

**Cible :** Alone in the dark from **Kharneth**  
**Type:** Serial only  
**URL:** <http://kharneth.free.fr>  
**Niveau:** facile  
**Auteur :** Requiem [FRET]

**Outils utilisés :**  
PEiD, OllyDbg.



Alone in the dark fut en son temps le titre de toute une série de jeux bien connus, mais ce qui nous intéresse aujourd'hui est un crackme de **Kharneth**. Sur le site on peut trouver les indications suivantes :

« *MASM. Emulation d'instructions. Algo très simple.* »

En réalité ce qui m'a attiré dès le départ, c'est qu'il soit codé en asm, c'est si agréable. Le crackme est de type *serial only*. De la même manière que dans le DBKeygenme du même auteur, l'algorithme en lui-même n'est pas très dur mais il nous faut auparavant bien analyser le code pour pouvoir le comprendre et ne pas se faire dérouter.

Un scan sous PEiD ne nous révèle rien de particulier concernant un éventuel packeur, on va donc directement l'ouvrir dans Olly. La première protection de ce crackme réside dans la manière dont il a été codé, on ne trouve rien dans les SDR, ni dans les imports...comment savoir où se déroule la vérification.

Si on rentre un serial du type « 123 » pour tester, une message box nous indiquant l'erreur apparaît. C'est déjà une indication que l'on va mettre à profit. Si l'on étudie le code, on voit que les API sont toutes appelées de manière indirecte.

L'idée va donc être de chercher à placer un BP à l'intérieur même des **API** sensibles et l'on verra bien où l'on atterrira au retour de l'**API**, c'est-à-dire pour nous, on souhaite que ce soit dans la routine de vérification, voici la procédure à suivre:

On commence par se placer sur la table de correspondance des adresses des API et l'on regarde celles qui nous intéressent : **GetDlgItemTextA** et **MessageBoxA**. Click-droit, Go To, et on se déplace à l'adresse concernée :

00401516	68	DB 68	CHAR 'h'
00401517	CC	INT3	
00401518	.-FF25 14204000	JMP DWORD PTR DS:[<&kernel32.ExitProcess>]	kernel32.ExitProcess
0040151E	.-FF25 10204000	JMP DWORD PTR DS:[<&kernel32.GetModuleHandleA>]	kernel32.GetModuleHandleA
00401524	.-FF25 20204000	JMP DWORD PTR DS:[<&user32.DialogBoxParamA>]	user32.DialogBoxParamA
0040152A	.-FF25 1C204000	JMP DWORD PTR DS:[<&user32.EndDialog>]	user32.EndDialog
00401530	.-FF25 24204000	JMP DWORD PTR DS:[<&user32.GetDlgItemTextA>]	user32.GetDlgItemTextA
00401536	.-FF25 28204000	JMP DWORD PTR DS:[<&user32.MessageBoxA>]	user32.MessageBoxA
0040153C	.-FF25 00204000	JMP DWORD PTR DS:[<&comctl32.InitCommonControls>]	comctl32.InitCommonControls
00401542	.-FF25 08204000	JMP DWORD PTR DS:[<&gdi32.GetStockObject>]	GDI32.GetStockObject
00401548	00	DB 00	
00401549	00	DB 00	
0040154A	00	DB 00	

DS:[00402024]=77D6AC06 (user32.GetDlgItemTextA)

Address	Hex dump	ASCII
00403000	03 00 00 00 00 00 00 00	*.....
00403008	43 6F 6E 67 72 61 74 75	Congratu
00403010	6C 61 74 69 6F 6E 73 21	lations!

