Ce document comprend le tut original de DeathSpawn (http://www.geocities.com/imdeathspawn), une traduction et quelques commentaires complémentaires afin de vous permettre de démarrer et comprendre quelques aspects du code de ce crackme, extrêmement simple mais appropriée à une première approche de l'utilisation de OllyDbg. Le crackme est notamment accessible sur la page de DeathSpawn (http://www.geocities.com/imdeathspawn/haque-abex5.zip).

CommComm, le 7 février 2004.
http://commcomm.free.fr/abextut05.pdf

Subject: Abex CrackMe5
Tools: OllyDbg
Difficulty: Newbie
Goal: Find serial number
Author: DeathSpawn
Included: This tutorial, CrackMe5

Let's run the file to see what happens first. Ok, so we are looking for a serial number in this program. Let's enter our handy 7777777 so it's easy to look for inside of the program. Hit the magic button and we get "The serial you entered is not correct". Bummer, but what did you expect ;)

Now let's load up OllyDbg and our program to find out what's going on. First step inside of Olly is to check out the text referrences. So right click and choose "search for" then choose "all referrenced string texts". Now let's look for the code that we got earlier. We find it along with a few other ASCII referrences. A couple of them look like serial numbers, so let's give 'em a try. Perhaps the programmer hard coded them into the program. So we enter L2C-5781 and 4562-ABEX respectively, but both come up with the bad code popup. Oh well, can't be that lucky every time :)

So let's double click on the ASCII "Yep, you entered the correct serial!". Now we are looking at the actual code here inside of the program. Now if we look a few lines above, we will see our bad code popup information as well. So we are in the right area to begin our search. Now if we look at the following lines of code, we see something very interesting:

```
004010FF  |. /74 16        JE SHORT abexcm5.00401117       ;  JUMP OVER BAD CODE IF EQUAL
00401101  |. |6A 00        PUSH 0                          ; /Style = MB_OK|MB_APPLMODAL
00401103  |. |68 34244000  PUSH abexcm5.00402434           ; |Title = "Error!"
00401108  |. |68 3B244000  PUSH abexcm5.0040243B           ; |Text = "The serial you entered is not correct!"
0040110D  |. |FF75 08      PUSH [ARG.1]                    ; |hOwner = 00401000
00401110  |. |E8 56000000  CALL <JMP.&USER32.MessageBoxA>  ; \MessageBoxA
00401115  |. |EB 16        JMP SHORT abexcm5.0040112D      ;  JUMP OVER GOOD CODE
```

Now the section here is fully commented. All of the comments in all caps are comments by me and the rest is done by OllyDbg for us. So the line 004010FF jumps over the bad "the serial you entered is not correct!" section if something is equal (JE = Jump on equal). Then the line 00401115 jumps over the good code after running the information for the bad code (JMP = unconditional jump).

So let's examine the code line directly above that JE jump:

```
004010FC  |.  83F8 00      CMP EAX,0                       ;  COMPARE SOME STUFF HERE
```

Interesting. Let's place a breakpoint here by hitting the F2 key. Now you will notice the line number highlights up in red, which tells us that there is a breakpoint set here. Now let's hit the F9 key to run the program and let's enter our 7777777 number again to see what happens.

Now after we enter our serial number and hit the magic button, we come to a stop on the compare line. Nothing really much to see here, but if we look down and to the left at our memory dump section, we can see something very interesting there. Look under the ASCII section and you will see the following:

L2C-57816784-ABEX

Interesting. Now, if we look above in our CPU section again we will also notice that we see this information repeated 3x directly above our breakpoint. Now Olly has nicely commented this information for us:

```
004010CF  |.  68 FD234000   PUSH abexcm5.004023FD           ; /StringToAdd = "L2C-5781"
004010D4  |.  68 00204000   PUSH abexcm5.00402000           ; |ConcatString = "L2C-57816784-ABEX"
004010D9  |.  E8 63000000   CALL <JMP.&KERNEL32.lstrcatA>    ; \lstrcatA
004010DE  |.  68 5C224000   PUSH abexcm5.0040225C           ; /StringToAdd = "6784-ABEX"
004010E3  |.  68 00204000   PUSH abexcm5.00402000           ; |ConcatString = "L2C-57816784-ABEX"
004010E8  |.  E8 54000000   CALL <JMP.&KERNEL32.lstrcatA>    ; \lstrcatA
004010ED  |.  68 24234000   PUSH abexcm5.00402324           ; /String2 = "7777777"
004010F2  |.  68 00204000   PUSH abexcm5.00402000           ; |String1 = "L2C-57816784-ABEX"
004010F7  |.  E8 51000000   CALL <JMP.&KERNEL32.lstrcmpiA>   ; \lstrcmpiA
```

