CANter: data-link layer detection of drop-and-spoof attacks on CAN and CAN FD

Stefano Longari[1,*], Gregorio Galletti[1], Jan Holle[2] and Stefano Zanero[1]

[1]*Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Via Ponzio 34/5, 20133, Milan, Italy*
[2]*ETAS GmbH, Borsigstraße 24, 70469 Stuttgart, Germany*

Abstract

The automotive industry has experienced significant growth and innovation in recent years, resulting in increasingly complex in-vehicle networks and a higher number of Electronic Control Units (ECUs) on-board. Communication between ECUs is primarily accomplished using the Controller Area Network (CAN) protocol, which is considered the standard in the industry due to its reliable and efficient transmission. However, security concerns have arisen due to the fact that these protocols were initially designed without much consideration for security, leaving vehicles vulnerable to attackers who can transmit spoofed messages on the bus. Current Intrusion Detection Systems (IDSs) detect these messages, but an attacker can avoid defenses through drop-and-spoof attacks, by first disconnecting the target ECU and then spoofing the messages.

In response to these vulnerabilities, this paper presents CANter , an IDS designed to detect and respond to attacks on in-vehicle networks. CANter utilizes the CAN and CAN FD specifications, as well as a frequency analysis of time intervals between frames on the network, to detect drop-and-spoof attacks without requiring any modifications to the existing network structure or ECUs, being installed as a stand-alone device on the network.

Keywords

Controller Area Network, Intrusion Detection System, Error Frame Attacks

1. Introduction

CAN is the de-facto standard network protocol for automotive on-board communication. It is a reliable, medium-high bandwidth serial bus communications protocol, and currently, the automotive industry is also adopting an updated version of CAN, the CAN FD protocol. However, the lack of security features in CAN and CAN FD design poses significant concerns [1], and defense techniques, such as secure network architectures [2], authentication protocols [3, 4], and IDSs [5, 6], have been proposed to address them. However, these techniques may not be suitable for all ECUs and may be evaded by knowledgeable attackers, e.g., thanks to drop-and-spoof attacks: such attacks exploit the fault confinement protocol of CAN and CAN FD, designed to isolate malfunctioning ECUs from the rest of the network to prevent them from causing further problems; When an ECU generates too many errors, it is considered to be faulty and is temporarily removed from the bus by the protocol. However, an attacker can exploit this by intentionally causing errors on the network [7]: the attacker sends sequences of bits over messages being sent by other ECUs. These sequences are designed to trigger the fault confinement protocol, causing the targeted ECU to be incapable of communicating on the network. At this point the attacker can send spoofed messages impersonating the target ECU without being detected by standard IDSs, potentially disrupting the functionality of the vehicle.

To detect these types of attacks, which as shown by many researchers [8] can lead to safety-critical outcomes, in this paper we present CANter , a reliable IDS against drop-and-spoof attacks for CAN and CAN FD networks. CANter monitors the traffic and keeps track of the error counters of ECUs, detecting nodes' disconnections and consequent spoofed messages. Our approach is inspired by prior work [9], solving its vulnerabilities, extending it to CAN FD, and providing additional frequency-based

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT
[*]Corresponding author.
stefano.longari@polimi.it (S. Longari); gregorio.galletti@mail.polimi.it (G. Galletti); jan.holle@etas.com (J. Holle); stefano.zanero@polimi.it (S. Zanero)
0000-0002-7533-4510 (S. Longari)

analysis to detect attacks. Our frequency analysis performed on CAN frames represents a further security feature that leads CANter to detect also novel implementations of drop-and-spoof attacks in the literature [10, 11, 12, 13]. Furthermore, our IDS only requires the installation of the monitoring unit to the existing network, without modifying the ECUs or the network topology as required by other solutions.

Our contributions can be summarized as follows:

- We introduce CANter , an IDS against drop-and-spoof attacks that monitors the traffic and keeps track of the error counters of ECUs to detect disconnections and subsequent spoofed messages.
- We design a frequency analysis algorithm specifically tailored towards detecting attackers attempting to evade our IDS.
- We present a proof-of-concept implementation of our IDS on a real-world CAN network, demonstrating its detection capabilities against drop-and-spoof attacks.

2. Primer on CAN

In this section we present the elements of the CAN standard that are necessary for the understanding of the approach. We refer the interested reader to the CAN standard documentation [14]. The CAN specification standardized in ISO 11898 [14] defines the data link layer and a portion of the physical layer. The physical layer defines *dominant* (logical value 0) and *recessive* (logical value 1) bits. A dominant bit state overrides a recessive one.

