



ESPILON
SECURITY RESEARCH

SECURITY ADVISORY

Out-of-bounds write in BTstack's A2DP stream-endpoint discovery

CVE-2026-93015

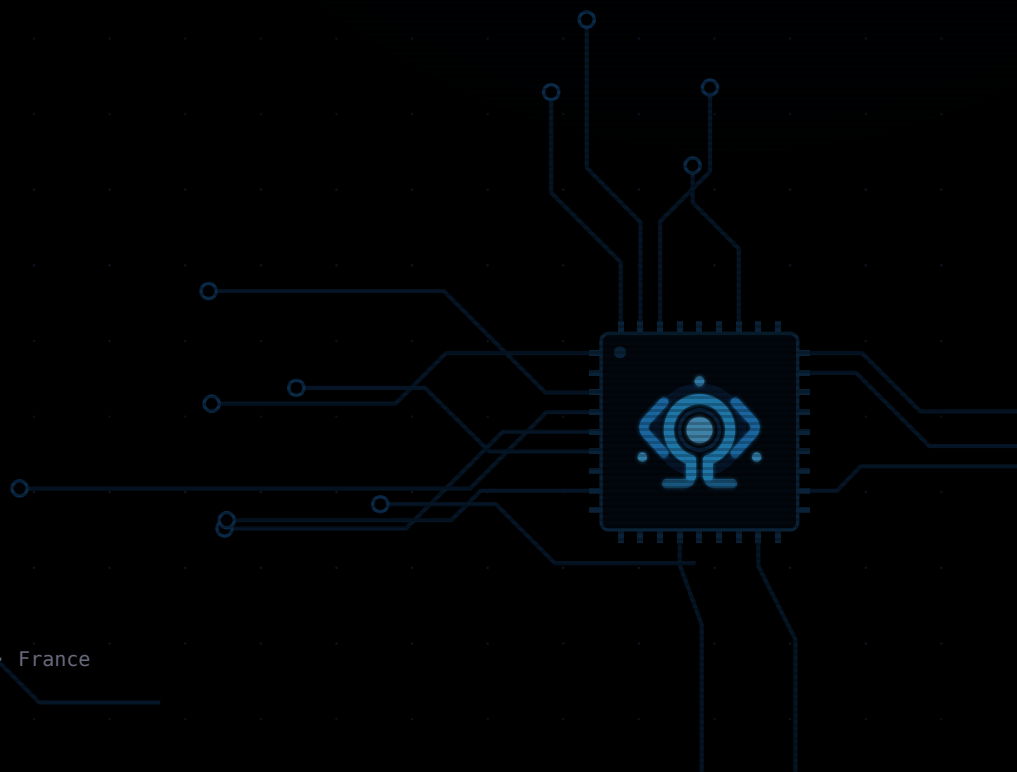
CVSS 7.0 · HIGH

BlueKitchen · BTstack (A2DP)

CVE PUBLISHED 17 Sep 2026

Marouane El Moufid (Eun0us)

espilon.net · Association loi 1901 · France



01 Vulnerability

7.0

HIGH

CVSS 4.0 BASE SCORE · VULNCHECK (CNA)

AV:A / AC:L / AT:N / PR:L / UI:N / VC:N / VI:L / VA:H / SC:N / SI:N / SA:N

CVSS 3.1: 6.3 (Medium) · AV:A / AC:L / PR:L / UI:N / S:U / C:N / I:L / A:H

CVE	CVE-2026-93015
Component	BTstack, src/classic/a2dp.c, A2DP SEP discovery
Affected	BTstack through v1.8.2 (analysed at master 431d58d5613f, 2026-08-18)
Fixed	commit fc208fc ("a2dp: ignore additional stream endpoints"), first tag v1.8.3
Type	Out-of-bounds write (CWE-787) via missing array-index validation (CWE-1284)
Coordination	VulnCheck (CNA); CVE assigned 2026-09-17
Credit	Epsilon · Marouane El Moufid (Eun0us)

Issue

BTstack runs A2DP stream-endpoint (SEP) discovery whenever a peer establishes an AVDTP signalling connection, including on an incoming connection. The peer's DISCOVER response is parsed into a fixed static table of AVDTP_MAX_SEP_NUM (10) entries. The store into that table increments a counter and writes at the resulting index without ever comparing the counter to the table bound, while the number of entries comes from the length of the peer's response. A bonded peer that returns more than ten matching endpoints writes past the end of a 1520-byte global object, by up to roughly 75 KB. The written bytes are essentially zeroes: the result is a loss of availability of the audio profile, not a code-execution primitive.

Timeline

DATE	EVENT
2026-09-06	Reported to VulnCheck (CNA) and to BlueKitchen
2026-09-17	CVE-2026-93015 assigned by VulnCheck
2026-10-23	BTstack v1.8.3 tagged, includes fix fc208fc

Mitigation

Update to BTstack v1.8.3 or apply commit fc208fc, which stops recording endpoints once AVDTP_MAX_SEP_NUM is reached. For a code base that cannot update, bound the store: compare the running counter against AVDTP_MAX_SEP_NUM before writing, and stop recording (or reject the response) once the table is full. The constant is already defined in the file. To keep a working link, ignore endpoints beyond the tenth rather than aborting discovery.

