In-Depth Analysis of Apple Safari WebKit CSS "format" Memory Corruption Vulnerability (CVE-2010-0046)

Table of Contents

Introduction

A vulnerability exists in Apple Safari when processing certain CSS arguments, which could be exploited by attackers to compromise a vulnerable system.

Tested Versions

The vulnerability was analyzed on Windows XP SP3 with Apple Safari version 4.0.4.

Fixed Versions

The vulnerability is fixed in Apple Safari version 4.0.5.

Technical Details

A memory corruption vulnerability exists in the WebKit module embedded with Apple Safari. The flaw is triggered when handling unexpected arguments passed to the CSS "format()" function.

The CSS "*format()*" function is used to specify the format of a font. It is part of the "*@font-face*" keyword and can be used as follows:

```
@font-face {
   font-family: "MyFont";                                   // Name of the custom font
   src: url(http://domain.tld/font.ttf) format(TrueType);   // URL and format of the
                                                            // custom font
}
```

Such code is parsed via the "*CSSParser::parseFontFaceSrc()*" function in "*WebCore/css/CSSParser.cpp*".

First, the function looks for the "*format(*" keyword and if it is found, the supplied argument is used to set the font format:

```
3158            CSSParserValue* a = args->current();
3159            uriValue.clear();
3160            parsedValue = CSSFontFaceSrcValue::createLocal(a->string);

             // If function's name is format(.
3161         } else if (equalIgnoringCase(val->function->name, "format(") &&
3162                                     allowFormat && uriValue) {
3162            expectComma = true;
3163            allowFormat = false;

                // Sets the format using the given argument.
3164            uriValue->setFormat(args->current()->string);
3165            uriValue.clear();
3166            m_valueList->next();
3167            continue;
3168         }
3169      }
```

The same code can be seen in assembly :

```
.text:016AEBE0 loc_16AEBE0:
.text:016AEBE0
.text:016AEBE0    push    offset aFormat_0 ; "format("
.text:016AEBE5    call    equalIgnoringCase              // Checks for "format("
.text:016AEBEA    add     esp, 4
.text:016AEBED    test    al, al                          // Checks if the keyword is
                                                          // found
.text:016AEBEF    jz      loc_16AECC7                     // Jcc is taken when the
                                                          // keyword is not found
.text:016AEBF5    cmp     [esp+68h+var_52], 0             // Checks for allowFormat
.text:016AEBFA    jz      loc_16AECC7                     // Jcc is taken if False
.text:016AEC00    test    ebx, ebx                        // Checks for uriValue
.text:016AEC02    jz      loc_16AECC7                     // Jcc is taken if False
.text:016AEC08    mov     eax, [edi+70h]
.text:016AEC0B    cmp     eax, ebp
.text:016AEC0D    mov     [esp+68h+var_53], 1             // Set expectComma to True
.text:016AEC12    mov     [esp+68h+var_52], 0             // Set allowFormat to False
.text:016AEC17    jnb     short loc_16AEC24
.text:016AEC19    mov     ecx, [edi+8]
.text:016AEC1C    lea     eax, [eax+eax*2]                // Computes args->current()
.text:016AEC1F    lea     eax, [ecx+eax*8]                // Computes args->current()
.text:016AEC22    jmp     short loc_16AEC26
[...]
.text:016AEC26 loc_16AEC26:
.text:016AEC26    mov     edx, [eax+0Ch]                  // Compute
                                                          // args->current()->string
.text:016AEC29    mov     eax, [eax+8]
.text:016AEC2C    push    edx
.text:016AEC2D    lea     edi, [esp+6Ch+var_3C]
.text:016AEC31    call    setFormat                       // Set the format using the
                                                          // given argument
.text:016AEC36    mov     eax, edi
.text:016AEC38    mov     ecx, ebx
```

As it can be seen around "*args->current()->string*", any argument supplied via the CSS "*format()*" function will be interpreted as a "*CSS_STRING*" object.

If a non-string value is used with the "*format()*" function, a memory corruption occurs. This can be achieved for example by supplying one the following CSS functions as an argument to "*format()*":

- *attr()*
- *counter()*
- *calc()*

When a legitimate "*CSS_STRING*" object is given as argument to "*format()*", the following layout can be observed in memory.

Offset	String pointer	String size	Memory	Memory
7FE9E810	***80 E7 F5 7F***	***20 00 00 00***	*01 00 10 00*	*01 00 00 0*
7FF5E780	→ ***UTF8 "TrueType)}"***			

When any of the above functions is given as argument to "*format()*", the following layout can be observed in memory.

Offset	String pointer	Memory	Memory	Memory
7FE9E888	***40 E7 F5 7F***	*A0 DB 12 00*	*01 00 10 00*	*00 00 1E 44*
7FF5E740	→ ***D0 6C F6 7F***			
7FF66CD0	→ ***UTF8 "counter(a))}"***			

In such case, the DWORD following the pointer to string is used as the string size: 0x0012dba0 (*A0 DB 12 00*).