Very nice. So, from the comments, we can see that the string L2C-5781 is added a blank and then to the string 6784-ABEX and then finally compared with our string in pulled in from line 004010ED. So let's hit F9 again. Then hit Ctrl+F2 to restart the program. Finally, hit F9 to run the program one last time and let's enter that suspicious serial information that we just saw earlier.

Bang! We get the proper popup code, "Yep, you entered the correct serial!" So we have cracked the CrackMe 5. Not too bad and now we also understand what the 2 ASCII strings were from earlier that we tried to enter as serial numbers.

Now this brings up something interesting because why would the program add the first and second strings together to blanks? Now there is an interesting code protection which takes the name of the hard drive and converts it, then uses it inside of the serial comparision. Now my hard drive is not named anything, so that could be why the blanks. So I will try to rename my hard drive as DS and run our proggie inside of Olly again. Once I do this, I hit my breakpoint and find the following:

```
004010ED  |.  68 24234000   PUSH abexcm5.00402324           ; /String2 = "7777777"
004010F2  |.  68 00204000   PUSH abexcm5.00402000           ; |String1 = "L2C-5781FU6762-ABEX"
004010F7  |.  E8 51000000   CALL <JMP.&KERNEL32.lstrcmpiA>   ; \lstrcmpiA
```

So we see that the program converts my DS into FU and then adds the L2C-5781 and 6782-ABEX to both sides, then compares it vs the serial entered before. So we can look into the hard drive name conversion algorithm, but since this is just a newbie tut and Olly shows us exactly what to look for, regardless of what the name of the hard drive. We will look into conversion algorithms later. So for now, enjoy :)

That about wraps it up for this tut.

Talk at ya later.....

DS

Sujet: Abex CrackMe5
Outils: OllyDbg
Difficulté: Newbie
But: Trouver un serial
Auteur: DeathSpawn
Inclus: This tutorial, CrackMe5

Lançons d'abord le fichier pour voir ce qui se passe. Bon, on voit qu'on est à la recherche d'un serial dans ce programme. Entrons notre 7777777, bien pratique à chercher à l'intérieur du programme. On appuie sur le bouton magique et on obtient "The serial you entered is not correct". Sale coup, mais vous vous attendiez à quoi ;)

Maintenant, chargeons OllyDbg et notre programme [File – Open] pour voir ce qui se trame. Le premier pas dans Olly consiste à jeter un œil aux références texte. Donc, on clique droit et on choisit « Search for » puis «All referenced text strings" (1). Maintenant, cherchons le code qu'on a obtenu tout à l'heure. On le trouve parmi quelques autres chaînes ASCII.

Text strings referenced in abexcm5:CODE

Address	Disassembly	Text string
00401024	CALL <JMP.&KERNEL32.ExitProcess>	(Initial CPU selection)
0040109E	PUSH abexcm5.004023F3	ASCII "4562-ABEX"
004010CF	PUSH abexcm5.004023FD	ASCII "L2C-5781"
00401103	PUSH abexcm5.00402434	ASCII "Error!"
00401108	PUSH abexcm5.0040243B	ASCII "The serial you entered is not correct!"
00401119	PUSH abexcm5.00402406	ASCII "Well Done!"
0040111E	PUSH abexcm5.00402411	ASCII "Yep, you entered a correct serial!"