Data frames., This type of frame carries actual data from the transmitting node to one or more receiving nodes. The length of the data can range from 0 to 8 bytes, and it is identified by an 11-bit or 29-bit identifier (ID). To simplify the discussion, we will only mention the relevant fields of an 11-bit CAN frame:

- *Start of Frame (SOF):* marks the beginning of Data frames, represented by a single dominant bit.
- *Identifier (ID):* composed of 11 (standard) or 29 (extended) bits, frame IDs must be unique.
- *Data:* up to 64 bits of application data may be transmitted. The data field is encoded with vendors dependent rules that can be very different between vehicles.
- *Cyclic Redundancy Check (CRC):* the CRC sequence is calculated using as coefficients the de-stuffed bit stream until this field. The CRC delimiter is a recessive bit to signal the end of the sequence.
- *Acknowledgement (ACK):* a transmitting node sends a recessive bit. Receiving nodes send a dominant bit during the ACK slot, overwriting the recessive bit and acknowledging correct frames.
- *End of Frame (EOF):* composed by seven recessive bits.

Arbitration. is handled through CSMA/BA (Carrier Sense Multiple Access / Bitwise Arbitration), a non-destructive bitwise arbitration method. This arbitration takes place without corruption or delay of the higher-priority message. To support non-destructive bitwise arbitration, the transmitting node must also monitor the state of the bus to see whether the logic state it is trying to send actually appears on the bus. While transmitting the ID of a data frame, at each bit all transmitting nodes ensure that the written bit is visible on the bus. If a node transmits a recessive bit but reads a dominant one, it loses arbitration. Through this process, lower IDs have a higher probability of transmission in case of collision.

Bit Timing and Synchronization. Given the absence of a global clock, the CAN protocol is asynchronous and thus it needs a method to keep each node on the bus synchronized with the others. To do so it uses bit stuffing to add one bit to the sequence when five consecutive bits with the same value are detected. This method ensures a falling or rising edge every five bits and keeps nodes synchronized. The stuffed bit is set to the opposite value and removed by the CAN controller while the frame is being received.

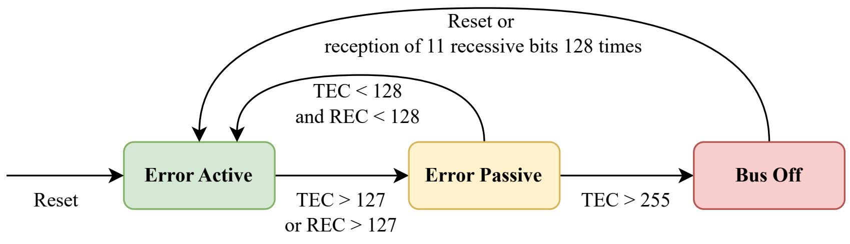

Figure 1: Node states finite state machine in CAN

Error Generation and Error Frames. Five different types of errors (not mutually exclusive) can occur during a communication: **a bit error** is detected if the monitored bit value is different from the sent bit value, except during arbitration, in the ACK slot, or over a Passive error flag. A **stuff error** is detected at the bit time of the 6th consecutive equal bit level in a message field. A **CRC error** is detected when the calculated result is different from the received CRC. A **form error** is detected when a fixed-form bit field has one or more illegal bits. Finally, an **ACK error** is detected by a transmitter when no dominant bit is read during the ACK slot.

Whenever an error is detected either by a transmitter or a receiver, that node starts sending an *error frame*. The error frame starts with the error flag, which consists of 6 consecutive dominant bits. This is purposefully designed to break the bit-stuffing rule so that every other node monitoring the bus will read, recognize and send its error flag. The superposition of error flags due to this event may lead to 0-6 extra dominant bits. An important distinction to make is that the error flag can be either Active (6 dominant bits as written above) or Passive (6 recessive bits) based on the detecting node's state, as described in the fault confinement paragraph. The error delimiter ends the error frame and is composed of 8 consecutive recessive bits. As soon as the error frame is sent, the transmitter of the corrupted frame attempts to retransmit the frame.

Fault Confinement. The CAN protocol implements a fault confinement mechanism to detect faulty nodes and prevent them to cause high bus loads, assigning a state to every node in the network. In *error active* state the node can normally communicate on the bus and, upon detecting an error sends an active error flag. In *error passive* state the node takes part in bus communication but when an error has been detected it sends a passive error flag. Finally, in *bus off* state the node is not allowed to have any influence on the bus. To move between states, each node implements two error counters: A Transmit Error Counter (TEC) and Receive Error Counter (REC), which are used to determine the state of the node itself, as represented in Figure 1. These counters are modified according to the following rules:

