

Le ScrewMe #2 by Dynasty : Analyse et Keygenning

Outils nécessaires

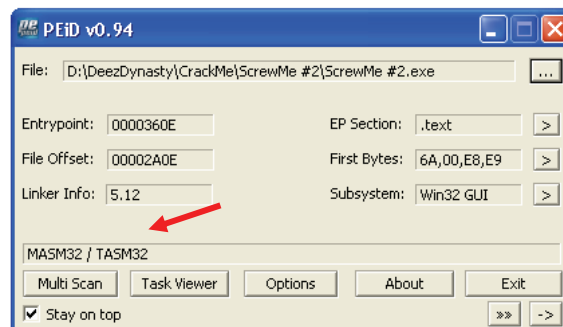
- Le ScrewMe#2 de Dynasty ([téléchargeable ici](#))
- PEID
- OllyDBG (+ plugins HideDebugger, CommandBar)

Sommaire

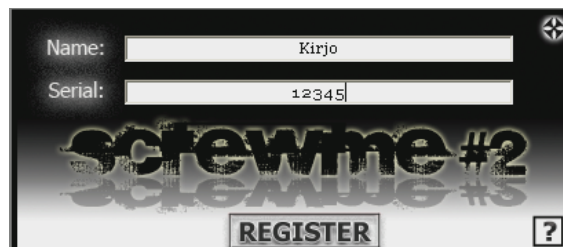
- 1 - Introduction
- 2 - Passer les anti-debug
- 3 - Etude de la routine de création du sérial
- 4 - Analyse du Keyfile
- 5 - Conclusion

1 - Introduction

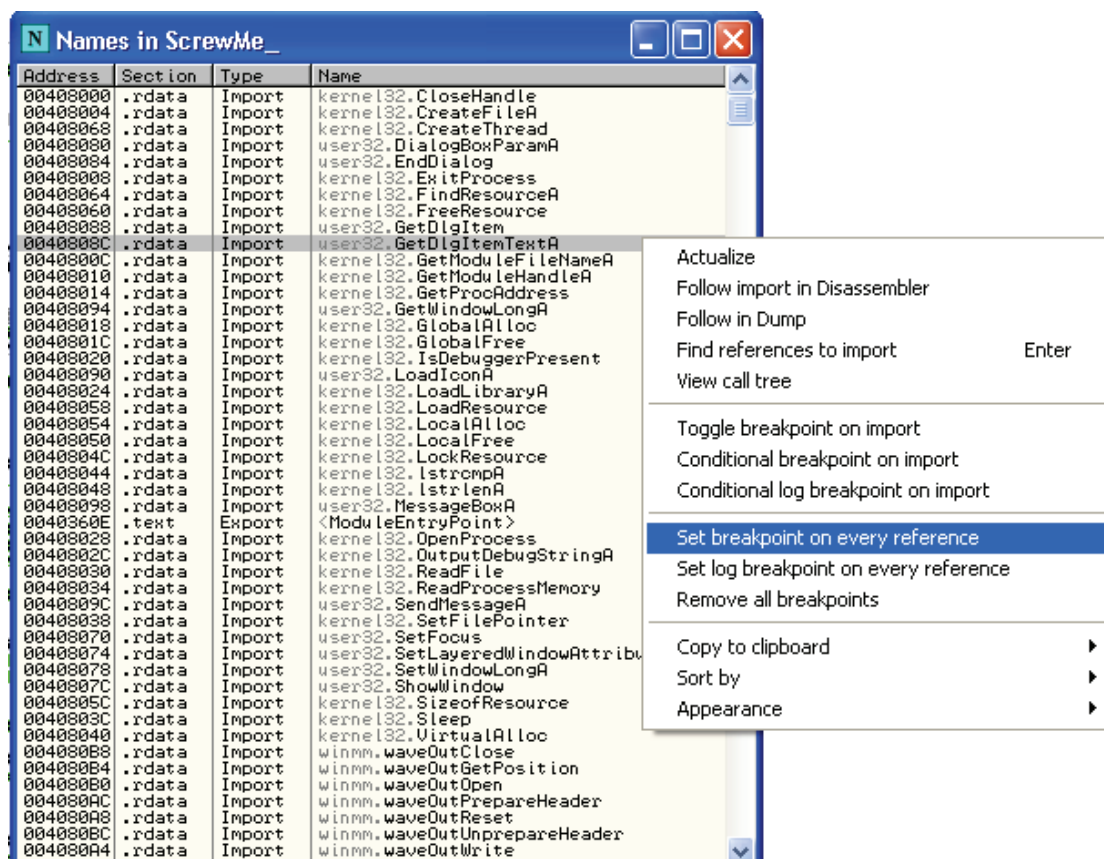
On va s'attaquer dans ce tutorial au ScrewMe #2 by Dynasty. Premier réflexe, l'analyse avec PEID :




OK, pas de packer, on peut commencer. On ouvre le ScrewMe #2, on saisit notre name, un serial bidon et on clique sur "REGISTER" :






