ffjpeg "jfif_decode" function heap-overflow vulnerabilities #28

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

yangjiageng commented on Jul 2, 2020

ffjpeg "jfif_decode" function heap-overflow vulnerabilities

Description:
There are two heap-overflow bugs in jfif_decode(void *ctxt, BMP *pb) function at ffjpeg/src/jfif.c: line 544 & line 545
An attacker can exploit this bug to cause a Denial of Service (DoS) by submitting a malicious jpeg image.
We finded the integer pointer array variable yuv_datbuf[] which cannot have bound sanity, so the using of variable yuv_datbuf[] is dangerous.
As the issue 27 (#27) showed, the using of yuv_datbuf[] caused security vulnerabilities.
We tracked the using of yuv_datbuf, and finded two heap-overflow bugs, at ffjpeg/src/jfif.c: line 544 & line 545 :
usrc = yuv_datbuf[2] + uy * yuv_stride[2] + ux;
vsrc = yuv_datbuf[1] + vy * yuv_stride[1] + vx;

We used asan to recognize these vulnerabilities, the output of asan as follow:

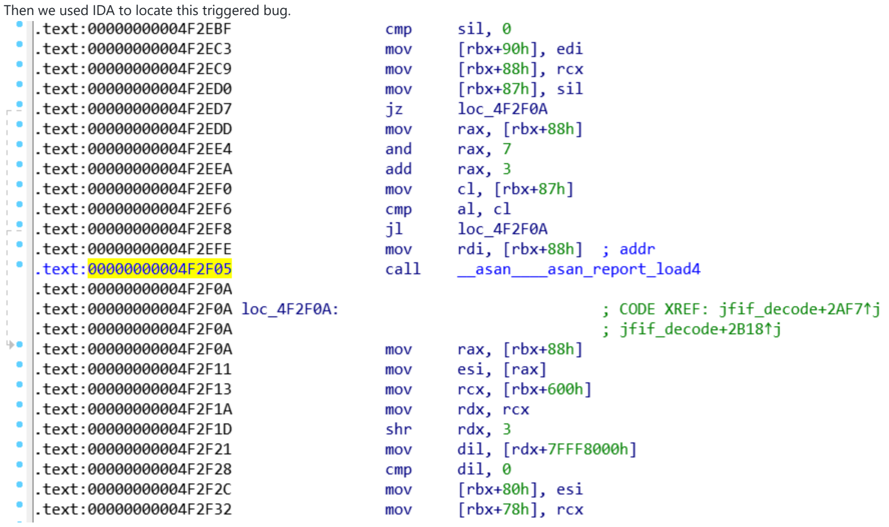
```
==40953==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000010 at pc 0x0000004f2f0a bp 0x7ffecaf45690 sp 0x7ffecaf45688
READ of size 4 at 0x602000000010 thread T0
#0 0x4f2f09 in jfif_decode (/root/ffjpeg/src/ffjpeg+0x4f2f09)
#1 0x4eb545 in main (/root/ffjpeg/src/ffjpeg+0x4eb545)
#2 0x7fbe45680b96 in __libc_start_main /build/glibc-OTsEL5/glibc-2.27/csu/../csu/libc-start.c:310
#3 0x41ac89 in _start (/root/ffjpeg/src/ffjpeg+0x41ac89)

0x602000000011 is located 0 bytes to the right of 1-byte region [0x602000000010,0x602000000011)
allocated by thread T0 here:
#0 0x4a71a0 in malloc /root/llvm-project/llvm/projects/compiler/lib/asan/asan_malloc_linux.cc:145
#1 0x4f1457 in jfif_decode (/root/ffjpeg/src/ffjpeg+0x4f1457)
#2 0x4eb545 in main (/root/ffjpeg/src/ffjpeg+0x4eb545)
#3 0x7fbe45680b96 in __libc_start_main /build/glibc-OTsEL5/glibc-2.27/csu/../csu/libc-start.c:310

SUMMARY: AddressSanitizer: heap-buffer-overflow (/root/ffjpeg/src/ffjpeg+0x4f2f09) in jfif_decode
Shadow bytes around the buggy address:
0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa[01]fa fa fa 01 fa fa fa fa fa fa fa fa fa
0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
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
==40953==ABORTING
```

the asan telled us there is a heap-buffer-overflow on address 0x602000000010 at pc 0x0000004f2f0a bp 0x7ffecaf45690 sp 0x7ffecaf45688

