Apple - QuickTime Player
Déréférence Arbitraire de Pointeur

Fournisseur:
Apple

Système affecté :
QuickTime Player v7.1.5 for Windows
QuickTime Player v7.2 for Windows
QuickTime Player v7.3 for Windows

Chronologie de divulgation :
29/08/2007 : Notification initiale chez le fournisseur.
29/08/2007 : Réponse initiale du fournisseur.
12/12/2007 : Divulgation publique en accord avec le fournisseur.

Références CVE :
CVE-2007-4707

Informations complémentaires :
http://docs.info.apple.com/article.html?artnum=307176-fr

Introduction :

Il existe dans le Parser de QuicktimePlayer une vulnérabilité au niveau du traitement des fichiers au format Flash (.swf). Cette vulnérabilité permet à un attaquant, à l'aide d'un fichier swf spécialement conçu, d'exécuter du code arbitraire à distance.

Description générale :

Le problème se trouve dans le module QuickTime.qts.
Lors du traitement d'un fichier .swf, le Parser utilise un champ décrivant le format des bitmaps d'une manière incorrecte.

Au delà de l'identification du format des images, le champ BitmapFormat est aussi utilisé comme index mémoire lors de la création de l'objet "Parser". Cet index mémoire découlant du BitmapFormat sert à choisir différents pointeurs de fonctions dans des tableaux de pointeurs prédéfinis dans la section data de QuickTime.qts. Ces pointeurs sont alors placé dans l'objet "Parser" lors de sa création en mémoire.

Le champ BitmapFormat n'étant pas vérifié lors du traitement de cette opération, il est tout à fait possible d'imposer, comme pointeurs de fonction, des valeurs se trouvant au delà des tableaux de pointeurs. Pour cela, il suffit de donner un BitmapFormat supérieur à 5 (bm32Bit).

Description technique :

Les animations Flash peuvent gérer des images sous 6 formats différents.

Voici les différentes valeurs que peuvent prendre le BitmapFormat:
enum { bm1Bit, bm2Bit, bm4Bit, bm8Bit, bm16Bit, bm32Bit };

Voici un exemple de Header d'un fichier swf travaillant avec des images en 32Bits.

swf header :

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000000   46 57 53 06 F1 32 00 00  68 00 19 28 00 05 0A 00   FWS.ñ2..h..(....
00000010   00 03 02 00 16 03 96 11  00 00 63 73 4D 6F 76 69   ......–...csMovi
00000020   65 46 50 53 00 07 03 00  00 00 1D 00 24 03 96 07   eFPS........$.–.
00000030   00 00 53 74 61 67 65 00  1C 96 14 00 00 73 63 61   ..Stage..–...sca
00000040   6C 65 4D 6F 64 65 00 00  6E 6F 53 63 61 6C 65 00   leMode..noScale.
00000050   4F 00 1F AF 00 00 00 00  00 00 00 00 00 00 00 00   O..¯............
00000060   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000070   00 00 00 3F 09 D9 30 00  00 02 00 05 A0 00 80 00   ...?.Ù0..... .€.
```

Object Header (Object stagDefineBitsLossless2):

```
00000073    093F      : tag record : (tag & 0x3F) = 0x3F  -> Long Header
                                     ID -> (tag>>6) = 0x24 -> stagDefineBitsLossless2
00000075    000030D9  : Length
00000079    0002      : character tag : 0x2 -> splaceCharacter

0000007B    05        : Bitmap format (0 to 5 -> 1Bit to 32Bits)
0000007C    00A0      : Bitmap width
0000007E    0080      : Bitmap height
```

Lors du traitement du fichier .swf, le Parser de QuickTime formate ces données dans un Objet SBitmapCore.

Récupération des champs à partir du fichier :

```
.text:66FAD2AB      mov cl, [eax+edi]      ; cl = Bitmap Format (0 to 5)
.text:66FAD2AE      inc eax
.text:66FAD2AF      mov [esp+868h+current_offset], eax
.text:66FAD2B3
       loc_66FAD2B3:
.text:66FAD2B3      movzx ebx, cl          ; ebx = Bitmap Format
.text:66FAD2B6      lea ecx, [eax+2]
.text:66FAD2B9      cmp ecx, edx
.text:66FAD2BB      lea esi, [eax+edi]
.
.
.
.text:66FAD3FE      mov edi, [esp+868h+p_SBitmapCore]
.text:66FAD402      xor ecx, ecx
.text:66FAD404      cmp ebx, bm8Bit
.text:66FAD407      setnle cl
.text:66FAD40A      push 1                 ; FLAG_allowPurge
.text:66FAD40C      lea edx, [esp+86Ch+p_SColorTable]
.text:66FAD410      dec ecx
.text:66FAD411      and ecx, edx
.text:66FAD413      push ecx               ; p_SColorTable
.text:66FAD414      push ebp               ; width
.text:66FAD415      push eax               ; height
.text:66FAD416      push ebx               ; <--- BitmapFormat
.text:66FAD417      mov ecx, edi
.text:66FAD419      call Create_SBitmapCore     ; --> Create SBitmapCore object
.
.
.
```

Création de l'objet SBitmapCore en mémoire :

```
Create_SBitmapCore proc near

    BitmapFormat= dword ptr  0Ch
    height= dword ptr  10h
    width= dword ptr  14h
    p_SColorTable= dword ptr  18h
    FLAG_allowPurge= dword ptr  1Ch

.text:66FDC6C0      push ebx
.text:66FDC6C1      push ebp
.text:66FDC6C2      xor ebp, ebp
.text:66FDC6C4      push esi
.text:66FDC6C5      mov esi, [esp+4+p_SColorTable]
.
.
.
       loc_66FDC6F2:
.text:66FDC6F2      mov ecx, [esp+8+BitmapFormat]
.text:66FDC6F6      mov edx, [esp+8+width]
.text:66FDC6FA      mov [ebx+SBitmapCore.transparent], eax
.text:66FDC6FD      mov eax, [esp+8+height]
.text:66FDC701      push eax
.text:66FDC702      push ecx
.text:66FDC703      mov [ebx+SBitmapCore.bmFormat], ecx ; <-- Save BitmapFormat in
                                                        ;     SBitmapCore object
.text:66FDC706      mov [ebx+SBitmapCore.height], eax
.text:66FDC709      mov [ebx+SBitmapCore.width], edx
.
.
.
```

L'objet Parser est initialisé avec différents pointeurs de fonctions qui serviront au traitement de l'image. Ces fonctions sont dépendantes du format de l'image et sont donc choisies en fonction de celui-ci. Le tableau de pointeur de fonction se trouve dans la section rdata.

Le tableau de pointeurs de fonction :

```
Table2_interface_pointer :
.rdata:672EA0F4      dd 0                    ; bm1Bit  function
.rdata:672EA0F8      dd 0                    ; bm2Bit  function
.rdata:672EA0FC      dd 0                    ; bm4Bit  function
.rdata:672EA100      dd offset sub_66FCE190  ; bm8Bit  function
.rdata:672EA104      dd offset sub_66FCE5E0  ; bm16Bit function
.rdata:672EA108      dd offset sub_66FCEA50  ; bm32Bit function

Table1_interface_pointer :
.rdata:672EA10C      dd offset nullsub_1     ; bm1Bit  function
.rdata:672EA110      dd offset nullsub_1     ; bm2Bit  function
.rdata:672EA114      dd offset nullsub_1     ; bm4Bit  function
.rdata:672EA118      dd offset sub_66FCFB20  ; bm8Bit  function
.rdata:672EA11C      dd offset sub_66FCFC30  ; bm16Bit function
.rdata:672EA120      dd offset sub_66FCFD40  ; bm32Bit function
 .rdata:672EA124
.rdata:672EA128   ... following Datas ...
.rdata:672EA12C
.
.
.
```

La récupération des pointeurs des fonctions se fait par l'intermédiaire de la valeur BitmapFormat. BitmapFormat sert, ici, d'index de table de pointeur :

```
exemple : eax = BitampFormat value
          mov eax, ds:Table1_function_pointer[eax*4]
```

Malheureusement, aucune vérification n'est faite sur la valeur de BitmapFomat. Avec une valeur supérieure à 5 (bm32Bit), l'index de tableau dépasse la capacité du tableau et engendre des pointeurs invalides puisque récupérés dans un mauvais emplacement mémoire.

Voici le code qui s'occupe de cette opération :

```
; esi = pointer of Parser Object
.text:66FD1C33      mov ecx, [esi+34h]            ; Get pointer of SBitmapCore object
.text:66FD1C36      mov byte ptr [esi+32h], 0
.text:66FD1C3A      mov eax, [ecx+SBitmapCore.bmFormat]        ; Get Bitmap format !!!
.text:66FD1C3D      mov eax, ds:Table1_function_pointer[eax*4] ; Get pointer of
                                                               ; function_1 according
                                                               ; to Bitmat format
                                                               ; value !!!!
.text:66FD1C44      mov [esi+80h], eax                 ; Save Pointer of function_1 in
                                                       ; Parser Object (offset 0x80)
.
.
.
      loc_66FD1C70 :
.text:66FD1C70      mov eax, dword_6743B87C[eax*4]
.text:66FD1C77      cmp eax, edi
.text:66FD1C79      jz loc_66FD1FE6
.text:66FD1C7F      mov eax, [ecx+SBitmapCore.bmFormat] ; Get Bitmap format !!!
.text:66FD1C82      mov edx, [eax+ecx*4]                ; Get pointer of function_2
                                                        ; according to format value !!!!

.text:66FD1C85      pop edi
.text:66FD1C86      pop ebp
.text:66FD1C87      pop ebx
.text:66FD1C88      mov [esi+84h], edx         ; Save Pointer of function_2 in
.                                              ; Parser Object (offset 0x84)
.
.
```

Par la suite, la fonction_1 est appelée par un call [ebp+0x80] :

```
; ebp = pointer of Parser Object
.text:66FD0707      rep movsd
; If (SBitmapCore.bmFormat > 5 (bm32Bit) ) then pointer of function_1 is invalid !
.text:66FD0709      call dword ptr [ebp+80h]   ; --> Call function_1
.text:66FD070F      mov eax, [ebp+7Ch]
.text:66FD0712      add esp, 30h
.text:66FD0715      test eax, eax
.text:66FD0717      jz short loc_66FD0728
```

Dans ce contexte là, il est possible d'imposer un autre pointeur de fonction qui sera exécutée par la suite. Mais, il est impossible d'imposer l'adresse du nouveau pointeur. Le choix de cette valeur ne peut se faire qu'à partir des données qui suivent directement les tableaux de pointeurs dans la section rdata du module QuickTime.qts. Nous nous trouvons dans une section ReadOnly qui contient donc uniquement des constantes. De plus, le champ BitmapFormat est seulement défini sur un octet. La mémoire accessible ne peut alors pas dépasser les 0x3D8 octets qui suivent les tables de pointeurs.

$(2^{8bits}$ - total_table_entry)*sizeof DWORD = (0x100-0xA)*4 = 0x3D8

Il devient alors difficile d'exploiter cette faille en utilisant des adresses en dure.
Mais cette vulnérabilité est tout à fait exploitable dans l'environnement d'un navigateur internet du type Internet Explorer avec la technique du heap-spraying.
Il suffit pour cela de choisir une valeur au champ BitmapFormat qui permette de récupérer une valeur donnant une adresse satisfaisante pour le heap-spray. Et en regardant les constantes qui suivent les tableaux de pointeurs, on peut constater qu'il n'y a que l'embarra du choix.

Merci d'avoir lu ce papier jusqu'au bout. :)

That's all folks !
12 décembre 2007
Lionel d'Hauenens - www.laboskopia.com –