New issue

Jump to bottom

Integer overflow in bmp_load() resulting in heap overflow in jfif_encode() at jfif.c:763 #49

Open **0xdd96** opened this issue on Mar 25 · 7 comments

0xdd96 commented on Mar 25

version: master (commit caade60)
poc: poc
command: ./ffjpeg -e *poc*

Here is the trace reported by ASAN:

```
user@c3ae4d510abb:/path_to_ffjpeg/src$ ./ffjpeg -e poc
=================================================================
==17827==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x612000000148 at pc
0x555555567e84 bp 0x7fffffffe120 sp 0x7fffffffe110
READ of size 1 at 0x612000000148 thread T0
    #0 0x555555567e83 in jfif_encode /path_to_ffjpeg/src/jfif.c:763
    #1 0x555555556c63 in main /path_to_ffjpeg/src/ffjpeg.c:33
    #2 0x7ffff73bf0b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x240b2)
    #3 0x55555555704d in _start (/path_to_ffjpeg/src/ffjpeg+0x304d)

0x612000000148 is located 0 bytes to the right of 264-byte region [0x612000000040,0x612000000148)
allocated by thread T0 here:
    #0 0x7ffff769abc8 in malloc (/lib/x86_64-linux-gnu/libasan.so.5+0x10dbc8)
    #1 0x555555557987 in bmp_load /path_to_ffjpeg/src/bmp.c:48

SUMMARY: AddressSanitizer: heap-buffer-overflow /path_to_ffjpeg/src/jfif.c:763 in jfif_encode
```

This issue is the same as #38, but the fix to it (0fa4cf8) is not complete. An integer overflow is still possible in line 43. In the example below, when `width=1431655779`, `pb->stride=44` which bypasses the check in line 44. This will lead to a heap buffer flow in `jfif.c` as in the ASAN report above.

ffjpeg/src/bmp.c
Lines 41 to 47 in caade60

```
41      pb->width  = (int)header.biWidth  > 0 ? (int)header.biWidth  : 0;
42      pb->height = (int)header.biHeight > 0 ? (int)header.biHeight : 0;
43      pb->stride = ALIGN(pb->width * 3, 4);
```

```
44      if ((long long)pb->stride * pb->height >= 0x80000000) {
45          printf("bmp's width * height is out of range !\n");
46          goto done;
47      }
```

```
pwndbg> p pb
$3 = (BMP *) 0x7fffffffe370
pwndbg> p *(BMP *) 0x7fffffffe370
$4 = {
  width = 1431655779,
  height = 6,
  stride = 44,
  pdata = 0x555555576490
}
```

Marsman1996 commented on Mar 31 Contributor

I cannot reproduce the crash with the provided poc file.

```
❯ ./bin_asan/bin/ffjpeg -e ffjpeg-bmp_load-integer-overflow
=================================================================
==2742398==ERROR: AddressSanitizer: allocator is out of memory trying to allocate 0x1555555c00
bytes
    #0 0x4c5587 in calloc /home/ubuntu178/Downloads/llvm12/projects/compiler-
rt/lib/asan/asan_malloc_linux.cpp:154:3
    #1 0x507342 in jfif_encode
/home/ubuntu178/cvelibf/test/ffjpeg/caade60/build_asan/src/jfif.c:749:21
    #2 0x4fc0bf in main /home/ubuntu178/cvelibf/test/ffjpeg/caade60/build_asan/src/ffjpeg.c:33:16
    #3 0x7fa4e61cb0b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-
start.c:308:16

==2742398==HINT: if you don't care about these errors you may set allocator_may_return_null=1
SUMMARY: AddressSanitizer: out-of-memory /home/ubuntu178/Downloads/llvm12/projects/compiler-
rt/lib/asan/asan_malloc_linux.cpp:154:3 in calloc
==2742398==ABORTING
❯ ASAN_OPTIONS="allocator_may_return_null=1" ./bin_asan/bin/ffjpeg -e ffjpeg-bmp_load-integer-
overflow
❯ ls
bin_afl    bin_asan    build.sh   build_aflpp  build_normal  decode.bmp  ffjpeg-bmp_load-integer-
overflow
bin_aflpp  bin_normal  build_afl  build_asan   code          encode.jpg  ffjpeg-jfif_load-buffer-
overflow
```

I tested with gdb and find that the poc did not pass the check in jfif.c:752 and with poc the program does not execute the jfif.c:763.

```
(gdb) set args -e ./ffjpeg-bmp_load-integer-overflow
(gdb) b jfif.c:752
```

```
Breakpoint 1 at 0x48f9: file jfif.c, line 752.
(gdb) r
Starting program: /home/ubuntu178/cvelibf/test/ffjpeg/caade60/bin_normal/bin/ffjpeg -e ./ffjpeg-
bmp_load-integer-overflow

Breakpoint 1, jfif_encode (pb=0x7fffffffdd40) at jfif.c:752
752         if (!yuv_datbuf[0] || !yuv_datbuf[1] || !yuv_datbuf[2]) {
(gdb) p *(BMP *) pb
$1 = {width = 1431655779, height = 6, stride = 44, pdata = 0x555555561490}
(gdb) n
753             goto done;
(gdb)
850         if (yuv_datbuf[0]) free(yuv_datbuf[0]);
(gdb) p yuv_datbuf[0]
$2 = (int *) 0x0
```

Tested in Ubuntu 20.04, 64bit; Clang 12.0.0.

0xdd96 commented on Mar 31 — Author

Did you copy-paste the file from browser? The poc contains special characters in the end.

```
❯ xxd ffjpeg-bmp_load-integer-overflow
00000000: 5f55 5555 5555 553d 3635 7155 5555 5455  _UUUUUU=65qUUUTU
00000010: 5555 6355 5555 0600                      UUcUUU..
```

ps: I tried to copy the PoC from browser and encountered the same output as you :)
Maybe you could try download the PoC with the RAW link

My test environment is

```
user@c3ae4d510abb:$ cat /etc/issue
Ubuntu 20.04.4 LTS \n \l

user@c3ae4d510abb:$ gcc --version
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Marsman1996 commented on Apr 1 — Contributor

The poc file is downloaded from the browser, but the `xxd` shows same result as yours:

```
❯ xxd ffjpeg-bmp_load-integer-overflow
00000000: 5f55 5555 5555 553d 3635 7155 5555 5455  _UUUUUU=65qUUUTU
00000010: 5555 6355 5555 0600                      UUcUUU..
```

And I use the `wget` to download the poc file and get the same result:

```
❯ wget https://github.com/dandanxu96/PoC/raw/main/ffjpeg/ffjpeg-bmp_load-integer-overflow
--2022-04-01 11:53:22--  https://github.com/dandanxu96/PoC/raw/main/ffjpeg/ffjpeg-bmp_load-
integer-overflow
Resolving github.com (github.com)... 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/dandanxu96/PoC/main/ffjpeg/ffjpeg-bmp_load-integer-
overflow [following]
--2022-04-01 11:53:23--  https://raw.githubusercontent.com/dandanxu96/PoC/main/ffjpeg/ffjpeg-
bmp_load-integer-overflow
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 2606:50c0:8000::154,
2606:50c0:8002::154, 2606:50c0:8001::154, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|2606:50c0:8000::154|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 24 [application/octet-stream]
Saving to: ‘ffjpeg-bmp_load-integer-overflow.1’

ffjpeg-bmp_load-integer-overflow. 100%
[============================================================>]      24  --.-KB/s    in 0s

2022-04-01 11:53:24 (1.21 MB/s) - ‘ffjpeg-bmp_load-integer-overflow.1’ saved [24/24]

❯ xxd ffjpeg-bmp_load-integer-overflow.1
00000000: 5f55 5555 5555 553d 3635 7155 5555 5455  _UUUUUU=65qUUUTU
00000010: 5555 6355 5555 0600                      UUcUUU..
❯ ./bin_asan/bin/ffjpeg -e ffjpeg-bmp_load-integer-overflow.1
=================================================================
==2755917==ERROR: AddressSanitizer: allocator is out of memory trying to allocate 0x1555555c00
bytes
    #0 0x4c5587 in calloc /home/ubuntu178/Downloads/llvm12/projects/compiler-
rt/lib/asan/asan_malloc_linux.cpp:154:3
    #1 0x507342 in jfif_encode
/home/ubuntu178/cvelibf/test/ffjpeg/caade60/build_asan/src/jfif.c:749:21
    #2 0x4fc0bf in main /home/ubuntu178/cvelibf/test/ffjpeg/caade60/build_asan/src/ffjpeg.c:33:16
    #3 0x7f54a11b10b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/libc-
start.c:308:16

==2755917==HINT: if you don't care about these errors you may set allocator_may_return_null=1
SUMMARY: AddressSanitizer: out-of-memory /home/ubuntu178/Downloads/llvm12/projects/compiler-
rt/lib/asan/asan_malloc_linux.cpp:154:3 in calloc
==2755917==ABORTING
❯ ASAN_OPTIONS="allocator_may_return_null=1" ./bin_aflpp/bin/ffjpeg -e ffjpeg-bmp_load-integer-
overflow.1
❯ ls
bin_afl    bin_normal  build_aflpp   code        ffjpeg-bmp_load-integer-overflow
bin_aflpp  build.sh    build_asan    decode.bmp  ffjpeg-bmp_load-integer-overflow.1
bin_asan   build_afl   build_normal  encode.jpg  ffjpeg-jfif_load-buffer-overflow
```

```
❯ sha1sum ./ffjpeg-bmp_load-integer-overflow
3bc9c74cee80ff2a5c7c837f3d26abd5d7ae7205  ./ffjpeg-bmp_load-integer-overflow
❯ sha1sum ./ffjpeg-bmp_load-integer-overflow.1
3bc9c74cee80ff2a5c7c837f3d26abd5d7ae7205  ./ffjpeg-bmp_load-integer-overflow.1
```

I am doing research about vulnerability reproduction and analysis. Could you please provide more information, such as how you compile the target program?

Thanks for any reply!

0xdd96 commented on Apr 1 — Author

The complete compilation process is as follows. Feel free to ask if you encountered any problems.

```
user@c3ae4d510abb:$ git clone https://github.com/rockcarry/ffjpeg.git ffjpeg-caade60
Cloning into 'ffjpeg-caade60'...
remote: Enumerating objects: 438, done.
remote: Counting objects: 100% (88/88), done.
remote: Compressing objects: 100% (62/62), done.
remote: Total 438 (delta 52), reused 52 (delta 26), pack-reused 350
Receiving objects: 100% (438/438), 191.18 KiB | 1.86 MiB/s, done.
Resolving deltas: 100% (259/259), done.

user@c3ae4d510abb:$ cd ffjpeg-caade60/
user@c3ae4d510abb:$ ls
LICENSE  Makefile  README  docs  msvc  src

user@c3ae4d510abb:$ vim src/Makefile
CC      = gcc
AR      = ar
- CCFLAGS = -Wall
+ CCFLAGS = -Wall -g -fsanitize=address

user@c3ae4d510abb:$ make -j
src
make -C src
make[1]: Entering directory 'ffjpeg-caade60/src'
gcc -Wall -g -fsanitize=address -o color.o color.c -c
gcc -Wall -g -fsanitize=address -o dct.o dct.c -c
gcc -Wall -g -fsanitize=address -o quant.o quant.c -c
gcc -Wall -g -fsanitize=address -o zigzag.o zigzag.c -c
gcc -Wall -g -fsanitize=address -o bitstr.o bitstr.c -c
gcc -Wall -g -fsanitize=address -o huffman.o huffman.c -c
gcc -Wall -g -fsanitize=address -o bmp.o bmp.c -c
gcc -Wall -g -fsanitize=address -o jfif.o jfif.c -c
gcc    -c -o ffjpeg.o ffjpeg.c
ar rcs libffjpeg.a color.o dct.o quant.o zigzag.o bitstr.o huffman.o bmp.o jfif.o
gcc -Wall -g -fsanitize=address -o ffjpeg ffjpeg.o libffjpeg.a
make[1]: Leaving directory 'ffjpeg-caade60/src'

user@c3ae4d510abb:$ ./src/ffjpeg -e ../ffjpeg-bmp_load-integer-overflow
================================================================
```

```
==16175==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x612000000148 at pc
0x55555555e71e bp 0x7fffffffe190 sp 0x7fffffffe180
READ of size 1 at 0x612000000148 thread T0
    #0 0x55555555e71d in jfif_encode ffjpeg-caade60/src/jfif.c:763
    #1 0x555555556771 in main (ffjpeg-caade60/src/ffjpeg+0x2771)
    #2 0x7ffff73bf0b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x240b2)
    #3 0x55555555656d in _start (ffjpeg-caade60/src/ffjpeg+0x256d)

0x612000000148 is located 0 bytes to the right of 264-byte region [0x612000000040,0x612000000148)
allocated by thread T0 here:
    #0 0x7ffff769abc8 in malloc (/lib/x86_64-linux-gnu/libasan.so.5+0x10dbc8)
    #1 0x555555556c26 in bmp_load ffjpeg-caade60/src/bmp.c:48
    #2 0x55555555673a in main (ffjpeg-caade60/src/ffjpeg+0x273a)
    #3 0x7ffff73bf0b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x240b2)
```

GHTHYS commented on Apr 1

I can't reproduce it either.

ihopenot commented on May 8

this problem can lead to crash in 32 bit program, try compiling with -m32 option to reproduce it.

Marsman1996 commented on May 20 — Contributor

OK, I get it.

This is caused by the different size of `unsigned long`. In 64-bit program, `unsigned long` is 8 bytes while in 32-bit `unsigned long` is 4 bytes.
As a result, the `calloc` in jfif.c:749-751 in 32-bit is always success due to the overflow, which causes the check in jfif.c:752 fail to work.

ffjpeg/src/jfif.c
Lines 747 to 754 in caade60

```
747     jw = ALIGN(pb->width , 16);
748     jh = ALIGN(pb->height, 16);
749     yuv_datbuf[0] = calloc(1, sizeof(int) * jw * jh / 1);
750     yuv_datbuf[1] = calloc(1, sizeof(int) * jw * jh / 4);
751     yuv_datbuf[2] = calloc(1, sizeof(int) * jw * jh / 4);
752     if (!yuv_datbuf[0] || !yuv_datbuf[1] || !yuv_datbuf[2]) {
753         goto done;
754     }
```

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

4 participants