Arbitrary memory overwrite occurs when loading glyphs and rendering text with a malformed TTF file. #187

Closed **ch4rli3kop** opened this issue on Mar 19 · 2 comments

ch4rli3kop commented on Mar 19

Hello, I found a vulnerability in this project.

Summary

Arbitrary memory overwrite occurs when loading glyphs and rendering text with a malformed TTF file.

System Info

- Operating System: Ubuntu 20.04

Detailed Description

When the function `TTF_RenderText_Solid()` is executed, it internally calls `TTF_Size_Internal()` and `Render_Line()`. Since the code load and render glyph data after measuring bitmap size, if the measured size has a problem, it causes memory overflow or arbitrary memory write when rendering the data. The bitmap size of glyph affects variables `xstart` and `ystart`. And they are used to calculate the destination of `BG_SSE()`. Therefore a malformed TTF file that has manipulated glyph data will result in memory corruption.

If the rendered string has only characters that mapped malformed glyph data, `ft_failure` occurs when calling `FT_Render_Glyph`. But, if the string has a character that mapped normally formed glyph data in front of the mal-mapped character, the normal character is rendered with corrupted size while `FT_Render_Glyph` is normally called. So, the normal character's glyph data is overwritten to arbitrary memory addresses with corrupted `xstart` and `ystart`. The address will be heap or stack.

In the below code and attached malformed TTF file, a character "T" has normal glyph data and a character "S" has malformed glyph data. Since The address of the calculated destination with `xstart` and `ystart` is not a valid memory address, a segmentation fault occurs. debugged data is below.

```
Starting program: /home/ch4rli3kop/SDL/fuzz/OpenTTF crashed/access_violation_0000xxxxxxxxx9E4_0000xxx
[Thread debugging using libthread_db enabled]
```

```
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
INFO: Found charmap: platform id 0, encoding id 280

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7c25318 in _mm_load_si128 (__P=0x7fffd9fc71a0) at /usr/lib/gcc/x86_64-linux-gnu/9/include/
697       return *__P;

[ Legend: Modified register | Code | Heap | Stack | String ]
────────────────────────────────────────────────────────────────────────────────────────────────────
$rax   : 0x7fffd9fc71a0
$rbx   : 0x00555555555860  →  <__libc_csu_init+0> endbr64
$rcx   : 0x84100
$rdx   : 0x0
$rsp   : 0x007fffffffdb78  →  0x00555555edc5a0  →  0x00555555e51e00  →  0x0000000000000001
$rbp   : 0x007fffffffdbc0  →  0x007fffffffdc70  →  0x007fffffffdcc0  →  0x007fffffffdd50  →  0x007fff
$rsi   : 0x7fffd9fc71a0
$rdi   : 0x00555555ee2d78  →  0x00555555e2c15d  →  0x0000000000000000
$rip   : 0x007ffff7c25318  →  <BG_SSE+162> movdqa xmm0, XMMWORD PTR [rax]
$r8    : 0x00555555e2c150  →  0x0000000000000000
$r9    : 0x007fffffffd8b4  →  0x5555586000000000
$r10   : 0xfffffffffffff3cf
$r11   : 0x007ffffff98b8  →  0x00007fff00000001
$r12   : 0x00555555555300  →  <_start+0> endbr64
$r13   : 0x007fffffffe020  →  0x0000000000000002
$r14   : 0x0
$r15   : 0x0
$eflags: [zero carry parity adjust sign trap INTERRUPT direction overflow RESUME virtualx86 IDENTIFIC
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
────────────────────────────────────────────────────────────────────────────────────────────────────
0x007fffffffdb78│+0x0000: 0x00555555edc5a0  →  0x00555555e51e00  →  0x0000000000000001   ← $rsp
0x007fffffffdb80│+0x0008: 0x00005b0000006e ("n"?)
0x007fffffffdb88│+0x0010: 0x00000011f78f67c3
0x007fffffffdb90│+0x0018: 0x0000000000000000
0x007fffffffdb98│+0x0020: 0x00555555ee2d70  →  0x000000ff00000011
0x007fffffffdba0│+0x0028: 0x007fffffffdc70  →  0x007fffffffdcc0  →  0x007fffffffdd50  →  0x007fffffff
0x007fffffffdba8│+0x0030: 0x007ffff7c2582f  →  <Render_Line_SSE_Solid+272> add rsp, 0x20
0x007fffffffdbb0│+0x0038: 0x0000000000000000
────────────────────────────────────────────────────────────────────────────────────────────────────
   0x7ffff7c2530c <BG_SSE+150>     mov    rax, QWORD PTR [rbp-0x80]
   0x7ffff7c25310 <BG_SSE+154>     mov    QWORD PTR [rbp-0x70], rax
   0x7ffff7c25314 <BG_SSE+158>     mov    rax, QWORD PTR [rbp-0x70]
 → 0x7ffff7c25318 <BG_SSE+162>     movdqa xmm0, XMMWORD PTR [rax]
   0x7ffff7c2531c <BG_SSE+166>     movaps XMMWORD PTR [rbp-0x50], xmm0
   0x7ffff7c25320 <BG_SSE+170>     movdqa xmm0, XMMWORD PTR [rbp-0x50]
   0x7ffff7c25325 <BG_SSE+175>     movaps XMMWORD PTR [rbp-0x20], xmm0
   0x7ffff7c25329 <BG_SSE+179>     movdqa xmm0, XMMWORD PTR [rbp-0x60]
   0x7ffff7c2532e <BG_SSE+184>     movaps XMMWORD PTR [rbp-0x10], xmm0
────────────────────────────────────────────────────────────────────────────────────────────────────
    692  /* Create a vector with element 0 as *P and the rest zero.  */
    693
    694  extern __inline __m128i __attribute__((__gnu_inline__, __always_inline__, __artificial__))
    695  _mm_load_si128 (__m128i const *__P)
    696  {
           // __P=0x007fffffffdb50  →  0x00007fffd9fc71a0
 →  697    return *__P;
    698  }
    699
    700  extern __inline __m128i __attribute__((__gnu_inline__, __always_inline__, __artificial__))
    701  _mm_loadu_si128 (__m128i_u const *__P)
    702  {
──────────────────────────────────────────────────────────────────────────────────────────────────
[#0] Id 1, Name: "OpenTTF", stopped 0x7ffff7c25318 in _mm_load_si128 (), reason: SIGSEGV
──────────────────────────────────────────────────────────────────────────────────────────────────
[#0] 0x7ffff7c25318 → _mm_load_si128(__P=0x7fffd9fc71a0)
[#1] 0x7ffff7c25318 → BG_SSE(image=0x555555ee2d78, destination=0x7fffd9fc71a0 <error: Cannot access m
[#2] 0x7ffff7c259ec → Render_Line_SSE_Solid(font=0x555555edc5a0, textbuf=0x555555ed3170, xstart=0x254
[#3] 0x7ffff7c2abc4 → Render_Line(render_mode=RENDER_SOLID, subpixel=0x0, font=0x555555edc5a0, textbu
[#4] 0x7ffff7c2f631 → TTF_Render_Internal(font=0x555555edc5a0, text=0x7fffffffdce0 "TV", str_type=STR
  r = 0x6f,
  g = 0x6f,
  b = 0xff,
  a = 0xff
}, bg={
  r = 0x6f,
  g = 0x6f,
  b = 0xff,
  a = 0xff
}, render_mode=RENDER_SOLID)
[#5] 0x7ffff7c2f746 → TTF_RenderText_Solid(font=0x555555edc5a0, text=0x555555556035 "TV", fg={
  r = 0x6f,
  g = 0x6f,
  b = 0xff,
  a = 0x0
})
[#6] 0x5555555555f6 → fuzzme(file=0x7fffffffe37f "crashed/access_violation_0000xxxxxxxxx9E4_0000xxxxx
[#7] 0x55555555582e → main(argc=0x2, argv=0x7fffffffe028)
```

