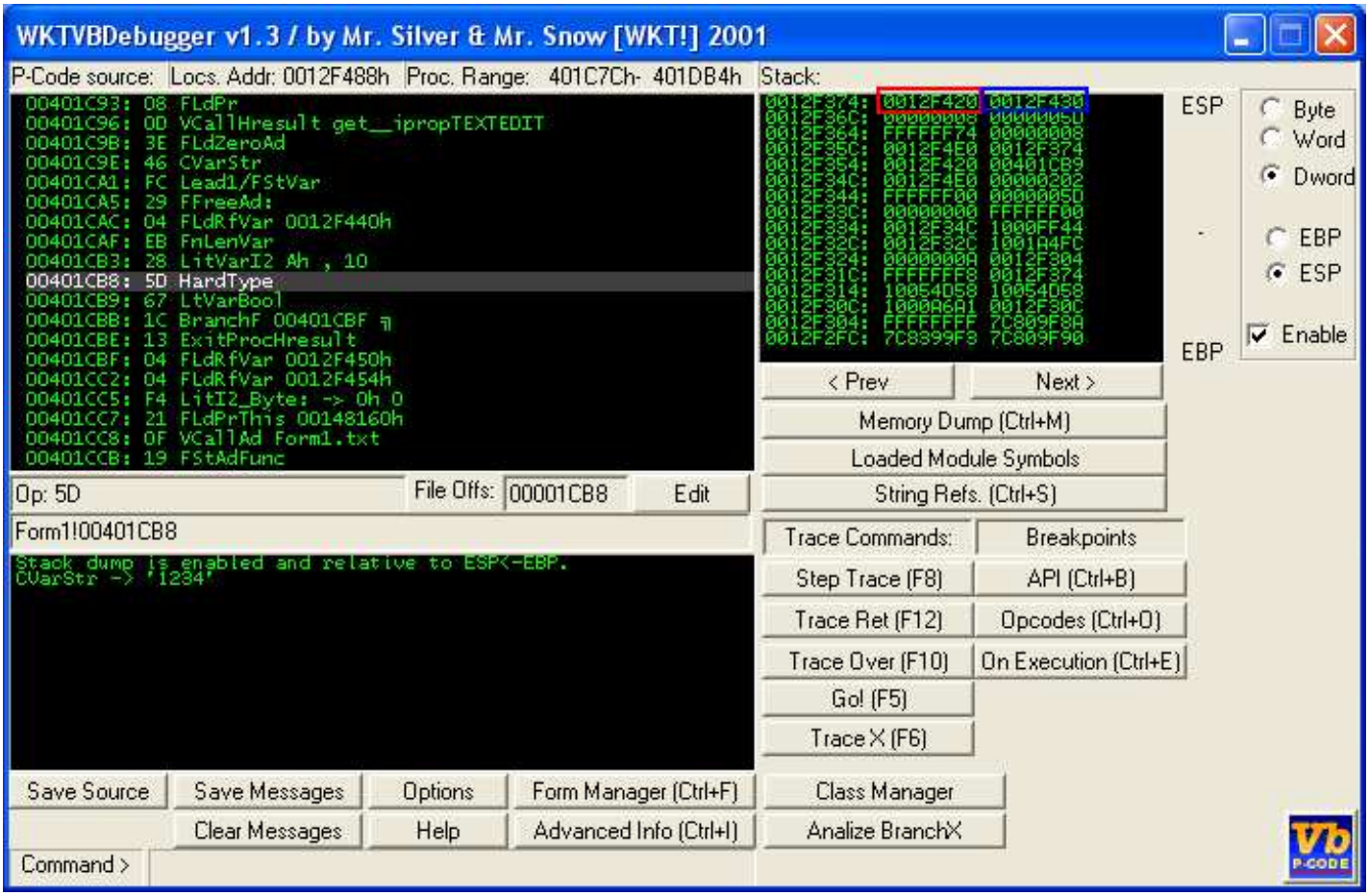
On trouve notre serial 1234 en unicode (zone entourée en rouge).
Conclusion, le paramètre passé à la fonction FnLen est notre serial stocké en 15FE9C
Notre serial 1234 comprend 4 caractères donc la fonction FnlLen devrait logiquement retourner 4 dans la pile. On exécute la ligne avec F8 pour voir. On obtient cet écran :



Le résultat de la fonction FnLenVar est stocké dans la première adresse de la pile 0012F430
On cliques sur « Memory Dump » et on rentre 0012F430. Voila ce qu'on obtient :



Comme tout à l'heure, le résultat est stocké dans la deuxième ligne. Il s'agit de 4 (et c'est bien le résultat qu'on attendait…) On exécute la ligne d'après avec F8
401CB3 LitVarI2 0012F420h Ah , 10
Cette instruction met la valeur A en hexadécimal (soit 10 en décimal) dans l'adresse 0012F420, et la pousse dans la pile. La pile ressemble donc à ça :



Deux valeurs ont été poussées sur la pile :
- 0012F420 (en rouge) contient 10 en décimal
- 0012F430 (en bleu) contient le nb de caractère de notre serial, ici 4 en décimal

On a ensuite les instructions suivantes :
00401CB8: 5D HardType
00401CB9: 67 LtVarBool
00401CBB: 1C BranchF 00401CBF
00401CBE: 13 ExitProcHresult

Il est probable que ces instructions fassent une comparaison entre les deux paramètres transmis à la pile (10 et 4). On exécute pas à pas jusqu'au BranchF.

C'est indiqué (No Jump), donc le saut ne va pas se faire. La ligne d'après s'exécute ExitProcHresult et le prog s'arrête. On a donc pas réussit à passer à aller bien loin. En fait on ne passe pas le test tant que le nb de caractère du serial est inférieur à 10. On re-essaye avec un serial de 10 caractères et on clique sur le bouton OK

Cette fois ci, on passe cette première condition, mais on arrive sur une deuxième condition :
(listing obtenu avec P32Dasm)

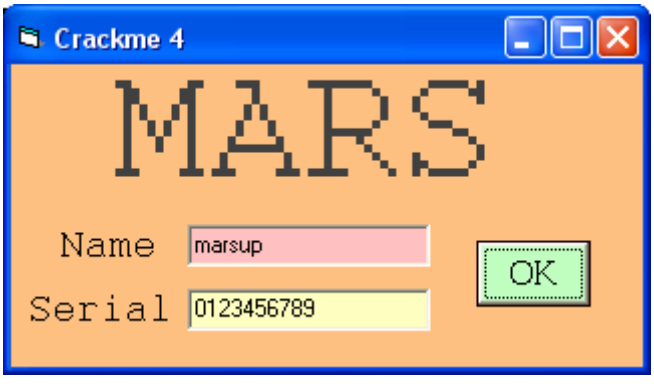
```
Form1 1.1 Command1.Click() ----
00001C7C:   04    FLdRfVar var_90
00001C7F:   04    FLdRfVar var_8C
00001C82:   F4    LitI2_Byte: 1 0x1
00001C84:   21    FLdPrThis
00001C85:   0F    VCallAd
00001C88:   19    FStAdFunc var_88
00001C8B:   08    FLdPr var_88
00001C8E:   0D    VCallHresult .(Index)
00001C93:   08    FLdPr var_8C
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 00001CBB
```

premiere condition sur la longueur du serial

```
00001CBF:   04    FLdRfVar var_90
00001CC2:   04    FLdRfVar var_8C
00001CC5:   F4    LitI2_Byte: 0 0x0
00001CC7:   21    FLdPrThis
00001CC8:   0F    VCallAd
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
```

deuxieme condition sur la longueur du nom

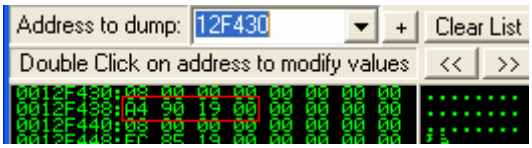
Cette deuxième partie ressemble beaucoup à la première partie. On récupère cette fois ci le nom avec TextBox.Get_Text() et on compare le nb de caractères du nom à 5. Si le nombre de caractères du nom est supérieur ou égal à 5, on passe la condition. On prend le nom marsup et le serial 0123456789



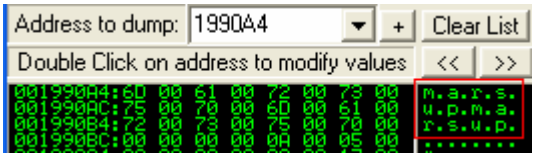
Une étrange fonction AddVar

Étudions les premières instructions :
00401D02: FLdRfVar 0012F400h → 0012F400= “marsup” est poussé sur la pile
00401D05: FLdRfVar 0012F400h → 0012F400= “marsup” est poussé sur la pile
00401D08: AddVar → que fait cette fonction ?

On exécute la fonction AddVar, et on regarde le résultat dans la pile
La valeur 0012F430 se retrouve sur la pile. On regarde cette valeur dans « Memory Dump »



On tape l’adresse entourée en rouge sur la ligne juste en dessous



On comprend donc maintenant ce qu’a fait la fonction AddVar : elle a additionnée le nom « marsup » deux fois pour faire un nom plus grand « marsupmarsup »

Les fonctions mid() et asc()

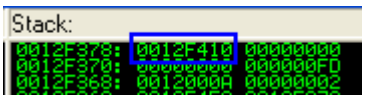
```
00001D0C:  FCF6 FStVar var_E0
00001D10:  28   LitVarI2: 1 0x1 var_110
00001D15:  04   FLdRfVar var_F0
00001D18:  28   LitVarI2: 10 0xA var_100
00001D1D:  FE68 ForVar For (counter = start) To (end)
00001D23:  28   LitVarI2: 1 0x1 var_B0
00001D28:  04   FLdRfVar var_F0
00001D2B:  FC22 CI4Var
00001D2D:  04   FLdRfVar var_A0
00001D30:  04   FLdRfVar var_D0
00001D33:  0A   ImpAdCallFPR4 Mid()
00001D38:  04   FLdRfVar var_D0
00001D3B:  FDFE 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:  FDFE CStrVarVal var_154
00001D60:  0B   ImpAdCallI2 Asc()
```

manipulation du serial

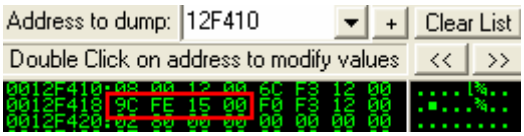
manipulation du nom

La suite du code c’est le début de boucle For Next avec des appels aux fonctions Mid et Asc
La partie entourée en rouge s’attaque ou serial, celle en bleu s’attaque au nom

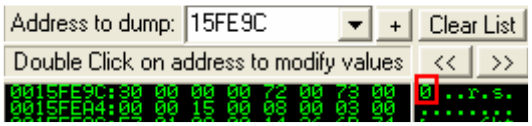
On cherche à comprendre ce que fait la fonction mid de dans la partie entourée en rouge.
Après avoir exécuté la fonction mid, on voit ceci tout en haut de la pile :



on tape l'adresse entourée de bleu dans la fenêtre « memory dump » et on a ça :

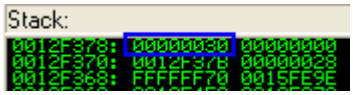


On tape l'adresse en rouge dans « memory dump »



La fonction mid() récupère donc simplement le premier caractère du serial « 0123456789 » donc 0

La fonction asc() qui suit récupère elle la valeur ascii de ce premier caractère 0. La valeur ascii de 0 est donc 30 en hexadécimal. On en a la confirmation quand on regarde la pile après l'exécution de la fonction asc()



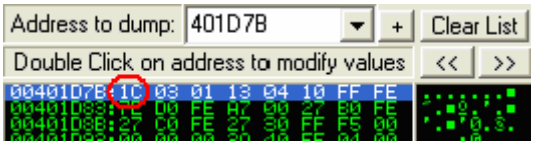
La partie rouge de la boucle For Next récupère donc le code ascii du premier caractère du serial
Que fait la partie bleue de la boucle For Next ?
→ Elle récupère le code ascii du premier caractère du nom. Pour la lettre « m », le code ascii est ainsi de 6D en hexadécimal

<u>La comparaison finale</u>

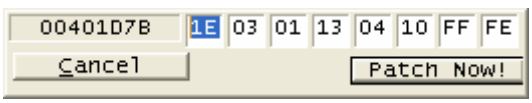
On étudie ensuite la fin de la boucle For Next. C'est la partie la plus importante puisqu'elle effectue la comparaison entre le serial et le nom. On va donc l'étudier en détail.

