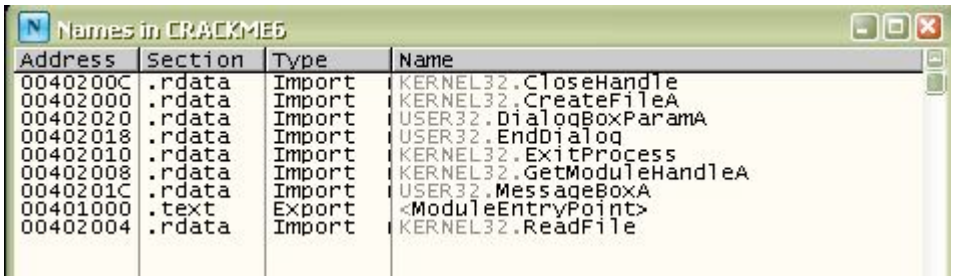
* Cracker un KeyfileMe
* Patcher pour avoir le GoodBoy
* Trouver le contenu du keyfile sans patcher

Observation du CrackMe:
• Boite de diaologue “Invalid Key” • Protection par KeyFile
Outils:
• OllyDbg 1.10
Objectifs:
• Obtenir un message de réussite • Trouver le contenu du KeyFileMe • Créer un KeyFileMe valide

On a affaire ici à un keyfileme très simple, vous le verrez très vite: le but est simplement de se familiariser avec un schéma de protection que nous n’avons pas encore étudié.

Premier reflexe, comme d’habitude, ouvrir PEiD et voir ce qu’il peut nous indiquer sur une protection supplémentaire du crackme, comme le packing ou encore la crypto. Ici, rien de particulier à signaler…

On ouvre donc le crackme dans OllyDebugger. Pour éviter d’utiliser la technique des Text Strings, on va donc faire “**CTRL + N**” dans Olly, et voir quelles API sont utilisées et celles qui pourraient nous intéresser. Il se trouve que, par chance, et vu la simplicité de programmation de ce crackme, on en a très peu.



Dans le cadre d'un KeyFileMe, on retrouve les deux principales qui peuvent nous intéresser: **CreateFileA** et **ReadFile** (sans parler de **MessageBoxA** que vous connaissez sans doute).
Voici ce que nous dit MSDN sur la function **CreateFile**:

```
HANDLE CreateFile(

  LPCTSTR lpFileName,     // pointer to name of the file
  DWORD dwDesiredAccess,          // access (read-write) mode
  DWORD dwShareMode,  // share mode
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,// pointer to security attributes
  DWORD dwCreationDistribution,  // how to create
  DWORD dwFlagsAndAttributes,    // file attributes
  HANDLE hTemplateFile   // handle to file with attributes to copy
 );


Parameters

lpFileName

Points to a null-terminated string that specifies the name of the object (file, pipe, mailslot,
communications resource, disk device, console, or directory) to create or open.

If *lpFileName is a path, there is a default string size limit of MAX_PATH characters. This
limit is related to how the CreateFile function parses paths.
```

… et sur **ReadFile**:

```
BOOL ReadFile(

  HANDLE hFile,    // handle of file to read
  LPVOID lpBuffer, // address of buffer that receives data
  DWORD nNumberOfBytesToRead,          // number of bytes to read
  LPDWORD lpNumberOfBytesRead,         // address of number of bytes read
  LPOVERLAPPED lpOverlapped   // address of structure for data
 );


Parameters

hFile

Identifies the file to be read. The file handle must have been created with
GENERIC_READ access to the file.

Windows NT

For asynchronous read operations, hFile can be any handle opened with the
```

FILE_FLAG_OVERLAPPED flag by the CreateFile function, or a socket handle returned by the socket or accept functions.

Windows 95

For asynchronous read operations, hFile can be a communications resource, mailslot, or named pipe handle opened with the FILE_FLAG_OVERLAPPED flag by CreateFile, or a socket handle returned by the socket or accept functions. Windows 95 does not support asynchronous read operations on disk files.

lpBuffer

Points to the buffer that receives the data read from the file.

nNumberOfBytesToRead

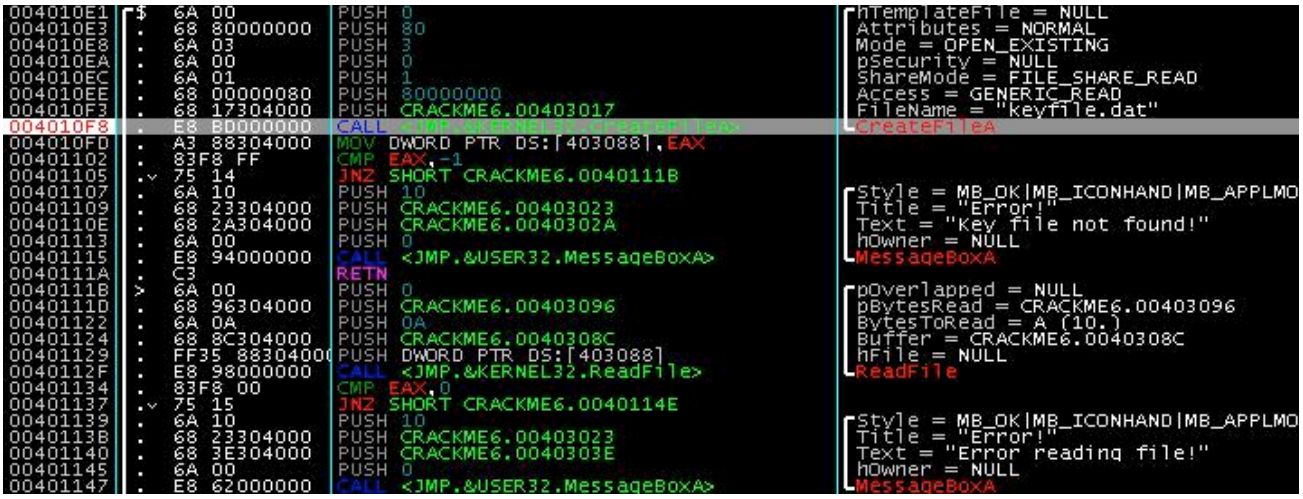
Specifies the number of bytes to be read from the file.

lpNumberOfBytesRead

Points to the number of bytes read. ReadFile sets this value to zero before doing any work or error checking. If this parameter is zero when ReadFile returns TRUE on a named pipe, the other end of the message-mode pipe called the WriteFile function with nNumberOfBytesToWrite set to zero.
If lpOverlapped is NULL, lpNumberOfBytesRead cannot be NULL.
If lpOverlapped is not NULL, lpNumberOfBytesRead can be NULL. If this is an overlapped read operation, you can get the number of bytes read by calling GetOverlappedResult. If hFile is associated with an I/O completion port, you can get the number of bytes read by calling GetQueuedCompletionStatus.

On va donc poser un BP sur CreateFileA puis lancer le debugging avec F9… Le crackme s'ouvre, on va alors cliquer sur "Check it!" et voir ce que ça nous donne.

On arrive ici:



On remarque bien sûr les lignes suivantes, qui vont nous donner des pistes quant à ce que le crackme va chercher:

```
004010E1  /$  6A 00        PUSH 0                          ; /hTemplateFile = NULL
004010E3  |.  68 80000000   PUSH 80                             ; |Attributes = NORMAL
004010E8  |.  6A 03        PUSH 3                          ; |Mode = OPEN_EXISTING
004010EA  |.  6A 00        PUSH 0                          ; |pSecurity = NULL
004010EC  |.  6A 01        PUSH 1                          ; |ShareMode =
FILE_SHARE_READ
004010EE  |.  68 00000080   PUSH 80000000                        ; |Access =
GENERIC_READ
004010F3  |.  68 17304000   PUSH CRACKME6.00403017               ; |FileName =
"keyfile.dat"
004010F8  |.  E8 BD000000   CALL <JMP.&KERNEL32.CreateFileA>        ;
\CreateFileA
```

Notre KeyFile doit s'appeler "***Keyfile.dat***" – on va alors créer un fichier à l'aide de notepad et changer l'extension, en respectant la casse ou bien le crackme risque de ne pas le reconnaître. Autant ne pas prendre de risque!