Deux d'entre elles ressemblent à des serials : essayons voir. Peut-être le programmeur les a-t-il codés en dur dans le programme ? Donc, on entre respectivement L2C-5781 et 4562-ABEX, mais les deux nous retournent le popup avec le mauvais message. Bon, on ne peut pas avoir de la chance à tous les coups :)

On double clique donc sur la chaîne « Yep, you entered the correct serial ». On cherche ainsi le véritable code dans le programme (2). Et si on regarde quelques lignes au-dessus, on verra aussi le texte de notre mauvais popup. On est donc bien dans la bonne zone du code pour entamer les recherches. Et si maintenant on regarde les lignes de code qui suivent, on voit quelque chose de très intéressant.

```
004010FF   |. 74 16          JE SHORT abexcm5.00401117          SAUTE PAR DESSUS LE MAUVAIS CODE SI EGAL
00401101   |. 6A 00          PUSH 0                             ; /Style = MB_OK|MB_APPLMODAL
00401103   |. 68 34244000    PUSH abexcm5.00402434              ; |Title = "Error!"
00401108   |. 68 3B244000    PUSH abexcm5.0040243B              ; |Text = "The serial you entered is not correct!"
0040110D   |. FF75 08        PUSH DWORD PTR SS:[EBP+8]          ; |hOwner
00401110   |. E8 56000000    CALL <JMP.&USER32.MessageBoxA>     ; \MessageBoxA
00401115   |. EB 16          JMP SHORT abexcm5.0040112D         SAUTE PAR DESSUS LE BON CODE
00401117   |> 6A 00          PUSH 0                             ; /Style = MB_OK|MB_APPLMODAL
00401119   |. 68 06244000    PUSH abexcm5.00402406              ; |Title = "Well Done!"
0040111E   |. 68 11244000    PUSH abexcm5.00402411              ; |Text = "Yep, you entered a correct serial!"
00401123   |. FF75 08        PUSH DWORD PTR SS:[EBP+8]          ; |hOwner
00401126   |. E8 40000000    CALL <JMP.&USER32.MessageBoxA>     ; \MessageBoxA
```

Là, la section est commentée de façon très complète. Tous les commentaires en majuscules sont de moi et le reste a été réalisé par Olly pour nous. Donc, la ligne 004010FF saute en 00401117, donc par dessus la mauvaise section de code « the serial you entered is not correct ! », si quelque chose [le contenu du registre EAX] est égal à 0 (JE = Jump on Equal, saut conditionnel). Puis de la ligne 00401115, après avoir généré l'information pour le mauvais code, on saute en 0040112D donc par-dessus le bon code (JMP = Jump, saut non conditionnel, effectué à chaque fois).

Examinons donc la ligne de code juste au-dessus de ce saut conditionnel JE [004010FC].

Intéressant. Plaçons un point d'arrêt à cet endroit (004010FC), avec la touche F2. Vous remarquerez maintenant que le numéro de la ligne est surligné en rouge, ce qui nous indique l'existence d'un point d'arrêt. Maintenant on actionne F9 pour lancer le programme [Run] et entrer à nouveau notre 7777777 puis voir ce qui se passe.

Une fois entré notre serial et cliqué sur le bouton magique, on arrive sur le point d'arrêt de la ligne de comparaison (004010FC : CMP EAX,0). Pas grand-chose à voir par ici en vérité (3), mais si on regarde dans la section du dump mémoire [fenêtre inférieure gauche], on peut y voir quelque chose de très intéressant. Regardez dans la colonne ASCCI et vous y verrez :

L2C-57816784-ABEX

Intéressant. Maintenant, si on regarde au dessus dans notre section CPU (le code du programme), on voit que cette information figure trois fois directement au dessus de notre point d'arrêt. Et Olly a gentiment commenté tout ça pour nous.

```
004010CF   |. 68 FD234000   PUSH abexcm5.004023FD                ; /StringToAdd = "L2C-5781"
004010D4   |. 68 00204000   PUSH abexcm5.00402000                ; |ConcatString = "L2C-57816784-ABEX"
004010D9   |. E8 63000000   CALL <JMP.&KERNEL32.lstrcatA>        ; \lstrcatA
004010DE   |. 68 5C224000   PUSH abexcm5.0040225C                ; /StringToAdd = "6784-ABEX"
004010E3   |. 68 00204000   PUSH abexcm5.00402000                ; |ConcatString = "L2C-57816784-ABEX"
004010E8   |. E8 54000000   CALL <JMP.&KERNEL32.lstrcatA>        ; \lstrcatA
004010ED   |. 68 24234000   PUSH abexcm5.00402324                ; /String2 = "7777777"
004010F2   |. 68 00204000   PUSH abexcm5.00402000                ; |String1 = "L2C-57816784-ABEX"
004010F7   |. E8 51000000   CALL <JMP.&KERNEL32.lstrcmpiA>       ; \lstrcmpiA
004010FC   |. 83F8 00       CMP EAX,0
```

Très sympa. Donc, d'après les commentaires, on peut voir que la chaîne L2C-5781 est complétée par une chaîne vide (4) puis par la chaîne 6784-ABEX et enfin comparée avec notre chaîne tirée de la ligne 004010ED [le serial proposé, 7777777]. On redonne un coup de F9 [Run pour terminer] et CTRL-F2 [Restart] pour recharger le programme dans Olly (5). Et enfin F9 pour relancer le programme une dernière fois et nous permettre d'entrer ce curieux serial [L2C-57816784-ABEX] que nous avons vu juste avant.

Bingo ! On obtient le bon popup. « Yep, you entered the correct serial! » (6). Nous avons donc craqué le CrackMe5. Pas si mal… et maintenant on comprend ce qu'étaient les deux chaînes que nous avions essayé d'entrer précédemment comme des serials.

Ceci montre quelque chose d'intéressant : pourquoi le programme ajoute-t-il la première et la seconde chaînes ensemble avec une chaîne vide ? Eh bien, c'est une protection intéressante qui prend le nom du disque dur et le convertit, puis l'utilise dans la comparaison du serial. Mon disque dur n'ayant pas de nom, ça explique la chaîne vide. Je vais donc essayer de renommer mon disque « DS » puis de relancer notre prog depuis Olly. Une fois ça effectué, je remets mon point d'arrêt et je trouve les choses suivantes :

```
004010ED   |. 68 24234000   PUSH abexcm5.00402324                ; /String2 = "7777777"
004010F2   |. 68 00204000   PUSH abexcm5.00402000                ; |String1 = "L2C-5781FU6762-ABEX"
004010F7   |. E8 51000000   CALL <JMP.&KERNEL32.lstrcmpiA>       ; \lstrcmpiA
```

On voit donc que le programme convertit mon DS en FU puis ajoute les chaînes « L2C-5781 » et « 6782-ABEX » de chaque côté (7), puis fait la comparaison avec le serial entré préalablement. On peut regarder dans l'algorithme de conversion du nom du disque dur, mais comme c'est juste un tut pour newbies et qu'Olly nous montre exactement quoi chercher, sans se soucier de ce qu'est le nom du disque dur. Nous regarderons les algorithmes de conversion plus tard (8). Donc pour le moment, bon amusement :)

Voilà ce tut à peu près bouclé comme ça (9).

A plus…

DS

Commentaires par CommComm.

(1) Pour faire apparaître « toutes les chaînes de caractères référencées » (« All referenced text strings »), il faut cliquer droit sur la fenêtre supérieure gauche (là où apparaît le code). Chacune des quatre fenêtres a en effet son propre menu contextuel. Seule la fenêtre supérieure gauche fait apparaître la fonction de recherche « Search for » qui permet, entre autres de créer une nouvelle fenêtre dans laquelle apparaîtront toutes les chaînes de caractères. Le « Search for » permet aussi d'accéder à la liste des API avec « Names » ou directement par CTRL-N).

(2) Le double clic est effectué sur le texte de la chaîne recherchée, à partir de la fenêtre des chaînes ASCII (« Text strings referenced in abexcm5:CODE »). Le double clic aura pour effet de réafficher la fenêtre principale de Olly « CPU - main thread, module abexcm5 » et de placer en haut de la fenêtre du code, surlignée en gris, la ligne contenant la chaîne cherchée. Attention toutefois, la chaîne peut se trouver à plusieurs endroits différents dans le code (ce n'est pas le cas ici).

(3) Notez quand même au passage que le registre EAX est égal à 1. On le voit soit dans la fenêtre supérieure droite consacrée aux registres, soit en bas de la fenêtre du code, au dessus de la ligne de titre de la section dump.

(4) En fait le schéma présenté par DeathSpawn ne montre pas la chaîne vide. En effet, c'est la chaîne vide qui reçoit le résultat de la concaténation. Donc, une fois le code effectué, les commentaires font apparaître non plus une chaîne vide mais la chaîne résultante. Avant l'exécution du ode, on voit effectivement la chaîne StringToAdd « L2C-5781 » poussée à partir de 004023FD et la chaîne vide (ConcatString) poussée à partir de 00402000. Une fois que lstrcatA a été exécutée en 004010D9, 00402000 contient « L2C-5781 » (concaténée avec une chaîne vide donc inchangée) puis une fois ajoutée la StringToAdd « 6784-ABEX » par lstrcatA en 004010E8, la ConcatString en 00402000 contient « L2C-57816784-ABEX ». C'est cette valeur qui apparaît dans le tut de DeathSpawn et que j'ai repris dans la traduction. Avant exécution du code, on voit bien la ConcatSTring vide apparaître effectivement dans les commentaires.

```
004010CF    |. 68 FD234000      PUSH ABEXCM5.004023FD          ; /StringToAdd = "L2C-5781"
004010D4    |. 68 00204000      PUSH ABEXCM5.00402000          ; |ConcatString = ""
004010D9    |. E8 63000000      CALL <JMP.&KERNEL32.lstrcatA>  ; \lstrcatA
004010DE    |. 68 5C224000      PUSH ABEXCM5.0040225C          ; /StringToAdd = "4562-ABEX"
004010E3    |. 68 00204000      PUSH ABEXCM5.00402000          ; |ConcatString = ""
004010E8    |. E8 54000000      CALL <JMP.&KERNEL32.lstrcatA>  ; \lstrcatA
```

Pour suivre l'évolution de la ConcatString en 00402000, mettez un point d'arrêt en 004010D4 puis avancez à coup de F8.

(5) Ce qui va terminer le programme, puisqu'après cet arrêt imposé il n'y en a plus d'autre. Dans certains programmes, on boucle en attendant un nouveau serial. Ici en revanche, on sort complètement du programme qu'il convient de recharger dans Olly avec CTRL-F2 (Menu Debug/Restart).

(6) Si l'on avait placé à nouveau un point d'arrêt en 004010FC, on aurait pu voir que cette fois le registre EAX était égal à 0. Si vous regarder la ligne 004010ED, vous verrez qu'elle contient, comme lors du premier essai, la valeur du serial entré. Ce n'est bien sûr plus 7777777, mais L2C-57816784-ABEX. Ce qui permet de comprendre ce qui se passe autour de notre point d'arrêt. Les deux chaînes figurant en 004010ED et 004010F2 sont comparées l'une à l'autre par l'API lstrcmpiA qui est en 004010F7. Celle-ci ignore la casse des chaînes comparées. Si les deux chaînes sont différentes, la fonction renvoie la valeur 1 dans le registre EAX. Si au contraire les deux chaînes sont identiques (à la casse près), l'API retourne 0 dans EAX. La fonction CMP EAX compare le contenu de EAX à 0 : si EAX = 0, on sautera par-dessus le code du mauvais popup pour atteindre bon grâce au saut conditionnel (JE). Si EAX=1, on ne saute pas et on rentre dans la section correspondant au mauvais popup. Ce n'est qu'à la fin de cette séquence affichant le mauvais popup, qu'on sautera par-dessus la séquence de bon code et qu'on se retrouvera en 0040112D pour terminer le programme. Si quand vous êtes sur le point d'arrêt de la ligne de comparaison d'EAX, vous redémarrez non par F9 mais par F7 ou F8 (mode « pas à pas »), vous verrez qu'en dessous de la section du code programme, il est indiqué « Jump is taken », signifiant que le saut conditionnel va être effectué (9).

(7) En réalité, la routine convertit effectivement bien le nom du disque « DS » en « FU » mais également les deux premiers caractères de la chaîne « 4562-ABEX » puisque ce sont quatre caractères et non deux seulement qui sont convertis comme indiqué dans l'approche sur la routine de conversion du nom du disque ci-dessous. Donc « DS » devient « FU » mais aussi « 45 » devient « 67 », le reste demeurant inchangé (6762-ABEX et non 6782-ABEX comme indiqué dans l'original). Si le nom du disque avait comporté plus de trois caractères, on

aurait retrouvé la chaîne « 4562-ABEX » inchangée. Dans le cas où le disque n'est pas nommé, ce sont les caractères « 4562 » qui sont convertis en « 6784 » (voir ci-dessous).

(8) En 00401099, on voit un appel (CALL) à l'API GetVolumeInformationA. Placer un point d'arrêt sur cette ligne. Lancer le crackme par F9. Quand on arrive sur la ligne d'appel de l'API, on voit en dessous que la ligne 0040109E pousse sur la pile une « StringToAdd », une chaîne à ajouter, qui figure en 004023F3 et qui est « 4562-ABEX ». Dans la ligne d'après, on voit que c'est une « ConcatString », une chaîne à concaténer à la suite de la précédente, qui est vide (deux guillemets accolés) et poussée depuis 0040225C. On continue l'exécution du programme avec F8 (pas F7 qui entrerait dans le détail de l'API, ce qui ne présente à ce stade aucun intérêt). Et hop… on voit que la fonction GetVolumeInformationA qui porte bien son nom a retourné le nom du volume (DS) dans la ConcatString en 0040225C, et que cette valeur est poussée sur la pile. On la voit aussi apparaître plus bas au fur et à mesure qu'on avance dans le code. Lorsque le disque n'est pas nommé, c'est une chaîne vide qui est utilisée. Encore trois coups de F8 pour passer l'API suivante (lstrcatA). On est donc arrivé en 004010AD et on prend le temps de regarder le registre EAX dans la fenêtre supérieure droite. Il contient « DS4562-ABEX », valeur figurant désormais en 0040225C (nouvelle ConcatString) renvoyée par l'API lstrcatA, qui se compose de la ConcatString initiale (« DS ») suivie de la StringToAdd (« 4562-ABEX »). MOV DL,2 met DL (registre EDX) à 2. Les quatre lignes suivantes vont ajouter 1 aux quatre caractères qui sont en 40225C, 40225D, 40225E et 40225F. Les caractères initiaux étant D, S, 4 et 5, on les voit passer respectivement à E, T, 5 et 6. La ConcatString et EAX contiennent alors ET5662-ABEX. En 4010CB, on décrémente ensuite DL qui initialement valait 2 et passe donc à 1. On teste ensuite cette valeur et on repart en 4010AF tant que la valeur décrémentée n'est pas nulle. Ce qui veut dire que le saut conditionnel JNZ ne sera effectué qu'une fois et qu'on sera au total passé deux fois dans la séquence (voir dans les commentaires « Jump is taken » ou « Jump is not taken »). Ce deuxième et dernier passage va donc créer une ConcatString FU6762-ABEX, les quatre premiers caractères étant passés de DS45 à ET56 puis à FU67. DL étant à 0, on ne repart pas en 4010AF et on continue en 4010CF où on pousse sur la pile la chaîne « L2C-5781 » qui existe initialement dans le code du programme.

(9) Encore un petit truc à ajouter. Sélectionnez la ligne 4010FF qui contient « 74 16 » le 74 indiquant un saut conditionnel « JE SHORT ». Clic-droit, puis dans le menu contextuel « Binary » et enfin Edit. Dans la boîte qui apparaît, là où le curseur clignote sur 74, entrez « EB » ce qui correspond au code d'un saut inconditionnel « JMP SHORT », comme il en existe un par exemple en 401115. La ligne « 74 16 » devient donc « EB 16 ». Que va-t-il se passer lors de l'exécution du crackme ? Vous avez compris : puisque le saut est inconditionnel, on sautera *toujours* par dessus la séquence du mauvais popup qui sera systématiquement ignorée. Bref, le programme est patché et la validation sera obtenue avec n'importe quel serial entré. Le menu View/patches (ou CTRL-P) permet d'ouvrir une fenêtre avec les patches en cours et l'indication de l'ancienne valeur et de la nouvelle. Si vous voulez réellement enregistrer la modification dans l'exécutable, la ligne patchée étant sélectionnée, faites un clic-droit puis « Copy to executable » et « Selection » (« All modifications » permettrait de valider tous les patches en cours). Une fenêtre avec le code modifié apparaît, indiquant dans la barre de titre le chemin du programme. La commande « Save File » du menu contextuel (clic-droit) permet d'enregistrer l'exécutable sous un autre nom et conserver l'original intact. Si vous fermez directement la fenêtre sans passer par le menu contextuel, un message d'alerte vous rappelle que vous allez modifier le programme. En cliquant «Non » vous n'enregistrez pas la modification. En cliquant « Oui », il vous restera à indiquer, comme avec « Save File » le nom du fichier dans lequel vous voulez sauvegarder les modifications.