1. When a receiving node detects an error, its REC is increased by 1, except when the detected error is a bit error during the transmission of an active error flag or an overload flag.
2. When a receiver detects a dominant bit as the first bit after sending an error flag the REC will be increased by 8.
3. When a transmitter sends an error flag the TEC is increased by 8. Exception 1: If the transmitter is error passive and detects an acknowledgment error because of not detecting a dominant ACK and does not detect a dominant bit while sending its passive error flag. Exception 2: If the transmitter sends an error flag because a stuff error occurred during arbitration, and should have been recessive, and has been sent as recessive but monitored as dominant. In exceptions 1 and 2 the TEC is not changed.
4. If a transmitter detects a bit error while sending an active error flag or an overload flag the TEC is increased by 8.
5. If a receiver detects a bit error while sending an active error flag or an overload flag the REC is increased by 8.
6. Any node tolerates up to 7 consecutive dominant bits after sending an active error flag, passive error flag or overload flag. After detecting the 14th consecutive dominant bit (in case of an active error flag

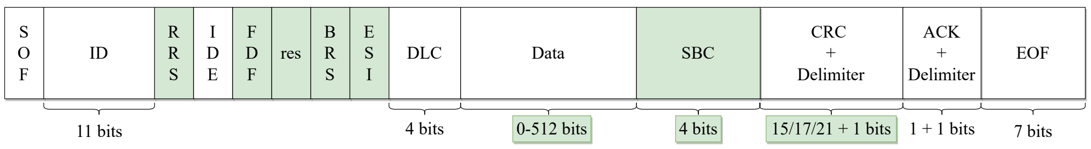

Figure 2: Fields of the CAN FD data frame, in green the differences from standard CAN frames.

or an overload flag) or after detecting the 8th consecutive dominant bit following a passive error flag, and after each sequence of additional eight consecutive dominant bits every transmitter increases its TEC by 8 and every receiver increases its REC by 8.

7. After the successful transmission of a message (getting ACK and no error until the end of frame is finished) the TEC is decreased by 1 unless it was already 0.
8. After the successful reception of a message (reception without error up to the ACK slot and the successful sending of the ACK bit), the REC is decreased by 1, if it was between 1 and 127. If the REC was 0, it stays 0, and if it was greater than 127, then it will be set to a value between 119 and 127.
9. A node is error passive when the TEC equals or exceeds 128, or when the REC equals or exceeds 128. An error condition letting a node become error passive causes the node to send an active error flag.
10. A node is bus off when the TEC is greater than or equal to 256.
11. An error passive node becomes error active again when both the TEC and REC are less than or equal to 127.
12. A node which is bus off is permitted to become error active again with its error counters both set to 0 after 128 occurrences of 11 consecutive recessive bits have been monitored on the bus.

2.1. Differences with CAN FD

Controller Area Network with Flexible Data-rate (CAN FD) was born to allow data rates exceeding 1 Mbit/s, with a payload length ranging from 0 to 64 bytes [14]. The idea behind CAN FD is that when just one node is transmitting, the bitrate can be raised because no nodes must be synchronized. Of course, the nodes must be re-synchronized before transmitting the ACK slot bit.

Frame Format. One of the most important differences between CAN and CAN FD is the frame format. Figure 2 shows the CAN FD frame format, highlighting the differences from standard CAN frames. The relevant new or modified fields in CAN FD are the following: *FD Format (FDF)*: is a recessive bit which distinguishes between CAN and CAN FD frames. *Bit Rate Switch (BRS)*: decides whether the bit-rate is switched inside a CAN FD format frame. If the bit is recessive the bit rate is switched, otherwise not. *Error State Indicator (ESI)*: dominant for error active and recessive for error passive nodes. *Stuff Bit Count (SBC)*: This field represents the number of dynamic stuff bits (modulo 8) in the frame to lower the risk of undetected errors. CRC: has a variable length of 15, 17, and 21 bits depending on the data length of the frame.

Bit Timing. The CAN FD specification introduces a second bit-time configuration into CAN networks. ISO 11898-1 [14] has been expanded to include the CAN FD format, including the nominal bit-rate and the data bit-rate. The nominal bit-rate is related to the network architecture in the arbitration phase, and the speed is limited to 1 Mbit/s. In the data phase, the speed is limited simply by the transceiver characteristics, with a theoretical maximum of 8 Mbit/s. However, there is a significant difference between theoretical and practical bit-rate [15]: CAN usually runs at $\simeq$ 0.5Mbit/s while CAN FD at $\simeq$ 2 Mbit/s.

3. Related Works and Motivation

Attacks on CAN can be divided into spoofing and Denial of Service. Eavesdropping is usually not considered a defensible threat since the bus cannot realistically be encrypted due to timing requirements.

