VUPEN security

VUPEN Security – Binary Analysis & Exploits Service

In-Depth Analysis of Apple Safari ColorSync Profile Handling Integer Overflow Vulnerability (CVE-2010-0040)

Table of Contents

Introduction

A vulnerability exists on Apple Safari for Windows when handling and displaying images with an embedded color profile, which may lead to arbitrary code execution.

This vulnerability was discovered by VUPEN Security.

Tested Versions

The vulnerability was analyzed on Windows XP SP3 with Apple Safari for Windows version 4.0.3.

Fixed Versions

The vulnerability was fixed in Apple Safari for Windows 4.0.5.

Technical Details

In color management, an ICC profile is a set of data that characterizes a color input or output device, or a color space, according to standards promulgated by the International Color Consortium (ICC).

Profiles describe the color attributes of a particular device or viewing requirement by defining a mapping between the device source or target color space and a profile connection space (PCS), that is, an independent and normalized color space.

Embedding of ICC profiles can be done within PICT, EPS, TIFF, JFIF (JPEG), and GIF image files.

Other file formats, such as ISO 15444-2 and proprietary file formats such as PSD, specify a "proprietary" (that is, not documented in ICC specification but rather in the file format itself) embedding of ICC profiles.

When loading an image with an embedded ICC profile, the Safari web-browser parses the ICC profile according to the ICC specification.

At some point it tries to parse and read the profile description from the related and relevant tag:

```
;
; In function starting at 0x11A05A0, CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                                  Comments
011A05E5    PUSH ECX                                 ; /Arg2 => OFFSET LOCAL.0
011A05E6    PUSH EAX                                 ; |Arg1
011A05E7    CALL CMCopyProfileDescriptionString
```

In the "CMCopyProfileDescriptionString()" function, there is a call to a sub-function located at 0x1067790. Note that the first parameter is the 'desc' tag:

```
;
; In CMCopyProfileDescriptionString function, starting at 0x1067E10
; CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                 Comments
01067EC6    XOR ESI,ESI             ; |
01067EC8    MOV EDI,64657363        ; | EDI = 'desc'
01067ECD    CALL 01067790           ; \CoreGraphics.01067790
```

The 'desc' tag (i.e. the "profileDescriptionTag") is one of the tags that can be found in the Tag Table (see the "Tag Table" structure in the "Detection" chapter of this documentation).

Below is an excerpt of an ICC profile from a JPEG file. These bytes represent the 'desc' tag structure in the tag table:

```
File template.jpeg
Address     Hex dump                                ASCII
0000610B    64 65 73 63|00 00 02 7C|00 00 00 29|    desc..|...)

Tag Signature ; offset to data ; length of data
```

Inside the aforementioned function, we find the following code:

```
;
; In function starting at 0x1067790 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                                 Comments
01067387    PUSH EDX                                ; |Arg3 => pointer for return value
01067388    PUSH ESI                                ; |Arg2 => ARG.EAX, 'desc'
01067389    PUSH EDI                                ; |Arg1 => pointer to tags
;[…]
0106738F    CALL CMGetProfileElement
```

This returns the length of data for the 'desc' tag (in our example : 0x29).

Then it allocates a buffer with the retrieved length:

```
;
; In function starting at 0x1067790 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                                 Comments
010673A1    MOV EAX,DWORD PTR SS:[LOCAL.4]          ; size of element
010673A5    PUSH EAX
010673A6    PUSH 1                                  ; number of elements
010673A8    CALL DWORD PTR DS:[<&MSVCR80.calloc>]
;[…]
010673B5  MOV DWORD PTR SS:[ESP+28],EBX             ; save pointer to allocation
```

After that, the code issues once again a call to the "CMGetProfileElement()" function, but this time it gets back the whole structure pointed by the "offset" member in the tag structure:

```
;
; In function starting at 0x1067790 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                                 Comments
010673C8    PUSH EBX                                ; /Arg4: allocated buffer
010673C9    LEA ECX,[ESP+20]                        ; |
010673CD    PUSH ECX                                ; |Arg3: 'desc' string
010673CE    PUSH ESI                                ; |Arg2
010673CF    PUSH EDI                                ; |Arg1
010673D0    CALL CMGetProfileElement
```

In our tests, the returned structure looks like this:

```
CPU Dump
Address     Hex dump                                            ASCII
034461F0    64 65 73 63|00 00 00 00|00 00 00 11|54 65 73 74|    desc.......Test
03446200    20 52 47 42|20 50 72 6F|66 69 6C 65|00 00 00 00|    RGB Profile....
03446210    00 FF FF 00|00 00 00 01|00                          ..... .

'desc' type tag
ASCII count
ASCII string
Unicode Code
Unicode Count
Script code
```

This structure is of type 'desc' (warning: do not confound the 'desc' tag, with the 'desc' type) which explains how to structure the data. See the 'desc' type structure in the "Detection" chapter.

The code then gets the size of the ASCII string in the right field (highlighted in green above) and then converts this size from big to little endian:

```
;
; In function starting at 0x1097260 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                                 Comments
01097304    MOV EAX,DWORD PTR DS:[EBX+8]            ; eax = size of ASCII string
01097307    MOV EDX,DWORD PTR SS:[ESP+1C]           ; edx = size of 'desc' type
0109730B    LEA EDI,[EBX+EDX]                       ; ebx = pointer to desc type
                                                    ; edi = end of desc type
0109730E    MOV EBP,EAX                             ; change endianness (start)
01097310    MOV ECX,EAX
01097312    MOV EDX,EAX
01097314    AND EBP,00FF0000
0109731A    SHL EDX,10
0109731D    SHR ECX,10
01097320    AND EAX,0000FF00
01097325    OR EBP,ECX
01097327    OR EDX,EAX
01097329    SHR EBP,8
0109732C    SHL EDX,8
0109732F    OR EBP,EDX                          ; EBP=ASCII string length (Big E)
```

Next, the code gets a pointer to the ASCII string and adds the length of the string to this pointer:

```
;
; In function starting at 0x1097260 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                             Comments
01097331    LEA ESI,[EBX+0C]                    ; esi = points on ASCII string
;[…]
0109733E    LEA EAX,[EBP+ESI]                   ; EAX = start of string + length
01097341    CMP EAX,EDI                         ; if not > end
01097343    JB SHORT 0109734C                   ; continue if eax < edi
01097345    XOR EBP,EBP                         ; otherwise EBP = 0
01097347    JMP 01097468                        ; go to exit
```

The code compares if the end of the string is located before the end of the 'desc' type.

The code then gets the length of the Unicode string (which is in a big-endian format in the file) and changes it to little endian. The code finally checks if the value is not 0.

Each time the code manipulates data, it tries to ensure that it is not out of bounds of the 'desc' type structure.

```
;
; In function starting at 0x1097260 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                          Comments
0109736D    ADD ESI,EBP                      ; add string length
0109736F    CMP ESI,EDI                      ; end of 'desc' type ?
01097371    JNB 01097468
01097377    ADD ESI,4                        ; points on Unicode string Length
0109737A    CMP ESI,EDI                      ; check if not out of bounds
0109737C    JNB 01097468
01097382    MOV ECX,DWORD PTR DS:[ESI]       ; get value from Unicode Length
01097384    PUSH ECX                         ; /Arg1
01097385    CALL ChangeEndianness            ; \CoreGraphics.ChangeEndianness
0109738A    ADD ESI,4                        ; next DWORD
0109738D    ADD ESP,4
01097390    CMP ESI,EDI                      ; check if not outside type structure
01097392    MOV EDX,EAX                      ; EDX = Unicode length
01097394    MOV DWORD PTR SS:[ESP+18],EDX
01097398    JB SHORT 010973A7
0109739A    MOV DWORD PTR SS:[ESP+18],0
010973A2    JMP 01097468
010973A7    TEST EDX,EDX                     ; check if length is not 0
010973A9    JE 01097441
```

Then the code tries to add the length of the Unicode string (EDX * 2) to the pointer in EDI (which points to the start of a Unicode string, if any):

```
;
; In function starting at 0x1097260 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                          Comments
010973AF    LEA EBX,[EDX*2+ESI]      ; get end of string /!\ Integer overflow /!\
010973B2    CMP EBX,EDI              ; check bounds
010973B4    MOV CL,1                 ; pick _swab() by default
010973B6    JB SHORT 010973C9
```

Then it tries to check if the pointer is beyond the 'desc' structure. The problem is that the LEA instruction at 0x10973AF is prone to an integer overflow if the value in EDX is big enough.

Exploitation

Continuing in the same function, we find the following code:

```
;
; In function starting at 0x1097260 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                          Comments
010973C9    MOV AL,BYTE PTR DS:[ESI]         ; script code (0xFFFE / 0xFEFF)
010973CB    CMP AL,0FE
```

```
010973CD    JNE SHORT 010973D5
010973CF    CMP BYTE PTR DS:[ESI+1],0FF
010973D3    JE SHORT 010973DF
010973D5    CMP AL,0FF
010973D7    JNE SHORT 010973EF
010973D9    CMP BYTE PTR DS:[ESI+1],0FE
010973DD    JNE SHORT 010973EF
010973DF    CMP AL,0FF
010973E1    JNE SHORT 010973E5
010973E3    XOR CL,CL                       ; CL = 0 => choose memcpy() function
010973E5    ADD ESI,2                       ; skip script code
010973E8    SUB EDX,1                       ; decrement Unicode length value
010973EB    MOV DWORD PTR SS:[ESP+18],EDX
```

This code checks if the WORD pointed by the crafted pointer is 0xFFFE or 0xFEFF. If it is one of these values, the code sets CL to 0 (which will later pick for the memcpy() function), increments the crafted pointer by 2 and decrements the Unicode string length value picked previously at 0x01097382.

Then, we go to these lines of code (either directly from the last line of the previous snippet or from 0x010674B7, if the WORD was not 0xFFFE or 0xFEFF):

```
;
; In function starting at 0x1097260 - CoreGraphics.dll module [codebase: 0x1011000]
;
Address     Command                                     Comments
010973EF    CMP WORD PTR DS:[EDX*2+ESI-2],0  ; (crafted_value * 2) + (pointer - 2) == 0 ?
010973F5    JE SHORT 010973FE
010973F7    ADD EDX,1
010973FA    MOV DWORD PTR SS:[ESP+18],EDX
010973FE    CMP DWORD PTR SS:[ESP+40],0
01097403    JE SHORT 0109743B
01097405    MOV EAX,DWORD PTR SS:[ESP+24]    ; 0x200 (const)
01097409    CMP EDX,EAX                      ; check crafted value against 0x200
0109740B    JNB SHORT 0109740F
0109740D    MOV EAX,EDX                      ; crafted value set as copy size
0109740F    TEST CL,CL                       ; check for which function to use
01097411    MOV EDX,DWORD PTR SS:[ESP+40]
01097415    LEA EAX,[EAX+EAX-2]              ; (EAX*2) - 2
01097419    MOV WORD PTR DS:[EDX+EAX],0
0109741F    PUSH EAX                        ; copy size parameter
01097420    JE SHORT 0109742C                ; selector between _swab() and memcpy()
```

The check against 0x200 is achieved to set a value for the copy size passed to "_swab()" or "memcpy()".

If the value is below 0x200, then the value is used. If the value is above 0x200, then the copy size is set to a maximum of 0x200. It is possible to reach the "_swab()" function [which is merely a memcpy() + byte swapping] which an overly large value (e.g. 0xFFFFFFFF).

```
; In function starting at 0x1097260 - CoreGraphics.dll module [codebase: 0x1011000]
01097422    PUSH EDX
01097423    PUSH ESI
01097424    CALL DWORD PTR DS:[<&MSVCR80._swab>]
0109742A    JMP SHORT 01097433
0109742C    PUSH ESI
0109742D    PUSH EDX
0109742E    CALL <JMP.&MSVCR80.memcpy>
```

Detection

Parse the Image to find any embedded ICC profile according to the ICC specification (see [ICC-SPEC] in the "References" chapter) or the file format specification if the embedding is proprietary.

Skip the "Profile Header" - which is 128 bytes long - and parse the Tag Table which is described as follow:

Byte Offset	Field Length (bytes)	Content	Encoding
0 - 3	4	Tag count	
4 - 7	4	Tag Signature	
8 - 11	4	Offset to beginning of tag data element	uInt32Number
12 - 15	4	Size of tag data element	uInt32Number
16 - (12n+3)	12n	Signature, offset and size respectively of subsequent n tags	

- Search for the 'desc' Tag Signature (named "profileDescriptionTag" in the specification).
    - If the 'desc' tag is found, get the 'Offset' and 'Size of tag data' values.
    - Go to the defined offset (which is from the beginning of the ICC profile).

If the type tag present at the offset is of type 'desc' (named "textDescriptionType" in the specification), then the structure is :

Byte Offset	Content	Encoded as...
0..3	'desc' (64657363h) type signature	
4..7	reserved, must be set to 0	
8..11	ASCII invariant description count, including terminating null (description length)	uInt32Number
12..n-1	ASCII invariant description	7-bit ASCII
n..n+3	Unicode language code	uInt32Number
n+4..n+7	Unicode localizable description count (description length)	uInt32Number
n+8..m-1	Unicode localizable description	

You should:

- Get the DWORD at offset 8 (from the start of the "textDescriptionType structure), which is the "ASCII invariant description count". Call it "ASCIILength"
- Skip "ASCIILegnth" byte.
- Skip a DWORD [skip Unicode Language code]
- Get the DWORD which is the Unicode string length. Call it "UnicodeLength"

Try to count the length of the Unicode string until you find a '\0\0' (two NULL bytes character indicating the end of a Unicode string).

You must also count these terminating NULL characters as part of the length of the string.

If "UnicodeLength" is greater than the actual and real length of the string, the image file is malicious.

Note: the "textDescriptionType" structure is no more defined in the last available specification. You should review the older specifications available at [ICC-SPEC2].

References

VUPEN/ADV-2010-0599:
http://www.vupen.com/english/advisories/2010/0599

Apple Security Advisory:
http://support.apple.com/kb/HT4070

[ICC-SPEC] Specification ICC.1:2004-10 (Profile version 4.2.0.0):
http://www.color.org/ICC1v42_2006-05.pdf

[ICC-SPEC2] Specification ICC.1:2001-04
http://www.color.org/ICC_Minor_Revision_for_Web.pdf

[ICC-EMBED] File formats supporting ICC profiles embedding:
http://www.color.org/profile_embedding.xalter

Changelog

2009-12-03: Vulnerability discovered by VUPEN and reported to Adobe
2010-03-12: Initial release