Alberto Penzo 2109317 2025/2026

CAN Bus-off attack report

Objectives

This project implements and evaluates a discrete-time software simulation of the CAN Bus-Off attack, a denial-of-service technique that exploits the error-handling mechanism to forcibly disconnect a target ECU from the CAN bus.
The specific objectives are:
- implement a faithful model of the ISO 11898-1 error-handling rules.
- Simulate the attacker's core capability: mirroring a victim's frame ID with corrupted data to provoke collisions and resetting the attacker's own TEC to zero after each collision using a custom controller.
- Measure and characterise the attack duration as a function of the victim's frame transmission period, and compare the asymmetric scenario (attacker TEC reset) against the symmetric one (neither TEC resets).

System Setup

No physical hardware or external simulation tools are used. The entire software is a custom-built discrete-time simulator written in C99, compiled and executed on a Linux host.

Simulation Architecture

The simulator is organised as independent modules, each with a dedicated header (.h) and implementation (.c) file:

- Frame module: defines the CAN frame struct (11-bit ID, 1-byte dlc, up to 8 data bytes) and the helper for registering periodic frames with an ECU.
- ECU module: models a single CAN node: TEC/REC counters, state machine (ERROR_ACTIVE / ERROR_PASSIVE / BUS_OFF), send/listen/outcome logic and attacker specific TEC reset.
- CANBus module: models the shared medium, arbitration, collision detection, and per-tick state management.
- Main module: orchestrate the simulation loop, CLI argument passing, CSV logging and test execution.

Simulation

The simulated bus carries exactly two nodes:

Victim ECU: a standard CAN node with no ability to modify its error counters. It transmits three periodic frames. The frame periods are multiples of a configurable base period p, set via the -p command-line flag (default p = 10 ticks).
Attacker ECU: a custom node that monitors the bus each tick, detects when the victim is transmitting, and immediately injects a jam frame carrying the same ID but with data[0] XOR'd with 0xFF. After each collision it reset its TEC to zero (this behaviour can be disabled with the -r flag), simulating a custom CAN controller that directly manipulates its internal TEC register.

Simulation Tick Model

Time advances in discrete ticks, each representing one complete frame transmission slot. Within each tick the following sequence executes deterministically:
Victim tick: the victim inspects its frame list and calls send() on any frame whose period divides the current tick number. If multiple frames are due, the one with the lowest ID is chosen.
Attacker tick: if the victim is currently transmitting, the attacker reads the victim's current_msg, copies its ID and DLC, corrupts data[0] ^= 0xFF, and calls send().
Bus tick: bus_process_tick() identifies the winning frame (lowest ID), then checks for a collision among nodes sharing that ID by comparing DLC and data with memcmp().
Outcome processing: each transmitting node calls check_transmission_outcome(), TEC +8 on collision, TEC −1 on clean success and updates its state via update_ecu_state().
Attacker TEC reset (optional): if collision_detected is true and the attacker was transmitting and the -r flag is set, attacker_reset_tec() sets the attacker's TEC to 0.
Listener update: non-transmitting nodes call listen(), REC +1 on collision, REC −1 on success.

Experiment 1 - Attack with TEC Reset vs Without

The Bus-Off attack requires asymmetry: the attacker must be able to keep its TEC at zero while the victim's TEC climbs. Without this capability, both nodes should reach BUS_OFF simultaneously, which constitutes a partial success (the victim is silenced) but also a self-defeating outcome (the attacker is also silenced).

The simulation is run twice with the default base period (p = 10) and MAX_TICKS = 10,000:
Run A (-r flag absent): attacker resets its TEC to 0 after every collision.
Run B (-r flag set): neither node resets its TEC; both accumulate errors symmetrically.
In both runs, the victim transmits frames 0x101 (period 10), 0x102 (period 20), and 0x103 (period 50). The attacker jams every transmission it detects. The TEC of both nodes and the victim's state are logged each tick.

The attack is successful in both runs in the sense that the victim reaches BUS_OFF at exactly tick 310. The difference lies in the attacker's fate. In Run A, the attacker's TEC is reset to zero after each of the 32 collisions and remains at zero throughout, leaving the

attacker in ERROR_ACTIVE at the end free to continue operating on the bus or to launch the same attack against another victim.
In Run B, the attacker's TEC mirrors the victim's exactly, because every collision increments both nodes' TEC by +8 and neither node ever achieves a successful transmission that would earn a −1 decrement. Both nodes reach TEC = 256 at tick 310 and enter BUS_OFF simultaneously.

Experiment 2 - Attack duration vs Victim Frame Period

Attack duration (measured in ticks to BUS_OFF) should scale linearly with the base frame period p, because the collision rate is inversely proportional to the period of the victim's shortest frame (0x101, period p). A longer period means fewer collisions per unit time, so more time is needed to accumulate the 256 TEC increments required for BUS_OFF.