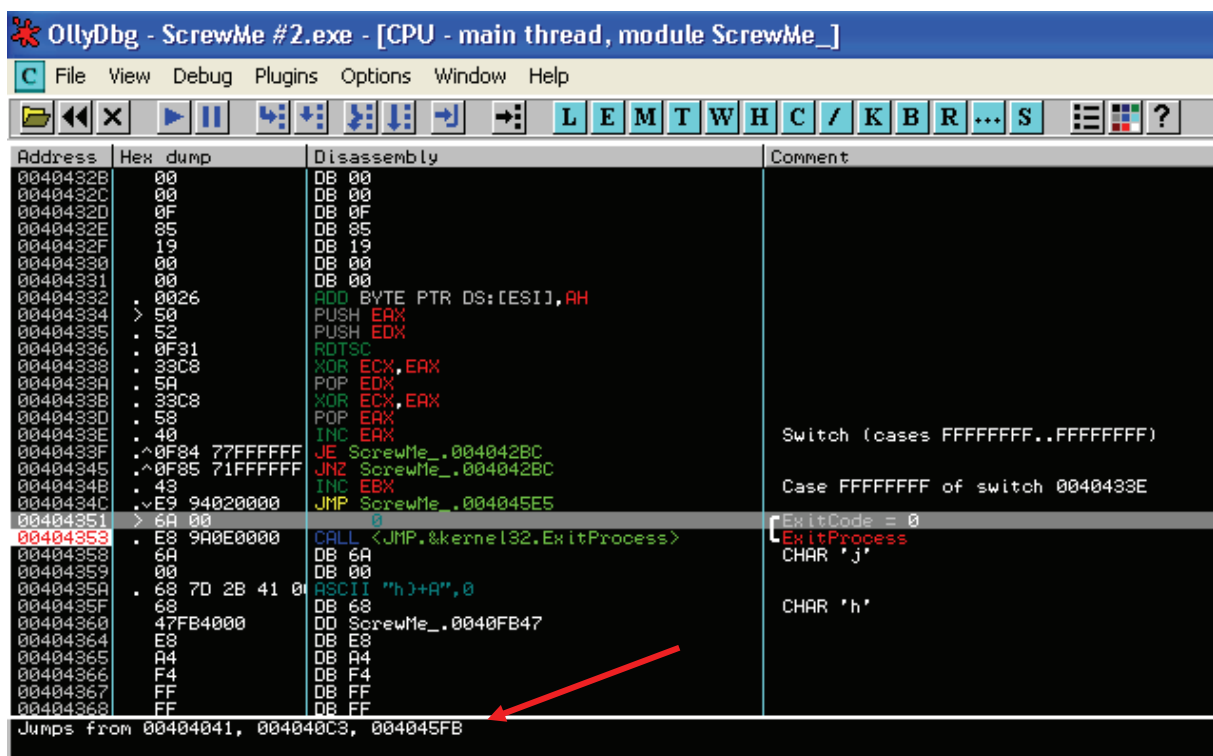
Et il ne se passe rien, donc pas d'indice. On ouvre le ScrewMe dans OLLYDBG, on fait un "CTRL + N" pour afficher la liste des fonctions :



On a notre classique **GetDlgItemTextA** (fonction qui récupère du texte dans une fenêtre, par exemple notre name), on fait un clic droit dessus, puis "**Set breakpoint on every reference**".
On ferme la fenêtre des Names, on fait "**F9**" (ou clic sur ) et le ScrewMe #2 se ferme immédiatement => on est visiblement en présence d'anti-debug. On va donc attaquer différemment

2 - Passer les anti-debug

On fait un restart program ("**CTRL + F2**" ou ) puis "**Alt + B**" (ou clic sur ) , on efface les BP que l'on avait posé et on ferme la fenêtre des BP. On fait un "**CTRL + N**" et on voit dans la liste un "**ExitProcess**" : est-ce que cela ne serait pas lui qui nous ferme le ScrewMe #2 ? Pour le savoir, clic droit et "**Set breakpoint on every reference**" (vous devriez avoir en bas à gauche de la fenêtre de Olly : 4 breakpoints set), on ferme la fenêtre des Names, on fait "**F9**" (ou clic sur ) et cette fois-ci le ScrewMe #2 s'ouvre normalement. On saisit un Name (Kirjo) et un serial bidon (12345) puis "**REGISTER**". Olly breake en **404353**. On remonte d'une ligne avec la touche "**Flèche Haut**" et on voit ceci :

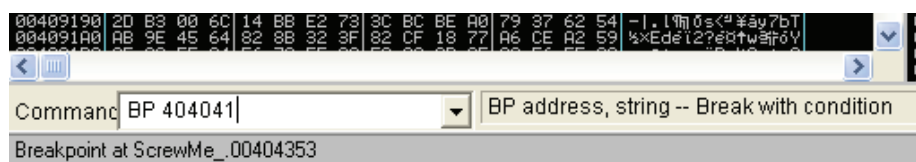



Address	Hex dump	Disassembly	Comment
0040432B	00	DB 00	
0040432C	00	DB 00	
0040432D	0F	DB 0F	
0040432E	85	DB 85	
0040432F	19	DB 19	
00404330	00	DB 00	
00404331	00	DB 00	
00404332	0026	ADD BYTE PTR DS:[ESI],AH	
00404334	50	PUSH EAX	
00404335	52	PUSH EDX	
00404336	0F31	RDTSC	
00404338	33C8	XOR ECX, EAX	
0040433A	5A	POP EDX	
0040433B	33C8	XOR ECX, EAX	
0040433D	58	POP EAX	
0040433E	40	INC EAX	
0040433F	77FFFFFF	JE ScrewMe_.004042BC	Switch (cases FFFFFFFF..FFFFFFF)
00404345	71FFFFFF	JNZ ScrewMe_.004042BC	
00404348	43	INC EBX	Case FFFFFFFF of switch 0040433E
0040434C	E9 94020000	JMP ScrewMe_.004045E5	
00404351	6A 00	0	ExitCode = 0
00404353	E8 9A0E0000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
00404358	6A	DB 6A	CHAR 'j'
00404359	00	DB 00	
0040435A	68 7D 2B 41 00	ASCII "h)+A",0	
0040435F	68	DB 68	CHAR 'h'
00404360	47FB4000	DD ScrewMe_.0040FB47	
00404364	E8	DB E8	
00404365	A4	DB A4	
00404366	F4	DB F4	
00404367	FF	DB FF	
00404368	FF	DB FF	

Jumps from 00404041, 004040C3, 004045FB

3 sauts pointent vers cette ligne : **404041**, **4040C3** et **4045FB**. On va se servir de ces sauts pour remonter le code et retrouver les anti-debugs ...

On commence par poser des BP sur ces 3 adresses :



et on fait un restart program ("**CTRL + F2**" ou ) on clique sur "**OUI**" dans la fenêtre qui s'affiche puis "**F9**", on saisit name et sérial, "**REGISTER**" et OLLY breake en **4040C3**. On tape ";" et on se rentre un petit commentaire pour mémoire (Jump vers ExitProcess par exemple).