Since CAN has no authorization or authentication method embedded in the protocol, an attacker can simply spoof a frame with any ID on the bus and this would be read by anyone on the bus. DoS on CAN, as in the vast majority of bus networks, is simple to implement. An attacker can spoof ID 0x00 to ensure winning arbitration and send as many frames as possible on the bus to block any other node from sending data. However, this is extremely easy to detect by a basic IDS and even by standard consistency checks on undefended ECUs [8]. Moreover, being CAN messages usually periodic, it is easy for an IDS to detect spoofing attempts in general, since even if the attacker writes valid frames of a given ID, the ECU that sends such ID continues to do so and incongruities can be detected, alongside a change in the frequency of the ID appearance on the bus [16]. To circumvent this, Palanca et al. [7] and Cho et al. [17], demonstrated targeted DoS attacks on CAN by exploiting its error-handling mechanism, forcing the victim ECU into a bus-off state, rendering it unable to transmit frames. Although not extremely useful for practical purposes by itself, the attacker can then implement a drop-and-spoof attack, sending spoofed frames impersonating the victim without triggering any incongruities on the bus. Multiple researchers followed their approach and have shown the feasibility to implement drop-and-spoof attacks through software-only solutions [10, 11, 12, 2], hence enabling them from remote, significantly increasing the risk associated with such attacks.

Multiple countermeasures have been proposed for CAN, amongst which the most effective ones have proven to be secure network architectures, authentication protocols, and IDSs. Secure network architectures are effective in dividing critical ECUs from those that are more easily accessible by an attacker [18, 2], but once the attacker obtains access to the critical network they lose effectiveness. Authentication protocols ensure that a message arrives from the valid sender, and many lightweight solutions have been proposed through the years [3, 4, 19, 20]. However, as also highlighted by Nowdehi et al. [21] and Ring et al. [22], authentication methods for CAN are limited by many factors, such as computation and network overheads, backward compatibility, and the necessity to implement them on all the nodes of the network that need to authenticate or be authenticated. Finally, Intrusion Detection Systems (IDS) recognize attackers on the network through different techniques. Signature-based detection uses a set of known signatures or previously-known attack patterns to detect attacks [23, 24], however, it cannot detect previously unknown attack patterns. Anomaly-based detection, on the contrary, uses knowledge of the normal behavior of the network to recognize patterns that do not follow such normality. The majority of IDSs for CAN fall in this category thanks to the ease of implementing machine-learning-based methods [25, 26, 6], and the possibility to use frequency as a feature [16, 27, 28, 29]. However, the majority of anomaly-based IDSs for CAN work from behind the CAN controller, which means that they have no view of the bus at bit level. Finally, specification-based detection works through a set of well-known rules describing the behavior of components of the network, e.g., ECUs or the CAN protocol specification. Matsumoto et al. [30], Parrot [31], and CopyCAN [9] are all examples of specification-based IDSs.

Motivation. While IDSs are one of the most commonly proposed solutions for CAN attacks, some implementations of packet drop attacks are by design impossible to detect by IDSs that do not rely on reading the bus directly and checking the bit value's correctness, potentially leading to drop-and-spoof attacks. This type of attack can be implemented using techniques such as CANflict [11], Palanca et al.'s attack [7], or Cho et al.'s attack [17]. Additionally, we discovered a vulnerability in a previously proposed IDS, CopyCAN [9] - which was meant to detect such attacks - where, by overwriting the protected ECU's passive error flag (composed of recessive bits) and simulating the finishing of the protected ECU's frame, the attacker was capable of not showing on the bus the triggering of the fault confinement mechanism, leading to a failure to detect passive error flags by other nodes on the bus. CopyCAN, being a node on the bus itself, would also fail to recognize the inconsistency. Furthermore, recent automotive adversary research has studied several new attacks, including CANnon [10], CANflict [11], WeepingCAN [13], and Serag et al.'s attack [12], which enable drop-and-spoof attacks and enable to implement them without physical access to the network, enabling much more threatening remotely-implemented attacks.

The objective of this paper is to present a solution capable of detecting drop-and-spoof attacks, also ensuring that the mentioned new attack techniques cannot evade detection on both the CAN and

CAN FD buses while addressing the current limitations of CopyCAN [9] through the integration of a secondary detection module.

4. Approach

CANter is designed as the combination of two components, a Finite State Machine (FSM) module, and a frequency analysis module.

The goal of the first component is to recognize whether a protected ECU goes into bus-off state, and then detect eventual attackers that attempt to spoof the victim. The first component is designed as a finite-state machine (FSM) that parses CAN FD frames reading the bus bit-by-bit. The proposed solution does not exclude CAN frames due to the compatibility granted by the frame format. The FSM follows the frame format and fault confinement rules to identify incorrect bits or error sequences. Each frame field has a corresponding FSM state, and the transitions are based on the bit count. The TEC of the protected ECU (or ECUs) is tracked and updated based on the successfully transmitted frames and the error frames. It is important to note that through the parsing of the ID, CANter can also detect any attack that exploits the use of invalid IDs, such as Serag et al.'s attack [12]. By implementing the custom CAN controller, we aim to overcome the limitations of existing solutions that rely on reading the bus through the standard CAN controller, which are not aware of errors on the bus. In addition to this first component, CANter includes a frequency analysis module for anomaly detection based on frame periodicity. A pattern of CAN frame transmission was identified and used to differentiate between regular frames and attack frames. Combined with the reading routine, the frequency analysis component allows for a more comprehensive approach to intrusion detection on the CAN bus.

4.1. Finite State Machine Module

As mentioned before, the core idea of this module is to obtain a custom CAN controller and use it to constantly read the bus bit by bit, detecting whether a protected ECU moves in bus-off state.

Rules Handling. We examine the fault confinement rules of the CAN and CAN FD protocols, to exploit rule (10) (as presented in Section 2) to recognize the transition to a bus off state when the TEC reaches a value of 256. In accordance with this rule, the REC is not considered for monitoring purposes. Furthermore, other rules that pertain exclusively to REC handling, namely rule (1), (2), (5), and (8), are also excluded from our analysis. On the other hand, rules regarding error passive state transitions, such as rules (9) and (11), are taken into consideration to signal to CANter frequency analysis module when a node enters an error passive state. To handle rule **(3)**, the TEC of the protected ECU is increased by 8 if an error flag is detected during transmission. Two exceptions must be considered: Exception 1 pertains to an error passive transmitter detecting an ACK error, in which case the TEC should not increase if no other active error flag is transmitted. Since the bit stuffing rule is not applied after the CRC delimiter, no further action is required for this exception. Exception 2 only occurs during arbitration and is outside the scope of our analysis. For rule **(4)**, the TEC of the protected ECU must be increased if a bit error is detected while sending an active error flag. While an attacker cannot force a dominant bit to become recessive to evade detection our IDS, this scenario is not detectable by CANter . We evaluate the frequency of occurrence of this event in Section 5. Rule **(6)** refers to a scenario where the protected ECU detects more than 7 consecutive dominant bits following transmission of an error flag. To the best of the authors' knowledge, there is no official justification for this 7th bit, but we assume it serves to verify the failure of the ECU. If an error delimiter does not follow these 7 bits and an 8th dominant bit is detected, the TEC is immediately increased by 8, and again for every subsequent sequence of 8 consecutive dominant bits. Note that the maximum dominant bit sequence in a functioning network should be limited to 12, with every error active node detecting a stuff error after receiving the first error flag (6 bits) and sending its own (the remaining 6). This scenario can arise due to incorrect node readings or faulty nodes flooding the bus. CANter lacks information on the type of error detected or the first bit of the error flag. To prevent exploitation by attackers, we increase the TEC by 8 after the

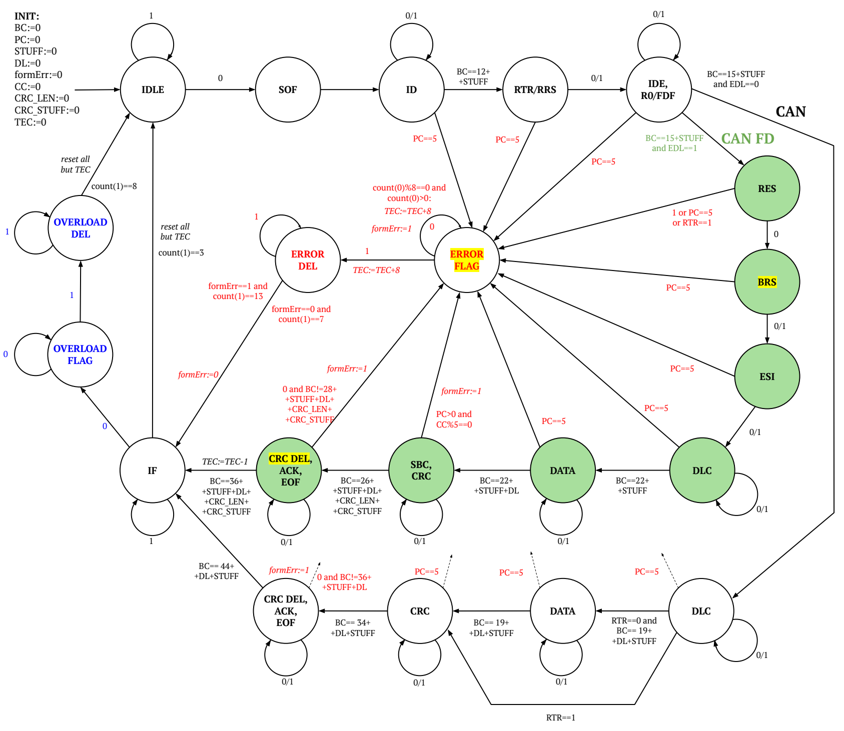

Figure 3: FSM representing the algorithm used by CANter . Green states are used to handle CAN FD frames, yellow highlighted states are states where a bit-rate switch might happen, and blue states are used to handle overload packets.

13th consecutive dominant bit. Rule **(7)** is easily handled by decreasing by 1 the TEC every time we parse a frame transmitted by the protected ECU without encountering any error. Rule **(12)** is handled by just resetting the TEC of the protected ECU after the mandatory time period, i.e. 128 occurrences of 11 consecutive recessive bits on the bus.

Algorithm. We proceed to explain the algorithm behind the FSM Module. For the sake of simplicity, we present the algorithm for 11-bit identifier CAN and CAN FD packets. In Figure 3 we show a graphical representation of the FSM. A set of *variables* are necessary to correctly manage the FSM: **BC**: Bit Counter, tracks the number of bits processed in the frame. **PC**: Polarity Counter, counts identical consequent bits. **STUFF**: stores the number of stuffed bits encountered. **DL**: Data Length, contains the length of the Data field in bits. **formErr**: a boolean variable used to handle form errors. **CC**: CRC Counter, keeps track of the number of processed CRC bits. **CRC LEN**: actual length of the CRC sequence. **CRC STUFF**: number of fixed stuffed bits inserted in the CRC sequence. **TEC**: variable to keep count of the TEC of the protected ECU.

The IDS starts when the vehicle is turned on and all ECUs have a TEC of 0. All variables are initialized to 0 and the entry state is IDLE. As soon as a dominant bit is detected, BC is increased and the IDS moves to the SOF state and then to the ID state. Here it starts checking for stuff bits and stuff errors through the PC and STUFF variables. Once the ID finishes, in the RTR/RRS state and in the following IDE state, the algorithm accepts and stores both recessive or dominant bits, through which it handles remote or data frames and 11 or 29 bits ID frames. The FDF bit is parsed and the algorithm moves to the RES state if the value is recessive (CAN FD) or to the DLC state if the value is dominant (CAN). In the case of a CAN FD frame, a dominant reserved bit is expected before moving to the BRS state. The value of this bit determines if there is a bit-rate switch to data bit-rate or not. The FSM's subsequent state is ESI in any case. The IDS then parses the DLC, through which DL, CRC LEN, and CRC STUFF variables are computed. The payload is then parsed in the DATA field, alongside SBC and CRC sequences. After this state, the PC is deactivated since the bit stuffing rule is deactivated. The algorithm reads the last ten bits of the frame, composed of the CRC delimiter, the ACK slot, the ACK delimiter, and 7 bits of

EOF. These bits should be recessive except for the ACK, where a dominant bit signals the successful reception of the frame by other nodes. If no error occurs, the TEC of the ID is decreased by 1. After an IFS of three recessive bits, all the variables are reset, except for TEC, and the algorithm waits for the next frame transmission in IDLE state. If one of these three bits is dominant, an overload frame must be parsed, accepting up to 12 consecutive dominant bits followed by 8 recessive bits. When the algorithm analyzes a standard CAN frame, it starts by parsing the FDF bit and then decodes the DLC and calculates the DL. If the saved RTR equals 0, the algorithm reads the payload, while if RTR is equal to 1, the algorithm skips directly to CRC, since it indicates a remote frame request. After reaching the CRC state, the bit stuffing rule is no longer applied, so the PC counter is deactivated. Once the CRC sequence has been read and parsed, the last bits of the frame are read, and if there are no errors, the TEC is decreased, and the algorithm moves to the IFS state. From this point on, the algorithm continues as described above.

Error handling is a fundamental part of the algorithm since it is crucial to correctly increase the TEC of the protected ECU: The algorithm monitors if an error flag is transmitted on the bus by checking for six consecutive identical bits while the bit stuffing rule is active. If this occurs, the algorithm moves to the ERROR FLAG state to handle the error and exceptions. The ERROR FLAG state contains the handling process for rule (6) where the algorithm checks the number of consecutive dominant bits on the bus, increasing TEC at the 14th bit and every 8 additional bits. Once the algorithm detects a recessive bit the error flag is finished, TEC is increased by 8, and the algorithm switches to the ERROR DEL state expecting 7 recessive bits to complete the error delimiter. For CAN FD frames, the PC counter is also used to check the CRC bit stuffing rule. The algorithm knows the position of the first CRC bit and subsequent bits after every fourth bit of the sequence. Using the CC counter, the algorithm checks if there is a bit with the same value as the preceding one in these positions (i.e. $CC\%5 == 0$ and $PC > 0$). If this occurs, the algorithm sets formErr to 1 and moves to the ERROR FLAG state to handle it, as specified by the CAN FD specification. For both CAN FD frames and CAN frames, the algorithm needs to handle a form error happening in the last ten bits of the frame, from CRC delimiter to EOF. If a dominant bit is detected among these fields except for the ACK, the algorithm sets the formError variable to 1 and moves to ERROR FLAG state. The algorithm then expects either an active error flag and its delimiter (i.e. 6 dominant and 8 recessive bits) or a passive error flag and its delimiter (i.e. 14 recessive bits). The role of formErr is to differentiate between the two situations: after reading the first bit of the flag, if it is dominant, the algorithm sets formErr back to 0 and waits for the dominant bits to finish before moving to ERROR DEL. If the bit is recessive, the algorithm keeps formErr at 1 and moves directly to ERROR DEL and then parses the remaining 13 bits. After that, the algorithm resumes execution from IFS state as in error-free transmission. When the TEC of an ECU reaches 256, the algorithm checks and counts every 11-bit-long sequence of recessive bits. When this counter reaches 128, the TEC is reset to 0, and the ECU is assumed to be connected again.

