Automotive CAN Protocol Libraries Multiple Buffer Overflows (CVE-2026-37534 through 37541, 42467-42469)

`advisory_automotive_v2.md`

Automotive CAN Protocol Libraries — Multiple Buffer Overflow Vulnerabilities

Reporter: Feng Ning, Innora Security Research (feng@innora.ai) **Disclosure**: 2026-04-30

CVE	Library / Product	Version	Type	CVSS
CVE-2026-37534	Open-SAE-J1939	≤ b6caf884	Integer Underflow → OOB Write	9.8
CVE-2026-42467	Open-SAE-J1939	≤ b6caf884	Stack Buffer Overflow	9.8
CVE-2026-37535	openxc/isotp-c	≤ 5a5d1924	OOB Read	7.5
CVE-2026-37536	miaofng/uds-c	commit e506334e	Stack Buffer Overflow	9.8
CVE-2026-37537	collin80/Open-SAE-J1939	≤ 744024d4	Integer Overflow	9.8

CVE	Library / Product	Version	Type	CVSS
CVE-2026-37538	socketcand	0.4.2	Stack Buffer Overflow	9.8
CVE-2026-37539	cannelloni	2.0.0	Heap Buffer Overflow (CAN frame parsing)	9.8
CVE-2026-37540	OpenAMP	2025.10.0	Integer Overflow — ELF loader	9.8
CVE-2026-37541	OVMS3	3.3.005	Stack Overflow — GVRET format	8.8
CVE-2026-42468	OVMS3	3.3.005	Stack Overflow — PCAP format	8.8
CVE-2026-42469	OVMS3	3.3.005	Stack Overflow — CANswitch format	8.8

Eleven vulnerabilities across eight libraries. All of them are deeply embedded in automotive ECUs, CAN bus gateways, and vehicle diagnostic tooling — production vehicles, industrial networks, research platforms.

CVE-2026-37534 / CVE-2026-42467 — Open-SAE-J1939

J1939 message parsing botches the `size - overhead` calculation when a malformed frame is received. The subtraction wraps, and the wrapped value drives out-of-bounds memory operations. Separately — same codebase, same commit boundary — there's a second independent stack buffer overflow tracked as CVE-2026-42467.

Attack vector: a crafted CAN frame, delivered over the network or directly on the local bus.

CVE-2026-37535 — openxc/isotp-c

ISO 15765-2 multi-frame reassembly in `isotp-c` has no check that the cumulative payload fits inside the receive buffer before writing frame data. Feed it an oversized sequence of ISO-TP frames and you get an out-of-bounds read. Every frame in the sequence is processed without accounting for what's already been deposited.

CVE-2026-37536 — miaofng/uds-c

Inside the UDS (ISO 14229) request builder in `uds.c`, service data gets `memcpy`'d into a fixed-size stack buffer. The length argument comes straight from the request structure — attacker-controlled, and there's no bounds check anywhere before the copy executes.

CVE-2026-37537 — collin80/Open-SAE-J1939

Packet length calculation overflows, producing a value small enough to pass allocation but large enough to blow past the buffer in the subsequent `memcpy`. Heap corruption follows.

CVE-2026-37538 — socketcand

In `socketcand.c`, `main()` copies a CAN frame identifier string into a stack buffer. Length is never validated. A sufficiently long CAN ID runs straight off the end.

CVE-2026-37539 — cannelloni

`parseCANFrame()` in `parser.cpp` handles CAN frames tunneled over UDP, TCP, or SCTP. Both the frame count and per-frame data length come from the attacker. Neither is bounds-checked before frame data is written to the receive buffer. Network-reachable, no authentication required, CVSS 9.8.

CVE-2026-37540 — OpenAMP ELF Loader

OpenAMP 2025.10.0 loads remote processor firmware via ELF images. When processing a `PT_LOAD` segment, the size calculation for the destination region uses 32-bit arithmetic on fields pulled directly from the ELF header. The multiplication overflows before the result is used to gate the `memcpy` length check. A crafted firmware image corrupts host memory during the load sequence.

CVE-2026-37541 / CVE-2026-42468 / CVE-2026-42469 — OVMS3

Three separate CAN log parsers in Open Vehicle Monitoring System 3 v3.3.005, each with the same class of bug:

- **CVE-2026-37541** — `canformat_gvret.cpp`, GVRET log format
- **CVE-2026-42468** — `canformat_pcap.cpp`, PCAP/pcapng format
- **CVE-2026-42469** — `canformat_canswitch.cpp`, CANswitch log format

In every case, a field length from the log file header drives a `memcpy` into a fixed-size stack buffer. No upper-bound check. Malformed log files trigger the overflow.

Innora Security Research — https://innora.ai