

# implanter un keygen dans un logiciel

## Plan du tutorial:

### Introduction

#### I) faire fonctionner le keygen de winzip dans le code d'EasyBanner

- 1) Copier le keygen avec ollydbg et le mettre dans le début du code de Easybanner
- 2) Modifier l'Etry Point du prog
- 3) Modifier les imports qui pointent sur rien
- 4) Corriger les références à la section data

#### II) faire un keygen fonctionnel dans EasyBanner

- 1) trouver la fonction qui calcule le serial et les paramètres à lui donner
- 2) adapter notre fichier keygen.exe
- 3) corriger la pile
- 4) pourquoi la partie du milieu est fausse et ne bouge pas en fonction du nom?
- 5) obtenir une troisieme partie de sérial correcte

### Conclusion

## Introduction

je cherchais un moyen de keygenner EasyBanner sans trop me fatiguer  
j'étais en train de lire le tut de Lise\_Grim sur la **Transplantation Keygennique**  
<http://nfotemple.free.fr/index2.html>

le seul problème avec Easybanner, c'est qu'il y a beaucoup de call dans ce prog  
quand on copie la routine avec ollydbg ou IDA, ça copie pas les call

je me suis donc intéressé à un outil dont parle Lise\_Grim TMG Ripper Studio  
<http://www.tuts4you.com/request.php?456>

ce prog permet de ripper la routine de création du serial --> il suffit juste de lui indiquer le début de la routine et les procédures à ne pas intégrer --> ensuite reste plus qu'à copier ce code dans un keygen comme dans l'exemple fourni avec winzip 8.0 et compiler avec MASM (pour télécharger winzip 8.0 <http://www.oldversion.com/download/winzip80.exe>)

Le problème, c'est que je maîtrise pas trop MASM et que j'ai vite des erreurs de compilation --> en fait, je préfère souvent coder en ASM directement dans ollydbg

Par contre j'ai beaucoup aimé le code minimal du Keygen (sans ressources et en très peu de lignes). Je me suis demandé s'il était possible d'intégrer ce Keygen dans le début de EasyBanner et de faire ensuite appel directement à la routine de création du serial d'EasyBanner pour afficher le serial --> la réponse est évidemment oui....

## D) faire fonctionner le keygen de winzip dans le code d'EasyBanner

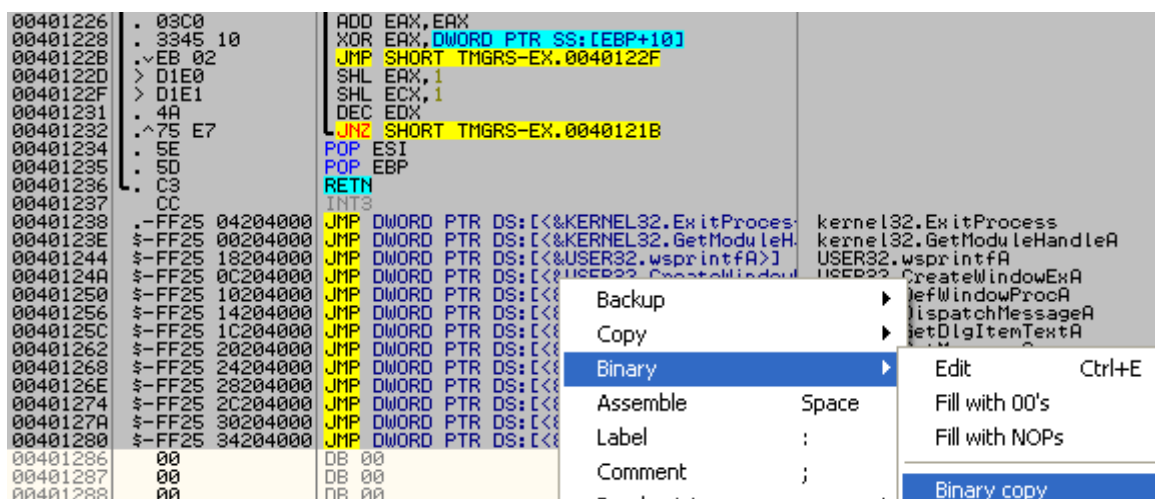
////////// PAR QUOI COMMENCER? //////////

On veut que le ce keygen implanté dans le code d'EasyBanner fonctionne.

On va copier le code du Keygen dans Easybanner et modifier l'EP d'Easybanner

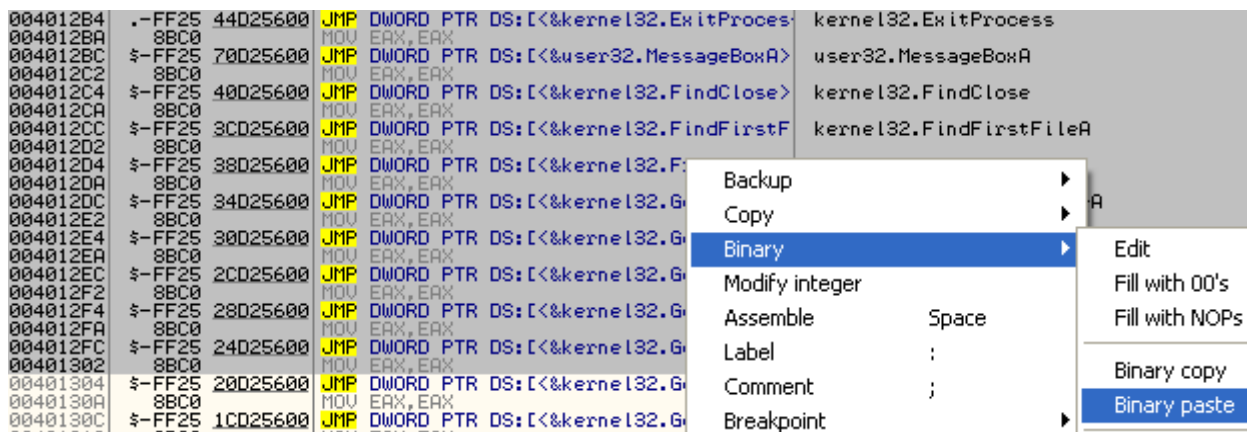
### 1) Copier le keygen avec ollydbg et le mettre dans le début du code de Easybanner

On ouvre le fichier TMGRS-EX.exe avec ollydbg et on sélectionne le code du crackme (adresses de 401000 à 401280), clic droit, binary, binary copy



on ouvre le fichier Easybanner avec ollydbg, on va tout au début du code en 401000, on sélectionne une partie de code suffisamment grande (adresses de 401000 à 401302 par exemple) clic droit, binary, binary paste  
On doit le copier impérativement avec une adresse de départ à 401000 car les adresses sont fixes et cela ne marcherait pas sinon. Le fait d'écraser une partie de EasyBanner dans ces adresses n'est pas gênant car on ne veut conserver que la partie d'EasyBanner qui contrôle ou génère le serial