Maintenant observons les lignes qui se trouvent au-dessus :

Address	Hex dump	Disassembly	Comment
00404073	. B9 04000000	MOV ECX,4	
00404078	. A1 95214100	MOV EAX,DWORD PTR DS:[412195]	
0040407D	. F7F1	DIV ECX	
0040407F	. 8B08	MOV EAX,EAX	
00404081	> A1 91214100	MOV EAX,DWORD PTR DS:[412191]	
00404086	. 8B0488	MOV EAX,DWORD PTR DS:[EAX+ECX*4]	
00404089	. 50	PUSH EAX	
0040408A	. 51	PUSH ECX	
0040408B	. 50	PUSH EAX	
0040408C	. 6A 00	PUSH 0	
0040408E	. 6A 10	PUSH 10	
00404090	. E8 8D110000	CALL <JMP.&kernel32.OpenProcess>	[ProcessId Inheritable = FALSE Access = VM_READ OpenProcess
00404095	. 8D35 95214100	LEA ESI,DWORD PTR DS:[412195]	
00404098	. 8D3D 99214100	LEA EDI,DWORD PTR DS:[412199]	
004040A1	. 57	PUSH EDI	
004040A2	. 6A 04	PUSH 4	
004040A4	. 56	PUSH ESI	[pBytesRead => ScrewMe_.00412199 BytesToRead = 4 Buffer => ScrewMe_.00412195 pBaseAddress = 4B064B hProcess
004040A5	. 68 4B064B00	PUSH 4B064B	
004040AA	. 50	PUSH EAX	
004040AB	. E8 84110000	CALL <JMP.&kernel32.ReadProcessMemory>	[ReadProcessMemory
004040B0	. 85C0	TEST EAX,EAX	
004040B2	. 74 14	JE SHORT ScrewMe_.004040C8	
004040B4	> 813E 4F4C4C59	CMP DWORD PTR DS:[ESI],594C4C4F ←	
004040B8	. 75 0C	JNZ SHORT ScrewMe_.004040C8	
004040BC	. C605 80214100	MOV BYTE PTR DS:[412180],1	
004040C3	. E9 89020000	JMP ScrewMe_.00404351	Jump vers ExitProcess
004040C8	> 59	POP ECX	
004040C9	. 58	POP EAX	
004040CA	. E2 B5	LOOPD SHORT ScrewMe_.00404081	
004040CC	. 803D 80214100	CMP BYTE PTR DS:[412180],1	
004040D3	. 7E F84 05110000	JE ScrewMe_.004051DE	
004040D9	. E9 34000000	JMP ScrewMe_.00404112	

On voit dans le rectangle rouge une boucle permettant de lister les processus en cours. A l'adresse **4040B4**, on a une comparaison entre le résultat du **ReadProcessMemory** et **594C4C4F**. Convertissons cette valeur en ASCII :

59 = Y 4C = L 4C = L 4F = O soit : **YLLO**

Vous l'aurez compris, le ScrewMe #2 recherche si on utilise OLLYDBG :

004040B4	CMP DWORD PTR DS:[ESI],594C4C4F	; Compare le résultat de ReadProcessMemory à " OLLY "
004040BA	JNZ SHORT ScrewMe_.004040C8	; Si négatif saut vers la suite de la boucle
004040BC	MOV BYTE PTR DS:[412180],1	; Si positif met 1 à l'adresse 412180
004040C3	JMP ScrewMe_.00404351	; Jump vers ExitProcess => BAD BOY
004040C8	POP ECX	; Récupère ECX sur la pile
004040C9	POP EAX	; Récupère AEX sur la pile
004040CA	LOOPD SHORT ScrewMe_.00404081	; Continue la boucle
004040CC	CMP BYTE PTR DS:[412180],1	; Compare le contenu de 412180 à 1
004040D3	JE ScrewMe_.004051DE	; Si égal saute vers 4051DE
004040D9	JMP ScrewMe_.00404112	; Sinon saute vers 404112

⇒ On pourrait se dire qu'il suffit de nopper le jump en **4040C3** mais dans ce cas on va mettre 1 en mémoire à l'adresse **412180**. Lors de la comparaison en **4040CC**, on va être égal à 1 et donc effectuer le saut en **4040D3** (Jump if Equal) qui nous enverra en **4051DE**, ce qui correspond à un **ExitProcess**.

On va passer proprement (sans nopper comme des cochons). On clique sur la ligne **4040B4**, on fait " **F2** " pour poser un BP, restart program (" **CTRL + F2** "), name, serial puis " **REGISTER** " et on break ici :

On continue à tracer avec " F8 " jusqu'à la ligne **404202**. On arrive sur une instruction **RDTSC** :

RDTSC - Read time stamp counter

Syntaxe : RDTSC

Retourne dans le couple de registre EDX:EAX le nombre de ticks* écoulés depuis la dernière remise à zéro du processeur (Reset). L'instruction lit simplement un registre spécial de 64 bits, nommé Time Stamp Counter (compteur temporel) et place le résultats dans les registres EDX et EAX. La partie haute du compteur temporel (32 bits de poids fort) est placée dans le registre EDX tandis que la partie basse (32 bits de poids faible) est placée dans le registre EAX.

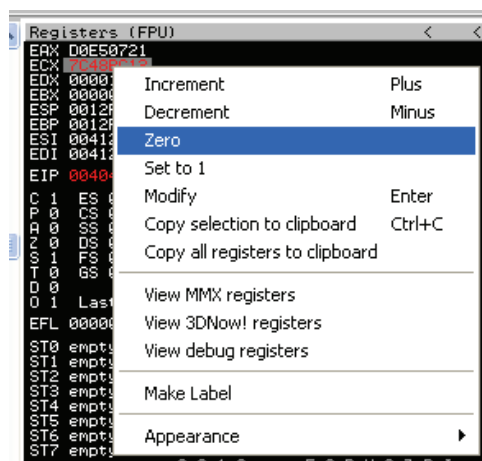
*Tick : Unité de mesure du temps. Quasiment toujours au pluriel, car un tick est très court : de l'ordre de un millionième de seconde pour une horloge, ou de un millième de seconde pour un timer.

