Cairo-1.15.4 Denial-of-Service Attack due to Logical Problem in Program

Overview

I have found a vulnerability of Cairo-1.15.4 when fuzzing HarfBuzz with AFL. Cairo is a 2d graphics library, and HarBuzz is an OpenType text shaping engine which contains a tool named *hb-view* to give a graphical view of text by Cairo, using a font provided by user. The crash happens when calling memcpy() with src=0, i.e. null pointer deference. src=0 is caused by failing to allocate again after memory is freed, and then the failing information is ignored when transmitted to upper function, which leads to the occurrence of calling memcpy(src=0). Owing to logical problem in program, this vulnerability can cause a Denial-of-Service attack with a crafted font file.

Software and Environments

Software:

HarfBuzz 1.4.5 (https://www.freedesktop.org/wiki/Software/HarfBuzz/)
Download by: git clone https://github.com/behdad/harfbuzz.git
Dependencies: FreeType (after harfbuzz is installed, reinstall FreeType), Cairo

FreeType 2.7.1 (https://www.freetype.org/index.html)
Download by: http://downloads.sourceforge.net/freetype/freetype-2.7.1.tar.bz2
Dependencies: HarfBuzz (first install without it; after it is installed, reinstall FreeType)

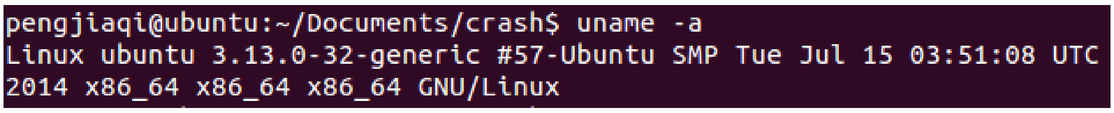
Cairo 1.15.4 (https://cairographics.org/)
Download by: git clone git://git.cairographics.org/git/cairo
Dependencies: Pixman, libpng, Xorg libraries

Pixman 0.34.0
Download by: git clone git://anongit.freedesktop.org/git/pixman

libpng: sudo apt-get install libpng12-dev

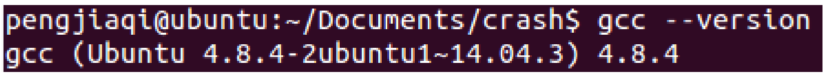
Xorg libraries: sudo apt-get install libx11-dev, libxrender-dev, libxft-dev

Operating System: Ubuntu 14.04 x86_64 Desktop



```
pengjiaqi@ubuntu:~/Documents/crash$ uname -a
Linux ubuntu 3.13.0-32-generic #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC
2014 x86_64 x86_64 x86_64 GNU/Linux
```

Compiler: gcc



```
pengjiaqi@ubuntu:~/Documents/crash$ gcc --version
gcc (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
```

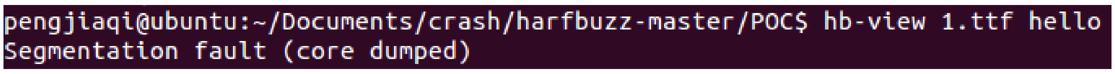
Reproducing

The crash can be reproduced in the following way:

```
cd /* path of freetype source code */
./configure --disable-shared; make; sudo make install

cd /* path of pixman source code */
./configure --disable-shared; make; sudo make install

cd /* path of cairo source code */
./autogen.sh --disable-shared; make; sudo make install

cd /* path of harfbuzz source code */
./autogen.sh --disable-shared; make; sudo make install

// reinstall FreeType
cd /* path of freetype source code */
./configure --disable-shared; make; sudo make install

cd /* path of harfbuzz source code */
hb-view or ./util/hb-view    /*path of PoC font file*/    /*any text*/
```

Exception

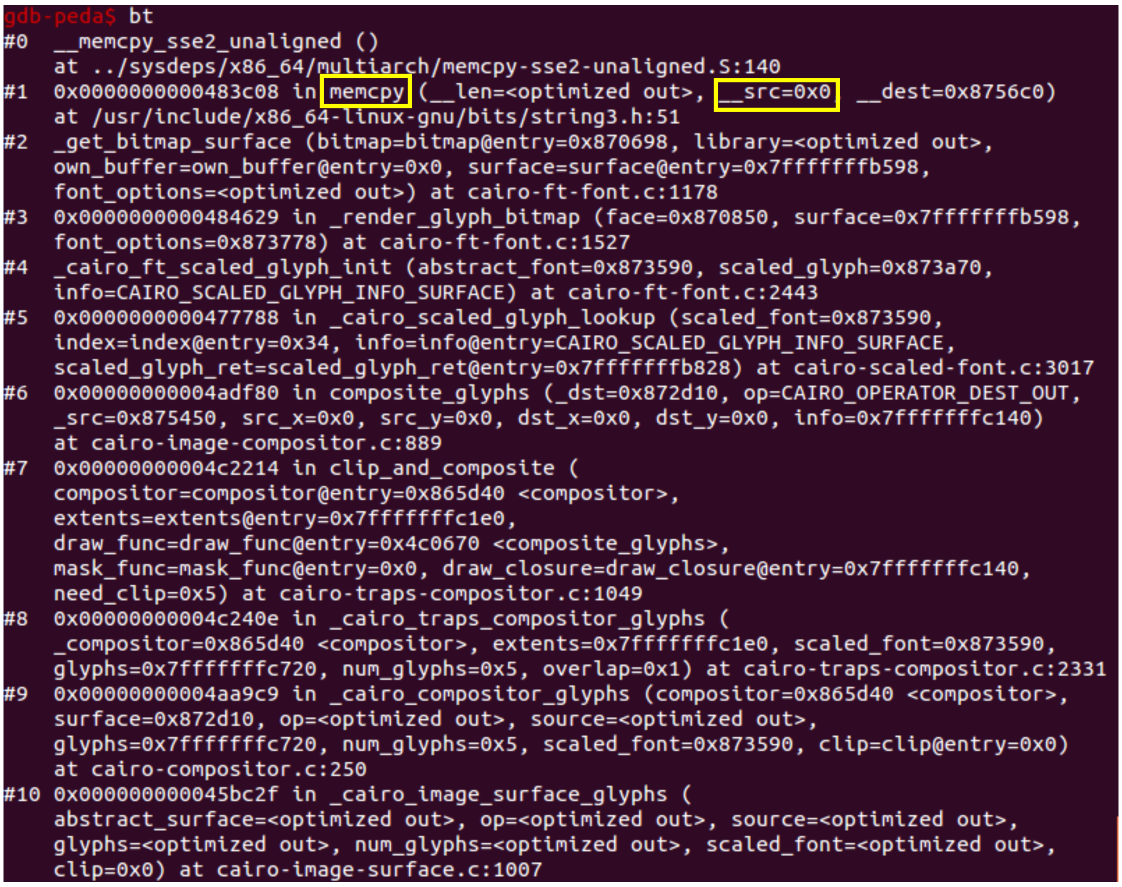
Run 'hb-view' with PoC (i.e. 1.ttf), throwing exception of "Segmentation fault " :



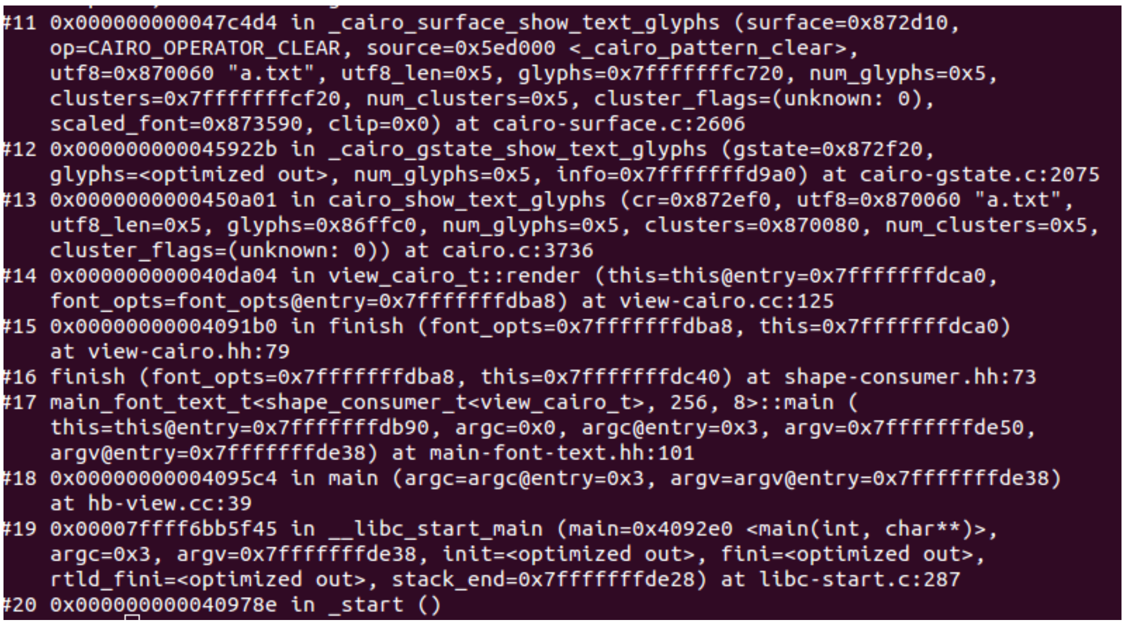
```
pengjiaqi@ubuntu:~/Documents/crash/harfbuzz-master/POC$ hb-view 1.ttf hello
Segmentation fault (core dumped)
```

Analysis

Here is the crash stack:



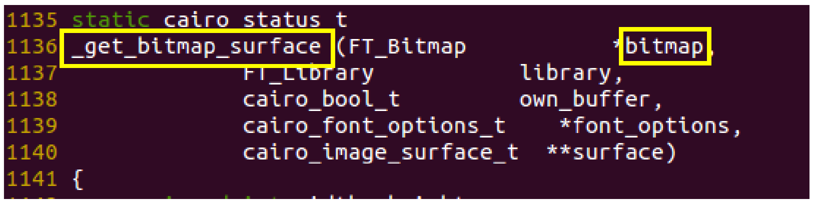
```
gdb-peda$ bt
#0  __memcpy_sse2_unaligned ()
    at ../sysdeps/x86_64/multiarch/memcpy-sse2-unaligned.S:140
#1  0x0000000000483c08 in memcpy (__len=<optimized out>, __src=0x0, __dest=0x8756c0)
    at /usr/include/x86_64-linux-gnu/bits/string3.h:51
#2  _get_bitmap_surface (bitmap=bitmap@entry=0x870698, library=<optimized out>,
    own_buffer=own_buffer@entry=0x0, surface=surface@entry=0x7fffffffb598,
    font_options=<optimized out>) at cairo-ft-font.c:1178
#3  0x0000000000484629 in _render_glyph_bitmap (face=0x870850, surface=0x7fffffffb598,
    font_options=0x873778) at cairo-ft-font.c:1527
#4  _cairo_ft_scaled_glyph_init (abstract_font=0x873590, scaled_glyph=0x873a70,
    info=CAIRO_SCALED_GLYPH_INFO_SURFACE) at cairo-ft-font.c:2443
#5  0x0000000000477788 in _cairo_scaled_glyph_lookup (scaled_font=0x873590,
    index=index@entry=0x34, info=info@entry=CAIRO_SCALED_GLYPH_INFO_SURFACE,
    scaled_glyph_ret=scaled_glyph_ret@entry=0x7fffffffb828) at cairo-scaled-font.c:3017
#6  0x00000000004adf80 in composite_glyphs (_dst=0x872d10, op=CAIRO_OPERATOR_DEST_OUT,
    _src=0x875450, src_x=0x0, src_y=0x0, dst_x=0x0, dst_y=0x0, info=0x7fffffffc140)
    at cairo-image-compositor.c:889
#7  0x00000000004c2214 in clip_and_composite (
    compositor=compositor@entry=0x865d40 <compositor>,
    extents=extents@entry=0x7fffffffc1e0,
    draw_func=draw_func@entry=0x4c0670 <composite_glyphs>,
    mask_func=mask_func@entry=0x0, draw_closure=draw_closure@entry=0x7fffffffc140,
    need_clip=0x5) at cairo-traps-compositor.c:1049
#8  0x00000000004c240e in _cairo_traps_compositor_glyphs (
    _compositor=0x865d40 <compositor>, extents=0x7fffffffc1e0, scaled_font=0x873590,
    glyphs=0x7fffffffc720, num_glyphs=0x5, overlap=0x1) at cairo-traps-compositor.c:2331
#9  0x00000000004aa9c9 in _cairo_compositor_glyphs (compositor=0x865d40 <compositor>,
    surface=0x872d10, op=<optimized out>, source=<optimized out>,
    glyphs=0x7fffffffc720, num_glyphs=0x5, scaled_font=0x873590, clip=clip@entry=0x0)
    at cairo-compositor.c:250
#10 0x000000000045bc2f in _cairo_image_surface_glyphs (
    abstract_surface=<optimized out>, op=<optimized out>, source=<optimized out>,
    glyphs=<optimized out>, num_glyphs=<optimized out>, scaled_font=<optimized out>,
    clip=0x0) at cairo-image-surface.c:1007
```



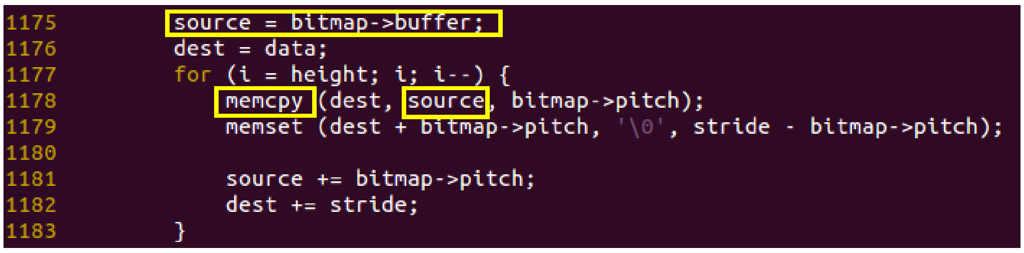
```
#11 0x000000000047c4d4 in _cairo_surface_show_text_glyphs (surface=0x872d10,
    op=CAIRO_OPERATOR_CLEAR, source=0x5ed000 <_cairo_pattern_clear>,
    utf8=0x870060 "a.txt", utf8_len=0x5, glyphs=0x7fffffffc720, num_glyphs=0x5,
    clusters=0x7fffffffcf20, num_clusters=0x5, cluster_flags=(unknown: 0),
    scaled_font=0x873590, clip=0x0) at cairo-surface.c:2606
#12 0x000000000045922b in _cairo_gstate_show_text_glyphs (gstate=0x872f20,
    glyphs=<optimized out>, num_glyphs=0x5, info=0x7fffffffd9a0) at cairo-gstate.c:2075
#13 0x0000000000450a01 in cairo_show_text_glyphs (cr=0x872ef0, utf8=0x870060 "a.txt",
    utf8_len=0x5, glyphs=0x86ffc0, num_glyphs=0x5, clusters=0x870080, num_clusters=0x5,
    cluster_flags=(unknown: 0)) at cairo.c:3736
#14 0x000000000040da04 in view_cairo_t::render (this=this@entry=0x7fffffffdca0,
    font_opts=font_opts@entry=0x7fffffffdba8) at view-cairo.cc:125
#15 0x00000000004091b0 in finish (font_opts=0x7fffffffdba8, this=0x7fffffffdca0)
    at view-cairo.hh:79
#16 finish (font_opts=0x7fffffffdba8, this=0x7fffffffdc40) at shape-consumer.hh:73
#17 main_font_text_t<shape_consumer_t<view_cairo_t>, 256, 8>::main (
    this=this@entry=0x7fffffffdb90, argc=0x0, argc@entry=0x3, argv=0x7fffffffde50,
    argv@entry=0x7fffffffde38) at main-font-text.hh:101
#18 0x00000000004095c4 in main (argc=argc@entry=0x3, argv=argv@entry=0x7fffffffde38)
    at hb-view.cc:39
#19 0x00007ffff6bb5f45 in __libc_start_main (main=0x4092e0 <main(int, char**)>,
    argc=0x3, argv=0x7fffffffde38, init=<optimized out>, fini=<optimized out>,
    rtld_fini=<optimized out>, stack_end=0x7fffffffde28) at libc-start.c:287
#20 0x000000000040978e in _start ()
```

From the #1 function of call stack, we can know the crash is caused by **null pointer deference**, where the src of memcpy() is 0.

In #2:



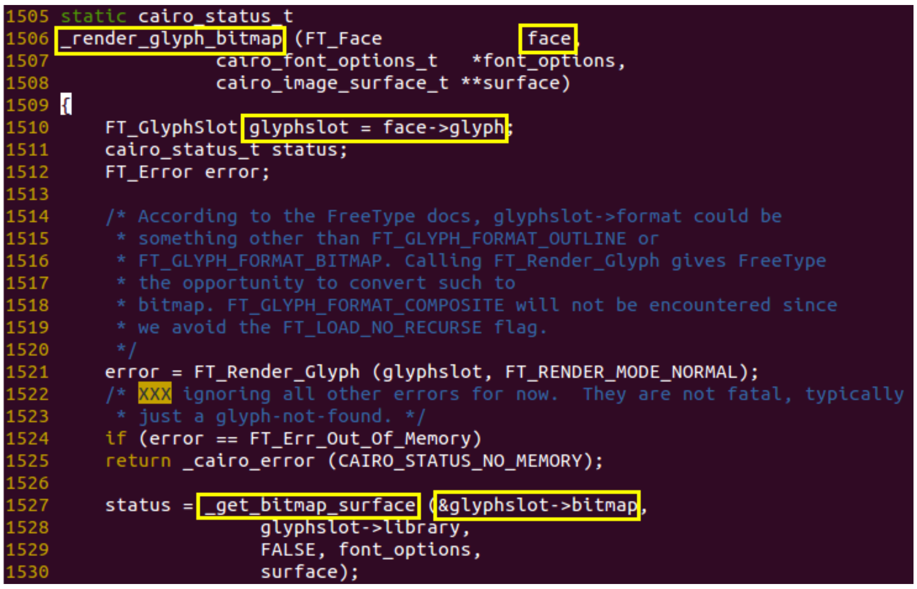
```
static cairo_status_t
_get_bitmap_surface (FT_Bitmap             *bitmap,
                     FT_Library             library,
                     cairo_bool_t           own_buffer,
                     cairo_font_options_t     *font_options,
                     cairo_image_surface_t  **surface)
{
```



```
        source = bitmap->buffer;
        dest = data;
        for (i = height; i--) {
            memcpy (dest, source, bitmap->pitch);
            memset (dest + bitmap->pitch, '\0', stride - bitmap->pitch);

            source += bitmap->pitch;
            dest += stride;
        }
```

src = source = **bitmap->buffer**

In #3:



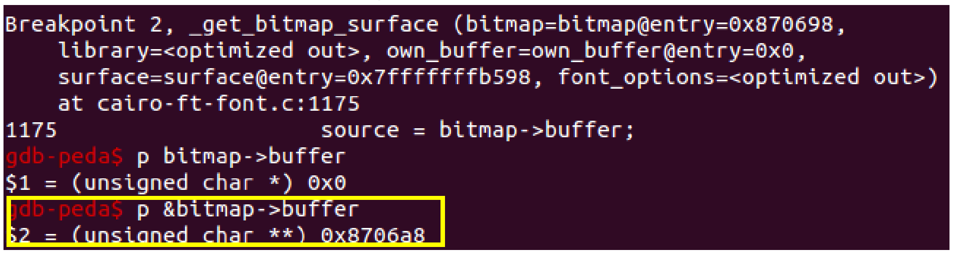
```
static cairo_status_t
_render_glyph_bitmap (FT_Face              face,
                      cairo_font_options_t   *font_options,
                      cairo_image_surface_t **surface)
{
    FT_GlyphSlot glyphslot = face->glyph;
    cairo_status_t status;
    FT_Error error;

    /* According to the FreeType docs, glyphslot->format could be
     * something other than FT_GLYPH_FORMAT_OUTLINE or
     * FT_GLYPH_FORMAT_BITMAP. Calling FT_Render_Glyph gives FreeType
     * the opportunity to convert such to
     * bitmap. FT_GLYPH_FORMAT_COMPOSITE will not be encountered since
     * we avoid the FT_LOAD_NO_RECURSE flag.
     */
    error = FT_Render_Glyph (glyphslot, FT_RENDER_MODE_NORMAL);
    /* XXX ignoring all other errors for now.  They are not fatal, typically
     * just a glyph-not-found. */
    if (error == FT_Err_Out_Of_Memory)
    return _cairo_error (CAIRO_STATUS_NO_MEMORY);

    status = _get_bitmap_surface (&glyphslot->bitmap,
                          glyphslot->library,
                          FALSE, font_options,
                          surface);
```

src = bitmap->buffer = **face->glyph->bitmap.buffer**

(Later, we will use ***buffer*** to indicate src)

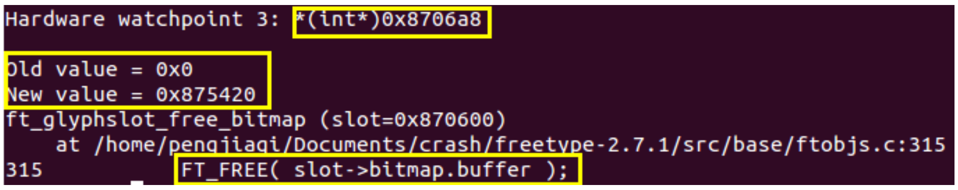
Next, we want to know when the *buffer* attribute is assigned/changed.

We use reverse execution of gdb.



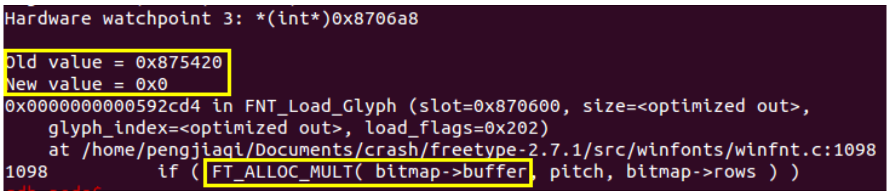
```
Breakpoint 2, _get_bitmap_surface (bitmap=bitmap@entry=0x870698,
    library=<optimized out>, own_buffer=own_buffer@entry=0x0,
    surface=surface@entry=0x7fffffffb598, font_options=<optimized out>)
    at cairo-ft-font.c:1175
1175                    source = bitmap->buffer;
gdb-peda$ p bitmap->buffer
$1 = (unsigned char *) 0x0
gdb-peda$ p &bitmap->buffer
$2 = (unsigned char **) 0x8706a8
```

First reverse:



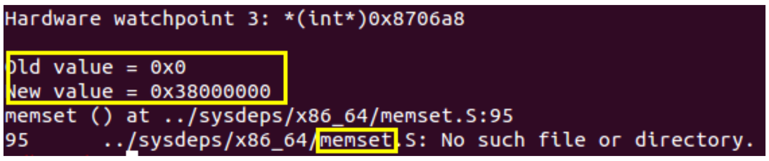
```
Hardware watchpoint 3: *(int*)0x8706a8

Old value = 0x0
New value = 0x875420
ft_glyphslot_free_bitmap (slot=0x870600)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:315
315          FT_FREE( slot->bitmap.buffer );
```

By calling FT_FREE(), buffer is set null. But before this, *buffer* must point to some memory. So, we continue reverse to see when the memory *buffer* points to is allocated.
Second reverse:



```
Hardware watchpoint 3: *(int*)0x8706a8

Old value = 0x875420
New value = 0x0
0x0000000000592cd4 in FNT_Load_Glyph (slot=0x870600, size=<optimized out>,
    glyph_index=<optimized out>, load_flags=0x202)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/winfonts/winfnt.c:1098
1098          if ( FT_ALLOC_MULT( bitmap->buffer, pitch, bitmap->rows ) )
```

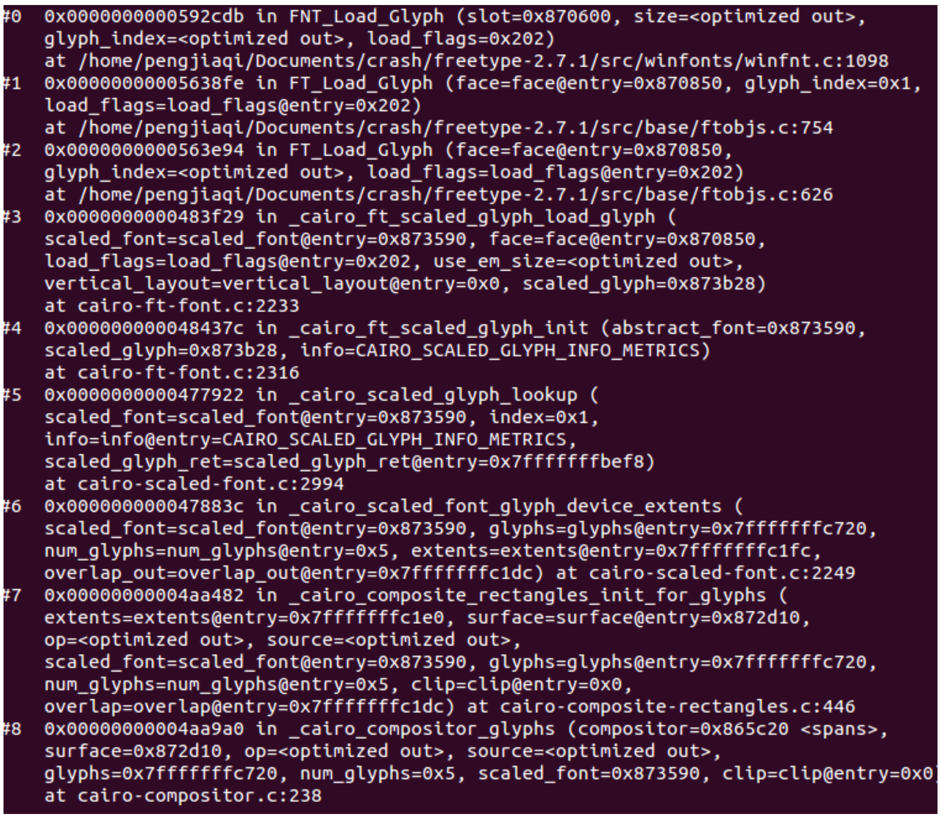
By calling FT_ALLOC_MULT(), *buffer* pointers to some allocated memory. We continue reverse to see any other operation on *buffer*.
Third reverse:



```
Hardware watchpoint 3: *(int*)0x8706a8

Old value = 0x0
New value = 0x38000000
memset () at ../sysdeps/x86_64/memset.S:95
95      ../sysdeps/x86_64/memset.S: No such file or directory.
```

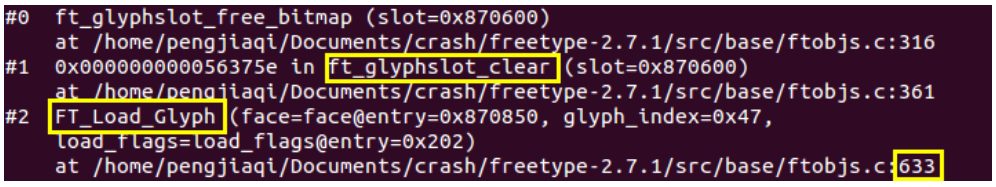
Here, we come to memset() of *buffer*, meaning on more information about *buffer* before. Since now, we know that there is **one allocation and** then **one free** operation on ***buffer***.

Next, we need to look carefully into the allocate and free.
First allocate, here is the call stack:



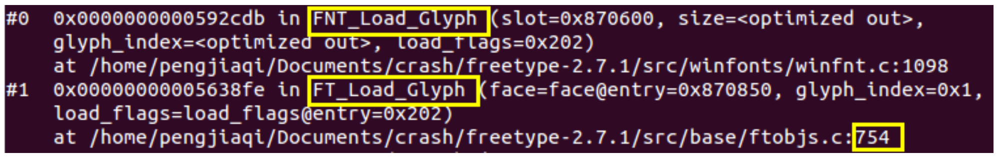
```
#0  0x0000000000592cdb in FNT_Load_Glyph (slot=0x870600, size=<optimized out>,
    glyph_index=<optimized out>, load_flags=0x202)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/winfonts/winfnt.c:1098
#1  0x00000000005638fe in FT_Load_Glyph (face=face@entry=0x870850, glyph_index=0x1,
    load_flags=load_flags@entry=0x202)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:754
#2  0x0000000000563e94 in FT_Load_Glyph (face=face@entry=0x870850,
    glyph_index=<optimized out>, load_flags=load_flags@entry=0x202)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:626
#3  0x0000000000483f29 in _cairo_ft_scaled_glyph_load_glyph (
    scaled_font=scaled_font@entry=0x873590, face=face@entry=0x870850,
    load_flags=load_flags@entry=0x202, use_em_size=<optimized out>,
    vertical_layout=vertical_layout@entry=0x0, scaled_glyph=0x873b28)
    at cairo-ft-font.c:2233
#4  0x000000000048437c in _cairo_ft_scaled_glyph_init (abstract_font=0x873590,
    scaled_glyph=0x873b28, info=CAIRO_SCALED_GLYPH_INFO_METRICS)
    at cairo-ft-font.c:2316
#5  0x0000000000477922 in _cairo_scaled_glyph_lookup (
    scaled_font=scaled_font@entry=0x873590, index=0x1,
    info=info@entry=CAIRO_SCALED_GLYPH_INFO_METRICS,
    scaled_glyph_ret=scaled_glyph_ret@entry=0x7fffffffbef8)
    at cairo-scaled-font.c:2994
#6  0x000000000047883c in _cairo_scaled_font_glyph_device_extents (
    scaled_font=scaled_font@entry=0x873590, glyphs=glyphs@entry=0x7fffffffc720,
    num_glyphs=num_glyphs@entry=0x5, extents=extents@entry=0x7fffffffc1fc,
    overlap_out=overlap_out@entry=0x7fffffffc1dc) at cairo-scaled-font.c:2249
#7  0x00000000004aa482 in _cairo_composite_rectangles_init_for_glyphs (
    extents=extents@entry=0x7fffffffc1e0, surface=surface@entry=0x872d10,
    op=<optimized out>, source=<optimized out>,
    scaled_font=scaled_font@entry=0x873590, glyphs=glyphs@entry=0x7fffffffc720,
    num_glyphs=num_glyphs@entry=0x5, clip=clip@entry=0x0,
    overlap=overlap@entry=0x7fffffffc1dc) at cairo-composite-rectangles.c:446
#8  0x00000000004aa9a0 in _cairo_compositor_glyphs (compositor=0x865c20 <spans>,
    surface=0x872d10, op=<optimized out>, source=<optimized out>,
    glyphs=0x7fffffffc720, num_glyphs=0x5, scaled_font=0x873590, clip=clip@entry=0x0
    at cairo-compositor.c:238
```

(ps: remaining functions in the allocate stack are the same as functions after #10 in crash stack)
For the free operation, the call stack is similar to allocate stack, except for the first 2/3 functions:



```
#0  ft_glyphslot_free_bitmap (slot=0x870600)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:316
#1  0x000000000056375e in ft_glyphslot_clear (slot=0x870600)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:361
#2  FT_Load_Glyph (face=face@entry=0x870850, glyph_index=0x47,
    load_flags=load_flags@entry=0x202)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:633
```

The first 2 functions in allocate stack:



```
#0  0x0000000000592cdb in FNT_Load_Glyph (slot=0x870600, size=<optimized out>,
    glyph_index=<optimized out>, load_flags=0x202)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/winfonts/winfnt.c:1098
#1  0x00000000005638fe in FT_Load_Glyph (face=face@entry=0x870850, glyph_index=0x1,
    load_flags=load_flags@entry=0x202)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:754
```

So, all allocate and free operation happens quite early (9 functions gap) before crash.
Next, we need to analysis FT_Load_Glyph():
It first calls ft_glyphslot_clear(), which definitely sets *buffer*=0 => free operation;
It then calls FNT_Load_Glyph(), which will do some comparisons:
    if all the cmp can succeed, we will come to the allocate operation;
    if one cmp fails, it will return some kind of error to FT_Load_Glyph().

Therefore, after calling FT_Load_Glyph():

if all comparisons succeed:

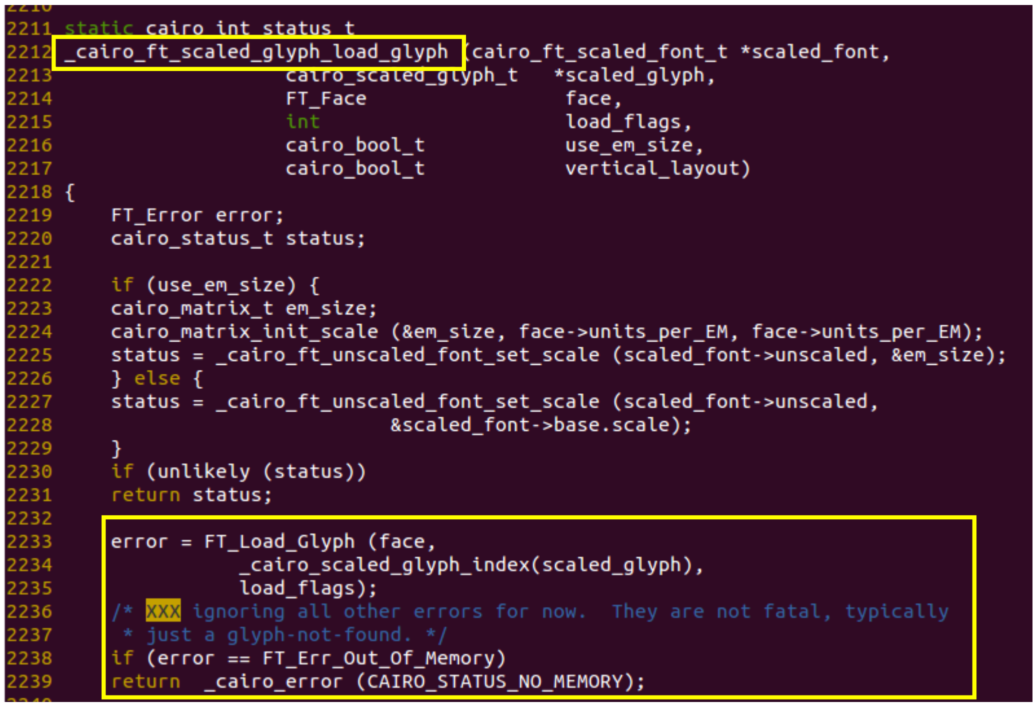
buffer != 0

else

buffer = 0

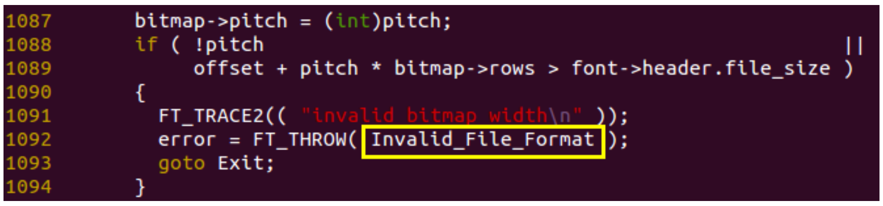
In this crash, buffer=0 is just because it fails some comparison.

We have mentioned, if buffer=0, it will return **error** to FT_Load_Glyph(), then return to upper caller _cairo_ft_scaled_glyph_load_glyph().
However, in _cairo_ft_scaled_glyph_load_glyph(), the error will be ignored except for FT_Err_Out_Of_Memory:



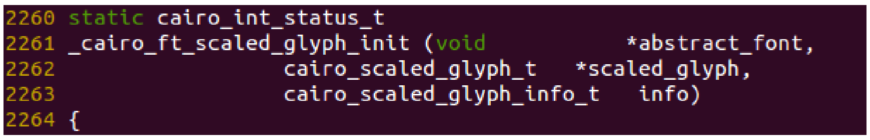
```
static cairo_int_status_t
_cairo_ft_scaled_glyph_load_glyph (cairo_ft_scaled_font_t *scaled_font,
                   cairo_scaled_glyph_t   *scaled_glyph,
                   FT_Face                face,
                   int                    load_flags,
                   cairo_bool_t           use_em_size,
                   cairo_bool_t           vertical_layout)
{
    FT_Error error;
    cairo_status_t status;

    if (use_em_size) {
    cairo_matrix_t em_size;
    cairo_matrix_init_scale (&em_size, face->units_per_EM, face->units_per_EM);
    status = _cairo_ft_unscaled_font_set_scale (scaled_font->unscaled, &em_size);
    } else {
    status = _cairo_ft_unscaled_font_set_scale (scaled_font->unscaled,
                          &scaled_font->base.scale);
    }
    if (unlikely (status))
    return status;

    error = FT_Load_Glyph (face,
              _cairo_scaled_glyph_index(scaled_glyph),
             load_flags);
    /* XXX ignoring all other errors for now.  They are not fatal, typically
     * just a glyph-not-found. */
    if (error == FT_Err_Out_Of_Memory)
    return  _cairo_error (CAIRO_STATUS_NO_MEMORY);
```

In this crash, the allocate operation fails due to the following cmp and returns error type - Invalid_File_Format:

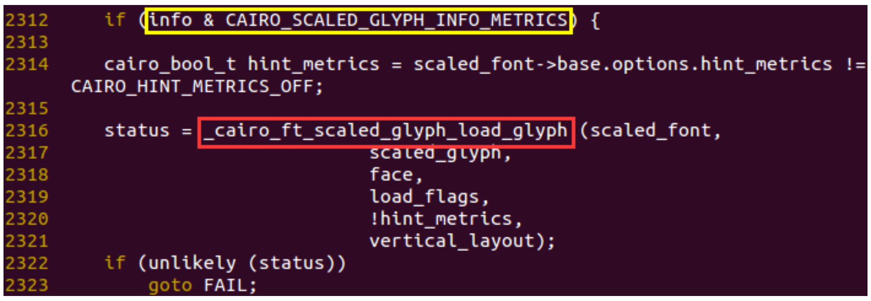


```
1087        bitmap->pitch = (int)pitch;
1088        if ( !pitch                                          ||
1089             offset + pitch * bitmap->rows > font->header.file_size )
1090        {
1091          FT_TRACE2(( "invalid bitmap width\n" ));
1092          error = FT_THROW( Invalid_File_Format );
1093          goto Exit;
1094        }
```

So, the error info will be ignored in _cairo_ft_scaled_glyph_load_glyph(), and when back to its caller _cairo_ft_scaled_glyph_init(), the return value will be success, so it won't go to FAIL branch to do some error handling.

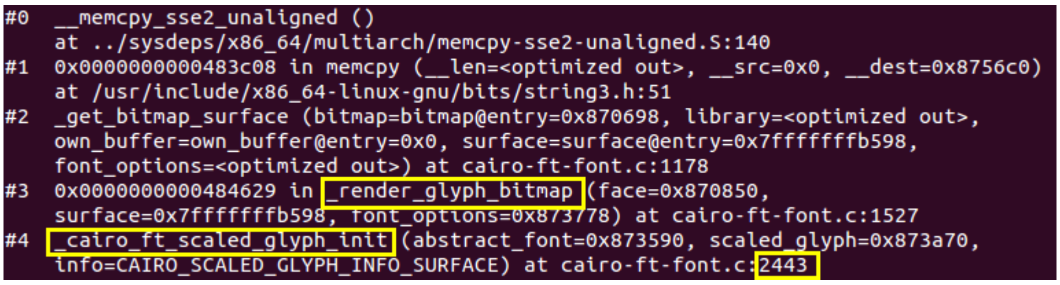


```
2260 static cairo_int_status_t
2261 _cairo_ft_scaled_glyph_init (void                   *abstract_font,
2262                              cairo_scaled_glyph_t   *scaled_glyph,
2263                              cairo_scaled_glyph_info_t   info)
2264 {
```

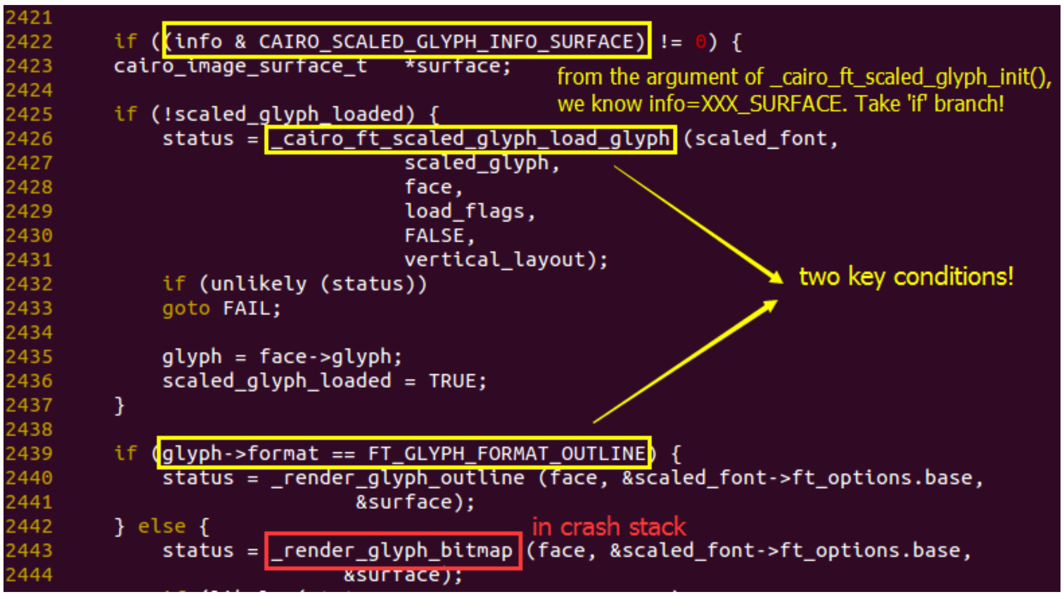


```
2312     if (info & CAIRO_SCALED_GLYPH_INFO_METRICS) {
2313
2314     cairo_bool_t hint_metrics = scaled_font->base.options.hint_metrics !=
  CAIRO_HINT_METRICS_OFF;
2315
2316     status = _cairo_ft_scaled_glyph_load_glyph (scaled_font,
2317                                                  scaled_glyph,
2318                                                  face,
2319                                                  load_flags,
2320                                                  !hint_metrics,
2321                                                  vertical_layout);
2322     if (unlikely (status))
2323         goto FAIL;
```

For the crash stack, it also contains _cairo_ft_scaled_glyph_init():



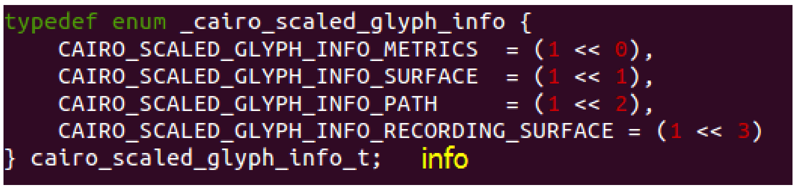
```
#0  __memcpy_sse2_unaligned ()
    at ../sysdeps/x86_64/multiarch/memcpy-sse2-unaligned.S:140
#1  0x0000000000483c08 in memcpy (__len=<optimized out>, __src=0x0, __dest=0x8756c0)
    at /usr/include/x86_64-linux-gnu/bits/string3.h:51
#2  _get_bitmap_surface (bitmap=bitmap@entry=0x870698, library=<optimized out>,
    own_buffer=own_buffer@entry=0x0, surface=surface@entry=0x7fffffffb598,
    font_options=<optimized out>) at cairo-ft-font.c:1178
#3  0x0000000000484629 in _render_glyph_bitmap (face=0x870850,
    surface=0x7fffffffb598, font_options=0x873778) at cairo-ft-font.c:1527
#4  _cairo_ft_scaled_glyph_init (abstract_font=0x873590, scaled_glyph=0x873a70,
    info=CAIRO_SCALED_GLYPH_INFO_SURFACE) at cairo-ft-font.c:2443
```



```
2421
2422     if ((info & CAIRO_SCALED_GLYPH_INFO_SURFACE) != 0) {
2423     cairo_image_surface_t    *surface;
2424
2425     if (!scaled_glyph_loaded) {
2426         status = _cairo_ft_scaled_glyph_load_glyph (scaled_font,
2427                                                      scaled_glyph,
2428                                                      face,
2429                                                      load_flags,
2430                                                      FALSE,
2431                                                      vertical_layout);
2432         if (unlikely (status))
2433         goto FAIL;
2434
2435         glyph = face->glyph;
2436         scaled_glyph_loaded = TRUE;
2437     }
2438
2439     if (glyph->format == FT_GLYPH_FORMAT_OUTLINE) {
2440         status = _render_glyph_outline (face, &scaled_font->ft_options.base,
2441                                         &surface);
2442     } else {
2443         status = _render_glyph_bitmap (face, &scaled_font->ft_options.base,
2444                                        &surface);
```

from the argument of _cairo_ft_scaled_glyph_init(),
we know info=XXX_SURFACE. Take 'if' branch!
two key conditions!
in crash stack

_cairo_ft_scaled_glyph_init() calls _cairo_ft_scaled_glyph_load_glyph() in different lines depending on the value of info. And the calling location of line 2316 or 2426 is excluded, which means if calling 2316, line 2426 won't be called and vice versa.



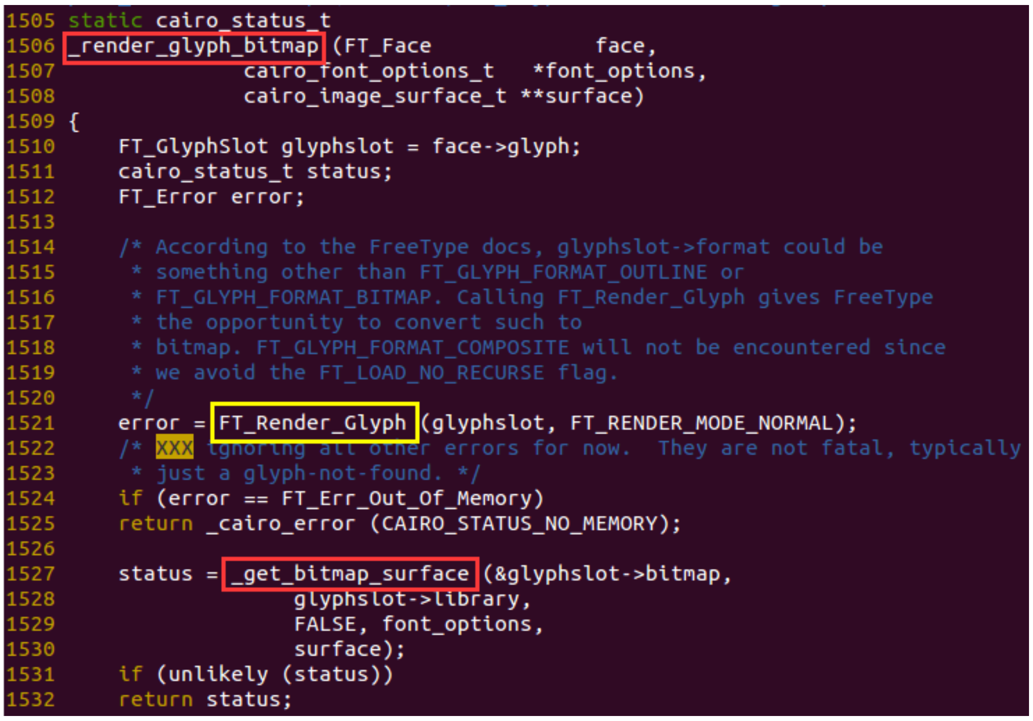
```
typedef enum _cairo_scaled_glyph_info {
    CAIRO_SCALED_GLYPH_INFO_METRICS   = (1 << 0),
    CAIRO_SCALED_GLYPH_INFO_SURFACE   = (1 << 1),
    CAIRO_SCALED_GLYPH_INFO_PATH      = (1 << 2),
    CAIRO_SCALED_GLYPH_INFO_RECORDING_SURFACE = (1 << 3)
} cairo_scaled_glyph_info_t;
```

Here, the caller of _cairo_ft_scaled_glyph_init() pass the argument *info* as CAIRO_SCALED_GLYPH_INFO_SURFACE, so the first if branch in line 2422 must take, meaning that in crash cairo_ft_scaled_glyph_load_glyph() is called in line 2426. Then, to reach render_glyph_bitmap(), which is in crash stack, we must satisfy the following two conditions:

(1)_cairo_ft_scaled_glyph_load_glyph() calls FT_Load_Glyph() and returns error (the type of error is not FT_Err_Out_Of_Memory, e.g. Invalid_File_Format)

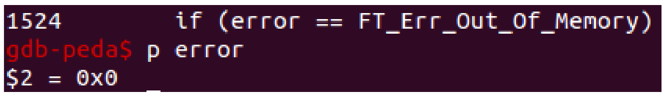
(2)glyph->format = face->glyph->format != FT_GLYPH_FORMAT_OUTLINE

For render_glyph_bitmap():



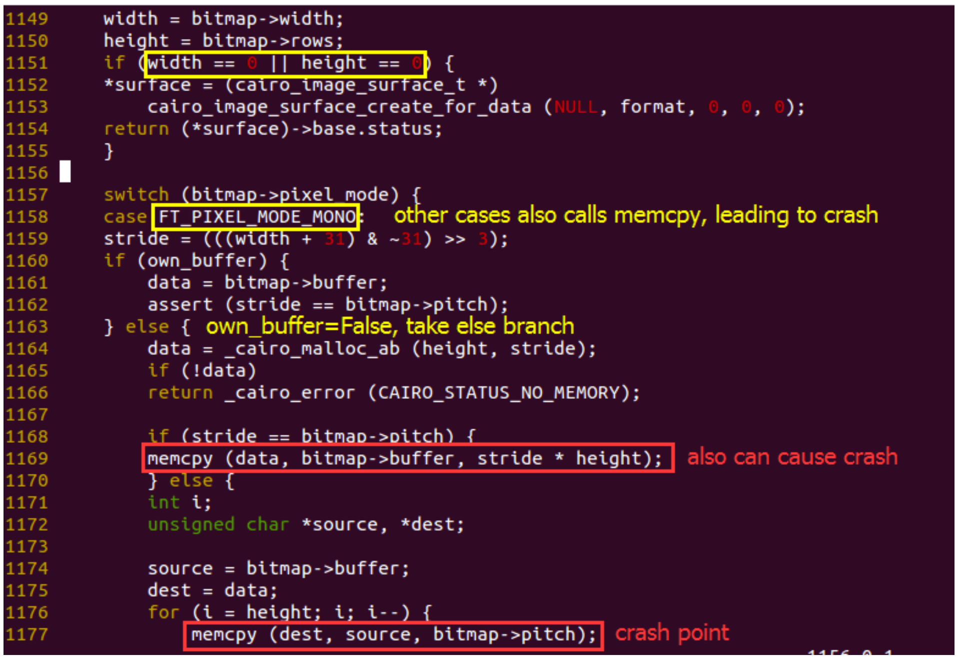
```
static cairo_status_t
_render_glyph_bitmap (FT_Face             face,
                      cairo_font_options_t   *font_options,
                      cairo_image_surface_t **surface)
{
    FT_GlyphSlot glyphslot = face->glyph;
    cairo_status_t status;
    FT_Error error;

    /* According to the FreeType docs, glyphslot->format could be
     * something other than FT_GLYPH_FORMAT_OUTLINE or
     * FT_GLYPH_FORMAT_BITMAP. Calling FT_Render_Glyph gives FreeType
     * the opportunity to convert such to
     * bitmap. FT_GLYPH_FORMAT_COMPOSITE will not be encountered since
     * we avoid the FT_LOAD_NO_RECURSE flag.
     */
    error = FT_Render_Glyph (glyphslot, FT_RENDER_MODE_NORMAL);
    /* XXX ignoring all other errors for now.  They are not fatal, typically
     * just a glyph-not-found. */
    if (error == FT_Err_Out_Of_Memory)
    return _cairo_error (CAIRO_STATUS_NO_MEMORY);

    status = _get_bitmap_surface (&glyphslot->bitmap,
                          glyphslot->library,
                          FALSE, font_options,
                          surface);
    if (unlikely (status))
    return status;
```