Enter expression to follow

77D6AC06

OK Cancel

Ensuite on pose le **Break Point** sur le **Ret** de l'API :

77D6AC06	8BFF	MOV EDI,EDI	
77D6AC08	55	PUSH EBP	
77D6AC09	8BEC	MOV EBP,ESP	
77D6AC0B	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
77D6AC0E	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
77D6AC11	E8 8EA6FBFF	CALL user32.GetDlgItem	
77D6AC16	85C0	TEST EAX,EAX	
77D6AC18	74 0E	JE SHORT user32.77D6AC28	
77D6AC1A	FF75 14	PUSH DWORD PTR SS:[EBP+14]	
77D6AC1D	FF75 10	PUSH DWORD PTR SS:[EBP+10]	
77D6AC20	50	PUSH EAX	
77D6AC21	E8 084CFDFF	CALL user32.GetWindowTextA	
77D6AC26	EB 0E	JMP SHORT user32.77D6AC36	
77D6AC28	837D 14 00	CMP DWORD PTR SS:[EBP+14],0	
77D6AC2C	74 06	JE SHORT user32.77D6AC34	
77D6AC2E	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]	
77D6AC31	C600 00	MOV BYTE PTR DS:[EAX],0	
77D6AC34	33C0	XOR EAX,EAX	
77D6AC36	5D	POP EBP	
77D6AC37	C2 1000	RETN 10	On pose un BP sur le ret qui sors de l'API
77D6AC39	90	NOP	

Ensuite on lance le crackme, on rentre un serial de test, bien sûr on break au niveau du **ret** (notre méthode est donc efficace), un **F7** pour sortir et nous voilà en devant la routine de vérification, elle est assez longue mais ne met en jeu que des opérations au final élémentaires du type **XOR**.

Comme **Kharneth** nous l'avais précisé, on trouve de l'émulation d'instructions, c'est-à-dire que certaines instructions relativement simples vont être réécrites...en plus compliquées !! C'est pourquoi on va devoir analyser correctement le code. On peut définir six parties dans le code :

- la 1<sup>ère</sup> vérifie la longueur du serial
- la 2<sup>nde</sup> le transforme en dword (suivant certaines règles)
- les 4 autres, vérifient les caractères du serial deux à deux.

Voici la première partie, qui constitue la récupération du serial pour le crackme. Le serial fera 8 caractères et pour simplifier la fonction mise en jeu ici agit comme une fonction StringToDword. Il faudra tout de même veiller à avoir le (caractère n) > (caractère n+1) sinon lors de certains or, on pourrai avoir des problèmes lors de la transformation.

00401285	. 50	PUSH EAX	la ou l'on atterit ??
00401286	. 8F05 00304000	POP DWORD PTR DS:[403000]	Ci-dessous, la routine de verification
00401288	. 51	PUSH ECX	
0040128C	. B9 04000000	MOV ECX,4	
00401291	> 50	PUSH EAX	
00401292	> FF0C24	DEC DWORD PTR SS:[ESP]	On soustrait 4 a la longueur
00401295	. ^E2 FB	LOOPD SHORT AloneInT.00401292	du name placee sur la pile
00401297	. 58	POP EAX	
00401298	. 59	POP ECX	
00401299	. ^78 56	JS SHORT AloneInT.004012F1	On doit donc avoir un serial 4 caractere minimum
0040129B	. 34 12	XOR AL,12	(longueur serial - 4) XOR 12
0040129D	. 8AC0	MOV AL,AL	
0040129F	. B1 16	MOV CL,16	
004012A1	. 2AC1	SUB AL,CL	
004012A3	. 9F	LAHF	(longueur serial - 4) XOR 12h = 16h
004012A4	. 80E4 40	AND AH,40	Load Register AH From Flags
004012A7	. 0FB6CC	MOVBX ECX,AH	on selectione le zero flag
004012A9	. ^E3 02	JECXZ SHORT AloneInT.004012AE	Le serial fera donc 8 caracteres
004012AC	. ^EB 05	JMP SHORT AloneInT.004012B3	Jump if Register (E)CX is Zero
004012AD	> ^E9 55010000	JMP AloneInT.00401408	
004012B3	> 68 4F304000	PUSH DWORD PTR DS:[40304F]	ASCII "123"
004012B8	. 59	POP ECX	met dans ECX l'adresse du serial
004012B9	. FF31	PUSH DWORD PTR DS:[ECX]	
004012BB	. 58	POP EAX	<=> mov eax, dword ptr ds:[ecx]
004012BC	. 35 14F5DE98	XOR EAX,98DEF514	on travaille sur les 4 premiers caracteres
004012C1	. 50	PUSH EAX	
004012C2	. 5A	POP EDX	<=> mov edx, eax
004012C3	. 35 24C5EEA8	XOR EAX,A8EEC524	
004012C8	. C1E8 04	SHR EAX,4	EAX = ( (4 premier char serial) XOR 30303030 ) SHR 4
004012CB	. 81F2 24C5EEA8	XOR EDX,A8EEC524	EDX = ( (4 premier char serial) XOR 30303030
004012D1	. 0AD0	OR DL,AL	
004012D3	. C1E8 04	SHR EAX,4	
004012D6	. 22F4	AND DH,AH	
004012D8	. C1E8 04	SHR EAX,4	
004012DB	. 0AF4	OR DH,AH	
004012DD	. 66:0915 043040	OR WORD PTR DS:[403004],DX	
004012E4	. 83C1 04	ADD ECX,4	
004012E7	. FF31	PUSH DWORD PTR DS:[ECX]	on travaille alors sur les 4 caracteres suivants
004012E9	. 58	POP EAX	
004012EA	. 35 14F5DE98	XOR EAX,98DEF514	
004012EF	. 50	PUSH EAX	
004012F0	. 5A	POP EDX	
004012F1	> 35 24C5EEA8	XOR EAX,A8EEC524	De toute cette partie ce qu'il faut retenir c'est qu'on
004012F6	. C1E8 04	SHR EAX,4	realise un string -> dword
004012F9	. 81F2 24C5EEA8	XOR EDX,A8EEC524	a condition que le caractere n soit inferieur au
004012FF	. 0AD0	OR DL,AL	caractere n+1, et ce pour les 8
00401301	. C1E8 04	SHR EAX,4	
00401304	. ^EB 06	JMP SHORT AloneInT.0040130C	
00401306	. E8	DB E8	
00401307	. ^E9 FC000000	JMP AloneInT.00401408	
0040130C	> 22F4	AND DH,AH	
0040130E	. C1E8 04	SHR EAX,4	
00401311	. 0AF4	OR DH,AH	
00401313	. 66:0915 063040	OR WORD PTR DS:[403006],DX	
0040131A	. FF35 04304000	PUSH DWORD PTR DS:[403004]	une fois le dword forme a partir des 8 premiers caracteres du serial
00401320	. 5B	POP EBX	on le met dans EBX

La structure à bien comprendre est avant tout, celle concernant l'instruction **lahf** (on la trouve notamment en 004012A3), elle remonte dans AH, la byte formée par les flags du processeur, ensuite le **and AH, 40** sélectionne le zéro flag. On déplace AH dans ECX et pour finir l'instruction **jecxz** vérifie si ECX = 0, la manière dont les sauts sont placés fait que l'on doit avoir ECX différent de zéro, donc le zéro flag présent, c'est-à-dire que les deux entités qui seront mises en jeu dans la structure de comparaison, devront être égales.