4.2. Frequency Analysis Module

The only assumption, which is consistent with almost all CAN communication [16], that is required for any frequency analysis module to work, is that the frames of the protected ECU have to be periodic. An empirical evaluation of the ReCAN dataset [32] also shows evidence of a fundamental pattern of CAN periodic packets: in attack-free scenarios, delays in the transmission of a packet are common. However, when this happens in attack-free scenarios, we observe a pattern (as visible in Figure 4) that we define as **late-early pattern**: if a frame B takes more than the average inter-arrival time from a frame A (late), then the inter-arrival time between frame B and the next frame C will be shorter than the average.

Moreover, as shown in Figure 5, when an attacker triggers an error frame on a victim's message, the victim will attempt to send that message again immediately after. Therefore, even if the attacker hides the victim's passive error flag, the perceived frequency on the bus will be increased for one frame. At this point, the attacker can either silence the new frame again - being recognized by the FSM module - or let it pass.

Starting from this intuition we design the frequency analysis module based on three elements: a

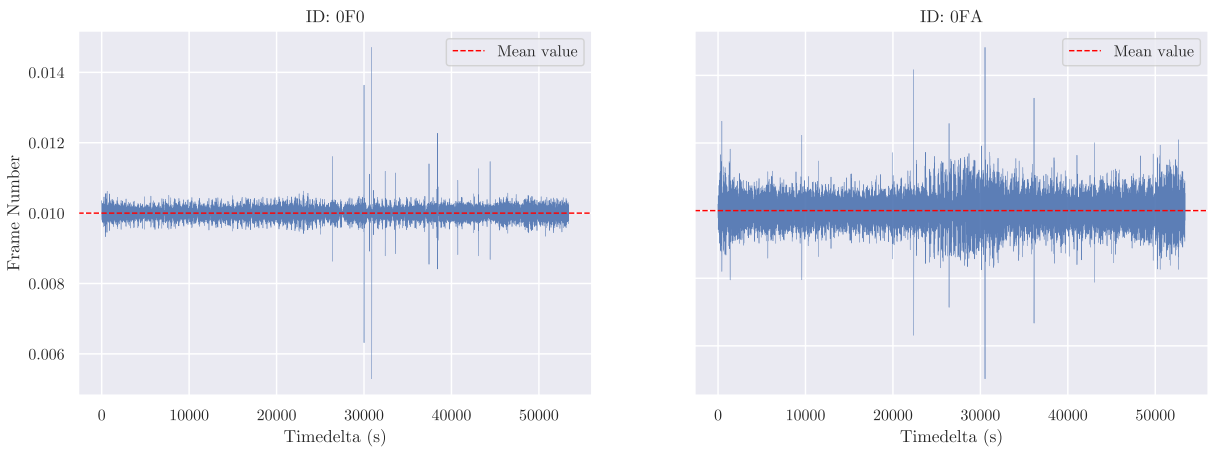

Figure 4: Packet inter-arrival times in relation to the average inter-arrival time of two different IDs from the ReCAN dataset.

Table 1
Statistics on inter-arrival times of periodic IDs.

ID	Total frames	# early	# late-early
0DE	53399	1	1
0EE	53287	2	2
0F0	53400	2	2
0FA	53399	0	0
0FF	53399	3	3
120	53397	56	32
192	53364	5	3

threshold, a **condition**, and a **counter**. The **threshold** is used to define whether a sequence of packets has to be analyzed. Specifically, considering an average packet inter-arrival time of x, we consider a threshold of $x/2$ to start the analysis. In fact, referencing Figure 5, either the inter-arrival time between frame Y^l and Y or the one between Y to Z will trigger such a threshold. The **condition** is necessary to control if the frame with the inter-arrival time below the threshold respects the late-early pattern. To do so we compute the mean m between the inter-arrival time that triggered the threshold and its predecessor. We then apply the *empirical rule* of Gaussian distributions (the inter-arrival times of all IDs in the ReCAN dataset [32] have proven to follow Gaussian distributions, as expected), which states that 99.7% of a Gaussian distribution lies within three standard deviations from the mean. Hence, if m is not into three standard deviations from the mean of its ID we flag the event.