Reproduce

compile the below code and run the program with a malformed TTF file. A malformed TTF file link is here

main.cpp

```
#include <cstdlib>
#include "SDL.h"
#include "SDL_ttf.h"

#pragma comment (lib, "SDL2")
#pragma comment (lib, "SDL2main")
#pragma comment (lib, "SDL2_ttf")


#define WIN_W 0x400
#define WIN_H 0x300

void render_ttf(char* file, SDL_Renderer* mRenderer) {
        TTF_Font* font;
        char buf[0x100] = {0,};
        if (TTF_Init() == -1) {
                SDL_Log("Failed to init ttf : %s", SDL_GetError());
                return;
        }

        font = TTF_OpenFont(file, 40);
        if (!font) {
                SDL_Log("Failed to open font : %s", SDL_GetError());
                return;
        }

        SDL_Color color = { 111, 111, 255 };
        SDL_Surface* surface = TTF_RenderText_Solid(font, "TS", color);
        SDL_Texture* texture = SDL_CreateTextureFromSurface(mRenderer, surface);
        int W = 0, H = 0;
        SDL_QueryTexture(texture, NULL, NULL, &W, &H);
        SDL_Rect dstrect = { 0, 0, W, H };
        SDL_RenderCopy(mRenderer, texture, NULL, &dstrect);

        SDL_RenderPresent(mRenderer);

/*      SDL_Event event;
        int done = 0;
        while (!done) {
                SDL_PollEvent(&event);
                if (event.type == SDL_QUIT)
                { done = 1; }
        }
*/
        SDL_DestroyTexture(texture);
        SDL_FreeSurface(surface);

        TTF_CloseFont(font);
        TTF_Quit();
}

int main(int argc, char* argv[]) {

        SDL_Window* mWindow;
        SDL_Renderer* mRenderer;

        int sdlResult = SDL_Init(SDL_INIT_VIDEO);
        if (sdlResult) {
                SDL_Log("Unable to initialize SDL: %s", SDL_GetError());
                return false;
        }

        mWindow = SDL_CreateWindow(
                "SDL Font Test",
                100,
                100,
                WIN_W,
                WIN_H,
                SDL_WINDOW_RESIZABLE
        );
        if (!mWindow) {
                SDL_Log("Failed to create window : %s", SDL_GetError());
                return false;
        }

        mRenderer = SDL_CreateRenderer(
                mWindow,
                -1,
                SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC
        );

        if (!mRenderer) {
                SDL_Log("Failed to render window : %s", SDL_GetError());
                return false;
        }

        SDL_SetRenderDrawColor(mRenderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
        SDL_RenderClear(mRenderer);

        render_ttf(argv[1], mRenderer);

        SDL_DestroyRenderer(mRenderer);
        SDL_DestroyWindow(mWindow);
        SDL_Quit();
}
```

compile & run

```
ch4rli3kop@ubuntu:~/SDL/fuzz$ g++ -o OpenTTF OpenTTF.cpp -D_REENTRANT -I/usr/local/include/SDL2 -L/us
ch4rli3kop@ubuntu:~/SDL/fuzz$ ./OpenTTF crashed/access_violation_0000xxxxxxxxx9E4_0000xxxxxxxxx800_1
Segmentation fault (core dumped)
```

Conclusion

In my thought, the part of the responsibility for this vulnerability partially rests with FreetypeFont. Usually, opening malformed TTF file results in an error code. But in here, it doesn't. I will report this issue to FreetypeFont. However, I think it would be good to add the routine that checks the range of variables `xstart` and `ystart` before calling `Render_Line`.

1bsyl added a commit that referenced this issue on Mar 19

`Fixed bug #187 - Arbitrary memory overwrite occurs when loading glyph…` … `09a2294`

1bsyl commented on Mar 19 Contributor

Thanks for the test-case !

the issue is that the font (indeed malformed, but acceptable) gives big width/height. and the final size wasn't calculated with 64 bits precision (eg badly calculated).

(even if glyph goes outside, it gets clipped/truncated. if badly loaded, it get rejected)

❤️ 1

1bsyl closed this as completed on Mar 19

1bsyl added a commit that referenced this issue on Mar 19

`More integer overflow (see bug #187)` … db1b41a

smcv commented on May 7 Contributor

CVE-2022-27470 has apparently been assigned to this issue.

This was referenced on May 9

Add inlines for overflow detection libsdl-org/SDL#5643

Merged

Sync up Create_Surface_LCD with other surface creation #203

Merged

slouken pushed a commit that referenced this issue on May 9

`Use 64-bit arithmetic in Create_Surface_LCD` … c8553b7

smcv mentioned this issue on May 9

Check for overflow more carefully #204

Merged

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

3 participants