New issue

Jump to bottom

ffjpeg "jfif_decode" function stack-buffer-overflow vulerability #27

Open **yangjiageng** opened this issue on Jul 2, 2020 · 1 comment

yangjiageng commented on Jul 2, 2020

ffjpeg "jfif_decode" function stack-buffer-overflow vulerability

Description:
There is a stack-buffer-overflow bug in jfif_decode(void *ctxt, BMP *pb) function at ffjpeg/src/jfif.c:513:28
An attacker can exploit this bug to cause a Denial of Service (DoS) by submitting a malicious jpeg image.
The bug is caused by the dangerous pointer variable using as follow:
x = ((mcui % mcuc) * mcuw + h * 8) * jfif->comp_info[c].samp_factor_h / sfh_max;
y = ((mcui / mcuc) * mcuh + v * 8) * jfif->comp_info[c].samp_factor_v / sfv_max;
idst = yuv_datbuf[c] + y * yuv_stride[c] + x;
the variable yuv_datbuf is an integer pointer array, but there is no security check before the using of yuv_datbuf (jfif.c: line 513).

We used AddressSanitizer instrumented in ffjpeg and triggered this bug, the output of asan as follow:

==4436==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fff9fa0bfd8 at pc 0x0000004f345f bp 0x7fff9f
READ of size 8 at 0x7fff9fa0bfd8 thread T0
#0 0x4f345e in jfif_decode /root/Downloads/fuzz_code/ffjpeg/src/jfif.c:513:28
#1 0x4ed951 in main /root/Downloads/fuzz_code/ffjpeg/src/ffjpeg.c:24:9
#2 0x7fcbfd63eb96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
#3 0x41b929 in _start (/root/Downloads/fuzz_code/ffjpeg/src/ffjpeg+0x41b929)

Address 0x7fff9fa0bfd8 is located in stack of thread T0 at offset 280 in frame
#0 0x4f120f in jfif_decode /root/Downloads/fuzz_code/ffjpeg/src/jfif.c:378

This frame has 5 object(s):
[32, 160) 'ftab' (line 381)
[192, 208) 'dc' (line 382)
[224, 236) 'yuv_stride' (line 387)
[256, 280) 'yuv_datbuf' (line 389) <== Memory access at offset 280 overflows this variable
[320, 576) 'du' (line 476)
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
(longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /root/Downloads/fuzz_code/ffjpeg/src/jfif.c:513:28 in jfif_decod
Shadow bytes around the buggy address:
0x100073f397a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100073f397b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100073f397c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100073f397d0: 00 00 00 00 00 00 00 00 f1 f1 f1 f1 00 00 00 00
0x100073f397e0: 00 00 00 00 00 00 00 00 00 00 00 00 f2 f2 f2 f2
=>0x100073f397f0: 00 00 f2 f2 00 04 f2 f2 00 00 00[f2]f2 f2 f2 f2
0x100073f39800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100073f39810: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100073f39820: f3 f3 f3 f3 f3 f3 f3 f3 00 00 00 00 00 00 00 00
0x100073f39830: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x100073f39840: 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
==4436==ABORTING

We could clearly observe the stack overflow in jfif_decode function at 0x4f345e, and the variable yuv_datbuf was overflowing.

Lastly, we used GDB to debug this bug, the GDB outputs:

```
Reading symbols from ./ffjpeg...done.
gdb-peda$ set args -d poc_stack.fuzz
gdb-peda$ b * 0x4f345e
Breakpoint 1 at 0x4f345e: file jfif.c, line 513.
gdb-peda$ r
Starting program: /root/Downloads/fuzz_code/ffjpeg/src/ffjpeg -d poc_stack.fuzz
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.

[----------------------------------registers-----------------------------------]
RAX: 0x0
RBX: 0x7fffffffe160 --> 0x7ffff6e90510 (<flush_cleanup>: mov rax,QWORD PTR [rip+0x360379] # 0x7ffff71f0890 <run_fp>)
RCX: 0xffffffffbff --> 0x0
RDX: 0x0
RSI: 0x7474b8 --> 0xfede00 --> 0x1000100 --> 0x0
RDI: 0x7fffffffdff8 --> 0xb40 ('@\x0b')
RBP: 0x7fffffffe3d0 --> 0x7fffffffe4c0 --> 0x50ee40 (<__libc_csu_init>: push r15)
RSP: 0x7fffffffded8 --> 0x4f345f (<jfif_decode+8799>: call 0x4b6cc0 <__asan::__asan_report_load8(__sanitizer::uptr)>)
RIP: 0xffffffffcd4b6cc0
R8 : 0xfede00 --> 0x1000100 --> 0x0
R9 : 0xffffffffc06 --> 0x0
R10: 0x7fffffffe110 --> 0x3f8000003f800
R11: 0xffffffffc1a --> 0x0
R12: 0x1
R13: 0x1fb
R14: 0x0
R15: 0x0
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0xffffffffcd4b6cc0
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffded8 --> 0x4f345f (<jfif_decode+8799>: call 0x4b6cc0 <__asan::__asan_report_load8(__sanitizer::uptr)>)
0008| 0x7fffffffdee0 --> 0x41b58ab3
0016| 0x7fffffffdee8 --> 0x522e48 ("5 32 128 8 ftab:381 192 16 6 dc:382 224 12 14 yuv_stride:387 256 24 14 yuv_datbuf:389 320 256 6 du:476")
0024| 0x7fffffffdef0 --> 0x4f1200 (<jfif_decode>: push rbp)
0032| 0x7fffffffdef8 --> 0x0
0040| 0x7fffffffdf00 --> 0x6110000002c0 --> 0x2c600000200 --> 0x0
0048| 0x7fffffffdf08 --> 0x611000000400 --> 0x42900000300 --> 0x0
0056| 0x7fffffffdf10 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xffffffffcd4b6cc0 in ?? ()
```

We ensured there is a stack overflow bugs, which will be used to finish a DoS attack.

You can reproduce this stack overflow vulnerability by the follow step:
ffjpeg -d PoC_stackoverflow_ffjpeg

yangjiageng mentioned this issue on Jul 2, 2020

ffjpeg "jfif_decode" function heap-overflow vulnerabilities #28

Open

rockcarry commented on Jul 27, 2020 — Owner

lastest code can't reproduce this issue.
please check and test. **@yangjiageng**

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants