Padua (PD)
29th December 2025

UNIVERSITÀ DEGLI STUDI DI PADOVA

**CYBER-PHYSICAL SYSTEMS AND IOT SECURITY**
**A.Y 2025/2026**

---

# Report 1 for the Course on Cyber-Physical Systems and IoT Security

**Riccardo Scalco (2155352)**

**Reference Paper:** WeepingCAN: A Stealthy CAN Bus-off Attack

---

**Source Code:** https://github.com/richis02/Report-1- - -Cyber-Physical-Systems

## 1 Objectives

The primary objective of this report is to replicate the WeepingCAN attack logic described in the reference paper using a custom Python simulator. Specifically, the objectives are:

- To simulate the node state transitions (Error Active → Error Passive → Bus Off) within the CAN protocol.
- To implement the distinctive WeepingCAN mechanisms:
  - disabling automatic retransmissions;
  - injecting recessive bits to induce errors;
  - eliminating the need for fabricated preceded messages;
  - performing randomized bit error injection.
- To demonstrate how the skipping attack strategy allows the attacker to recover their TEC and evade detection. In particular, the aim is to validate the following stability condition:

$$\exists\, v \in M_{\mathcal{V}} \; s.t. \; 8 > \sum_{m' \in M_{\mathcal{V}}} \frac{v_T}{m'_T} < \sum_{m \in M_{\mathcal{A}}} \frac{v_T}{m_T}$$

## 2 System Setup

To validate the proposed attack, I developed a custom Discrete Event Simulator in Python. The simulation environment models a simplified CAN Bus connecting two ECUs: the *Victim* and the *Attacker*. The software architecture is structured as follows:

- The class `CANFrame` encapsulates the CAN message structure, focusing specifically on the `id` and `data` fields; I omitted the other standard CAN fields for simulation simplicity. It provides the following methods:
  - `to_bit_string()`: converts the frame fields into a binary string representation.
  - `create_attack_frame(victim_frame)`: generates the attack payload by identifying a dominant bit ('0') in the victim's frame and flipping it to recessive ('1'). If the target payload consists entirely of recessive bits, the method returns an impossibility flag, as the attack cannot be executed.

- The class `CANNode` models a generic ECU. It maintains internal state variables (`tec`, `state`) and the transmission interval (`period`). Key methods include:
  - `event_transmit_success(), event_transmit_error()`: manage the Transmit Error Counter (TEC) logic, handling increments (+8) upon error and decrements (-1) upon successful transmission.
  - `_update_state_machine()` (private): enforces the CAN protocol state transitions (Error Active → Error Passive → Bus-off) based on the current TEC value.
- The class `WeepingAttacker`, which inherits from `CANNode`, implements the specific attack logic. It introduces the following attributes:
  - `target_period`: stores the estimated transmission interval of the victim node.
  - `skip_strategy`: defines the attack frequency (i.e., the number of cycles to skip between injections) to optimize stealth.

  The primary methods are:
  - `sync_with_bus(...)`: determines synchronization status with the victim. While less critical in a discrete-event simulation compared to a physical deployment, it models the "learning phase" of the attack.
  - `predict_window(...)`: predicts the precise time window for the victim's next transmission based on previous observations.
  - `attempt_attack(victim_frame)`: executes the core attack decision logic, determining whether to inject an error or apply the skipping strategy for the current cycle.
  - `recover_tec()`: simulates the transmission of legitimate background traffic to decrement the attacker's TEC (recovery phase), assuming the availability of valid messages.

# 3 Experiments

The previously described classes and methods enable a complete simulation of the experiment outlined in Section IV.D of the reference paper. Additionally, a `SimulationConfig` class was introduced to efficiently manage the simulation environment. Modifying the parameters within this class allows for the customization of key experimental variables.
The core execution is handled by the `run_simulation(...)` method, which orchestrates the entire process. Inside this method, both the Attacker and Victim nodes are initialized with their respective parameters. The Victim's CAN ID is fixed, under the assumption that the Attacker has obtained this information via prior analysis. The Victim's message payload is generated randomly at the start of the simulation but remains constant throughout the execution. Conversely, the specific bit position targeted by the Attacker's injection is randomized for each attack attempt. All simulation events are logged into a CSV file for subsequent analysis.

| Time_ms | Victim_TEC | Attacker_TEC | Victim_State | Action | Bus_Status |
|---|---|---|---|---|---|
| 87 | 0 | 0 | ERROR_ACTIVE | IDLE | IDLE |
| 88 | 0 | 0 | ERROR_ACTIVE | IDLE | IDLE |
| 89 | 0 | 0 | ERROR_ACTIVE | IDLE | IDLE |
| 90 | 8 | 8 | ERROR_ACTIVE | ATTACK_COLLISION | VICTIM_TX |
| 91 | 7 | 8 | ERROR_ACTIVE | VICTIM_RETRANSMIT_SUCCESS | VICTIM_TX |
| 92 | 7 | 7 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 93 | 7 | 6 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 94 | 7 | 5 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 95 | 7 | 4 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 96 | 7 | 4 | ERROR_ACTIVE | IDLE | IDLE |
| 97 | 7 | 4 | ERROR_ACTIVE | IDLE | IDLE |
| 98 | 7 | 4 | ERROR_ACTIVE | IDLE | IDLE |
| 99 | 7 | 4 | ERROR_ACTIVE | IDLE | IDLE |
| 100 | 6 | 4 | ERROR_ACTIVE | SKIPPING_TX | VICTIM_TX |
| 101 | 6 | 3 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 102 | 6 | 2 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 103 | 6 | 1 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 104 | 6 | 0 | ERROR_ACTIVE | ATTACKER_RECOVERY | ATTACKER_BG_TX |
| 105 | 6 | 0 | ERROR ACTIVE | IDLE | IDLE |

Figure 1: Example of file csv

For the experimental parameters, a Victim transmission period of 10ms was assumed. In the event of an attack (e.g., occurring at $t = 30$ms), the execution sequence proceeds as follows:

a. At $t = 30$, the Victim attempts to transmit, and the Attacker performs the bit-flip injection. Consequently, the TEC of both nodes increases.

b. At $t = 31$, the Victim successfully retransmits the message (as the Attacker disables retransmission).

c. In the subsequent time slots (from $t = 32$ to $t = 32 + n$), the Attacker transmits $n$ legitimate recovery messages to restore its TEC (up to a maximum of 8 messages). Various values for $n$ were tested, and the results are discussed in the following section.

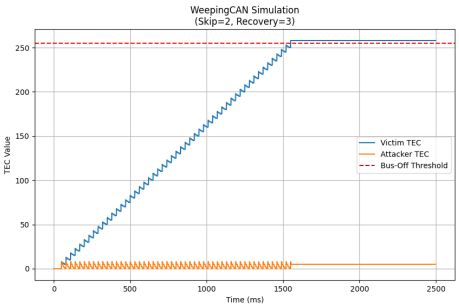
Alternatively, when the Attacker employs the *Skipping Strategy*, the Victim's message is transmitted successfully without interference. However, the Attacker still executes the TEC recovery phase during these idle slots to ensure its error counter remains low.
In the case of an attack, the output console shows the information of the victim's and attacker's frames. It also indicates the point of collision.

```
Collision Analysis @ Bit 30
Victim:          00000010000110100000010110110101000111101111001111010011000101010000005010 01
Attacker:        00000010000110100000010110110111000111101111001111010011000101010000005010 01
Collision Point:                                    ^
Result: Attacker detects Bit Error -> Sends Error Flag.
```

