Integer Underflow in Zephyr in IEEE 802154 Fragment Reassembly Header Removal

High d3zd3z published GHSA-w44j-66g7-xw99 on Oct 12, 2021

Package
zephyr (west)

Affected versions	Patched versions
>=2.4.0	2.5.0

Description

3. Integer Underflow in IEEE 802154 Fragment Reassembly Header Removal

- Bug Description: Incomplete check of minimum IEEE 802154 fragment size leading to an integer underflow.
- Bug Result: Underflown size value is used in a call to memmove, leading to a large out-of-bounds write in a network buffer.
- Bug Impact: At a minimum, the firmware will crash (denial of service). The resulting memory corruption may be exploitable for RCE on the board. The proof-of-concept generated by the fuzzer crashes in the kernel function z_time_slice because of the kernel struct "z_kernel" being corrupted.

Bug Details

- Affected code: IEEE 802154 fragment reassembly logic in subsys/net/l2/ieee802154/ieee802154_fragment.c#fragment_reconstruct_packet

High-Level reasoning for bug occurrence:

1. Initial frame validation in function ieee802154_validate_frame asserts the generic minimum length of IEEE802154_MIN_LENGTH, which only accounts for the initial mpdu header data.
2. For data frames (frame type IEEE802154_FRAME_TYPE_DATA), the (minimum) length of the data payload itself is not validated.
3. Then, fragments get gathered, until the full of fragments has arrived (function ieee802154_reassemble->fragment_add_to_cache->fragment_append)
4. At this point, the fragments are reassembled by the fragment_reconstruct_packet function. To extract only data, the header of each fragment is stripped from the packet. This happens in ieee802154_reassemble->fragment_add_to_cache->fragment_reconstruct_packet->fragment_remove_headers
5. While stripping the header, the size of the header is assumed from looking at the fragment's type, but the actual fragment size itself is not validated.
6. The length of the remaining data payload is then calculated as frag->len - hdr_len.
7. As a result, the following expression leads to an integer underflow and thus a large, out-of-bounds copy operation: memmove(frag->data, frag->data + hdr_len, frag->len - hdr_len);

Vulnerable code path:

1. Missing data payload size validation
   - ieee802154_recv->ieee802154_manage_recv_packet->ieee802154_reassemble->fragment_add_to_cache
   - Link:

     zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
     Line 503 in d969ace

     ```
     503     cache = get_reass_cache(size, tag);
     ```
2. Adding fragments
   - ieee802154_reassemble->fragment_add_to_cache->fragment_append
   - Link:

     zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
     Line 515 in d969ace

     ```
     515     fragment_append(cache->pkt, frag);
     ```
3. Reconstructing packet
   - ieee802154_reassemble->fragment_add_to_cache->fragment_reconstruct_packet
   - Link:

     zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
     Line 471 in d969ace

     ```
     471     fragment_remove_headers(pkt);
     ```
4. Stripping headers from fragments leads to OOB memmove for small data payload sized
   - ieee802154_reassemble->fragment_add_to_cache->fragment_reconstruct_packet->fragment_remove_headers
   - Link:

     zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
     Line 443 in d969ace

     ```
     443     memmove(frag->data, frag->data + hdr_len, frag->len - hdr_len);
     ```

Proposed Fix

- Before adding a fragment to the cache, validate it's size to at least be able to hold it's header
  - NET_6LO_DISPATCH_FRAG1 <-> NET_6LO_FRAG1_HDR_LEN
  - NET_6LO_DISPATCH_FRAGN <-> NET_6LO_FRAGN_HDR_LEN
  - Link:

    zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
    Line 491 in d969ace

    ```
    491     size = get_datagram_size(pkt->buffer->data);
    ```

```
diff --git a/subsys/net/l2/ieee802154/ieee802154_fragment.c b/subsys/net/l2/ieee802154/ieee802154_fragm
ent.c
```

```
index 790c159b56..816888f524 100644
--- a/subsys/net/l2/ieee802154/ieee802154_fragment.c
+++ b/subsys/net/l2/ieee802154/ieee802154_fragment.c
@@ -484,6 +487,7 @@ static inline enum net_verdict fragment_add_to_cache(struct net_pkt *pkt)
        bool first_frag = false;
        struct frag_cache *cache;
        struct net_buf *frag;
+       uint8_t type;
        uint16_t size;
        uint16_t tag;

@@ -494,6 +498,15 @@ static inline enum net_verdict fragment_add_to_cache(struct net_pkt *pkt)
        tag = get_datagram_tag(pkt->buffer->data +
                               NET_6LO_FRAG_DATAGRAM_SIZE_LEN);

+       /* Ensure large enough fragments */
+       type = pkt->buffer->data[0] & NET_FRAG_DISPATCH_MASK;
+       if (!((type == NET_6LO_DISPATCH_FRAG1 &&
+                       pkt->buffer->len > NET_6LO_FRAG1_HDR_LEN) ||
+                 (type == NET_6LO_DISPATCH_FRAGN &&
+                       pkt->buffer->len > NET_6LO_FRAGN_HDR_LEN))) {
+               return NET_DROP;
+       }
+
        /* If there are no fragments in the cache means this frag
         * is the first one. So cache Rx pkt otherwise not.
         */
```

Patches

This has been fixed in:

- main #31908
- v2.4: #33453
- v1.14: NA

For more information

If you have any questions or comments about this advisory:

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

embargo: 2021-04-14
zepsec: ZEPSEC-114

Severity

High 7.5 / 10

CVSS base metrics

Attack vector	Adjacent
Attack complexity	High
Privileges required	None
User interaction	None
Scope	Changed
Confidentiality	Low
Integrity	Low
Availability	High

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

CVE ID

CVE-2021-3321

Weaknesses

CWE-680