Then we used IDA to locate this triggered bug.



```
.text:00000000004F2EBF                 cmp     sil, 0
.text:00000000004F2EC3                 mov     [rbx+90h], edi
.text:00000000004F2EC9                 mov     [rbx+88h], rcx
.text:00000000004F2ED0                 mov     [rbx+87h], sil
.text:00000000004F2ED7                 jz      loc_4F2F0A
.text:00000000004F2EDD                 mov     rax, [rbx+88h]
.text:00000000004F2EE4                 and     rax, 7
.text:00000000004F2EEA                 add     rax, 3
.text:00000000004F2EF0                 mov     cl, [rbx+87h]
.text:00000000004F2EF6                 cmp     al, cl
.text:00000000004F2EF8                 jl      loc_4F2F0A
.text:00000000004F2EFE                 mov     rdi, [rbx+88h]  ; addr
.text:00000000004F2F05                 call    __asan____asan_report_load4
.text:00000000004F2F0A
.text:00000000004F2F0A loc_4F2F0A:                             ; CODE XREF: jfif_decode+2AF7↑j
.text:00000000004F2F0A                                         ; jfif_decode+2B18↑j
.text:00000000004F2F0A                 mov     rax, [rbx+88h]
.text:00000000004F2F11                 mov     esi, [rax]
.text:00000000004F2F13                 mov     rcx, [rbx+600h]
.text:00000000004F2F1A                 mov     rdx, rcx
.text:00000000004F2F1D                 shr     rdx, 3
.text:00000000004F2F21                 mov     dil, [rdx+7FFF8000h]
.text:00000000004F2F28                 cmp     dil, 0
.text:00000000004F2F2C                 mov     [rbx+80h], esi
.text:00000000004F2F32                 mov     [rbx+78h], rcx
```

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

```
gdb-peda$ b * 0x4f2f09
Breakpoint 1 at 0x4f2f09
gdb-peda$ r
Starting program: /root/ffjpeg/src/ffjpeg -d hh
[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: 0x3
RBX: 0x7fffffffdd20 --> 0x4a6e81 (<__interceptor_free(void*)+193>: test r13d,r13d)
RCX: 0x602000000001 --> 0x102ffffff000000
RDX: 0xc0400000002 --> 0x0
RSI: 0xffffefcb501 --> 0x0
RDI: 0x602000000010 --> 0xbe
RBP: 0x7fffffffe3f0 --> 0x7fffffffe540 --> 0x501c10 (<__libc_csu_init>: push r15)
RSP: 0x7fffffffda78 --> 0x4f2f0a (<jfif_decode+11050>: mov rax,QWORD PTR [rbx+0x88])
RIP: 0xffffffffcd4b4f00
R8 : 0x0
R9 : 0x0
R10: 0x7fffffffd1c0 --> 0x4a71a1 (<__interceptor_malloc(__sanitizer::uptr)+257>: test r13d,r13d)
R11: 0x2
R12: 0x7fffffffe420 --> 0xdd000000dc --> 0x0
R13: 0x80
R14: 0x10007fff7b50 --> 0xf1f1f1f1 --> 0x0
R15: 0x615000000080 --> 0xdd000000dc --> 0x0
EFLAGS: 0x10202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0xffffffffcd4b4f00
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffda78 --> 0x4f2f0a (<jfif_decode+11050>: mov rax,QWORD PTR [rbx+0x88])
0008| 0x7fffffffda80 --> 0x41b58ab3
0016| 0x7fffffffda88 --> 0x5155e8 ("6 32 128 4 ftab 192 16 2 dc 224 12 10 yuv_stride 256 12 10 yuv_height 288 24 10 yuv_datbuf 352 256 2 du")
0024| 0x7fffffffda90 --> 0x4f03e0 (<jfif_decode>: push rbp)
0032| 0x7fffffffda98 --> 0x3a (':')
0040| 0x7fffffffdaa0 --> 0x611000000180 --> 0x58c00000600 --> 0x0
0048| 0x7fffffffdaa8 --> 0x0
0056| 0x7fffffffdab0 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xffffffffcd4b4f00 in ?? ()
```

We ensured there is a heap overflow because of the dangerous using of the int pointer array variable yuv_datbuf
This is the analysis of line 544, and the analysis of line 545 is similar, so we do not dump the detail analysis.
You can reproduce this heap overflow vulnerability by the follow step:
ffjpeg -d PoC_heapoverflow_line544_ffjpeg
ffjpeg -d PoC_heapoverflow_line545_ffjpeg

rockcarry commented on Jul 27, 2020 — Owner

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

Marsman1996 mentioned this issue on Dec 1, 2021

Heap-buffer-overflows in jfif_decode() at jfif.c:552:31 and 552:38 #43

Closed

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants