binamuse
security research

Apple Core Graphics Framework PDF CCITT FaxDecode vulnerability

Apple Core Graphics framework fails to validate the input when parsing CCITT group 3 encoded data resulting in a heap overflow condition. A small heap memory allocation can be overflowed with controled data from the input resulting in arbitrary code execution in the context of Mobile Safari.

Aug 2014

Felipe Andres Manzano
feliam@binamuse.com

Contents

1 Target summary

Title: Apple Core Graphics Framework PDF FaxDecode memory corruption

Product: iOS mobile operating system.

Version: 6.1.x , 7.0.x, 7.1.1, 7.1.2

Product Homepage: apple.com

2 Vulnerability brief information

Vulnerability Class	Memory Corruption
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)
Reliability Rating	Completely (100%)

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 8: iPhone5 iOS-7.1.2
Attack Vector	Client-Side File Format
Exploitation Impact	Code Execution
Exploitation Context	mobile
Exploit Features	ASLR/DEP/Code signing bypass (needs memory layout leak)

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	**U**navailable	There is either no solution available or it is impossible to apply
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

Apple Core Graphics framework fails to validate the input when parsing CCITT group 3 encoded data. A small heap memory allocation can be overflowed with controled data from the input enabling arbitrary code execution in the context of Mobile Safari (A memory layout information leak is needed).

The Core Graphics framework is a C-based API that is based on the Quartz[1] advanced drawing engine. It provides low-level, lightweight 2D rendering. This is used in a wide range of applications to handle path-based drawing, transformations, color management, offscreen rendering, patterns, gradients and shadings, image data management, image creation, masking, and PDF document creation, display, and parsing.

`CoreGraphics` library implements the funtionality to load and save several graphic formats such as PDFs and it is used by most common applications that deal with images, including Safari, Preview, Skype, etc. The framework also implements the `x_alloc` heap, a set of low level procedures to manage an internal heap memory structure used when processing images of different types and PDF files

The functions `x_calloc` and `x_free` are used very frequently to allocate and deallocate memory fast. The `x_calloc` function behaves as a normal calloc except it allocates a bit extra memory for metadata. It actually allocates an extra `2*sizeof(void*)` bytes and pad the resulting size to the next 16 byte border. When the chunk is in use (allocated) this extra space is used to hold the size of the chunk, thus making it super fast to

free. On te other hand when the chunk is free the metadata space is used to form a free-list linked list, the first metadata value in a free chunk points to the next free chunk.

This is its pseudocode. Think `x_mem_alloc0_size` as a plain malloc.

```
void* __cdecl x_calloc(size_t nmemb, size_t size)
{
  void * buffer = x_mem_alloc0_size((nmemb * size + 31) & -16);
  *(size_t *)buffer = (nmemb * size + 31) & -16;
  return buffer + 16;
}
```

This function is prone to integer overflow because it doesn't check for `(nmemb * size)` to be less than `MAXUINT-16`. Then if a programmer tries to allocate a size in the range [-16,-1] `x_alloc` will alloc 0 or 16 bytes (instead of the inmense value requiered) without triggering any exception.

4.1 CCITTFaxDecode group 3 x_calloc bug.

A `x_calloc` bug is triggered when decoding CCITT group 3 compressed data. The `CCITTFaxDecode` pdf filter decodes image data that has been encoded using either Group 3 or Group 4 CCITT facsimile (fax) encoding. CCITT encoding is designed to achieve efficient compression of monochrome (1 bit per pixel) image data at relatively low resolutions, and so is useful only for bitmap image data, not for colour images, grayscale images, or general data.

Any PDF Stream can be compressed using a different set of filters as described in PDFSPEC::7.3.8::Stream Objects [2]. Under certain conditions the implelentation of the `CCITTFaxDecode` filter fails to allocate the correct amount of memory for its internal state. It allocates an overly small array that can be overflowded with controlled data. The decoding parmeters of any pdf filter are controled via the `/decodeparams` field of the stream dictionary. The `CCITTFaxDecode` filter parses its decoding parameters at the function `pdf_source_create_ccitt_fax_filter` (see appendix for pseudocode). After the decode parameters are parsed an validated the `CCITTFaxDecode` internal state is allocated at functions `pdf_FaxDecodeStateAlloc`. Setting a `/Columns` value of `0x3fffffff-4` enables the attacker to reach `x_calloc` with a size in the `[16,-1]` range (without trigering any exceptions). Thus, `pdf_FaxDecodeStateAlloc` will reseve a small amount of memory for holding a potentially big array of encoded CCITT group3 1D runlenghts values, as explained in [3]. PDF CCITTFaxDecode filter can handle a number of CCITT encodings, `K=0` selects the vulnerable code (G31D). An example of `/DecodeParams` dictionary nedded to reach the vulnerable code follows:

```
/DecodeParms << /Rows 0
                /EndOfBlock true
                /K 0
                /BlackIs1 true
                /EncodedByteAlign false
                /Columns 1073741819
                /EndOfLine false
             >>
```