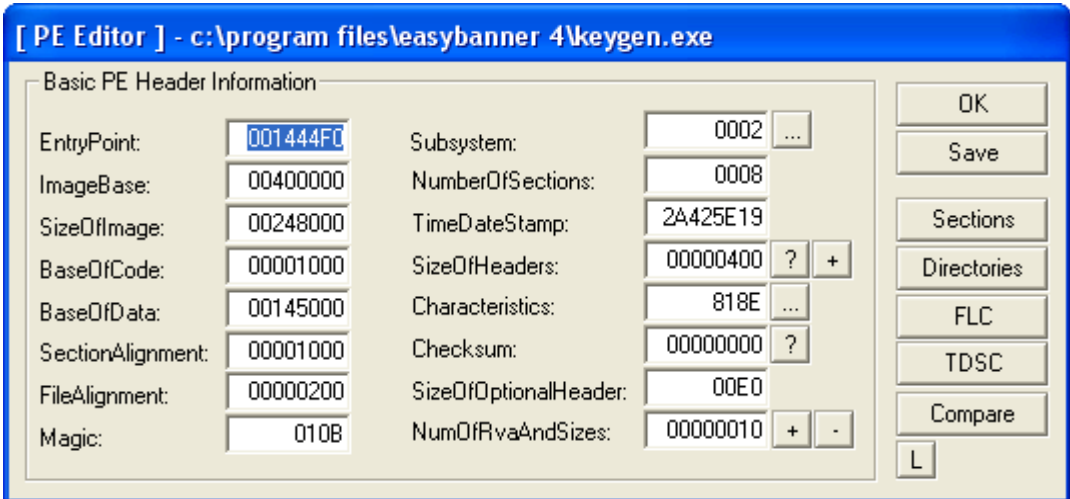
Remarque: si on veut faire quelque chose de plus propre, on peut essayer de coder le Keygen avec du code relogeable --> comme ça on peut le coller n'importe où  
<http://assembly.ifrance.com/CodeReloageable.txt>



On sauve tout ça avec ollydbg --> clic droit copy to executable, copy all, oui pour le message d'avertissement, clic droit dans la nouvelle fenetre, save file et on le nome Kegen.exe

## 2) Modifier l'Etry Point du prog

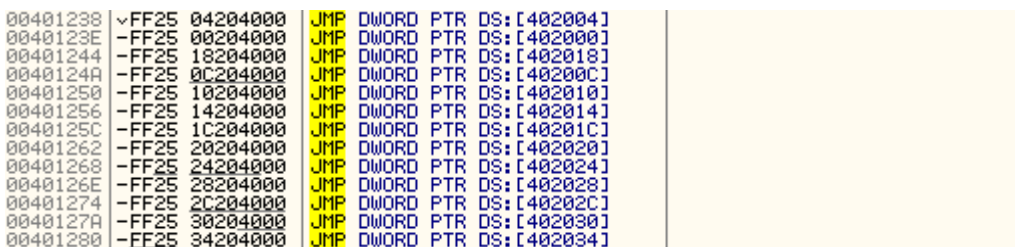
c'est pas trop compliqué ça --> l'EP est à 005444F0, et nous, on veut que Keygen.exe démarre au début de notre code en 00401000 --> on prend LordPE, on selectionne Keygen.exe et dans le champ Entry Point, on mets 00001000 à la place (soit 401000 – 400000)



//////////////////// OUI MAIS CA MARCHE PAS !!!! //////////////////////

si on lance notre exécutable, ça marche pas --> pas trop étonnant en fait ;)  
on ouvre Key.gen.exe avec ollydbg, (on interrompt l'analyse en appuyant sur la barre d'espace) et on voit qu'il faut qu'on modifie aussi deux choses: les imports et les références aux datas.

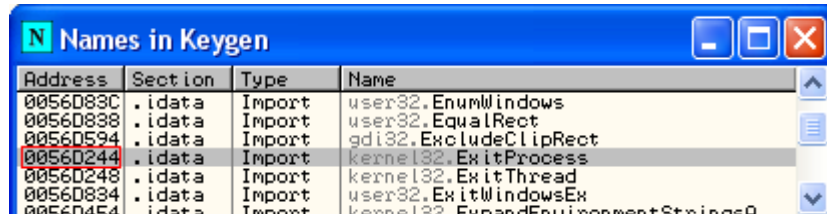
## 3) modifier les imports qui pointent sur rien:



dans le fichier TMGRS-EX.exe, on avait ça au niveau des imports:

00401238	.-FF25 04204000	JMP	DWORD	PTR	DS:[<&KERNEL32.ExitProcess>]	kernel32.ExitProcess
0040123E	.-FF25 00204000	JMP	DWORD	PTR	DS:[<&KERNEL32.GetModuleHandleA>]	kernel32.GetModuleHandleA
00401244	.-FF25 18204000	JMP	DWORD	PTR	DS:[<&USER32.wsprintfA>]	USER32.wsprintfA
0040124A	.-FF25 0C204000	JMP	DWORD	PTR	DS:[<&USER32.CreateWindowExA>]	USER32.CreateWindowExA
00401250	.-FF25 10204000	JMP	DWORD	PTR	DS:[<&USER32.DefWindowProcA>]	USER32.DefWindowProcA
00401256	.-FF25 14204000	JMP	DWORD	PTR	DS:[<&USER32.DispatchMessageA>]	USER32.DispatchMessageA
0040125C	.-FF25 1C204000	JMP	DWORD	PTR	DS:[<&USER32.GetDlgItemTextA>]	USER32.GetDlgItemTextA
00401262	.-FF25 20204000	JMP	DWORD	PTR	DS:[<&USER32.GetMessageA>]	USER32.GetMessageA
00401268	.-FF25 24204000	JMP	DWORD	PTR	DS:[<&USER32.PostQuitMessage>]	USER32.PostQuitMessage
0040126E	.-FF25 28204000	JMP	DWORD	PTR	DS:[<&USER32.RegisterClassExA>]	USER32.RegisterClassExA
00401274	.-FF25 2C204000	JMP	DWORD	PTR	DS:[<&USER32.SetDlgItemTextA>]	USER32.SetDlgItemTextA
0040127A	.-FF25 30204000	JMP	DWORD	PTR	DS:[<&USER32.ShowWindow>]	USER32.ShowWindow
00401280	.-FF25 34204000	JMP	DWORD	PTR	DS:[<&USER32.TranslateMessage>]	USER32.TranslateMessage

il faut donc qu'on se débrouille pour les sauts pointent sur les imports  
pour la première ligne --> jmp dword ptr ds: [402004] il faut qu'elle pointe sur  
Kernel32.ExitProcess --> clic droit dans ollydbg, search for, name (label) in current module  
et on voit les imports classés par ordre alphabétique



Pour faire référence à ExitProcess, il faut remplacer le jmp dword ptr ds: [402004]  
par un jmp dword ptr ds: [56D244] (valeur entourée en rouge)  
On fait pareil pour tous les imports qu'on trouve

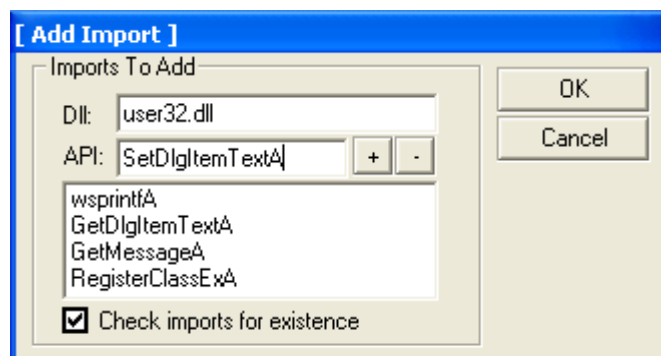
00401238	.-FF25 44D25600	JMP	DWORD	PTR	DS:[<&kernel32.ExitProcess>]	JMP	DWORD	PTR	DS:[56D244]
0040123E	.-FF25 24D25600	JMP	DWORD	PTR	DS:[<&kernel32.GetModuleHandleA>]	JMP	DWORD	PTR	DS:[56D224]
00401244	.-FF25 18204000	JMP	DWORD	PTR	DS:[402018]				
0040124A	.-FF25 04D65600	JMP	DWORD	PTR	DS:[<&user32.CreateWindowExA>]	JMP	DWORD	PTR	DS:[56D604]
00401250	.-FF25 90D85600	JMP	DWORD	PTR	DS:[<&user32.DefWindowProcA>]	JMP	DWORD	PTR	DS:[56D890]
00401256	.-FF25 78D85600	JMP	DWORD	PTR	DS:[<&user32.DispatchMessageA>]	JMP	DWORD	PTR	DS:[56D878]
0040125C	.-FF25 1C204000	JMP	DWORD	PTR	DS:[40201C]				
00401262	.-FF25 20204000	JMP	DWORD	PTR	DS:[402020]				
00401268	.-FF25 04D65600	JMP	DWORD	PTR	DS:[<&user32.PostQuitMessage>]	JMP	DWORD	PTR	DS:[56D6D4]
0040126E	.-FF25 28204000	JMP	DWORD	PTR	DS:[402028]				
00401274	.-FF25 2C204000	JMP	DWORD	PTR	DS:[40202C]				
0040127A	.-FF25 34D65600	JMP	DWORD	PTR	DS:[<&user32.ShowWindow>]	JMP	DWORD	PTR	DS:[56D634]
00401280	.-FF25 24D65600	JMP	DWORD	PTR	DS:[<&user32.TranslateMessage>]	JMP	DWORD	PTR	DS:[56D624]

La il nous manque 5 imports qu'on trouve pas dans le prog:

```
00401244 JMP DWORD PTR DS:[<&USER32.wsprintfA>]
0040125C JMP DWORD PTR DS:[<&USER32.GetDlgItemTextA>]
00401262 JMP DWORD PTR DS:[<&USER32.GetMessageA>]
0040126E JMP DWORD PTR DS:[<&USER32.RegisterClassExA>]
00401274 JMP DWORD PTR DS:[<&USER32.SetDlgItemTextA>]
```

*Remarque:* on pourrait entrer un JMP USER32.wsprintfA directement dans ollydbg mais cela ne fonctionnerait que sur notre ordinateur car cela crée un jmp direct vers la fonction (chez moi un JMP 77D1A2DE) et l'adresse est différente pour chacun.

Pour ajouter les fonctions qui manquent, on charge Keygen.exe avec LordPe, on clique sur Directories, puis sur le bouton [...] en face d'import table, clic droit sur une dll, add import et dans la fenêtre qui s'affiche on met user32.dll dans le champ DLL, et dans le champ API, on rentre nos fonctions en cliquant sur le bouton [+] pour les ajouter l'une après l'autre dans la fenêtre du bas



[ ImportTable ]					
DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk
comdlg32.dll	00000000	00000000	00000000	0016FD58	0016D9DC
winmm.dll	00000000	00000000	00000000	0016FD9E	0016D9EC
user32.dll	00000000	00000000	00000000	0016FDD4	0016D9FC
kernel32.dll	00000000	00000000	00000000	0016FF3A	0016DA48
user32.dll	00000000	00000000	00000000	00248000	0024805C
ThunkRVA	ThunkOffset	ThunkValue	Hint	ApiName	
0024805C	0022085C	0024800B	0000	wsprintfA	
00248060	00220860	00248017	0000	GetDlgItemTextA	
00248064	00220864	00248029	0000	GetMessageA	
00248068	00220868	00248037	0000	RegisterClassExA	
0024806C	0022086C	0024804A	0000	SetDlgItemTextA	
Number Of Thunks: 5h / 5d (FirstThunk chain)					<input type="checkbox"/> View always FirstThunk

On valide sur Ok et on sauve--> LordPe vient de nous créer une nouvelle section .Silvana avec nos cinq imports --> il reste plus qu'à modifier les cinq jmp dword ptr ds: [xxxxxxx] qui restent  
 Dans ollydbg clic droit, search for Name in current module --> on cherche le premier import wsprintfA

0056D2F8	.idata	Import	kernel32.WritePrivateProfileStringA
0056D2F4	.idata	Import	kernel32.WriteProcessMemory
0064805C	.Silvana	Import	user32.wsprintfA

clic droit sur wsprintfA et follow in dump. Pour que les imports soient visibles dans la fenetre de dump, clic droit long, Adress with ASCII dump et on retrouve les adresses de nos 5 imports:

00648038	67655200	.Reg	
0064803C	65747369	iste	
00648040	616C4372	rCla	
00648044	78457373	ssEx	
00648048	00000041	A...	
0064804C	44746553	SetD	
00648050	7449676C	lgIt	
00648054	65546D65	emTe	
00648058	00417478	xtA.	
0064805C	77D1A2DE	id0w	user32.wsprintfA
00648060	77D6AC06	*%iw	user32.GetDlgItemTextA
00648064	77D3EA45	E0ew	user32.GetMessageA
00648068	77D24315	SCew	user32.RegisterClassExA
0064806C	77D360D5	'ew	user32.SetDlgItemTextA
00648070	00000000	....	

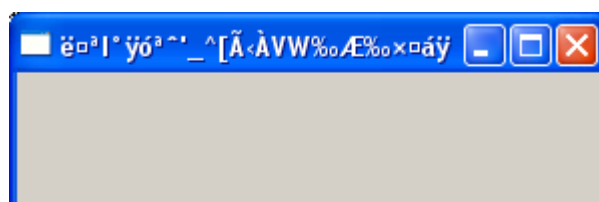
Pour regler définitivement le problème des imports, on modifie le code comme ceci dans ollydbg:

00401238	-FF25 44D25600	JMP DWORD PTR DS:[&kernel32.ExitProcess]	
0040123E	-FF25 24D25600	JMP DWORD PTR DS:[&kernel32.GetModuleHandleA]	
00401244	-FF25 5C806400	JMP DWORD PTR DS:[&user32.wsprintfA]	JMP DWORD PTR DS:[64805C]
0040124A	-FF25 04D65600	JMP DWORD PTR DS:[&user32.CreateWindowExA]	
00401250	-FF25 90D85600	JMP DWORD PTR DS:[&user32.DefWindowProcA]	
00401256	-FF25 78D85600	JMP DWORD PTR DS:[&user32.DispatchMessageA]	
0040125C	-FF25 60806400	JMP DWORD PTR DS:[&user32.GetDlgItemTextA]	JMP DWORD PTR DS:[648060]
00401262	-FF25 64806400	JMP DWORD PTR DS:[&user32.GetMessageA]	JMP DWORD PTR DS:[648064]
00401268	-FF25 D4D65600	JMP DWORD PTR DS:[&user32.PostQuitMessage]	
0040126E	-FF25 68806400	JMP DWORD PTR DS:[&user32.RegisterClassExA]	JMP DWORD PTR DS:[648068]
00401274	-FF25 6C806400	JMP DWORD PTR DS:[&user32.SetDlgItemTextA]	JMP DWORD PTR DS:[64806C]
0040127A	-FF25 34D65600	JMP DWORD PTR DS:[&user32.ShowWindow]	
00401280	-FF25 24D65600	JMP DWORD PTR DS:[&user32.TranslateMessage]	

Remarque: Au lieu de recréer une section avec des imports, on peut aussi modifier le nom de fonctions existantes dont on ne se sert pas pour les renommer en wsprintfA, GetDlgItemTextA.. --> pour faire ça, clic droit sur une fonction, Edit, et on change l'Api name, mais il faut que le nom de la fonction ai autant de lettres ou plus que celui qu'on veut mettre.

#### 4) Corriger les références à la section data

Quand on execute le crackme, on obtient ça:





Dans le prog original, toutes les références aux noms, ou les buffers font référence à la section .data du prog (adresses de 403000 à 403FFF):

Dans notre fichier keygen.exe, les adresses en 403000 ne font plus référence à rien:

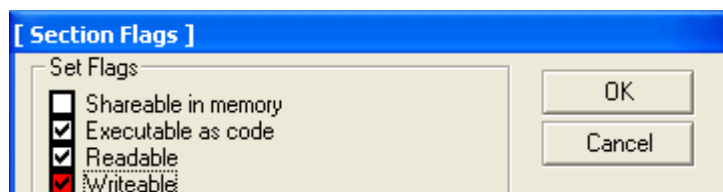
voilà le contenu de la section .data dans le fichier original

Dans notre fichier keyen.exe, on trouve du code de EasyBanner à cette adresse:

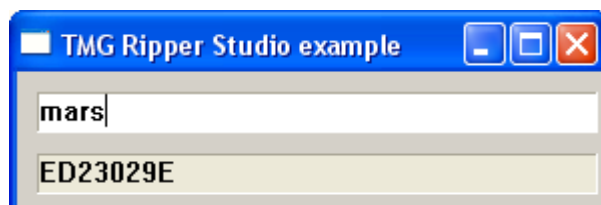
on va donc copier le contenu de la section .data originale dans notre fichier keygen.exe (juste les adresses de 403000 à 403060) Pour ça, clic droit dans la fenetre de dump, binary, binary copy et binary paste dans le fichier keygen.exe et copy to executable file.

Il reste un dernier détail pour que tout fonctionne --> mettre la section code qui s'étend des adresses 401000 à 545000 en writeable car sinon, quand notre keygen va vouloir écrire dans le buffer en 0040303C, cela va créer une erreur « Access violation when writting to [0040303C] » dans ollydbg.

Dans LordPe, on clique sur sections, sur la section CODE, clic droit, edit sections header et on clique sur le bouton [...] dans la fenetre qui s'affiche. On clique sur Writeable et on sauve:



Bon, maintenant, on lance Keygen.exe et on tape notre nom pour obtenir un serial valide pour winzip 8.0



Le keygen a l'air de marcher aussi bien que l'original : )

## II) faire un keygen fonctionnel dans EasyBanner

### 1) trouver la fonction qui calcule le serial et les paramètres à lui donner

On ouvre Easybanner avec ollydbg, et grâce aux indications des membres du forum, on arrive rapidement sur la routine qui vérifie le serial --> le call 52DAA0

```
0052E4C6 MOV EAX,DWORD PTR DS:[EBX+308] *TRegform.username:Tedit
0052E4CC CALL EasyBann.00478688 ->Controls.TControl.GetText(TControl):TCaption;
0052E4D1 MOV EAX,DWORD PTR SS:[EBP-24]
0052E4D4 POP EDX
0052E4D5 CALL EasyBann.0052DAA0 le serial est calcule la dedans
```

A la ligne 52E4D5 (c'est à dire avant d'entrer dans le call), on a ça dans les registres et dans la pile:

```
Registers (FPU)
EAX 00C2EEB8 ASCII "MARSUP"
ECX 0012EA30
EDX 00C3B904 ASCII "11111111111111111111"
EBX 00C04000
ESP 0012EC38
EBP 0012EC84
ESI 00C0C49C
EDI 0012EE00
EIP 0052E4D5 EasyBann.0052E4D5

0012EC38 0012EFB4 Pointer to next SEH record
0012EC3C 0052E68F SE handler
0012EC40 0012EC84
0012EC44 0046F804 Entry address
0012EC48 00C0BD6C
0012EC4C 00000000
0012EC50 00000000
0012EC54 00000000
0012EC58 00000000
0012EC5C 00000000
0012EC60 00C2EEB8 ASCII "MARSUP"
0012EC64 00C3B904 ASCII "11111111111111111111"
0012EC68 00C3B8E0 ASCII "11111111111111111111"
0012EC6C 00C0C0B4 ASCII "MARSUP"
0012EC70 00C3B898 ASCII "11111111111111111111"
0012EC74 00C3B8BC ASCII "11111111111111111111"
0012EC78 00C1E44C ASCII "MARSUP"
0012EC7C 00C0DB6C ASCII "MARSUP"
0012EC80 00000000
```

et quand on rentre dans le call, on voit notre serial pour MARSUP dans la pile à cette adresse:  
0052DB73 MOV EAX,DWORD PTR SS:[EBP-14]

```
0012EC38 Pointer to next SEH record
0052DBAA SE handler
0012EC30
00C04000
00C0FA04 ASCII "MARSUP"
00C3B948 ASCII "9CAED7791D1B748C"
00C3B928 ASCII "8A17D171F9AE4991"
00C3B968 ASCII "EB40-8A17D171F9AE4991-9CAED7791D1B748C"
79D7AE9C
8C741B1D
00C3B904 ASCII "11111111111111111111"
00C0FA04 ASCII "MARSUP"
```

La question qu'on se pose: quels sont les parametres donc le call 52DAA0 (call de vérif du serial) à besoin pour calculer le bon sérial et effectuer la vérif? --> est ce qu'il a besoin de plein de paramètres ou seulement de deux?  
--> Il suffit d'essayer pour savoir...

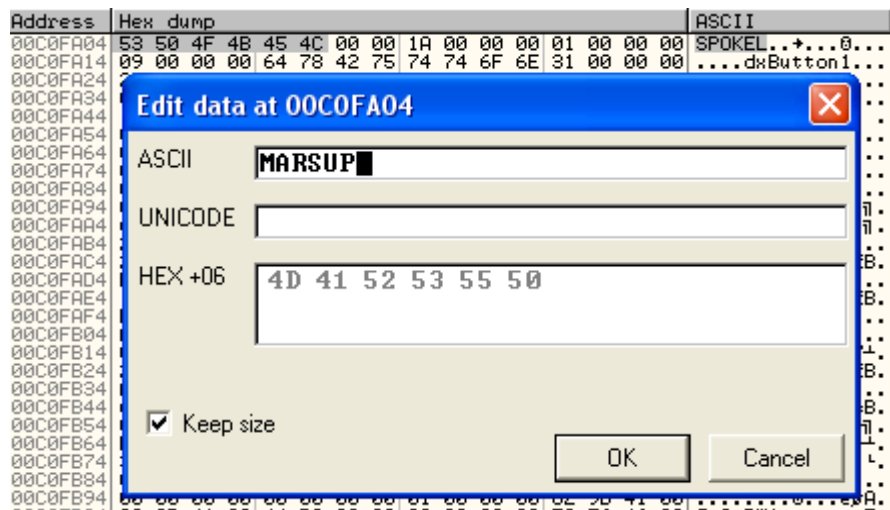
Pour le nom SPOKEL, le bon serial est "EB40-CCB78F0B436F2392-9CAED7791D1B748C"  
On rentre comme nom SPOKEL et on met un bp sur le call 52DAA0

voilà ce qu'on a dans les registres et dans la pile:

```
Registers (FPU)
EAX 00C0FA04 ASCII "SPOKEL"
ECX 0012EA30
EDX 00C3B904 ASCII "11111111111111111111"
EBX 00C04000
ESP 0012EC38
EBP 0012EC84
ESI 00C0C49C
EDI 0012EE00
EIP 0052E4D5 EasyBann.0052E4D5

0012EC38 0012EFB4 Pointer to next SEH record
0012EC3C 0052E68F SE handler
0012EC40 0012EC84
0012EC44 0046F804 Entry address
0012EC48 00C0BD6C
0012EC4C 00000000
0012EC50 00000000
0012EC54 00000000
0012EC58 00000000
0012EC5C 00000000
0012EC60 00C0FA04 ASCII "SPOKEL"
0012EC64 00C3B904 ASCII "11111111111111111111"
0012EC68 00C3B8E0 ASCII "11111111111111111111"
0012EC6C 00C0C0B4 ASCII "SPOKEL"
0012EC70 00C3B898 ASCII "11111111111111111111"
0012EC74 00C3B8BC ASCII "11111111111111111111"
0012EC78 00C0DB6C ASCII "SPOKEL"
0012EC7C 00C2EEB8 ASCII "SPOKEL"
0012EC80 00000000
```

--> on cliques sur le registre eax, clic droit follow in dump et on modifie SPOKEL en MARSUP



On execute dans ollydbg, et on regarde si cela a modifié quelque chose:  
le bon sérial est devenu ASCII "EB40-8A17D171F9AE4991-9CAED7791D1B748C" c'est a dire  
le sérial pour MARSUP

ça à l'air très simple --> le call 52DAA0 semble avoir besoin de deux paramètres pour effectuer sa vérification:

- notre nom dans eax
- notre serial dans edx

## 2) adapter notre fichier keygen.exe

On reprend notre fichier Keygen.exe --> comment affiche t'il le serial pour winzip?

```
00401167 PUSH 1E
00401169 PUSH TMGRS-EX.0040303C
0040116E PUSH 64
00401170 PUSH ESI
00401171 CALL <JMP.&USER32.GetDlgItemTextA>
00401176 TEST EAX,EAX
00401178 JE SHORT TMGRS-EX.00401199
0040117A PUSH TMGRS-EX.0040305A
0040117F PUSH TMGRS-EX.0040303C
00401184 CALL TMGRS-EX.0040119A
00401189 ADD ESP,8
0040118C PUSH TMGRS-EX.0040305A
00401191 PUSH 65
00401193 PUSH ESI
00401194 CALL <JMP.&USER32.SetDlgItemTextA>

[Count = 1E (30.)
Buffer = TMGRS-EX.0040303C
ControlID = 64 (100.)
hWnd
GetDlgItemTextA

[Arg2 = 0040305A
Arg1 = 0040303C
TMGRS-EX.0040119A

[Text = ""
ControlID = 65 (101.)
hWnd
SetDlgItemTextA
```



La fonction GetDlgItemTextA récupère le serial entré dans le buffer en 40303C  
 Ensuite on pousse deux parametres dans la pile: 40305A, l'endroit ou on veut qu'il écrive le bon serial et 40303C le buffer qui contient notre nom. On appelle ensuite le call 40119A qui est chargé de calculer le bon sérial et de le stocker à l'adresse 40305A  
 la fonction SetDlgItemTextA est ensuite chargée d'afficher le bon serial qui se trouve à l'adresse 40305A

Deja, on peut noper le code entre les adresses 40119A et 401236 puisque c'est la procédure de calcul du serial dans winzip et qu'on en a plus besoin ....

voilà comment on peut modifier notre code (modifications en rouge):

00401167	PUSH 1E	Count = 1E (30.)
00401169	PUSH TMGRS-EX.00403040	Buffer = TMGRS-EX.00403040
0040116E	PUSH 64	ControlID = 64 (100.)
00401170	PUSH ESI	hWnd
00401171	CALL <JMP.&USER32.GetDlgItemTextA>	GetDlgItemTextA
00401176	TEST EAX,EAX	
00401178	JE SHORT TMGRS-EX.00401199	
0040117A	MOV EAX, TMGRS-EX.00403040	ASCII "TMG Ripper Studio example"
0040117F	MOV EDX, TMGRS-EX.00403010	
00401184	CALL 0052DAA0	
00401189	ADD ESP, 8	
0040118C	PUSH EAX	Text
0040118D	NOP	
0040118E	NOP	
0040118F	NOP	
00401190	NOP	
00401191	PUSH 65	ControlID = 65 (101.)
00401193	PUSH ESI	hWnd
00401194	CALL <JMP.&USER32.SetDlgItemTextA>	SetDlgItemTextA

0052DB73	MOV EAX, DWORD PTR SS:[EBP-14]
0052DB76	JMP EasyBann.00401189

explications : on met notre nom (récupéré en 403040) dans eax, et on met notre serial dans edx (ici j'ai mis n'importe quoi « TMG Ripper studio example » car ce qui est en edx n'a pas d'importance.

On appelle le call de verification avec un CALL 0052DAA0

normalement dans le call de verfication à la ligne 52DB76 (c'est a dire juste après l'instruction MOV EAX DWORD [EBP-14], on a le bon serial dans eax donc on revient vers notre procédure de keygen avec une jmp 401189. Ensuite, au lieu d'afficher la valeur 40305A, on affiche eax qui contient notre serial grace à la fonction SetDlgItemTextA.

On execute notre prog, on met un breakpoint au niveau du push eax et on rentre une lettre --> on voit qu'il a trouvé le serial "EB40-816AF1BAA8781905-B891B024318A49C4" pour la lettre A.

Malheureusement, le prog plante juste après car les adresses de retour dans la pile ne sont plus bonnes et ça l'amène n'importe ou dans le code

### 3) corriger la pile

Normalement pour quitter une sous routine et revenir dans la routine appelante, on utilise l'instruction RET --> la en utilisant un JMP 401189 pour le faire revenir, on a fait le bourrin. Quand une instruction RET est appelée, elle va chercher la valeur au sommet de la pile et reviens à cet endroit. Ainsi le code suivant:

PUSH 401010 --> met la valeur 401010 au sommet de la pile  
 RET

équivaut à un JMP 401010

dans notre exemple, à l'adresse 52DB76, on a 0012FCFC au sommet de la pile. Si on execute un RET juste après, ça fera donc un JMP 0012FCFC et notre prog va planter. Nous ce qu'on veut, c'est revenir en 401189 --> on va donc chercher cette valeur pour voir si elle n'est pas restée dans la pile un peu plus bas --> et c'est le cas en effet (valeur entourée en bleu)

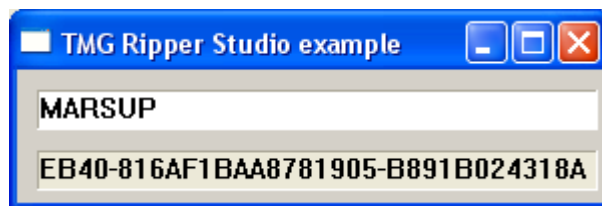
0012FC3C	0012FCFC	Pointer to next SEH record
0012FC40	0052DBAA	SE handler
0012FC44	0012FC6C	
0012FC48	00000000	
0012FC4C	00000000	
0012FC50	00C0006C	ASCII "B891B024318A49C4"
0012FC54	00C0002C	ASCII "816AF1BAA8781905"
0012FC58	00C0008C	ASCII "EB40-816AF1BAA8781905-B891B024318A49C4"
0012FC5C	24B091B8	
0012FC60	C4498A31	
0012FC64	00403010	ASCII "TMG Ripper Studio example"
0012FC68	00000000	
0012FC6C	0012FC78	
0012FC70	00401189	RETURN to Keygen1.00401189 from Keygen1.0052DAA0
0012FC74	0040104D	RETURN to Keygen1.0040104D from Keygen1.00401167
0012FC78	0012FCA4	
0012FC7C	77D18709	RETURN to user32.77D18709
0012FC80	00330298	

Avant d'exécuter notre ret, il faut donc faire revenir la valeur 00401189 au sommet de la pile.  
Le sommet de la pile est en 12FC3C et la valeur 401189 est en 12FC70 (la valeur juste devant)  
 $12FC70 - 12FC3C = 34$

les instructions suivantes a la place du JMP 401189 nous permettent de revenir en 401189  
0052DB76 ADD ESP,34  
0052DB79 RETN

0012FC70	00401189	RETURN to Keygen1.00401189 from Keygen1.0052DAA0
0012FC74	0040104D	RETURN to Keygen1.0040104D from Keygen1.00401167
0012FC78	0012FCA4	
0012FC7C	77D18709	RETURN to user32.77D18709
0012FC80	00330298	
0012FC84	00000111	
0012FC88	04000064	

Maintenant, ça doit rouler --> on rentre MARSUP comme sérial et voilà ce qu'on obtient:



Plusieurs observations:

- > ce sérial n'est pas celui attendu : EB40-8A17D171F9AE4991-9CAED7791D1B748C
- > quelque soit le nom rentré, le serial reste identique
- > la troisieme partie du sérial devrait être fixe à 9CAED7791D1B748C
- > le probleme vient du call 0052DAA0 qui retourne toujours le même serial dans eax

#### 4) pourquoi la partie du milieu est fausse et ne bouge pas en fonction du nom?

Bon la c'est le moins marrant, il faut débbugger et comparer l'exécution de notre keygen au prog original  
En tracant dans le call 0052DAA0, on remarque une première différence notable dans l'exécution à cet endroit:

0052DADF CALL EasyBann.00408804 ; CE CALL MET MARSUP DANS LA PILE EN 12EC10  
0052DAE4 MOV EDX,DWORD PTR SS:[EBP-20]

quand on s'arrete en 52DAE4, la pile ressemble à ça dans le prog original

0012EC00	0012EC38	Pointer to next SEH record
0012EC04	0052DBAA	SE handler
0012EC08	0012EC30	
0012EC0C	00C04000	
0012EC10	00C0FA04	ASCII "MARSUP"
0012EC14	00000000	
0012EC18	00000000	
0012EC1C	00000000	
0012EC20	004786B3	RETURN to EasyBann.004786B3 from EasyBann.0047860C
0012EC24	0012EE00	
0012EC28	00C3B8EC	ASCII "1111111111111111"
0012EC2C	00C2EEB8	ASCII "MARSUP"

Dans notre keygen, la pile ressemble à ça:

0012FA4C	0012FB0C	Pointer to next SEH record
0012FA50	0052DBAA	SE handler
0012FA54	0012FA7C	
0012FA58	00000000	
0012FA5C	00000000	
0012FA60	00000000	
0012FA64	00000000	
0012FA68	00000000	
0012FA6C	0012FA88	
0012FA70	00401176	RETURN to Keygen1.00401176 from <JMP.&user32.GetDlg
0012FA74	00403010	ASCII "TMG Ripper Studio example"
0012FA78	00403040	ASCII "MARSUP"



pas de MARSUP ici

On va donc rentrer dans le CALL EasyBann.00408804 pour comparer les executions. Heureusement pour nous, ce call n'est pas bien grand. Voilà le prog original:

00408804	PUSH EBX	
00408805	PUSH ESI	
00408806	PUSH EDI	
00408807	MOV EDI,EDX	
00408809	MOV ESI,EAX	
0040880B	MOV EAX,ESI	
0040880D	CALL EasyBann.00404604	
00408812	MOV EBX,EAX	
00408814	MOV EAX,EDI	
00408816	MOV EDX,EBX	
00408818	CALL EasyBann.00404990	
0040881D	MOV EDX,ESI	
0040881F	MOV ESI,DWORD PTR DS:[EDI]	
00408821	TEST EBX,EBX	EBX = 6
00408823	JE SHORT EasyBann.0040883A	jump is not taken
00408825	MOV AL,BYTE PTR DS:[EDX]	
00408827	CMP AL,61	
00408829	JB SHORT EasyBann.00408831	
0040882B	CMP AL,7A	
0040882D	JA SHORT EasyBann.00408831	
0040882F	SUB AL,20	
00408831	MOV BYTE PTR DS:[ESI],AL	
00408833	INC EDX	
00408834	INC ESI	
00408835	DEC EBX	
00408836	TEST EBX,EBX	
00408838	JNZ SHORT EasyBann.00408825	
0040883A	POP EDI	
0040883B	POP ESI	
0040883C	POP EBX	
0040883D	RETN	

et notre keygen:

00408804	PUSH EBX	
00408805	PUSH ESI	
00408806	PUSH EDI	
00408807	MOV EDI,EDX	
00408809	MOV ESI,EAX	
0040880B	MOV EAX,ESI	
0040880D	CALL Keygen1.00404604	
00408812	MOV EBX,EAX	
00408814	MOV EAX,EDI	
00408816	MOV EDX,EBX	
00408818	CALL Keygen1.00404990	
0040881D	MOV EDX,ESI	
0040881F	MOV ESI,DWORD PTR DS:[EDI]	
00408821	TEST EBX,EBX	ebx toujours a 0
00408823	JE SHORT Keygen1.0040883A	toujours is taken
00408825	MOV AL,BYTE PTR DS:[EDX]	
00408827	CMP AL,61	
00408829	JB SHORT Keygen1.00408831	
0040882B	CMP AL,7A	
0040882D	JA SHORT Keygen1.00408831	
0040882F	SUB AL,20	
00408831	MOV BYTE PTR DS:[ESI],AL	
00408833	INC EDX	
00408834	INC ESI	
00408835	DEC EBX	
00408836	TEST EBX,EBX	
00408838	JNZ SHORT Keygen1.00408825	
0040883A	POP EDI	
0040883B	POP ESI	
0040883C	POP EBX	
0040883D	RETN	

apparemment, le responsable de la valeur dans ebx est le premier call (CALL 404604). On rentre dedans: voilà son contenu:

```
00404604 TEST EAX,EAX
00404606 JE SHORT EasyBann.0040460B
00404608 MOV EAX,DWORD PTR DS:[EAX-4]
0040460B RETN
```

