DOS: Incorrect 802154 Frame Validation for Omitted Source / Dest Addresses

Moderate **ceolin** published **GHSA-94jg-2p6q-5364** on Oct 5, 2021

Package
zephyr (west)

Affected versions	Patched versions
> v2.4.0	v2.5.0

Description

1. Incorrect 802154 Frame Validation for Omitted Source / Dest Addresses

- Bug Description: Improper processing of omitted source and destination addresses in ieee802154 frame validation (ieee802154_validate_frame)
- Bug Result: Value "2" treated as a struct pointer within 802154 logic. More specifically, bytes at addresses 2-8 are endianness swapped (initial stack pointer and reset vector address in case IVT is mapped at address 0). This may change the initial stack pointer and reset vector address on some systems. Depending on the values and memory layout, this might lead to attacker-controlled code being executed during device reset.
- Bug Impact: In most cases, denial of service (crash of firmware). In niche situations, RCE might be achieved by an attacker.

Bug Details

- Affected code: 802154 Frame parsing and validation, starting in subsys/net/l2/ieee802154/ieee802154.c#ieee802154_recv

High-Level reasoning for bug occurrence:

1. Source and Destination addresses are validated in ieee802154_recv->ieee802154_validate_frame->validate_addr
2. Address validation may result in assignment of NULL source and dest addresses
3. Assigning NULL seems to be intended behavior during frame validation
4. However, the potential assignment of NULL is not properly handled while processing DATA frames, and the assignment of the source/dest addresses in set_pkt_ll_addr function may result in the assignment of an invalid pointer in the set_pkt_ll_addr function.
5. This invalid pointer is then used during DATA frame processing in ieee802154_manage_recv_packet->sys_mem_swap, resulting in the use of a corrupted pointer and thus a crash

Vulnerable code path:

1. ieee802154_recv->ieee802154_validate_frame->validate_addr may return NULL pointer
   - This is the case for the IEEE802154_ADDR_MODE_NONE address type and for short packets. For too short packets, any address type can be indicated (including IEEE802154_ADDR_MODE_EXTENDED)
   - As soon as an address with a type other than IEEE802154_ADDR_MODE_NONE and a corrupt pointer is assigned, the following processing logic will wrongly operate on this data
   - Link:

     zephyr/subsys/net/l2/ieee802154/ieee802154_frame.c
     Line 120 in d969ace

     ```
     120       return NULL;
     ```
2. ieee802154_recv->ieee802154_validate_frame treats this as intended behavior, not indicating failed frame validation when NULL is returned for an address of type other than IEEE802154_ADDR_MODE_NONE
   - Consequently, the statement: mpdu->mhr.src_addr = validate_addr(...)
   - May result in a NULL pointer assignment mpdu->mhr.src_addr==NULL
   - Link:

     zephyr/subsys/net/l2/ieee802154/ieee802154_frame.c
     Line 442 in d969ace

     ```
     442       mpdu->mhr.dst_addr = validate_addr(p_buf, &p_buf, &length,
     ```

Bug Manifestation Path a) DATA frames
3. During DATA Frame processing, this NULL value is passed on to set_pkt_ll_addr
- Code: set_pkt_ll_addr(net_pkt_lladdr_src(pkt), mpdu.mhr.fs->fc.pan_id_comp, mpdu.mhr.fs->fc.src_addr_mode, mpdu.mhr.src_addr);
- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154.c
Line 209 in d969ace

```
209       set_pkt_ll_addr(net_pkt_lladdr_src(pkt), mpdu.mhr.fs->fc.pan_id_comp,
```

4. Within set_pkt_ll_addr, the address argument is not checked against NULL
- It is used in the following assignment: addr->addr = ll->plain.addr.ext_addr;
- For ll being a NULL pointer, this results in addr->addr==2, which is a non-NULL, corrupted pointer
- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154.c
Line 110 in d969ace

```
110       addr->addr = ll->plain.addr.ext_addr;
```

5. In further 6LoWPAN processing in ieee802154_manage_recv_packet, the packet containing the corrupted address is then used in case it is not NULL
- The call sys_mem_swap(net_pkt_lladdr_src(pkt)->addr, net_pkt_lladdr_src(pkt)->len) then results in addresses 2-8 being byte-swapped
- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154.c
Line 134 in d969ace

```
134       if (net_pkt_lladdr_src(pkt)->addr &&
```