00401321	. C0C3 04	ROL BL,4	
00401324	. C0C7 04	ROL BH,4	
00401327	. 0FCB	BSWAP EBX	
00401329	. C0C7 04	ROL BH,4	
0040132C	. C0C3 04	ROL BL,4	
0040132F	. 8A15 9C124000	MOV DL,BYTE PTR DS:[40129C]	on se sert d'une byte du code : 12h
00401335	. 80F2 F6	XOR DL,0F6	DL = E4
00401338	. 80F3 C5	XOR BL,0C5	( 7eme et 8eme caracteres ) XOR C5 = E4 (cf ci dessous)
0040133B	. 8AC3	MOV AL,BL	
0040133D	. 8ACA	MOV CL,DL	CL = E4
0040133F	. 2AC1	SUB AL,CL	la manipulation sur le serial doit nous donner AL = CL
00401341	. 9F	LAHF	Load Register AH From Flags
00401342	. 80E4 40	AND AH,40	
00401345	. 0FB6CC	MOVZX ECX,AH	
00401348	. >E3 02	JECXZ SHORT AloneInT.0040134C	Jump if Register (E)CX is Zero
0040134A	. >EB 05	JMP SHORT AloneInT.00401351	il nous faut prendre ce saut
0040134C	. >E9 B7000000	JMP AloneInT.00401408	
00401351	. >C1EB 08	SHR EBX,8	
00401354	. 8A15 9B124000	MOV DL,BYTE PTR DS:[40129B]	on se sert d'une byte du code : 34h
0040135A	. 80F2 5E	XOR DL,5E	DL = 6A
0040135D	. 80F3 29	XOR BL,29	( 5eme et 6eme caracteres ) XOR 29 = 6A (cf ci dessous)
00401360	. 8AC3	MOV AL,BL	
00401362	. 8ACA	MOV CL,DL	
00401364	. 2AC1	SUB AL,CL	
00401366	. 9F	LAHF	
00401367	. 80E4 40	AND AH,40	Ici c'est le meme principe
0040136A	. 0FB6CC	MOVZX ECX,AH	on pourrait ecrire
0040136D	. >E3 02	JECXZ SHORT AloneInT.00401371	cmp dl, bl
0040136F	. >EB 05	JMP SHORT AloneInT.00401376	je suite
00401371	. >E9 92000000	JMP AloneInT.00401408	jne out
00401376	. >C1EB 08	SHR EBX,8	
00401379	. 8A15 9A124000	MOV DL,BYTE PTR DS:[40129A]	on se sert d'une byte du code : 56h
0040137F	. 80F2 8A	XOR DL,8A	DL = DC
00401382	. 80F3 B9	XOR BL,0B9	( 3eme et 4eme caracteres ) XOR B9 = DC (cf ci dessous)
00401385	. 8AC3	MOV AL,BL	On connait le principe
00401387	. 8ACA	MOV CL,DL	
00401389	. 2AC1	SUB AL,CL	
0040138B	. 9F	LAHF	
0040138C	. 80E4 40	AND AH,40	
0040138F	. 0FB6CC	MOVZX ECX,AH	
00401392	. >E3 02	JECXZ SHORT AloneInT.00401396	
00401394	. >EB 02	JMP SHORT AloneInT.00401398	
00401396	. >EB 70	JMP SHORT AloneInT.00401408	
00401398	. >C1EB 08	SHR EBX,8	
0040139B	. 8A15 99124000	MOV DL,BYTE PTR DS:[401299]	on se sert d'une byte du code : 78h
004013A1	. 80F2 35	XOR DL,35	DL = 4Dh
004013A4	. 80F3 CA	XOR BL,0CA	( 1eme et 2eme caracteres ) XOR CA = 4D (cf ci dessous)
004013A7	. 8AC3	MOV AL,BL	
004013A9	. 8ACA	MOV CL,DL	
004013AB	. 2AC1	SUB AL,CL	
004013AD	. 9F	LAHF	
004013AE	. 80E4 40	AND AH,40	
004013B1	. 0FB6CC	MOVZX ECX,AH	
004013B4	. >E3 02	JECXZ SHORT AloneInT.004013B8	
004013B6	. >EB 02	JMP SHORT AloneInT.004013BA	Jump vers message de reussite.

Je crois que le code est suffisamment commenté, on retrouve quatre fois la structure de comparaison que j'ai décrite précédemment, on vérifie les caractères du serial deux à deux, en se servant de bytes présentes dans le code du crackme, rien de réellement dur à reverser, principalement des xor.

Au final si l'on reverse cette routine on trouve comme serial : « 87654321 ». Nous sommes maintenant registered en bonne et due forme, donc crackme cracked. Après avoir localisée la routine de vérification, il est vraiment intéressant de faire le travail inverse pour voir quelles instructions sont émulées par le code.

**Greetz :** Kharneth pour son crackme, mes FRETiens préférés : Neitsa, ++ Meat, eedy31, DarKPhoeniX, \_TwG\_, BeatriX, tout ceux que je connais.

. Requiem [FRET] .