Au debut, eax = l'adresse qui contient notre nom MARSUP. Si il y quelque chose, on ne saute pas et on passe à la ligne suivante MOV EAX,DWORD PTR DS:[EAX-4]

Cette ligne met dans eax ce qui est à l'adresse eax-4 --> ça met dans eax ce qui se trouve 4 octets avant l'adresse qui contient notre nom MARSUP. Dans le prog original [eax-4] semble correspondre au nombre de caractères du sérial. Ensuite eax est mis dans ebx une fois sorti du call

Voilà le dump pour le prog original:

Address	Hex dump	ASCII
00C111FC	16 00 00 00 4C 8C 41 00 08 04 C0 00 0E 00 00 00	....LIA.iEt.B...
00C1120C	1C 00 00 00 16 00 00 00 02 00 00 00 06 00 00 00	.....@.....
00C1121C	4D 41 52 53 55 50 00 00 2A 00 00 00 84 60 42 00	MARSUP.*...a'B.
00C1122C	00 00 00 00 38 87 47 00 88 10 C1 00 EC 5A BB 00	....88G.e!..yZq.
00C1123C	00 00 00 00 08 00 00 FF 60 00 00 00 00 00 00 00	.....
00C1124C	26 00 00 00 A0 34 47 00 88 10 C1 00 00 00 00 00	%...s4G.e!.....
00C1125C	00 00 00 00 00 00 00 00 00 00 00 00 34 96 47 00	.....4uG.....
00C1126C	88 10 C1 00 6A 00 00 00 84 31 47 00 00 00 00 00	e!..j...ä!G.....
00C1127C	00 00 00 00 DC 12 C1 00 04 13 C1 00 24 13 C1 00	.....!..!..s!l..
00C1128C	00 00 00 00 00 00 00 00 20 00 CC 00 00 00 00 00	.....lf.....
00C1129C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00C112AC	20 38 1B 00 FF FF FF FF 00 00 00 00 00 00 00 00	8+.....
00C112BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00C112CC	88 10 C1 00 00 00 00 00 00 00 00 2A 00 00 00	e!..*.....
00C112DC	84 60 42 00 00 00 00 00 0C 90 42 00 74 12 C1 00	a'B.....EB.t!..

On a le nom MARSUP et le nombre de caracteres du serial (ici 6) 4 bytes plus haut.

Dans notre keygen, voilà ce qu'on a dans le dump:

Address	Hex dump	ASCII
00403000	55 67 6C 79 57 69 6E 64 7C 77 43 6C 61 73 73 00	UglyWindowClass.
00403010	54 4D 47 20 52 69 70 70 65 72 20 53 74 75 64 69	TMG Ripper Studi
00403020	6F 20 65 78 61 6D 70 6C 65 00 45 44 49 54 00 00	o example.EDIT..
00403030	00 00 00 25 30 34 58 25 30 34 58 00 00 00 00 00	...%04X%04X.□
00403040	4D 41 52 53 55 50 00 00 00 00 00 00 00 00 00 00	MARSUP.....
00403050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

On peut essayer de voir quelle procédure met le nombre de caracteres en [eax-4]

--> il s'agit du call juste avant notre CALL 52DAA0 qui calcule le sérial

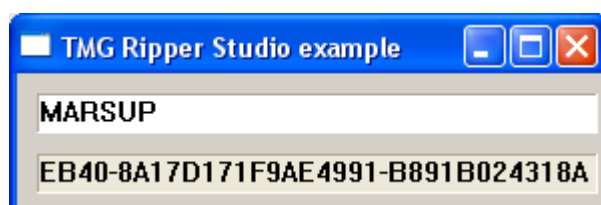
0052E4C6	MOV EAX,DWORD PTR DS:[EBX+308]	*TRegform.username:TEdit
0052E4CC	CALL EasyBann.00478688	->Controls.TControl.GetText(TControl):TCaption;
0052E4D1	MOV EAX,DWORD PTR SS:[EBP-24]	
0052E4D4	POP EDX	
0052E4D6	CALL EasyBann.0052DAA0	le serial est calcule la dedans

en fait le call 478G88 GetText (Tcontrol) ne fait pas que récupérer notre username, il met le nombre de caractères du nom 4 octets avant le nom.

Maintenant qu'on sait ça, on peut corriger notre Keygen pour qu'il fonctionne --> juste après la fonction GetDlgItemTextA, on a le nombre de caractères dans eax --> on va sauver cette valeur 4 octets avant 433040 l'adresse ou le nom est stocké:

00401171	CALL <JMP.&user32.GetDlgItemTextA>	
00401176	TEST EAX,EAX	
00401178	JE SHORT Keygen1.00401199	
0040117A	MOV BYTE PTR DS:[40303C],AL	met le nb de caracteres 4 octets avant l'adresse du nom
0040117F	MOV EAX,Keygen1.00403040	met l'adresse du nom dans eax
00401184	MOV EDX,EAX	met la meme chose dans edx
00401186	CALL Keygen1.0052DAA0	appelle la routine de verif du serial
00401188	ADD ESP,8	
0040118E	PUSH EAX	
0040118F	NOP	
00401190	NOP	
00401191	PUSH 65	
00401193	PUSH ESI	
00401194	CALL <JMP.&user32.SetDlgItemTextA>	

On sauve, on execute notre Keygen et voilà ce qu'on obtient:



le bon serial pour MARSUP est EB40-8A17D171F9AE4991-9CAED7791D1B748C  
donc la première et la deuxième partie sont bonnes maintenant mais la troisième partie du sérial n'est pas correcte

## 5) obtenir une troisieme partie de serial correcte

La, on peut tricher --> cette troisieme partie semble être toujours la même quelque soit le nom rentré et aussi quelque soit l'ordinateur (les serials postés sur le forum ont tous cette troisième partie identique) De plus, si on souhaite payer une licence, il nous demande uniquement notre nom et pas un code, un fichier, le nom de notre systeme d'exploitation --> on peut donc raisonnablement penser que cette partie est fixe pour tout le monde.