Bug Manifestation Path b) MAC_COMMAND frames

3. ieee802154_validate_frame->(validate_addr)->validate_payload_and_mfr->validate_mac_command->validate_mac_command_cfi_to_mhr will similarly use the provided destination address, resulting in a crash

- The crash occurs while making sure that the destination is a broadcast statement in the following statement: mhr->dst_addr->plain.addr.short_addr != IEEE802154_BROADCAST_ADDRESS
- This is the case for frames of types IEEE802154_CFI_BEACON_REQUEST or IEEE802154_CFI_COORDINATOR_REALIGNEMENT with NULL addresses in mode IEEE802154_ADDR_MODE_SHORT

Annotated Source Code Snippets

```
struct ieee802154_address_field_plain {
        uint16_t pan_id;
        struct ieee802154_address addr;
} __packed;

struct ieee802154_address_field {
        union {
                struct ieee802154_address_field_plain plain;
                struct ieee802154_address_field_comp comp;
        };
} __packed;

// addr->addr = ll->plain.addr.ext_addr;
// For ll==0, this means that addr->addr=2;



static inline struct ieee802154_address_field *
validate_addr(uint8_t *buf, uint8_t **p_buf, uint8_t *length,
              enum ieee802154_addressing_mode mode,
              bool pan_id_compression)
{
        ...
    if (mode == IEEE802154_ADDR_MODE_SHORT) {
                len += IEEE802154_SHORT_ADDR_LENGTH;
        } else {
                /* IEEE802154_ADDR_MODE_EXTENDED */
                len += IEEE802154_EXT_ADDR_LENGTH;
        }

    // BUG: IEEE802154_ADDR_MODE_EXTENDED and len > *length lead to NULL
        if (len > *length) {
                return NULL;
        }
    ...
}



bool ieee802154_validate_frame(uint8_t *buf, uint8_t length,
                               struct ieee802154_mpdu *mpdu)
{
    ...
    mpdu->mhr.dst_addr = validate_addr(p_buf, &p_buf, &length,
                                       mpdu->mhr.fs->fc.dst_addr_mode,
                                       false);

    // This can be NULL
        mpdu->mhr.src_addr = validate_addr(p_buf, &p_buf, &length,
                                       mpdu->mhr.fs->fc.src_addr_mode,
                                       (mpdu->mhr.fs->fc.pan_id_comp));

    // Bug: Missing check for null pointer assignment of src_addr
    ...
}



static enum net_verdict ieee802154_recv(struct net_if *iface,
                                        struct net_pkt *pkt)
{
    ...
    if (!ieee802154_validate_frame(net_pkt_data(pkt),
                                   net_pkt_get_len(pkt), &mpdu)) {
                return NET_DROP;
        }
    ...
    // BUG: mpdu.mhr.src_addr can be NULL
    set_pkt_ll_addr(net_pkt_lladdr_src(pkt), mpdu.mhr.fs->fc.pan_id_comp,
                        mpdu.mhr.fs->fc.src_addr_mode, mpdu.mhr.src_addr);

        set_pkt_ll_addr(net_pkt_lladdr_dst(pkt), false,
                        mpdu.mhr.fs->fc.dst_addr_mode, mpdu.mhr.dst_addr);
    ...
}



static inline void set_pkt_ll_addr(struct net_linkaddr *addr, bool comp,
                                   enum ieee802154_addressing_mode mode,
                                   struct ieee802154_address_field *ll)
{
        if (mode == IEEE802154_ADDR_MODE_NONE) {
                return;
        }

        if (mode == IEEE802154_ADDR_MODE_EXTENDED) {
                addr->len = IEEE802154_EXT_ADDR_LENGTH;

                if (comp) {
                        addr->addr = ll->comp.addr.ext_addr;
                } else {
            // BUG: for null ll pointer, this mean addr->addr = 2
                        addr->addr = ll->plain.addr.ext_addr;
                }
        } else {
                /* ToDo: Handle short address (lookup known nbr, ...) */
                addr->len = 0U;
                addr->addr = NULL;
        }
        addr->type = NET_LINK_IEEE802154;
}


enum net_verdict ieee802154_manage_recv_packet(struct net_if *iface,
                                               struct net_pkt *pkt,
                                               size_t hdr_len)
{
    ...
        /* Upper IP stack expects the link layer address to be in
         * big endian format so we must swap it here.
         */
    // BUG: this null pointer check is bypassed, as net_pkt_lladdr_src(pkt)->addr==2 in the vuln code path
    // -> This leads to endianness swap of addresses 2-8
        if (net_pkt_lladdr_src(pkt)->addr &&
            net_pkt_lladdr_src(pkt)->len == IEEE802154_EXT_ADDR_LENGTH) {
                sys_mem_swap(net_pkt_lladdr_src(pkt)->addr,
                             net_pkt_lladdr_src(pkt)->len);
        }
```