If FT_Render_Glyph() succeeds or doesn't return FT_Err_Out_Of_Memory when fails, we will come to get_bitmap_surface(). Here, the return error is 0, which means successfully return. So, we don't take this condition into account, which seems easy to satify.



```
1524        if (error == FT_Err_Out_Of_Memory)
gdb-peda$ p error
$2 = 0x0
```

For _get_bitmap_surface():



```
1149     width = bitmap->width;
1150     height = bitmap->rows;
1151     if (width == 0 || height == 0) {
1152     *surface = (cairo_image_surface_t *)
1153         cairo_image_surface_create_for_data (NULL, format, 0, 0, 0);
1154     return (*surface)->base.status;
1155     }
1156 
1157     switch (bitmap->pixel_mode) {
1158     case FT_PIXEL_MODE_MONO:
1159     stride = (((width + 31) & ~31) >> 3);
1160     if (own_buffer) {
1161         data = bitmap->buffer;
1162         assert (stride == bitmap->pitch);
1163     } else {
1164         data = _cairo_malloc_ab (height, stride);
1165         if (!data)
1166         return _cairo_error (CAIRO_STATUS_NO_MEMORY);
1167 
1168         if (stride == bitmap->pitch) {
1169         memcpy (data, bitmap->buffer, stride * height);
1170         } else {
1171         int i;
1172         unsigned char *source, *dest;
1173 
1174         source = bitmap->buffer;
1175         dest = data;
1176         for (i = height; i; i--) {
1177             memcpy (dest, source, bitmap->pitch);
```

other cases also calls memcpy, leading to crash

own_buffer=False, take else branch

also can cause crash

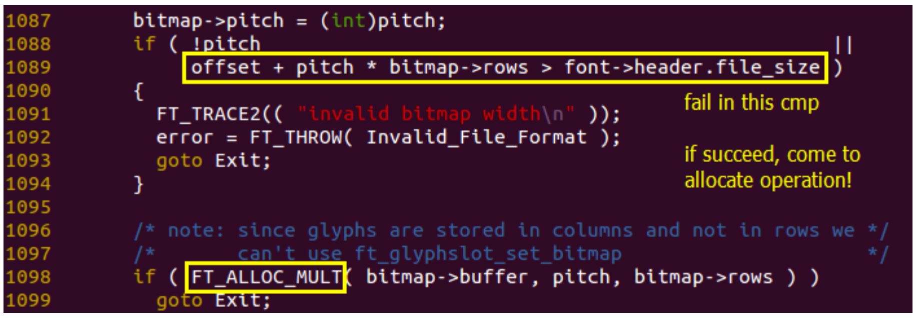
crash point

If **(3)bitmap->width!=0 && bitmap->rows!=0**, we will mostly come to memcpy(), which can cause null pointer deference.

In conclusion, to cause null pointer deference, the following three conditions are the least to satisfy:

(1)_cairo_ft_scaled_glyph_load_glyph() calls FT_Load_Glyph() and returns error (the type of error is not FT_Err_Out_Of_Memory, e.g. Invalid_File_Format)

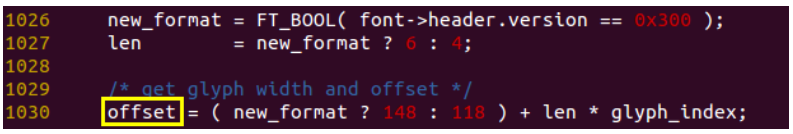
In the free operation discussed above, FT_Load_Glyph() calls FNT_Load_Glyph() and fails in line 1088:



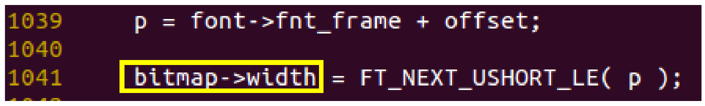
```
1087        bitmap->pitch = (int)pitch;
1088        if ( !pitch                                           ||
1089             offset + pitch * bitmap->rows > font->header.file_size )
1090        {
1091          FT_TRACE2(( "invalid bitmap width\n" ));
1092          error = FT_THROW( Invalid_File_Format );
1093          goto Exit;
1094        }
1095 
1096        /* note: since glyphs are stored in columns and not in rows we */
1097        /*       can't use ft_glyphslot_set_bitmap                     */
1098        if ( FT_ALLOC_MULT( bitmap->buffer, pitch, bitmap->rows ) )
1099          goto Exit;
```

fail in this cmp

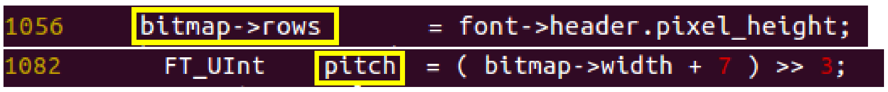
if succeed, come to allocate operation!



```
1026     new_format = FT_BOOL( font->header.version == 0x300 );
1027     len        = new_format ? 6 : 4;
1028 
1029     /* get glyph width and offset */
1030     offset = ( new_format ? 148 : 118 ) + len * glyph_index;
```



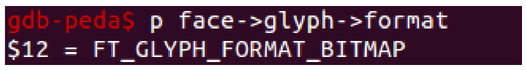
```
1039      p = font->fnt_frame + offset;
1040 
1041     bitmap->width = FT_NEXT_USHORT_LE( p );
```



```
1056     bitmap->rows        = font->header.pixel_height;
1082        FT_UInt    pitch  = ( bitmap->width + 7 ) >> 3;
```

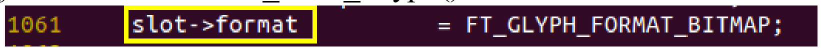
From screenshot, we know that **result of the critical cmp operation depends *font*.**

(1) glyph->format = face->glyph->format != FT_GLYPH_FORMAT_OUTLINE



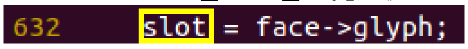
```
gdb-peda$ p face->glyph->format
$12 = FT_GLYPH_FORMAT_BITMAP
```

format is set in FNT_Load_Glyph():



```
slot->format          = FT_GLYPH_FORMAT_BITMAP;
```

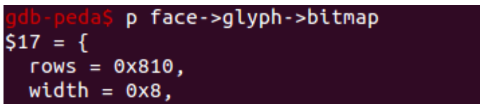
slot is set in FT_Load_Glyph():



```
slot = face->glyph;
```

So, after calling FNT_Load_Glyph(), face->glyph->format=FT_GLYPH-FORMAT-BITMAP.

(2) bitmap->width!=0 && bitmap->rows!=0 (bitmap=&face->glyph->bitmap)

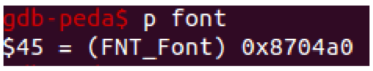


```
gdb-peda$ p face->glyph->bitmap
$17 = {
  rows = 0x810,
  width = 0x8,
```

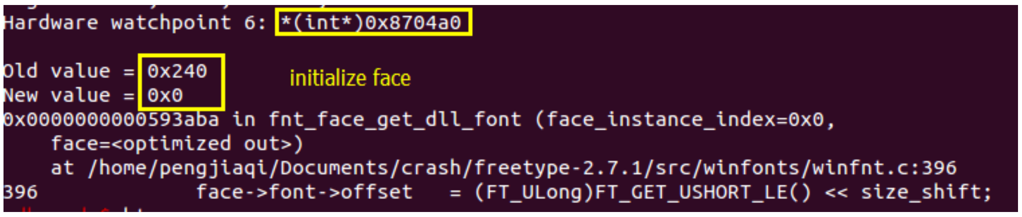
width and *rows* are also set in FNT_Load_Glyph(), which has been shown in the first condition (1).

Therefore, all conditions to crash are directly related to *font*.