The simulation is run with TEC reset enabled and the -n flag (no log, for speed) all over integer periods from 1 to 100: $p \in \{1, 2, \ldots, 100\}$. The ticks-to-BUS_OFF value is captured from the TICKS_TO_BUS_OFF output line.

The relationship is perfectly linear: ticks = 31 × p in all cases.
The linear relationship confirms that the attack duration is entirely determined by the victim's shortest frame period. An automotive ECU transmitting at a typical period of 10 ms would require only 310 ms to be silenced. Safety-critical ECUs which may transmit at 10–5 ms periods, could be taken offline in under 200 ms.

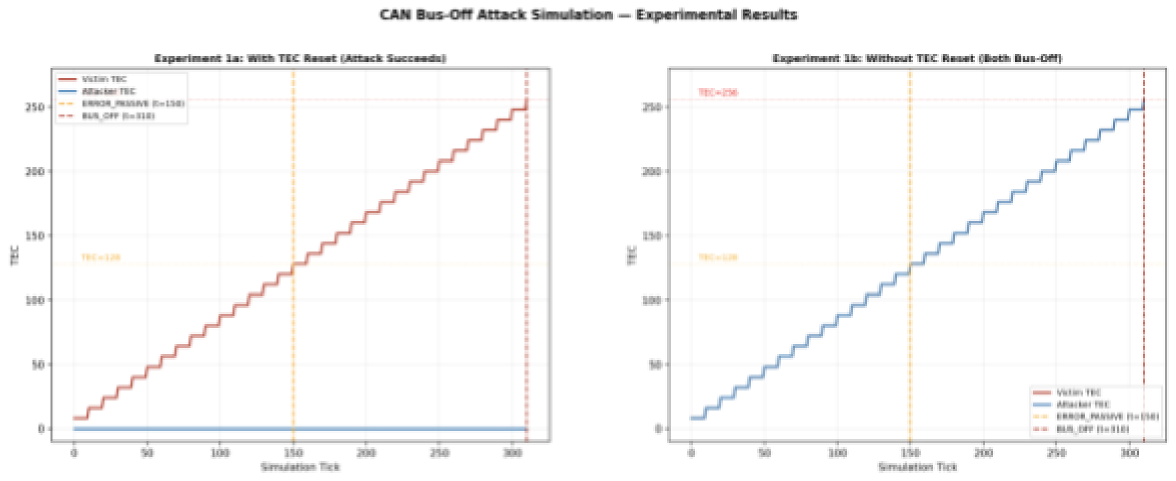

Figure 1. TEC progression with TEC reset enabled vs TEC reset disabled. In both cases the victim ECU reaches Bus-Off state.

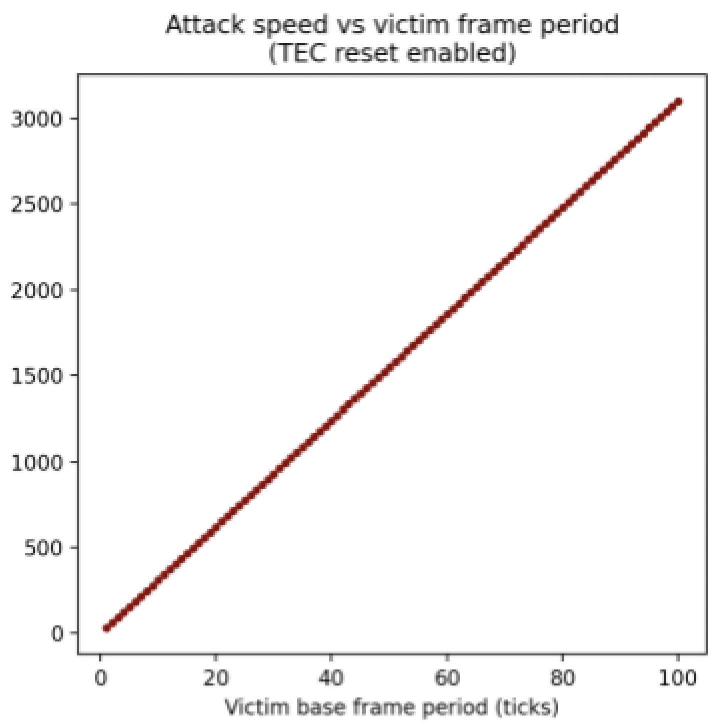

Figure 2. Ticks to Bus-Off vs base period. The relationship is perfectly linear (y=31x).

Reference paper

Error Handling of In-vehicle Networks Makes Them Vulnerable