After `pdf_FaxDecodeStateAlloc` silently fails to reserve enough space to decode a line of potentially `0x3fffffff-4` runlengths codes the actual decoding is done at the function `pdf_FaxDecode`. This function decodes the stream of CCITT G31D encoded runlengths found in the input. CCITT group 3 works in a line by line pase, at each line it will iterativelly consume white an black runlengths. Each decoded runlength is appended to in the internaly allocated array of 16 bit words. The decompression algorithm is best described in [3]. As the attaker controls the raw input it may craft a list of CCITT runlength codes to overwrite the overlly short 16bit word internall array with a controlled amount of controlled data.

Function lookup table based on firmware `iPhone5,1-7.1.2`.

```
0x2D57C824  pdf_FaxDecodeStateAlloc
0x2D57EF64  pdf_source_create_ccitt_fax_filter
0x2D57C8D8  pdf_FaxDecode
```

Assume `dyld_shared_cache` base addres is 0x2c00000.

5 Exploitation

A 100% reliable PoC exploit is attached. This exploit needs a companion information leak vulnerability to bypass ASLR, DEP and Code signing iOS exploit mitigations. The exploit is presented as a cgi script that expects to get the `dyld_shared_cache`, shellcode address and iOS version as `GET` parameters. It executes arbitrary code in the context of Safari Mobile. It was tested on a iPhone 5 version 7.1.2.

6 References

1 http://en.wikipedia.org/wiki/Quartz_(graphics_layer)