Regardons le code :

00404202	RDTSC	; Lit le compteur temporel
00404204	MOV ECX,EAX	; Met dans EAX la partie basse de EAX
00404206	RDTSC	; Lit le compteur temporel
00404208	SUB ECX,EAX	; Soustrait la nouvelle partie basse à l'ancienne et la met dans EAX
0040420A	NOT ECX	; Inversion logique de ECX
0040420C	CMP ECX,5000	; Compare ECX à 5000
00404212	JG SHORT ScrewMe_.00404212	; Si plus grand, saut vers 404212
00404214	PREFIX REP	; Junk code (code inutile)
00404215	JE ScrewMe_.0040414A	; Junk code (code inutile)
0040421B	JNZ ScrewMe_.0040414A	; Si différent de zéro, saut vers 40414A

En résumé, le code relève le compteur temporel, exécute une instruction, relève à nouveau le compteur temporel et regarde combien de temps il a fallu pour exécuter cette instruction. Si le temps d'exécution est trop élevé, cela signifie que l'on est en train de tracer le code : on a beau être rapide, le temps de faire un " F7 " ou un " F8 " sera toujours plus important que le temps qu'il faut au processeur pour faire un **MOV ECX,EAX**. On rentre alors dans une boucle infinie (**404212** saute vers **404212**) et on est bloqué.

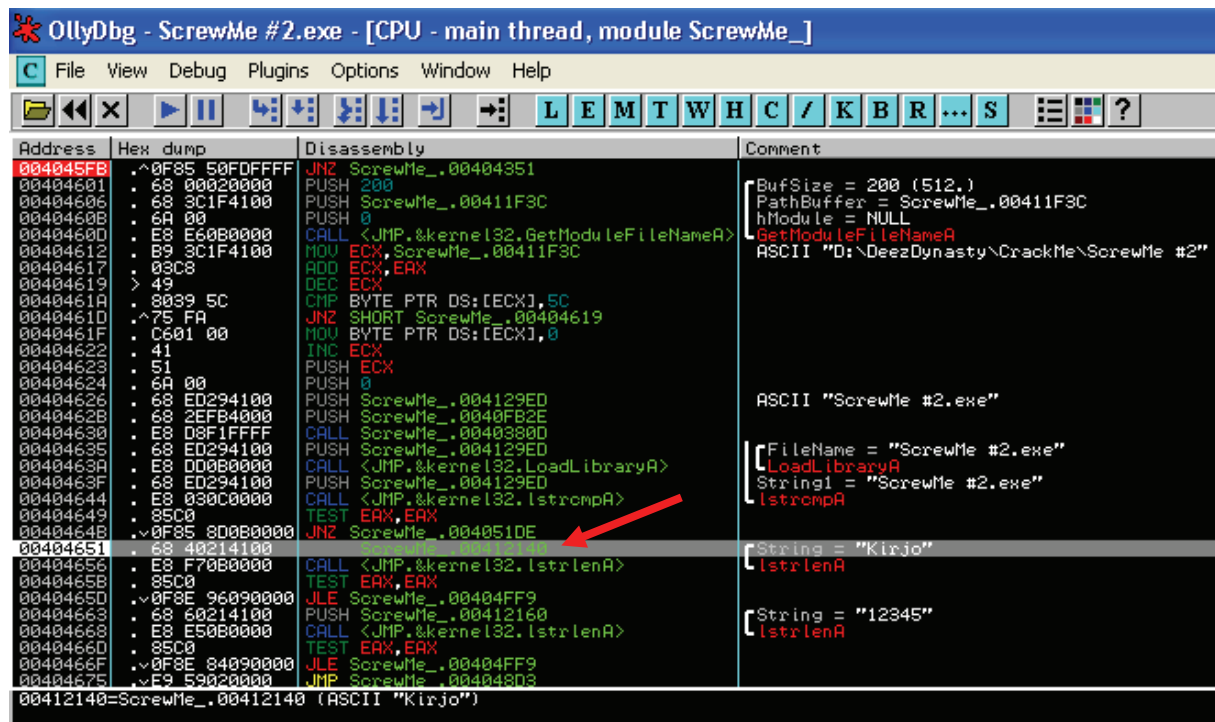
Pour passer il suffit de tracer jusqu'à **40420C**, on sélectionne **ECX** dans la fenêtre des registres, clic droit et zéro :



Ca y est, on est plus rapide qu'un processeur, on peut continuer à tracer !

4 fois " F8 " et on a de nouveau un contrôle RDTSC, même opération : quand on arrive sur le **COMP ECX,5000** on met **ECX** à zéro.

On continue à tracer avec " F8 " (attention : il faudra mettre **ECX** à zéro en **40422C** et **4040F0**) jusqu'à **40419A**. Là une messagebox va vous demander si vous êtes sûr de vouloir continuer, dites OUI et continuez à tracer jusqu'à **404651**. Vous devriez avoir ceci :



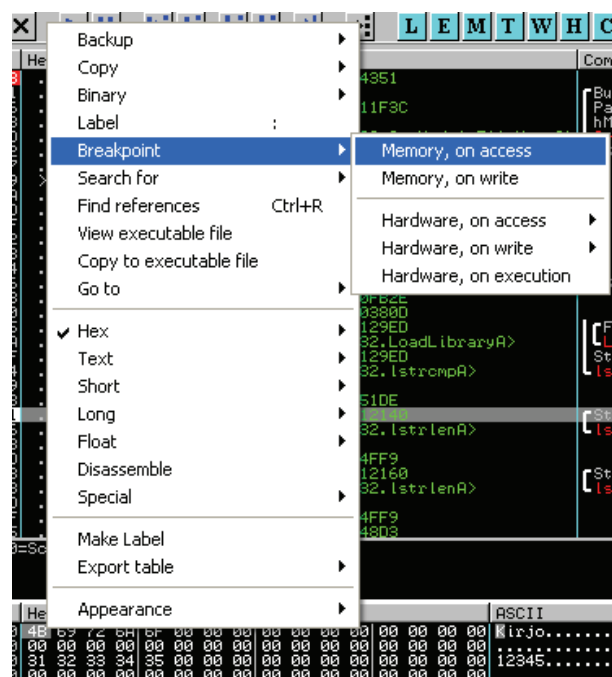
On peut voir que notre name est stocké en **412140**.

On va donc taper dans la CommandBar :



Pour afficher le name dans la fenêtre de dump.

On clique sur la première lettre du nom pour la sélectionner puis clic droit, **breakpoint** et **Memory, on access** :



Même opération avec la première lettre du sérial (31, 2 ligne en dessous).

Dorénavant, OLLY breakera à chaque fois que le ScrewMe #2 voudra accéder au name ou au sérial.

On fait 5 fois "F9" pour arriver en 404980, puis 2 fois "F7" pour arriver ici :

On peut voir sur cette ligne que le premier chiffre de notre sérial est comparé à "6".

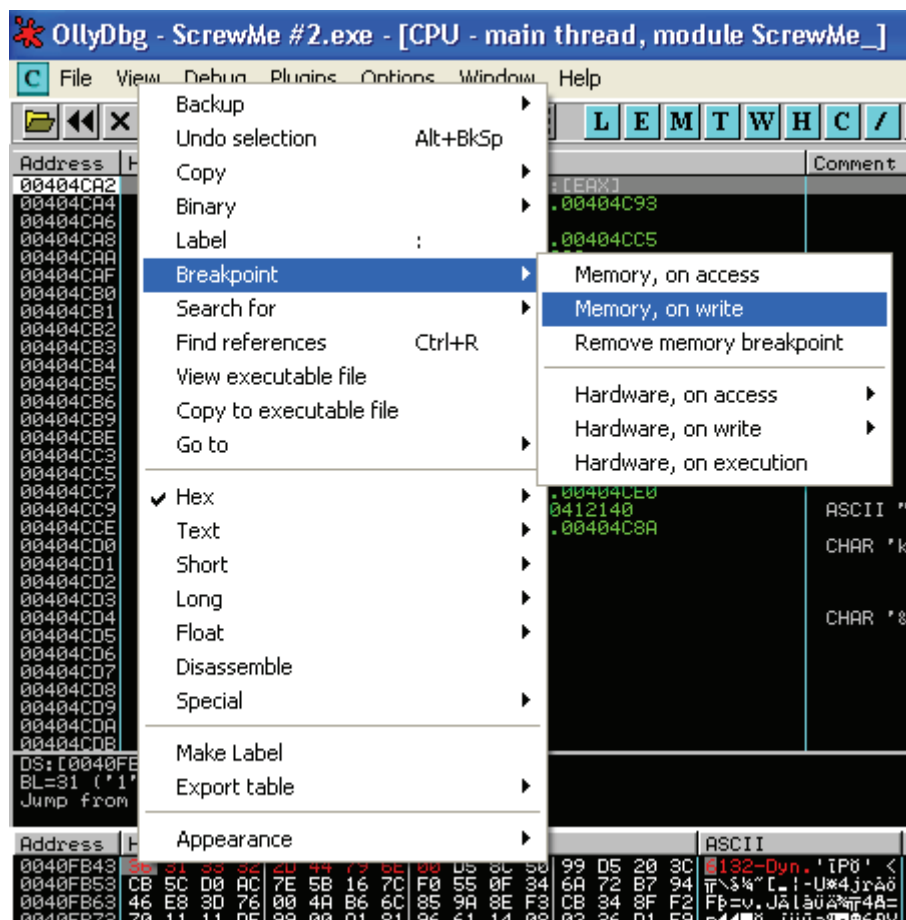
Si on regarde dans la fenêtre des registres, on peut voir que **EAX** est égal à "6132-Dyn". Ne serait ce pas notre sérial ? On regarde dans la pile et on voit ça :

Le sérial calculé par le ScrewMe #2 est stocké en 40FB43.

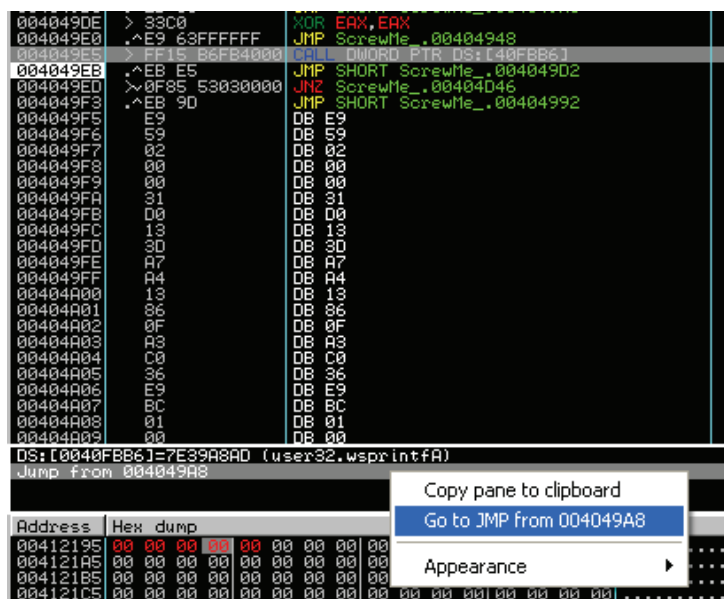
On va se servir de ça pour remonter jusqu'à la routine de création en retrouvant la ligne de le code qui écrit à cette adresse.

Dans la commandbar, on tape "d 40FB43" pour afficher le serial dans la fenêtre de dump :

On clique sur le premier bit du sérial (**36**), clic droit, breakpoint, **Memory, on write** :



On fait "**F9**", on repose le **BP 4040B4** dans la commandbar puis on clique sur "**REGISTER**".
 OLLY breake en **4040B4**, on refait la petite manip pour changer le **Y** en **D**, "**F2**" sur la ligne **4040B4** pour enlever le **BP**, "**F9**" et on breake dans le module **User32**. On fait "**CTRL + F9**" (ou) pour aller à la ligne **RET**, "**F8**" pour exécuter le **RET** jusqu'à revenir dans le ScrewMe à la ligne **4049EB**. On remonte d'une ligne avec "Flèche Haut", et on voit : **Jump from 4049A8**. On sélectionne la ligne, clic droit puis **Goto Jump from 4049A8** :



On continue ainsi à remonter en trouvant le jump d'où l'on vient à chaque fois. En **404CE0**, on a 2 possibilités: **404987** ou **404CC7**. Le **404987** nous emmène vers le jump en **40498D** que l'on a déjà vu, il s'agit d'une boucle, on choisit donc de remonter vers **404CC7**. En **40CC9**, notre name est mis dans **ESI**, c'est le début du calcul de notre sérial, on continue à remonter jusqu'en **4049A1** où on récupère la longueur du name, c'est le début de la routine de calcul de notre sérial. Ouf !

On note cette adresse et on va pouvoir étudier la routine en détail.

3 – Etude de la routine de création du sérial

Avant de commencer, on nettoie un peu. On fait "**Alt + B**" (ou clic sur **B**), on efface les BP que l'on avait posé, on ferme la fenêtre des BP, dans la commandbar, on tape **BP 4040B4** (anti OLLY) et **BP 4049A1** (routine du sérial) puis "**F9**". On clique sur "**REGISTER**", on break en **4040B4**, on modifie Y en D (voir plus haut), on enlève le BP en **4040B4**, "**F9**" et on break au début de la routine.

On a vu en cherchant le début de la routine que les instructions étaient alternées avec des jumps. **Dynasty** cherche à nous compliquer le travail, on va se le simplifier. On va tracer le code avec "**F8**" en copiant chaque ligne contenant une instruction dans le bloc note. En **404987**, on retourne dans la boucle, pour ne pas tout tracer, on pose un BP en **40498D** puis "**F9**". On continue avec "**F8**" jusqu'en **404D61**. Ça nous donne le code ci-dessous :

0040499C	CALL <JMP.&kernel32.lstrlenA>	; Récupère la longueur du name
00404964	TEST EAX,EAX	; Vérifier qu'un name a été saisi
00404CB6	CMP EAX,8	; Compare la longueur du name à 8
0040494D	JA ScrewMe_.00404FF9	; Si supérieur, sort de la routine
00404CC9	MOV ESI,ScrewMe_.00412140	; Met le name dans ESI
00404C8A	MOV EDI,ScrewMe_.00412160	; Met le serial saisi dans EDI
00404CA6	XOR EBX,EBX	; Met EBX à zéro
00404CC5	XOR EDX,EDX	; Met EDX à zéro
00404CF5	PUSH EAX	; Sauvegarde EAX sur la pile
004049DE	XOR EAX,EAX	; Met EAX à zéro
00404948	MOV AL,BYTE PTR DS:[EDX+ESI]	; Met le char n°EDX+1 du name dans EAX
004049BC	ADD EBX,EAX	; Additionne dans EBX : EAX+EBX
00404CEF	INC EDX	; Ajoute 1 à EDX
00404958	POP EAX	; Récupère EAX sur la pile (longueur du name)
00404D2A	CMP EAX,EDX	; Compare EAX à EDX
00404987	JNZ ScrewMe_.00404CE0	; Si EAX <> EDX jump vers 404CF5
00404CBE	MOV EAX,0C	; Met C(h) dans EAX
00404C9E	MUL EBX	; Met dans EAX : EBX*EAX
004049B9	PUSH EAX	; Met EAX sur la pile
004049D7	PUSH ScrewMe_.00412C45	; Passe le format "%d-Dyn" en argument
004049A3	PUSH ScrewMe_.0040FB43	; Passe la chaîne en argument
004049E5	CALL DWORD PTR DS:[40FBB6]	; Appelle la fonction User32.WsprintfA
00404D61	MOV EAX,ScrewMe_.0040FB43	; Met la chaîne formatée dans EAX

Pour résumé :

- On additionne les valeurs hexa de chaque caractère du name
- On multiplie par C (12 en décimal)
- On convertit en décimal
- On ajoute "-Dyn "

Et on a notre sérial. On fait "**Alt + B**" (ou clic sur **B**), on efface les BP que l'on avait posé, on ferme la fenêtre des BP, on pose un BP en **404D61**, dans la commandbar, on tape **BP 4040B4** (anti OLLY) puis "**F9**". On rentre notre sérial (**6132-Dyn**), on clique sur "**REGISTER**", on passe la routine anti OLLY, "**F9**", et on break en **404D61**.

On trace avec " **F8** " (en **404CA2** on voit que notre sérial est comparé caractère par caractère au sérial calculé par le ScrewMe #2), on passe tout le name, tout est bon on continue à tracer pour voir apparaître notre messagebox Good Boy mais au lieu de ça on arrive en **404D0E** sur un **CALL ScrewMe_.00403B4C**.
Rentrons dans ce CALL (" **F7** ") et traçons jusqu'en **403BDA** :

00403BDA	PUSH 0	; /hTemplateFile = NULL
00403BDC	PUSH 20	; Attributes = ARCHIVE
00403BDE	PUSH 3	; Mode = OPEN_EXISTING
00403BE0	PUSH 0	; pSecurity = NULL
00403BE2	PUSH 3	; ShareMode = FILE_SHARE_READ FILE_SHARE_WRITE
00403BE4	PUSH 80000000	; Access = GENERIC_READ
00403BE9	PUSH ScrewMe_.0040FED0	; FileName = " 6132-Dyn.nob "
00403BEE	CALL <JMP.&kernel32.CreateFileA>	; \CreateFileA
00403BA4	CMP EAX,-1	

On peut voir qu'il cherche si il existe un fichier nommé " **6132-Dyn.nob** ". On a donc affaire à un Keyfile.
On pose un BP sur l'adresse de vérification (**403BA4**), on remet le BP en **4040B4** (anti OLLY) puis " **F9** ".

On ouvre Notepad, on tape une chaîne bidon (**54321**) et on l'enregistre dans le répertoire du ScrewMe #2 sous **6132-Dyn.nob**. On revient sous OLLY, clic sur " **REGISTER** ", on passe l'anti OLLY, " **F9** " pour breaker en **403BA4**. On va maintenant analyser le Keyfile.

4 – Analyse du Keyfile

On voit de suite que **EAX** est différent de **-1**, donc on est bon, on continue.

On trace avec " **F8** ", en **403B9D** on lit le contenu du fichier .nob, on continue jusqu'en **403BC1**.

Voilà le code que l'on a alors :

00403BC1	MOV ECX,DWORD PTR SS:[EBP-4]	; Met 13 dans ECX
00403BC4	JMP SHORT ScrewMe_.00403BF5	; Saut vers 403BF5
00403BF5	MOV ESI,ScrewMe_.00410F14	; Met l'adresse 410F14 dans ESI
00403BFA	MOV EDI,ScrewMe_.0040FEF0	; Met "chaîne du nob" dans EDI
00403BFF	XOR EBX,EBX	; Met EBX à 0
00403C01	XOR EDX,EDX	; Met EDX à 0
00403C03	DEC ECX	; Décrémente ECX
00403C04	CMP ECX,0	; Compare ECX à 0
00403C07	JE SHORT ScrewMe_.00403C7A	; Si = 0 saut vers 403C7A >> Sortie
00403C09	MOV BL,BYTE PTR DS:[EDI]	; Met dans EBX la valeur hexa du caractère n°x de la chaîne
00403C0B	MOV DL,BYTE PTR DS:[EDI+1]	; Met dans EDX la valeur hexa du caractère n°x+1 de la chaîne
00403C0E	SUB BL,30	; Soustrait 48 à EBX
00403C11	MOV EAX,DWORD PTR DS:[40A004]	; Met dans EAX la longueur de chaîne restante
00403C16	CMP EAX,0	; Compare EAX à 0 (longueur chaîne)
00403C19	JE SHORT ScrewMe_.00403C6A	; Si = 0 saut vers 403C6A >> Sortie
00403C1B	MOV BYTE PTR DS:[ESI],DL	; Transfère à l'adresse ESI le contenu de EDX (caractère n°x+1 de la chaîne)
00403C1D	INC DWORD PTR SS:[EBP-8]	; Incrémente le haut de la pile
00403C20	CMP DWORD PTR SS:[EBP-8],1024	; Compare le haut de la pile à 4132
00403C27	JGE SHORT ScrewMe_.00403C7A	; Si >= saut vers 403C7A >> Sortie
00403C29	INC ESI	; Incrémente ESI
00403C2A	DEC BL	; Décrémente EBX
00403C2C	JNZ SHORT ScrewMe_.00403C1B	; Si EBX <>0 boucle vers 403C1B
00403C2E	ADD EDI,2	; Ajoute 2 à EDI (passe à l'emplacement suivant)
.../...		


.../...		
00403C31	CMP DL,0	; Compare EDI à zéro (vérifie si il y a encore un caractère dans EDI)
00403C34	JNZ SHORT ScrewMe_.00403C09	; Si <> 0 boucle vers 403C09
00403C36	MOV DWORD PTR SS:[EBP-4],ECX	; Pousse ECX dans la pile
00403C39	PUSH ScrewMe_.00410F14	; Passe la chaîne créée en argument
00403C3E	CALL <JMP.&kernel32.lstrlenA>	; Récupère la longueur de la chaîne
00403C43	PUSH EAX	; Met la longueur de la chaîne sur la pile
00403C44	MOV EDI,ScrewMe_.00410F14	; Met la chaîne créée dans EDI
00403C49	MOV ESI,ScrewMe_.0040FEF0	; Met la chaîne du nob dans ESI
00403C4E	XOR ECX,ECX	; Met ECX à zéro
00403C50	MOV CL,BYTE PTR DS:[EDI]	; Met le caractère n°x+1 dans ECX
00403C52	MOV BYTE PTR DS:[ESI],CL	; Met le caractère n°x+1 à l'adresse ESI
00403C54	INC EDI	; Incrémente EDI
00403C55	INC ESI	; Incrémente ESI
00403C56	DEC EAX	; Décrémente EAX
00403C57	JNZ SHORT ScrewMe_.00403C50	; Si EAX <> 0 boucle vers 403C50
00403C59	MOV ECX,DWORD PTR SS:[EBP-4]	; Met dans ECX la longueur de la chaîne du nob
00403C5C	MOV DWORD PTR DS:[ESI],0	; Met 0 à l'adresse ESI
00403C62	MOV DWORD PTR DS:[ESI],0	; Met 0 à l'adresse ESI
00403C68	JMP SHORT ScrewMe_.00403BF5	; Boucle vers 403BF5

En résumé pour une chaîne que l'on appellera **abcdefghij** :

- On crée une nouvelle chaîne
- On met [valeur hexa du caractère **a** – 30(h)] fois le caractère **b**
- On met [valeur hexa du caractère **c** – 30(h)] fois le caractère **d**
- On met [valeur hexa du caractère **e** – 30(h)] fois le caractère **f**
- Ainsi de suite jusqu'à la fin de la chaîne
- On prend la nouvelle chaîne créée et on recommence la boucle 11 fois de plus sauf si la chaîne créée comporte plus de 4132 caractères (voir ligne **403C27**).

On voit qu'on a 2 sorties possibles : **403C6A** et **403C7A**. On regarde ces adresses :

00403C6A	PUSH DWORD PTR DS:[40A000]	; /hObject = 000001D8 (window)
00403C70	CALL <JMP.&kernel32.CloseHandle>	; \CloseHandle
00403C75	JMP ScrewMe_.00403D35	

⇒ Ferme le fichier nob qu'il vient de lire et saute vers **403D35**, ce qui correspond à un **RET**, donc **BAD BOY** ! (on peut le voir en cliquant sur  et en rentrant l'adresse de destination du saut).

00403C7A	PUSH DWORD PTR DS:[40A000]	; /hObject = 000001D8 (window)
00403C80	CALL <JMP.&kernel32.CloseHandle>	; \CloseHandle
00403C85	PUSH ScrewMe_.00410F14	; /String = "4444444....."
00403C8A	CALL <JMP.&kernel32.lstrlenA>	; \lstrlenA
00403C8F	TEST EAX,EAX	

⇒ Ferme le fichier nob qu'il vient de lire, récupère la longueur de la chaîne que l'on vient de créer et commence à travailler dessus, on est donc sur la vérification du résultat de la boucle d'où l'on vient. C'est la sortie qu'il faut prendre pour aller vers le Good Boy.

On va maintenant étudier la routine de comparaison :

00403C85	PUSH ScrewMe_.00410F14	; Passe la chaîne créée en argument
00403C8A	CALL <JMP.&kernel32.lstrlenA>	; Récupère la longueur de la chaîne et la met dans EAX
00403C8F	TEST EAX,EAX	; Vérifie EAX
00403C91	JLE ScrewMe_.00403D35	; Si EAX = 0 sort vers Bad Boy
00403C97	PUSH ScrewMe_.00412160	; Passe le sérial en argument
00403C9C	CALL <JMP.&kernel32.lstrlenA>	; Récupère la longueur du sérial et le met dans EAX
00403CA1	TEST EAX,EAX	; Vérifie EAX
00403CA3	JLE ScrewMe_.00403D35	; Si EAX = 0 sort vers Bad Boy
00403CA9	PUSH ScrewMe_.00412160	; Passe le sérial en argument
00403CAE	CALL <JMP.&kernel32.lstrlenA>	; Récupère la longueur du sérial et le met dans EAX
00403CB3	TEST EAX,EAX	; Vérifie EAX
00403CB5	CMP EAX,8	; Compare la longueur du sérial à 8
00403CB8	JA SHORT ScrewMe_.00403D35	; Si plus petit sort vers Bad Boy
00403CBA	MOV ESI,ScrewMe_.00410F14	; Met la chaîne dans ESI
00403CBF	MOV EDI,ScrewMe_.00412160	; Met le sérial dans EDI
00403CC4	XOR EBX,EBX	; Met EBX à zéro
00403CC6	XOR EDX,EDX	; Met EDX à zéro
00403CC8	PUSH EAX	; Sauvegarde EAX sur la pile
00403CC9	XOR EAX,EAX	; Met EAX à zéro
00403CCB	MOV AL,BYTE PTR DS:[EDX+ESI]	; Met le caractère EDX+1 dans EAX
00403CCE	ADD EBX,EAX	; Met dans EBX : EBX+EAX
00403CD0	INC EDX	; Ajoute 1 à EDX
00403CD1	POP EAX	; Récupère EAX (longueur du sérial)
00403CD2	CMP EAX,EDX	; Compare EAX à EDX (position dans la chaîne)
00403CD4	JNZ SHORT ScrewMe_.00403CC8	; Si inférieur boucle vers 403CC8
00403CD6	MOV EAX,0C	; Met 12 dans EAX
00403CDB	MUL EBX	; Multiplie EBX par 12 et met le résultat dans EAX
00403CDD	PUSH EAX	; Sauvegarde EAX
00403CDE	PUSH ScrewMe_.00412C45	; Passe le format "%d-Dyn" en argument
00403CE3	PUSH ScrewMe_.0040FB43	; Passe le résultat en argument
00403CE8	CALL DWORD PTR DS:[40FBB6]	; Formate le résultat
00403CEE	MOV EAX,ScrewMe_.0040FB43	; Met le résultat formaté dans EAX
00403CF3	MOV BL,BYTE PTR DS:[EDI]	; Récupère caractère n°x du sérial
00403CF5	CMP BL,BYTE PTR DS:[EAX]	; Compare avec caractère n°x du résultat
00403CF7	JNZ SHORT ScrewMe_.00403D35	; Si différent petit sort vers Bad Boy
00403CF9	INC EAX	; Ajoute 1 à EAX
00403CFA	INC EDI	; Ajoute 1 à EDI
00403CFB	CMP BL,0	; Vérifie si on est à la fin de la comparaison
00403CFE	JNZ SHORT ScrewMe_.00403CF3	; Si non, boucle vers 403CF3 (caractere suivant)
00403D00		; Continue vers messagebox Good Boy

En résumé :

- On vérifie que le sérial ne soit pas inférieur à 8
- On récupère les 8 premiers caractères de la chaîne et on additionne leurs valeurs hexa
- On multiplie le résultat par C(h) et on convertit en décimal
- On concatène le tout avec "-Dyn"
- On compare si c'est égal au sérial

Soit pour Kirjo :

Ce qui nous donne pour les 8 premiers caractères de la chaîne :

Page 14 sur 15

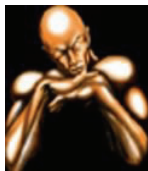
5 – Conclusion

On a maintenant tous les éléments pour coder notre keygen :

- Vérifier que le name soit au maximum de 8 caractères
- Additionner les valeurs décimales de chaque caractère du name
- Multiplier le résultat par 12
- Ajouter au résultat la chaîne " –Dyn "
- Vérifier que le sérial trouvé fasse 8 caractères
- Créer un fichier .nob avec le sérial comme nom
- Ecrire dans le fichier .nob une chaîne selon le calcul vu ci-dessus
- Afficher le sérial

Voilà, j'ai essayé d'être le plus détaillé et le plus clair possible. Si vous avez des questions ou des commentaires, rendez vous sur le [forum](#) de DeezDynasty.

A bientôt,



Kirjo
Mai 2008

Remerciements

- A Dynasty pour son crackme
- A Dynasty et Ezequi3l pour leur site et leur forum
- A tous les membres du forum qui le font vivre
- A tous ceux qui liront ce tutorial

D'autres tutos et crackmes sur :