02 Technical analysis

Root cause

The discovered endpoints land in one static table of ten `avdtp_sep_t` entries (152 bytes each, 1520 bytes total). The bound `AVDTP_MAX_SEP_NUM` appears only at the declaration and in the reset `memset`; it is never used in a comparison. The store in the SEP-found handler post-increments its counter and writes unconditionally:

• `src/classic/a2dp.c`, table and the unchecked store (`btstack 431d58d5613f`, abridged)

```
#define AVDTP_MAX_SEP_NUM 10
static uint16_t    a2dp_config_process_sep_discovery_count;
static avdtp_sep_t a2dp_config_process_sep_discovery_seps[AVDTP_MAX_SEP_NUM];

// a2dp.c:550, AVDTP_SUBEVENT_SIGNALING_SEP_FOUND
if ((sep.type == matching_type) && (sep.in_use == false)) {
    a2dp_config_process_sep_discovery_seps[
        a2dp_config_process_sep_discovery_count++] = sep;    // a2dp.c:568
}
```

The only guard before the write is a content filter on the endpoint type and its in-use flag. It decides *which* endpoints are kept, not *how many*.

Where the length comes from

Nothing upstream bounds it either. The DISCOVER response is parsed two bytes at a time, for as long as the packet lasts, and each pair becomes one endpoint. The SEID is range-checked; the endpoint type and in-use flag that decide whether the entry is kept, and the media type, are taken verbatim from the wire.

• `src/classic/avdtp_initiator.c:194`, the DISCOVER response loop (abridged)

```
for (i = offset; i < size; i += 2) {
    if (!avdtp_initiator_have_bytes((uint16_t)i, size, 2u)) break;
    sep.seid      = packet[i] >> 2;          // range-checked 0x01..0x3E
    sep.in_use    = (packet[i] >> 1) & 0x01;  // verbatim from the wire
    sep.media_type = packet[i+1] >> 4;       // verbatim from the wire
    sep.type      = (packet[i+1] >> 3) & 0x01; // verbatim from the wire
    avdtp_signaling_emit_sep(connection->avdtp_cid, sep);
}
```

The loop bound is the signalling packet length, capped only by the AVDTP L2CAP MTU of 1017 bytes. At two bytes per endpoint, an attacker drives about 507 endpoints, i.e. roughly 497 out-of-bounds records of 152 bytes, about 75 KB past the 1520-byte table. Events are delivered synchronously to the store, with no queue and no back-pressure.

Reachability

The A2DP signalling service is registered at BTstack's default security level, which requires an encrypted link, so the attacker is a bonded peer within Bluetooth range. This is not pre-authentication. There is no user interaction beyond the existing pairing: on an incoming connection the victim starts discovery itself and sends the DISCOVER command, so a peer that merely connects makes the victim ask the question whose answer overflows.

Impact, measured rather than inferred

Each out-of-bounds record is a zero-initialised descriptor in which only four low-entropy fields are attacker-influenced, so the overflow zeroes adjacent globals rather than placing chosen values. There is no demonstrated control-flow hijack and no information disclosure. The layout was read from the linked binaries with `nm`, on two compilers at two optimisation levels. In every build the first out-of-bounds record lands on function pointers the library dereferences, with two distinct outcomes:

BUILD	OBSERVED RESULT
clang -00 / -02	AVDTP sink and source callbacks zeroed; event delivery permanently severed, no abort
gcc -00	same silent severing of event delivery
gcc -02	connection-list head zeroed; next lookup returns NULL; btstack_assert aborts at a2dp.c:576

Both outcomes are a loss of availability of the audio profile: the process either aborts, or keeps running while the profile stops responding. Which one occurs is decided by the compiler and flags of the firmware build. The layout of a specific Cortex-M image was not measured; the out-of-bounds write is present in every build that was.

Proof

poc.c links the real upstream sources, with no part of the affected code reimplemented, and drives the genuine AVDTP signalling entry point. AddressSanitizer reports a 152-byte write landing exactly at the end of the 1520-byte table, with the whole call chain inside BTstack frames:

```

• AddressSanitizer: global-buffer-overflow (minimal ll-endpoint trigger)

WRITE of size 152 at 0x... thread T0
#1 a2dp_config_process_avdtp_event_handler      src/classic/a2dp.c:568
#2 a2dp_sink_packet_handler_internal           src/classic/a2dp_sink.c:167
#3 avdtp_emit_sink_and_source                  src/classic/avdtp.c:97
#4 avdtp_signaling_emit_sep                    src/classic/avdtp_util.c:657
#5 avdtp_initiator_handle_general_connection_responses avdtp_initiator.c:205
#6 avdtp_initiator_stream_config_subsm         src/classic/avdtp_initiator.c:452
#7 handle_l2cap_data_packet_for_signaling_connection avdtp.c:603
#8 avdtp_packet_handler                        src/classic/avdtp.c:886

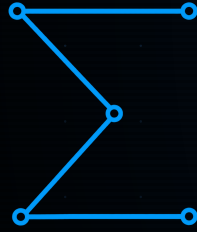
0x... is located 0 bytes after global variable
'a2dp_config_process_sep_discovery_seps' (a2dp.c:76) of size 1520

```

A negative control carrying exactly ten endpoints runs to completion with no report, so the harness discriminates rather than crashing on anything. The overflow also reproduces without a sanitizer in the gcc -O2 build, as the assertion abort.

Why the upstream fuzzer does not reach it

BTstack ships an AVDTP fuzz harness (test/fuzz/fuzz_avdtp.c). It installs empty callbacks at the AVDTP layer and never initialises A2DP or registers the A2DP packet handler, so the handler containing the defect is never on the event path. The sink is structurally outside the fuzzed surface no matter how long a campaign runs. Registering the A2DP layer the way a real application does is what makes it reachable.



ESPILON

SECURITY RESEARCH

Embedded, IoT and OT security research

espilon.net

eun0us@espilon.net

Association loi 1901 • France