# RCE/DOS: Linked-list corruption leading to large out-of-bounds write while sorting for forged fragment list in Zephyr

High d3zd3z published GHSA-fj4r-373f-9456 on Oct 12, 2021

Package

**zephyr** (west)

Affected versions

>=2.4.0

Patched versions

2.5.0

Description

## 1. Improper IEEE 802.15.4 Fragment Reconstruction Sorting For Forged Fragments

- Bug Description: The fragment reassembly logic improperly sorts forged fragment lists that miss a FRAG1 fragment, leading to a network buffer out of bounds write.
- Bug Result: An attacker can forge a fragment list that initially creates a cyclic fragment list, which leads to an integer underflow followed by a large out-of-bounds copy of a network buffer
- Bug Impact: Large out-of-bounds write which leads to a corruption of kernel data structures. This is likely exploitable for remote arbitrary code execution during a context switch, and at least leads to a crash inside the network stack (DoS).

### Bug Details

- Affected code: Fragment reassembly logic, starting in subsys/net/l2/ieee802154/ieee802154_fragment.c

High-Level reasoning for bug occurrence:

1. The IEEE 802.15.4 fragment reassembly logic assumes a sender to send properly formatted fragments, which includes a FRAG1 fragment
2. When all required fragments are collected, the fragments are sorted by its fragment_offset to form the full data packet
3. Building on the assumption of a FRAG1 fragment being present, the sorting logic implicitly assumes that during sorting, the first fragment never needs to be moved during sorting
4. By invalidating this assumption and providing a fragment with a large fragment offset as the first fragment, an attacker can trick the sorting logic into trying to insert a fragment in front of the first fragment, corrupting the list and leading to a cyclic list where a NULL-terminated singly-linked list is expected
5. When the supposed-to-be singly linked list is then iterated, header removal functions are repeatedly applied to the same fragments, leading to an integer overflow in the fragment's size field. This in turn causes a large out-of-bounds memmove operation, corrupting the data section, including important kernel structures.

Conceptual Example:
Consider the following fragment list to be sorted via fragment_reconstruct_packet (

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 450 in d969ace

```
450     static inline void fragment_reconstruct_packet(struct net_pkt *pkt)
```

, also pasted as code snippets below)

```
==== within reassemble ====
Packet @0x20108c04
        frag @0x201096d0 (6 bytes @0x2010c0e7): b'e02404ff4004'
        frag @0x20109700 (7 bytes @0x2010c1e7): b'e7ff7fff049a84'
        frag @0x20109730 (7 bytes @0x2010c2e7): b'e70180ff047f84'
        frag @0x20109838 (7 bytes @0x2010c867): b'e7ff7fff049a84'
        frag @0x20109898 (7 bytes @0x2010ca67): b'e7ff7fff049a84'
        frag @0x20109928 (20 bytes @0x2010cd67): b'e30980ff1b418780064f0040136c0b18ba40104f024004696a02ffd9e30c1501'
```

With the first byte indicating the fragment type (all of them being FRAG_N fragments as they are starting with a 0xe_ byte) fifth byte representing the fragment offset field inside the ieee802.15.4 fragment, this leads to the following iniatial list of fragments with given weights:
frag(0x40) -> frag(0x04) -> frag(0x04) -> frag(0x04) -> frag(0x04) -> frag(0x1b)

1. The logic will iterate starting from the first item
   prev = NULL, curr=frag(0x40), next=frag(0x04)
2. It will assign prev, as !prev holds, resulting in
   prev = frag(0x40), curr=frag(0x04), next=frag(0x04)
3. This will enter fragment_move_back(pkt, frag(0x04), frag(0x40)), but will immediately return as stop==pkt->buffer, resulting in
   prev = frag(0x04), curr=frag(0x04), next=frag(0x04)
4. This will enter fragment_move_back(pkt, frag(0x04), frag(0x04) again. This time, it will compare frag(0x04) to pkt->buffer, which is frag(0x40). This leads to the assignment `frag(0x04)->frags = pkt->buffer`. But as prev is NULL within fragment_move_back, pkt->buffer->frags remains the same. This creates the following circular list:
   frag(0x40) -> frag(0x04) -> frag(0x04)
   ^ ----------------------------|