Among many operations, the "*setFormat()*" function allocates memory in order to store a copy of the argument passed to "*format()*". The allocation is performed this way:

```
.text:01417B90 loc_1417B90:
.text:01417B90    lea     eax, [edi+edi+18h]        // String size * 2 + 0x18.
.text:01417B94    push    eax
.text:01417B95    call    WTF::fastMalloc(uint)     // Call to malloc().
.text:01417B9A    add     esp, 4
.text:01417B9D    test    eax, eax
.text:01417B9F    lea     ecx, [eax+18h]
```

By providing specially crafted data, various behaviors can occurs, potentially leading to code execution.

Exploitation

If the size of the provided string is too large, the allocation above fails and an access violation exception at *0xBBADBEEF* is thrown.

```
(adc.9ec): Access violation - code c0000005 (!!! second chance !!!)
eax=00000000 ebx=00000000 ecx=00ea7ea0 edx=00000005 esi=000f9223 edi=7ff90240 eip=00e0b1b1
esp=0012d56c ebp=00000000 iopl=0
nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000  efl=00000246

JavaScriptCore!JSC::CString::~CString+0x3f01:
00e0b1b1 c705efbeadbb00000000 mov dword ptr ds:[0BBADBEEFh],0 ds:0023:bbadbeef=????????
```

However, if the allocation succeeds, the argument to "*format()*" is copied to the newly allocated buffer. The string size is used as the size argument of "*memcpy()*". If the value is too large, the source may reach the end of the page and an access violation exception is thrown.

```
(290.164): Access violation - code c0000005 (!!! second chance !!!)
eax=801b3ef0 ebx=0012d88c ecx=00086fbc edx=00000000 esi=7ff98000 edi=7fc4f868 eip=7814500a
esp=0012d844 ebp=0012d84c iopl=0
nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000         efl=00000212

// ESI has reached the end of the page.
MSVCR80!memcpy+0x5a:
7814500a f3a5          rep movs dword ptr es:[edi],dword ptr [esi]
```

If both of the allocation and "*memcpy()*" succeed, the argument is checked against known supported format.

```
.text:0178AD4D loc_178AD4D:
.text:0178AD4D    push    offset aTruetype ; "truetype"
.text:0178AD52    call    strcasecmp_like                  // Look for truetype.
.text:0178AD57    add     esp, 4
.text:0178AD5A    test    al, al
.text:0178AD5C    jnz     short loc_178AD91                // Return 1 if found.
.text:0178AD5E    mov     eax, [esi+0Ch]
.text:0178AD61    push    offset aOpentype ; "opentype"
.text:0178AD66    call    strcasecmp_like                  // Look for opentype.
.text:0178AD6B    add     esp, 4
```

```
.text:0178AD6E    test    al, al
.text:0178AD70    jnz     short loc_178AD91                  // Return 1 if found.
.text:0178AD72    mov      esi, [esi+0Ch]
.text:0178AD75    push    offset aSvg      ; "svg"
.text:0178AD7A    mov      eax, esi
.text:0178AD7C    call    strcasecmp_like                    // Look for svg.
.text:0178AD81    add      esp, 4
.text:0178AD84    test    al, al
.text:0178AD86    jnz     short loc_178AD91                  // Return 1 if found.
.text:0178AD88    xor      eax, eax                          // Else, return 0.
.text:0178AD8A    pop      edi
.text:0178AD8B    pop      esi
.text:0178AD8C    pop      ebx
.text:0178AD8D    add      esp, 8
.text:0178AD90    retn

.text:0178AD91 loc_178AD91:                                  // Return 1 if found.
.text:0178AD91    pop      edi
.text:0178AD92    pop      esi
.text:0178AD93    mov      eax, 1
.text:0178AD98    pop      ebx
.text:0178AD99    add      esp, 8
.text:0178AD9C    retn
.text:0178AD9C sub_178ACC0      endp
```

If a non "*CSS_STRING*" object is found, the memory layout is a pointer to a pointer, the "*strcasencmp_like()*" cannot be fooled, so the function returns 0. If so, the previously allocated buffer is not used anymore.

In order to trigger the vulnerability and potentially execute arbitrary code, an attacker may need to prepare the heap in order to set a false string size which should large enough to fail either on the allocation or in the "*memcpy()*".

However, due to the nature of the vulnerability, reliable exploitation for code execution seems unlikely.

Detection

In order to detect exploits targeting this vulnerability, inspect web pages or CSS data including a "*format()*" function.

If the argument passed to "*format()*" is not a string (i.e. it includes non-alphanumeric characters), the web page is potentially malicious.

References

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

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

Changelog

2010-03-17: Initial release