00401D65: F4 LitI2_Byte: -> 2h 2 -→ pousse 2 dans la pile
00401D67: A9 AddI2 → additionne 2 à quelque chose
00401D68: CB NeI2 → effectue la comparaison
00401D69: 32 FFreeStr
00401D70: 36 FFreeVar -> 4
00401D7B: 1C BranchF 00401D7F → saut conditionnel
00401D7E: 13 ExitProcHresult → fin du programme si on ne saute pas
00401D7F: 04 FLdRfVar 0012F3F0h
00401D82: FE Lead3/NextStepVar

On peut déjà avoir une idée pour cracker ce crackme. On peut transformer le BranchF en Branch pour éviter le ExitProc qui suit
En p-code vb, l'opcode de l'instruction BranchF est 1C, et celui de Branch est 1E
On va donc remplacer l'opcode 1C par 1E
On clique sur « Memory dump » on entre l'adresse de l'instruction à modifier 00401D7B



on double clique sur l'opcode 1C, et dans la fenêtre qui s'affiche on rentre 1E

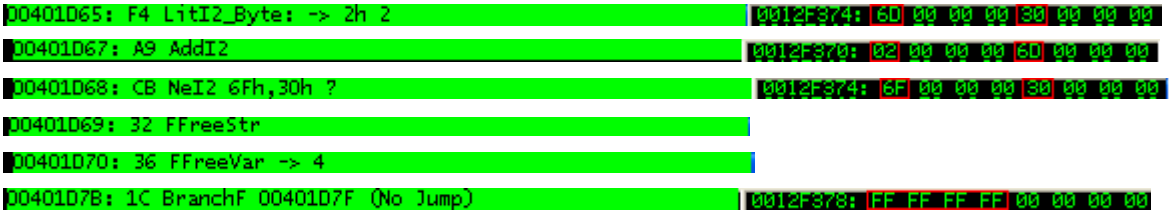


On clique sur Patch Now. → la modification est donc faite en mémoire. Si on veut que cette modification soit définitive, il faut le faire en dur avec un éditeur hexadécimal par exemple.

Remarque :
Pour inverser un BranchF en BranchT, on peut modifier l'opcode 1C en 1D
Pour noper une instruction, on remplace l'opcode par 00 (équivalent du 90 en assembleur)

La comparaison finale en détail

On vient de cracker le programme, maintenant ça serait bien de savoir ce qui est comparé pour pouvoir trouver un serial pour notre nom. Voila les premières instructions de la comparaison.



A la première ligne, on 6D et 30 dans la pile. Cela correspond au code ascii de la première lettre de notre nom « m » et du premier chiffre de notre serial « 0 »
L'instruction LitI2 2h 2 de la première ligne va ajouter la valeur 2 dans la pile

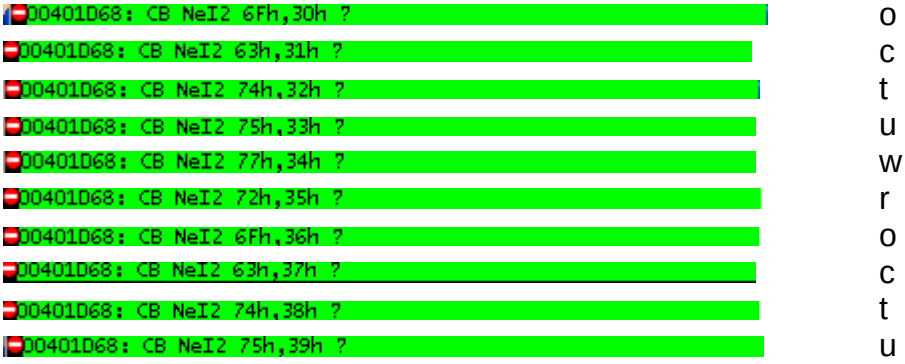
A la deuxième ligne, on a 2 et 6D (code ascii de « m ») dans la pile.
L'instruction AddI2 va ajouter les deux premiers paramètres de la pile→ =6F

A la troisième ligne, on a 6F (somme de 2 et 6D) et 30 (code ascii de « 0 ») dans la pile
L'instruction NeI2 va comparer les deux premiers paramètres de la pile 6F et 30

La quatrième et cinquième ligne ne font rien

A la sixième ligne, on a FFFFFFFF dans la pile
6F est différent de 30 donc la comparaison a échoué.

Si la comparaison entre la première lettre du nom et du serial est bonne, on effectue une comparaison entre la deuxième lettre du nom et du serial et ainsi de suite jusqu'à la 10eme lettre. On modifie le BranchF en Branch, on met un breakpoint sur la ligne 00401D68 NeI2 (en double cliquant sur la ligne)
On clique sur F5, et on regarde à chaque fois la comparaison :



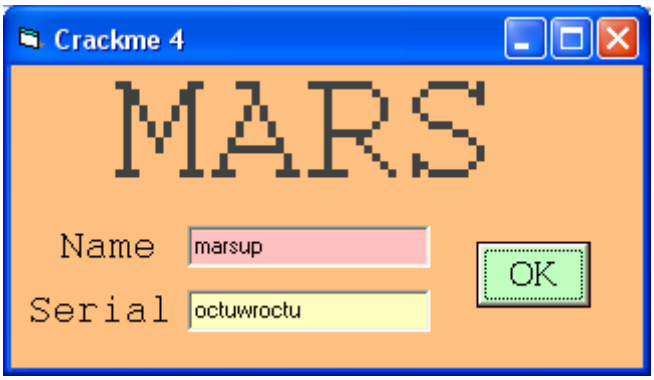
o
c
t
u
w
r
o
c
t
u

au lieu de rentrer 0123456789 comme serial, il faut donc rentrer le serial suivant :
octuwroctu
Pour trouver ce serial, on s'aide d'une table de correspondance entre code hexadécimal et caractère ascii

caractere ascii code hexa

a	61
b	62
c	63
d	64
e	65
f	66
g	67
h	68
i	69
j	6A
k	6B
l	6C
m	6D
n	6E
o	6F
p	70
q	71
r	72
s	73
t	74
u	75
v	76
w	77
x	78
y	79
z	7A

On rentre le serial, et on obtient le message de réussite



Voila la portion de code qui s’exécute pour afficher le message de réussite

```
00001D88:  27   LitVar_Missing
00001D8B:  27   LitVar_Missing
00001D8E:  27   LitVar_Missing
00001D91:  F5   LitI4: 0 0x0
00001D96:  3A   LitVarStr: "serial correct"
00001D9B:  4E   FStVarCopyObj var_B0
00001D9E:  04   FLdRfVar var_B0
00001DA1:  0A   ImpAdCallFPR4 MsgBox()
00001DA6:  36   FFreeVar var_B0 var_D0 var_140 var_150
00001DB1:  13   ExitProcHresult
```

Comment trouver n'importe quel serial

On prend notre nom « marsup » on le double « marsupmarsup »
On prend la première lettre du nom « m » et on ajoute 2 a son code ascii → on obtient la lettre « o »
Pour trouver le serial, on a donc un décalage de deux lettres par rapport au nom :
m→o
a→c
r→t
s→u
u →w
p→r
m→o
a →c
r→t
s→u

Bonus, le code source du crackme 4

```
Public Sub Command1_Click()
serie = txt(1).Text
If Len(serie) < 10 Then Exit Sub
nom = txt(0).Text
If Len(nom) < 5 Then Exit Sub
nom = nom + nom
For i = 1 To 10
If Asc(Mid(serie, i, 1)) <> Asc(Mid(nom, i, 1)) + 2 Then Exit Sub
Next
MsgBox "serial correct"
End Sub
```

Remerciements

Merci à Jericho et à Beginner pour avoir rédigé un tutorial sur ce crackme
Un coucou à tous les membres de FC