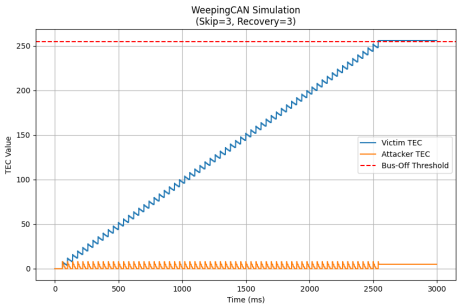
Figure 2: Output Console Example

# 4 Results and Discussion

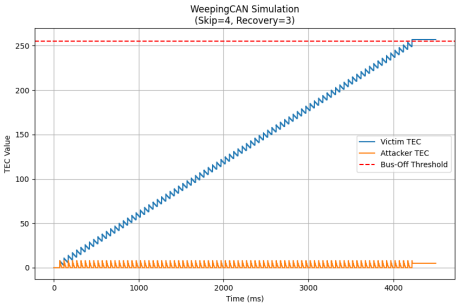
For all the subsequent results, I set the recovery value to 3. Then, I test the Weeping Attack with a skipping value from 2 to 7. I report all the graphs:



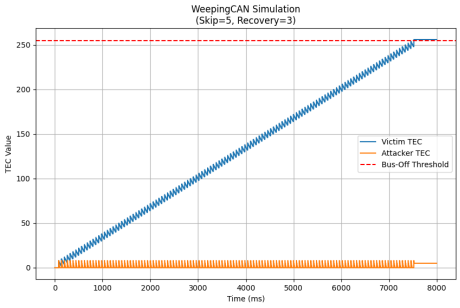

(a) SKIP=2



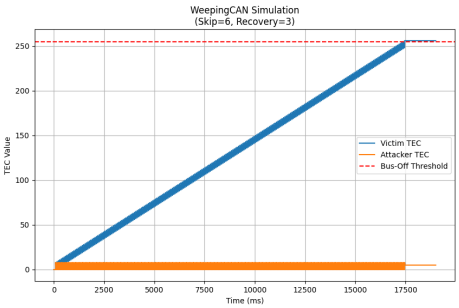

(b) SKIP=3



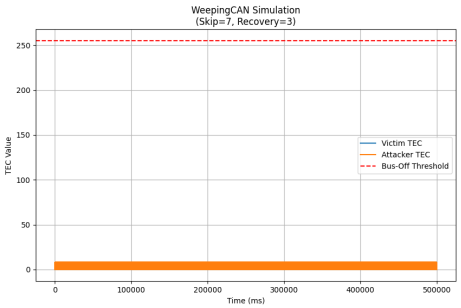

(c) SKIP=4




(d) SKIP=5




(e) SKIP=6




(f) SKIP=7

The simulation experiments provided a comprehensive validation of the WeepingCAN attack mechanics, confirming the theoretical models presented in the reference paper. The results obtained from the simulator highlight the relationship between the *Skipping Strategy*, the attacker's stealth, and the effectiveness of the Bus-off induction.

**Attack Effectiveness and Aggressive Strategy**
In the aggressive configuration ($k = 2$, Skip 2), the simulation successfully forced the Victim node into the *Bus-off* state ($TEC > 255$) in approximately 1500 ms. The graph shows a sharp increase in the Victim's TEC, thanks to the frequent injection of recessive bit errors.

**Stealth Optimization and Recovery**
Increasing the skip interval to $k = 6$ (Skip 6) drastically changed the attack dynamics. The time required to disable the Victim increased to approximately 17500 ms.

**Operational Boundaries and Failure Case ($k = 7$)**
To experimentally determine the upper bound of the attack's effectiveness, I simulated a scenario with a Skip strategy of $k = 7$. In this configuration, the attack **failed** to force the Victim into the Bus-off state. The Victim's TEC exhibited an oscillatory behavior without a growing trend. This result is mathematically consistent with the CAN fault confinement rules: the Victim accumulates a net +7 TEC during the attack phase (due to the error and the successful retransmission) but decrements its TEC by exactly 7 during the 7 skipped periodic cycles ($7 \times -1$). This results in a net change of zero ($\Delta TEC \approx 0$), creating a stable equilibrium where the Victim survives indefinitely. This finding experimentally verifies the theoretical limit of the WeepingCAN strategy against a standard periodic node.

**Validation of Stability and Stealth Features**
The experiments validated the stability condition described in Equation 1 of the paper. We confirmed that the attack is sustainable only when the Attacker's recovery rate (background traffic) is sufficient to neutralize the error penalty during the skip phase.