2 http://www.images.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf

3 http://tools.ietf.org/html/rfc804

A Listings

```
int __cdecl pdf_source_create_ccitt_fax_filter(void *arg0, int a1)
{
  signed int v2;
  ccitt *ccitt;
  int result;
  int K;
  bool flags_a;
  signed int ccitt_group_;
  int flags_b;
  int flags_c;
  int flags;
  char blackIsZERO;
  FaxDecodeState *state;
  int ccitt_group;
  char value_bool;
  int value_int;

  v2 = 3793108;
  ccitt = x_calloc(1, 32);
  result = 0;
  if ( ccitt )
  {
    ccitt->K = 0;
    ccitt->EOL = 0;
    ccitt->EncodedByteAlign = 0;
    ccitt->Columns = 1728;
    ccitt->Rows = 0;
    ccitt->EndOfBlock = 1;
    ccitt->BlackIs1 = 0;
    ccitt->DamagedRowsBeforeError = 0;
    if ( a1 )
    {
```

```
  if ( CGPDFDictionaryGetInteger(a1, "K", &value_int) )
    ccitt->K = value_int;
  if ( CGPDFDictionaryGetBoolean(a1, "EndOfLine", &value_bool) )
    ccitt->EOL = value_bool != 0;
  if ( CGPDFDictionaryGetBoolean(a1, "EncodedByteAlign",
     &value_bool) )
    ccitt->EncodedByteAlign = value_bool != 0;
  if ( CGPDFDictionaryGetInteger(a1, "Columns", &value_int) )
  {
    if ( value_int < 0 )
      pdf_error("/%s is outside the range of allowed values.",
        "Columns");
    else
      ccitt->Columns = value_int;
  }
  v2 = 3793108;
  if ( CGPDFDictionaryGetInteger(a1, "Rows", &value_int) )
  {
    if ( value_int < 0 )
      pdf_error("/%s is outside the range of allowed values.",
        "Rows");
    else
      ccitt->Rows = value_int;
  }
  if ( CGPDFDictionaryGetBoolean(a1, "EndOfBlock", &value_bool) )
    ccitt->EndOfBlock = value_bool != 0;
  if ( CGPDFDictionaryGetBoolean(a1, "BlackIs1", &value_bool) )
    ccitt->BlackIs1 = value_bool != 0;
  if ( CGPDFDictionaryGetInteger(a1, "DamagedRowsBeforeError",
     &value_int) )
  {
    if ( value_int < 0 )
      pdf_error("/%s is outside the range of allowed values.",
        "DamagedRowsBeforeError");
    else
      ccitt->DamagedRowsBeforeError = value_int;
  }
}
K = ccitt->K;
if ( K < 0 )
{
  ccitt_group_ = 4;
  flags_a = 0;
}
else
{
  flags_a = K > 0;
  ccitt_group_ = 3;
}
ccitt_group = ccitt_group_;
flags_b = flags_a + 2;
if ( !ccitt->EOL )
  flags_b = flags_a;
```

```
    flags_c = flags_b | 4;
    if ( !ccitt->EncodedByteAlign )
      flags_c = flags_b;
    if ( ccitt->EndOfBlock )
    {
      ccitt->Rows = 0;
      flags_c |= 8u;
    }
    flags = flags_c | 0x10;
    blackIsZERO = ccitt->BlackIs1 == 0;
    LOBYTE(ccitt->flags) = 0;
    if ( !blackIsZERO )
      flags = flags_c;
    ccitt->source = CGPDFSourceRetain(arg0);
    state = pdf_FaxDecodeStateAlloc(ccitt->Columns, ccitt_group, flags);
    ccitt->fax_decode_state = state;
    if ( !state
      || (state->getc = (ccitt_fax_filter_getc + v2 - 3793108),
          ccitt->fax_decode_state->stream = arg0,
          (result = CGPDFSourceCreateFilter(
                      ccitt,
                      (ccitt->Columns + (((ccitt->Columns + 7) >> 31) >>
                         29) + 7) >> 3,
                      &pdf_source_create_ccitt_fax_filter_callbacks + v2
                         - 3793108)) == 0) )
    {
      ccitt_fax_filter_finalize(ccitt);
      result = 0;
    }
  }
  return result;
}

FaxDecodeState *__cdecl pdf_FaxDecodeStateAlloc(int columns, int group,
   int flags)
{
  FaxDecodeState *retval;
  int size;
  FaxDecodeState *state;
  int N;
  _WORD *buf;
  unsigned int last;
  int ptr;
  char is4;

  retval = 0;
  size = columns;
  if ( columns <= 0x3FFFFFFF )
  {
    state = x_calloc(1, 0x38);
    if ( state )
    {
      state->group = group;
```

```
      state->flags = flags;
      state->columns = columns;
      state->columns_rnd = (columns + (((columns + 7) >> 31) >> 29) + 7)
         >> 3;
      is4 = (group == 4) | flags & 1;
      if ( is4 )
        size = (2 * columns + 62) & 0xFFFFFFC0;
      N = 2 * size + 3;
      if ( N >= 0 )
      {
        buf = x_malloc(2 * N);
        state->line_buffer = buf;
        if ( buf )
        {
          state->line_buffer2 = buf;
          last = size >> 1;
          if ( is4 & 1 )
            size = size >> 1;
          state->N = size;
          if ( is4 & 1 )
          {
            ptr = &buf[last];
            state->tail = ptr;
          }
          else
          {
            state->tail = 0;
            ptr = 0;
          }
          state->code_bitcount = 0;
          state->bitaccu = 0;
          state->field_18 = 0;
          retval = state;
          if ( ptr )
          {
            *ptr = LOWORD(state->columns);
            *(ptr + 2) = 0;
            retval = state;
          }
        }
      }
    }
  }
  return retval;
}
```

```
00000000 ccitt           struc ; (sizeof=0x20)
00000000 source          dd ?
00000004 flags           dd ?
00000008 K               dd ?
0000000C Rows            dd ?
00000010 Columns         dd ?
00000014 EOL             db ?
00000015 EncodedByteAlign db ?
```

```
00000016 EndOfBlock      db ?
00000017 BlackIs1        db ?
00000018 DamagedRowsBeforeError dd ?
0000001C fax_decode_state dd ?                   ; offset
00000020 ccitt           ends
00000020
00000000 ; ---------------------------------------------------------------------------
00000000
00000000 FaxDecodeState  struc ; (sizeof=0x38)
00000000 group           dd ?
00000004 flags           dd ?                    ; 0x10 BlackIsZero0
00000004                                         ; 0x08 EndOfBlock
00000004                                         ; 0x04 EncodedByteAlign
00000004                                         ; 0x02 EOL
00000004                                         ; 0x01 K>0
00000008 columns_rnd     dd ?
0000000C columns         dd ?
00000010 bitaccu         dd ?
00000014 code_bitcount   dd ?
00000018 field_18        dd ?
0000001C line_buffer     dd ?                    ; offset
00000020 tail            dd ?                    ; offset
00000024 line_buffer2    dd ?                    ; offset
00000028 N               dd ?
0000002C field_2C        dd ?
00000030 getc            dd ?                    ; offset
00000034 stream          dd ?                    ; offset
00000038 FaxDecodeState  ends
00000038
```