Vulnerable code path:

1. ieee802154_reassemble->fragment_add_to_cache
   - Fragments are collected until the full size is present
   - Link:

     zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
     Line 517 in d969ace

     ```
     517     if (fragment_cached_pkt_len(cache->pkt) == cache->size) {
     ```

2. ieee802154_reassemble->fragment_add_to_cache->fragment_reconstruct_packet
   - fragment_reconstruct_packet is called to sort the fragments in the correct order
   - Link:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 522 in d969ace

```
522     fragment_reconstruct_packet(pkt);
```

3. ieee802154_reassemble->fragment_add_to_cache->fragment_reconstruct_packet->fragment_move_back

- First iteration: !prev, so assign prev=pkt->buffer
- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 462 in d969ace

```
462     prev = current;
```

- Second iteration: try to sort, but exit because of check pkt->buffer==stop
- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 415 in d969ace

```
415     while (current && current != stop) {
```

- Third iteration: Sort in the third fragment, realize that its fragment offset is smaller than the offset of the first fragment. As prev==NULL, do not update forward link in the process
- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 417 in d969ace

```
417     if (prev) {
```

- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 421 in d969ace

```
421     frag->frags = current;
```

4. ieee802154_reassemble->fragment_add_to_cache->fragment_reconstruct_packet

- Trigger header removal to only retain actual payload bytes
- Link:

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

```
471     fragment_remove_headers(pkt);
```

5. ieee802154_reassemble->fragment_add_to_cache->fragment_reconstruct_packet->fragment_remove_headers

- Assuming the NULL-terminated linked-list data structure, remove headers from each fragment
- Link:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 435 in d969ace

```
435     while (frag) {
```

- As we now corrupted the list to create a cyclic list, this leads to repeated fragment length reductions, until the length underflows to create a large unsigned size value. This leads to a large out-of-bounds memmove operation
- Link length reduction:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 444 in d969ace

```
444     frag->len -= hdr_len;
```

- Link memmove:

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

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

6. This out-of-bounds copy operation now corrupts the data section, which includes kernel structures such as the current thread context

## Annotated Source Code Snippets

```
static void fragment_move_back(struct net_pkt *pkt,
                               struct net_buf *frag, struct net_buf *stop)
{
        struct net_buf *prev, *current;
        prev = NULL;
        current = pkt->buffer;
        while (current && current != stop) {
                if (fragment_offset(frag) < fragment_offset(current)) {
                        if (prev) {
                                prev->frags = frag;
                        }
                        frag->frags = current;
                        break;
                }
                prev = current;
                current = current->frags;
        }
}
static inline void fragment_reconstruct_packet(struct net_pkt *pkt)
{
        struct net_buf *prev, *current, *next;
        prev = NULL;

        current = pkt->buffer;
        while (current) {
                next = current->frags;
                if (!prev || (fragment_offset(prev) >
                              fragment_offset(current))) {
                        prev = current;
                } else {
                        fragment_move_back(pkt, current, prev);
                }
                current = next;
```

```
        }
        /* Let's remove now useless fragmentation headers */
        fragment_remove_headers(pkt);
}
```

## Proposed Fix

The fragmentation logic has to explicitly handle the case in which no FRAG1 fragment is present. This check could be performed just before performing the fragment reassembly logic:
Link:

zephyr/subsys/net/l2/ieee802154/ieee802154_fragment.c
Line 522 in d969ace

```
522     fragment_reconstruct_packet(pkt);
```

If the goal is to be more forgiving during the parsing, the sorting logic could be adapted to cater for the case where the first element is not already sorted into the list correctly and allow the pkt->buffer (which is the pointer to the first net_buf fragment) reference to be updated.

### Patches

This has been fixed in:

- main #31908

### For more information

If you have any questions or comments about this advisory:

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

embargo: 2021-04-20
zepsec: ZEPSEC-118

**Severity**

High 7.1 / 10

**CVSS base metrics**

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

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

**CVE ID**

CVE-2021-3330

**Weaknesses**

CWE-787