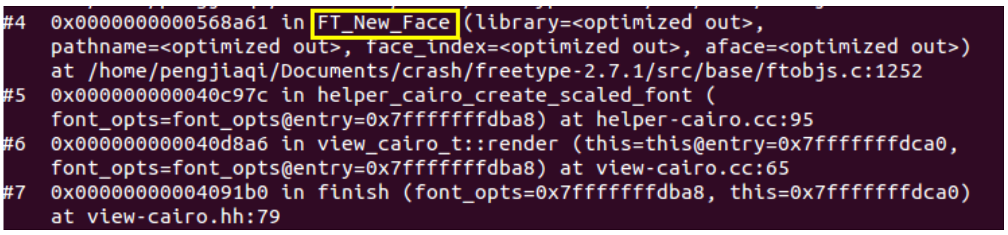
We continue to see when font is set.



```
gdb-peda$ p font
$45 = (FNT_Font) 0x8704a0
```

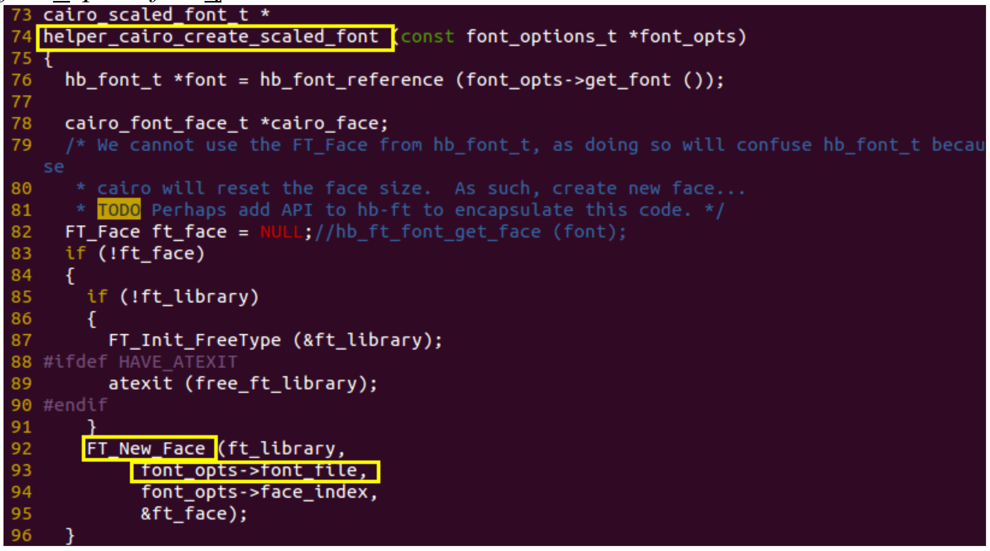


```
Hardware watchpoint 6: *(int*)0x8704a0

Old value = 0x240
New value = 0x0
0x0000000000593aba in fnt_face_get_dll_font (face_instance_index=0x0,
    face=<optimized out>)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/winfonts/winfnt.c:396
396             face->font->offset   = (FT_ULong)FT_GET_USHORT_LE() << size_shift;
```

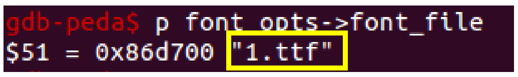


```
#4  0x0000000000568a61 in FT_New_Face (library=<optimized out>,
    pathname=<optimized out>, face_index=<optimized out>, aface=<optimized out>)
    at /home/pengjiaqi/Documents/crash/freetype-2.7.1/src/base/ftobjs.c:1252
#5  0x000000000040c97c in helper_cairo_create_scaled_font (
    font_opts=font_opts@entry=0x7fffffffdba8) at helper-cairo.cc:95
#6  0x000000000040d8a6 in view_cairo_t::render (this=this@entry=0x7fffffffdca0,
    font_opts=font_opts@entry=0x7fffffffdba8) at view-cairo.cc:65
#7  0x00000000004091b0 in finish (font_opts=0x7fffffffdba8, this=0x7fffffffdca0)
    at view-cairo.hh:79
```

In FT_New_Face(), ***face*** is initialized. And face is initialized according to *font_opts->font_file.*



```
cairo_scaled_font_t *
helper_cairo_create_scaled_font (const font_options_t *font_opts)
{
  hb_font_t *font = hb_font_reference (font_opts->get_font ());

  cairo_font_face_t *cairo_face;
  /* We cannot use the FT_Face from hb_font_t, as doing so will confuse hb_font_t because
   * cairo will reset the face size.  As such, create new face...
   * TODO Perhaps add API to hb-ft to encapsulate this code. */
  FT_Face ft_face = NULL;//hb_ft_font_get_face (font);
  if (!ft_face)
  {
    if (!ft_library)
    {
      FT_Init_FreeType (&ft_library);
#ifdef HAVE_ATEXIT
      atexit (free_ft_library);
#endif
    }
    FT_New_Face (ft_library,
         font_opts->font_file,
         font_opts->face_index,
         &ft_face);
  }
```

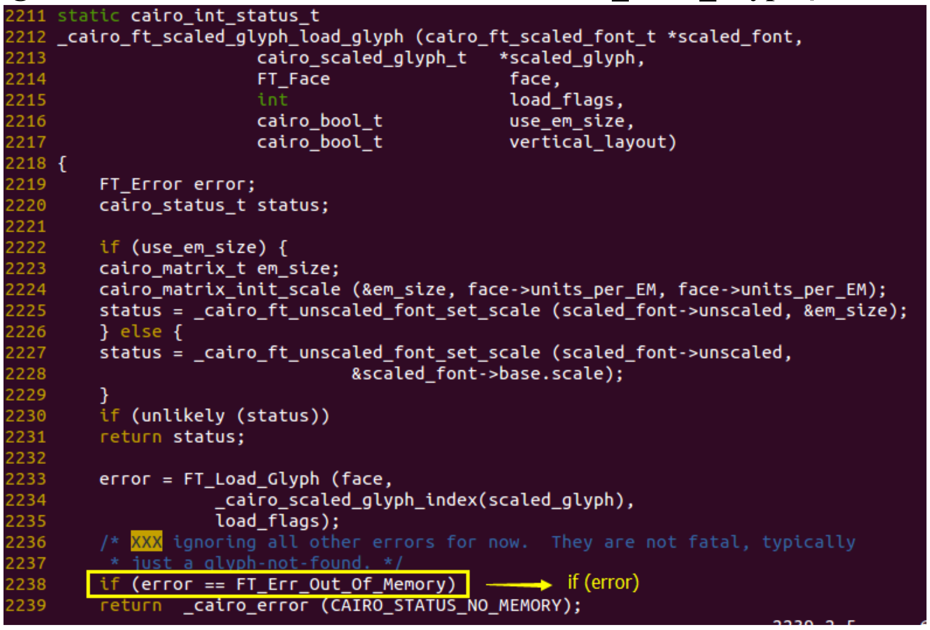


```
gdb-peda$ p font_opts->font_file
$51 = 0x86d700 "1.ttf"
```

font_opts->font_file is just our PoC font file.
So, *face* is controlled by PoC and the conditions analyzed above to crash are all dependent on *face*, meaning that an attacker can craft a font file that causes null pointer deference in memcpy(), making a Denial_of_Service attack.

Patch

The crash is caused by null pointer deference in memcpy, and src=0 is caused by failing to allocate after free. Further, when allocation fails, the current function will return error, but the error will be ignored by upper function (except error type is FT_Err_Out_Of_Memory). So, the upper function will think allocation operation is successfully done and next call render_glyph_bitmap(), which will finally call memcpy() with src=pointer_to_allocated_memory=0.
So, the patch is: **in _cairo_ft_scaled_glyph_load_glyph() @cairo-ft-font.c, don't ignore the error information return from FT_Load_Glyph()**.



```
2211 static cairo_int_status_t
2212 _cairo_ft_scaled_glyph_load_glyph (cairo_ft_scaled_font_t *scaled_font,
2213                     cairo_scaled_glyph_t   *scaled_glyph,
2214                     FT_Face                 face,
2215                     int                     load_flags,
2216                     cairo_bool_t            use_em_size,
2217                     cairo_bool_t            vertical_layout)
2218 {
2219     FT_Error error;
2220     cairo_status_t status;
2221 
2222     if (use_em_size) {
2223     cairo_matrix_t em_size;
2224     cairo_matrix_init_scale (&em_size, face->units_per_EM, face->units_per_EM);
2225     status = _cairo_ft_unscaled_font_set_scale (scaled_font->unscaled, &em_size);
2226     } else {
2227     status = _cairo_ft_unscaled_font_set_scale (scaled_font->unscaled,
2228                           &scaled_font->base.scale);
2229     }
2230     if (unlikely (status))
2231     return status;
2232 
2233     error = FT_Load_Glyph (face,
2234                _cairo_scaled_glyph_index(scaled_glyph),
2235                load_flags);
2236     /* XXX ignoring all other errors for now.  They are not fatal, typically
2237      * just a glyph-not-found. */
2238     if (error == FT_Err_Out_Of_Memory)
2239     return  _cairo_error (CAIRO_STATUS_NO_MEMORY);
```

Change line 2238 to if(error), which will handle all kinds of error.

Or, the patch can be: add some checks (if src!=0) before each memcpy() in _get_bitmap_surface()@ cairo-ft-font.c, which may be a little troublesome but can also avoid this Denial-of-Service attack.

Author

Name: Jiaqi Peng, Bingchang Liu of VARAS@IIE
Organization: IIE (http://iie.ac.cn/)