Proposed Fix

- During address validation in validate_addr, ensure that frame validation fails if no valid address is present outside the IEEE802154_ADDR_MODE_NONE address type.
  - In case the packet is too short to hold the actual address, either change the address mode to IEEE802154_ADDR_MODE_NONE or (probably the better idea) outright drop the packet
  - Link:

    zephyr/subsys/net/l2/ieee802154/ieee802154_frame.c
    Line 120 in d969ace

    ```
    120     return NULL;
    ```
  - This could be implemented as ieee802154_validate_frame returning false in case mpdu->mhr.dst_addr or mpdu->mhr.src_addr are NULL, but not of type IEEE802154_ADDR_MODE_NONE
  - Links:
    - Destination address:

      zephyr/subsys/net/l2/ieee802154/ieee802154_frame.c
      Line 442 in d969ace

      ```
      442     mpdu->mhr.dst_addr = validate_addr(p_buf, &p_buf, &length,
      ```
    - Source address:

      zephyr/subsys/net/l2/ieee802154/ieee802154_frame.c
      Line 446 in d969ace

      ```
      446     mpdu->mhr.src_addr = validate_addr(p_buf, &p_buf, &length,
      ```
  - Alternatively, change the validate_addr prototype to return false for failed address asignments (more specifically, the too-short-packet case), similar to other validation functions
  - Link:

    zephyr/subsys/net/l2/ieee802154/ieee802154_frame.c
    Line 119 in d969ace

    ```
    119     if (len > *length) {
    ```

Fix code for

zephyr/subsys/net/l2/ieee802154/ieee802154_frame.c
Line 442 in d969ace

```
442     mpdu->mhr.dst_addr = validate_addr(p_buf, &p_buf, &length,
```

```
diff --git a/subsys/net/l2/ieee802154/ieee802154_frame.c b/subsys/net/l2/ieee802154/ieee802154_frame.c
m
index 1d479c85d9..56c0b01204 100644
--- a/subsys/net/l2/ieee802154/ieee802154_frame.c
+++ b/subsys/net/l2/ieee802154/ieee802154_frame.c
@@ -443,10 +443,20 @@ bool ieee802154_validate_frame(uint8_t *buf, uint8_t length,
                                           mpdu->mhr.fs->fc.dst_addr_mode,
                                           false);

+       // Proposed fix: Propagate dst address validation
+       if (mpdu->mhr.dst_addr == NULL && mpdu->mhr.fs->fc.dst_addr_mode != IEEE802154_ADDR_MODE_NONE)
 {
+               return false;
+       }
+
        mpdu->mhr.src_addr = validate_addr(p_buf, &p_buf, &length,
                                           mpdu->mhr.fs->fc.src_addr_mode,
                                           (mpdu->mhr.fs->fc.pan_id_comp));

+       // Proposed fix: Propagage src address validation
+       if (mpdu->mhr.src_addr == NULL && mpdu->mhr.fs->fc.src_addr_mode != IEEE802154_ADDR_MODE_NONE)
 {
+               return false;
+       }
```

Patches

master: #31908 (2.5.0)
v2.4.0: TBD

For more information

If you have any questions or comments about this advisory:

- Open an issue in zephyr
- Email us at Zephyr-vulnerabilities

embargo: 2020-04-14
zepsec: ZEPSEC-112

Severity

Moderate 6.5 / 10

CVSS base metrics

Attack vector	Network
Attack complexity	High
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	Low
Availability	High

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:H

CVE ID

CVE-2021-3319

Weaknesses

CWE-476 CWE-588