On commence à exécuter ligne par ligne en faisant F8, et on remarque qu'on passe au-dessus de la routine qui va nous afficher le message "Keyfile not found" – normal, puisqu'on vient de le créer. Le crackme compare le résultat obtenu avec -1:

00401102 |. 83F8 FF CMP EAX,-1 // cherche si le keyfile existe avec le nom correct
00401105 |. 75 14 JNZ SHORT CRACKME6.0040111B // s'il existe, on saute au-dessus du premier bad boy ;)

La routine qui suite va lire le contenu de ce fichier qu'on vient de créer.

```
0040111B  |> \6A 00        PUSH 0                          ; /pOverlapped = NULL
0040111D  |.  68 96304000   PUSH CRACKME6.00403096         ; |pBytesRead =
CRACKME6.00403096
00401122  |.  6A 0A        PUSH 0A                         ; |BytesToRead = A (10.)
00401124  |.  68 8C304000   PUSH CRACKME6.0040308C          ; |Buffer =
CRACKME6.0040308C
00401129  |.  FF35 88304000 PUSH DWORD PTR DS:[403088]      ; |hFile =
000000E0 (window)
0040112F  |.  E8 98000000   CALL <JMP.&KERNEL32.ReadFile>   ; \ReadFile
```

On voit que le fichier va chercher 10 Bytes dans le fichier. Si le nombre de bytes lu est correct, l'API retourne la valeur 0 – on va donc passer cela, il serait facile de patcher, surtout sur le reste du crackme pour obtenir le message de réussite, mais ce n'est pas notre but!

Le crackme compare le nombre de bytes à 0. On va pour l'instant se contenter de contourner ceci en rentrant n'importe quel chiffre dans le keyfile.dat (par exemple "1") pour lui montrer qu'il y a quelquechose dedans. Le saut est alors pris, et on passé la routine suivante qui nous dirait qu'il n'a pas pu lire le fichier correctement.

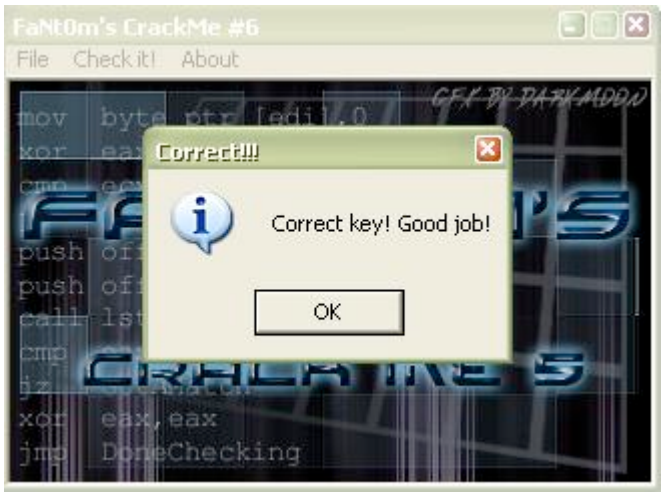
La routine qui suit est la plus importante: c'est celle qui va charger le contenu de notre keyfile et l'analyser…

```
0040114E 8D158C304000      lea edx, dword ptr [0040308C]   // charge le contenu de
notre keyfile
:00401154 83C205           add edx, 00000005               // regarde le 6ème chiffre de
notre keyfile
:00401157 803A31           cmp byte ptr [edx], 31          // regarde si c'est un 1
:0040115A 7510             jne 0040116C                    // saut vers bad boy si différent
de 1
:0040115C 83C203           add edx, 00000003               // regarde le 9ème chiffre du
keyfile
:0040115F 803A33           cmp byte ptr [edx], 33          // compare à 3
:00401162 7508             jne 0040116C                    // saut vers bad boy si different
de 3
:00401164 42               inc edx                         // regarde le 10ème chiffre du
keyfile
:00401165 803A30           cmp byte ptr [edx], 30          // compare à 0
:00401168 7402             je 0040116C                     // saut vers bad boy si c'est un 0
:0040116A EB15             jmp 00401181                    // saute vers le good boy
```

L’analyse est assez simple: le keyfile peut contenir n’importe quoi du moment qu’il y a

- 10 chiffres
- que le 6eme est bien un 6
- que le 9eme est bien un 3
- que le 10eme est different de 0

Vous pouvez donc essayer avec ces 4 parametres de remplir votre keyfile.dat, et voici le résultat:



Félicitations!

Dynasty – 04 Janvier 2008