Quand notre serial se trouve encore dans eax, on va utiliser des instructions ASM pour modifier la troisieme partie de serial:

00401186	CALL Keygen1.0052DAA0	appelle la routine de verif du serial
0040118B	ADD ESP,8	
0040118E	JMP SHORT Keygen1.0040119A	va en 40119A
00401190	PUSH EAX	
00401191	PUSH 65	
00401193	PUSH ESI	
00401194	CALL <JMP.&user32.SetDlgItemTextA>	
00401199	RETN	
0040119A	MOV DWORD PTR DS:[EAX+16],45414339	met 9CAE
004011A1	MOV DWORD PTR DS:[EAX+1A],39373744	met 0779
004011A8	MOV DWORD PTR DS:[EAX+1E],42314431	met 1D1B
004011AF	MOV DWORD PTR DS:[EAX+22],43383437	met 748C
004011B6	JMP SHORT Keygen1.00401190	retourne en 401190

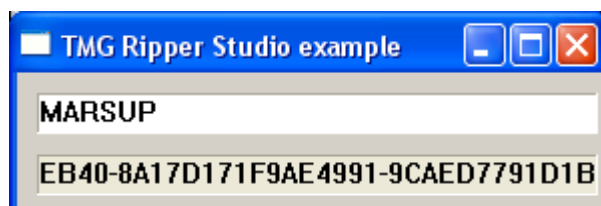
la premiere ligne

0040119A MOV DWORD PTR DS:[EAX+16],45414339

met 9CAE dans [eax+16]. Pour obtenir 9CAE, on prend la valeur ASCII de chaque caractère la valeur ASCII de 9 est 39, C est 43, A est 41, E est 45 ce qui fait au final 9CAE

---> on procede de la même façon pour la suite pour au final obtenir une troisieme partie valide: 9CAED7791D1B748C

On essaye notre keygen une derniere fois, et ça marche.....



(il reste plus qu'a agrandir un peu le champ texte pour que le serial soit complètement visible)

--> mettre 180 a la place de la valeur entourée en rouge et 160 a la place des valeurs entourés en bleu

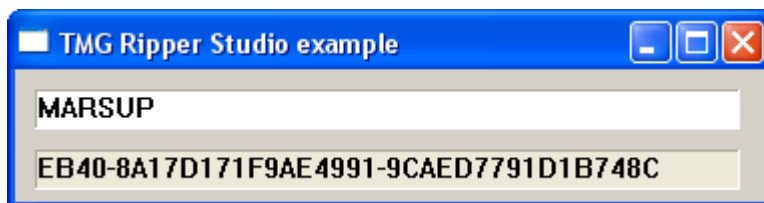
004010AB	PUSH 0	
004010AD	PUSH DWORD PTR SS:[EBP+8]	
004010B0	PUSH 0	
004010B2	PUSH 0	
004010B4	PUSH 64	
004010B6	PUSH 12C	
004010B8	PUSH 0	
004010BD	PUSH 80000000	
004010C2	PUSH 0CF0000	
004010C7	PUSH TMGRS-EX.00403010	
004010CC	PUSH TMGRS-EX.00403000	
004010D1	PUSH 20000	
004010D6	CALL <JMP.&USER32.CreateWindowExA>	CreateWindowExA
004010DB	PUSH DWORD PTR SS:[EBP+14]	ShowState
004010DE	PUSH EAX	hWnd
004010DF	PUSH 0	
004010E1	PUSH DWORD PTR SS:[EBP+8]	
004010E4	PUSH 64	
004010E6	PUSH EAX	
004010E7	PUSH 14	
004010E9	PUSH 118	
004010EE	PUSH 0A	
004010F0	PUSH 0A	
004010F2	PUSH 50000000	
004010F7	PUSH 0	
004010F9	PUSH TMGRS-EX.0040302A	
004010FE	PUSH 20000	
00401103	CALL <JMP.&USER32.CreateWindowExA>	CreateWindowExA
00401108	PUSH 0	
0040110A	PUSH DWORD PTR SS:[EBP+8]	
0040110D	PUSH 65	
0040110F	PUSH DWORD PTR SS:[ESP+C]	
00401113	PUSH 14	
00401115	PUSH 118	
0040111A	PUSH 28	
0040111C	PUSH 0A	
0040111E	PUSH 50000000	
00401123	PUSH 0	
00401125	PUSH TMGRS-EX.0040302A	
0040112A	PUSH 20000	
0040112F	CALL <JMP.&USER32.CreateWindowExA>	CreateWindowExA

lParam = NULL
hInst
hMenu = NULL
hParent = NULL
Height = 64 (100.)
Width = 12C (300.)
Y = 0
X = 80000000 (-2147483648.)
Style = WS_OVERLAPPED WS_MINIMIZE
WindowName = "TMG Ripper Studio e
Class = "UglyWindowClass"
ExtStyle = WS_EX_STATICEDGE
CreateWindowExA
ShowState
hWnd
lParam = NULL
hInst
hMenu = 00000064 (window)
hParent
Height = 14 (20.)
Width = 118 (280.)
Y = A (10.)
X = A (10.)
Style = WS_CHILD WS_VISIBLE
WindowName = NULL
Class = "EDIT"
ExtStyle = WS_EX_STATICEDGE
CreateWindowExA
lParam = NULL
hInst
hMenu = 00000065
hParent
Height = 14 (20.)
Width = 118 (280.)
Y = 28 (40.)
X = A (10.)
Style = WS_CHILD WS_VISIBLE 800
WindowName = NULL
Class = "EDIT"
ExtStyle = WS_EX_STATICEDGE
CreateWindowExA



Cette fois ci, c'est bon



*Remarque : Peut être cette dernière solution ne vous satisfait pas entièrement. Vous voulez peut être savoir pourquoi notre keygen ne trouve pas une troisième partie de serial correcte par le calcul. En traçant et en comparant l'original et notre keygen, on voit que la différence vient du contenu d'une adresse mémoire: 547EA8*

*Dans le prog original, au départ cette adresse pointe sur 00000000. On met un breakpoint hardware on write byte sur cette valeur et on lance le prog. Dans les premiers call qui initialisent le prog, l'instruction suivante modifie le contenu de 547EA8 pour qu'il contienne F988*

*00509377 MOV DWORD PTR DS:[547EA8],EAX*

Address	Hex dump	ASCII
00547EA8	F9 88 00 00 00 00 00 00 03 00 00 00 01 00 00 00	--e.....0...

*Cette valeur [547EA8] est ensuite utilisée plus tard pour calculer la troisième partie du sérial. Voilà la ligne coupable:*

*005092E2 SUB AX,WORD PTR DS:[547EA8]*

*Dans notre Keygen, la valeur 547EA8 est à zero et reste à zero tout le temps. Si on veut obtenir une troisième partie de serial valide, il faut donc mettre F988 dans 547EA8*

## Conclusion

J'espère que vous aurez pris plaisir à lire ce tut. Au départ, je pensais finir ce keygen très rapidement mais je n'ai pas suffisamment fait attention aux valeurs utilisées par la routine de vérification du sérial. Grâce à cette méthode, nous obtenons un magnifique keygen de 2.1 Mo et beaucoup de code inutile --> c'est le gros inconvénient de cette technique et c'est peut être pour ça que je ne l'avais jamais vu utilisée 😊 Je remercie chaleureusement Ulysse\_31 pour ses conseils et son aide quand je bloque sur ses tuts. Je remercie aussi sp0ke pour son enthousiasme --> ce tut c'est de ta faute 😊. Je salue aussi tous les membres du forum XTX et plus particulièrement Kirjo (merci pour ton [tut](#) sur Easybanner), Z!PPer, coolmen, Burner. Je salue aussi tous les membres du forum DeezDynasty et en particulier Dynasty et Donald.

PS: dans l'archive,j'ai mis le logiciel EasyBanner ainsi mon fichier Keygen.exe

mars le 10/01/2009