An evaluation of various IDs in attack-free scenarios in Table 1 shows that, however, there may be valid events that are flagged as attacks through the previously defined condition. Therefore, to avoid false positives, we implement a **counter**: since the analysis is necessary for ECUs in error passive state, the attacker needs to increase the TEC of the victim by 128 to force the ECU into bus off state. Therefore, we use a counter that acts similarly to the TEC of the protected ECU. It is increased by 8

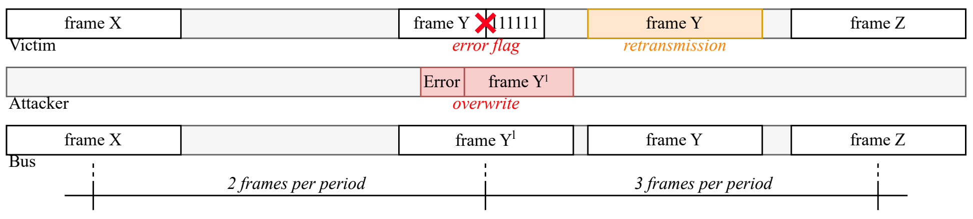

Figure 5: A representation of the evasion attempt scenario that shows the ID frequency increase.

Table 2
Tests executed to validate CANter 's approach alongside their results.

Test Number	Configuration	Traffic	Attack Detected	False Positives
1	CAN frames	-	✓	X
2	CAN FD frames, BRS = 0	-	✓	X
3	CAN FD frames, BRS = 1	-	✓	X
4	CAN and CAN FD frames	-	✓	X
5	CAN frames	✓	✓	X
6	CAN FD frames, BRS = 0	✓	✓	X
7	CAN FD frames, BRS = 1	✓	✓	X
8	CAN and CAN FD frames	✓	✓	X
9	CAN frames, Hiding Attempt	✓	✓	X
10	ReCAN [32] frequency evaluation	-	-	X

every time an event is flagged by the condition and decreased by 1 when a regular inter-arrival time is observed. If the counter becomes greater than 64, the sequence of events is considered an attack.

5. Evaluation

Our test setup is composed of an **attacker node** (Nucleo L552ZE-Q board + MCP2557FD transceiver) which implements the attack presented in CANflict [11], a **victim node** (Aurix TC399 Board - CAN0 Line), **CANter** (Nucleo L552ZE-Q board + MCP2557FD transceiver), and a **traffic generator** (Aurix TC399 Board - CAN0 Line). The nodes are connected to a 120Ω CAN bus line.

Our evaluation has three main objectives: **(1)** to demonstrate CANter 's ability to reliably detect drop-and-spoof attacks without any false positives on both CAN and CAN FD communication. **(2)** To evaluate the practical relevance of rules (4) and (6) which are designed to limit the effectiveness of CANter . **(3)** We aim to showcase the efficacy of the frequency analysis module in detecting attackers who try to evade detection by the FSM module.

To showcase the efficacy of CANter , we conducted the experiments outlined in Table 2 using the aforementioned test configuration. To ensure consistency, each test was repeated 50 times. In the absence of bit-rate switch activation (BRS = 0), the bus speed was set at 500Kb/s. When BRS = 1, the speed was boosted to 1Mb/s. It is important to note that it is the attacker's board that imposes a limit on the bus speed, resulting in inconsistent outputs when exceeding the stated throughput. Finally, the "hiding attempt" in test number 9 is executed by - every time the attacker silences a frame - letting the victim write one valid packet.

Tests from 1 to 7 demonstrate the effectiveness of the FSM module of CANter against drop-and-spoof attacks. No attack evades detection, and no attack-free event triggers the IDS. Tests from 5 to 9 are executed under traffic to attempt to trigger rules (4) and (6) by causing collisions between valid frames. Again, the tests show the effectiveness of CANter , and no false positive is generated, which is consistent with the findings in CopyCAN [9]. Finally, test number 9 implements an attempt to hide from CANter 's frequency analysis module, and also in this case the results show that the attacker is not capable of evading CANter . Regarding the frequency analysis module, it is important to note that evaluating all the ReCAN [32] dataset (test number 10) presented no occurrences of an ECU triggering the frequency analysis module detection.

6. Conclusions

In this paper, we proposed CANter , a novel Intrusion Detection System for CAN and CAN FD networks capable of detecting drop-and-spoof attacks. Our IDS monitors the traffic and keeps track of the error counters of ECUs, detecting nodes' disconnections and consequent spoofed messages. At the same time, it performs a frequency analysis to avoid evasion from knowledgeable attackers, solving the main vulnerability of previous related works. We demonstrated the feasibility of CANter by implementing a proof-of-concept testbed, and the experimental results shown that both components of CANter detected all attacks without ever generating false positive results.

Acknowledgments

This paper was accepted at ITASEC24 but included in the ITASEC25 proceedings. This work is supported by the MICS (Made in Italy – Circular and Sustainable) Extended Partnership and received funding from Next-Generation EU (Italian PNRR – M4 C2, Invest 1.3 – D.D. 1551.11-10-2022, PE00000004). CUP MICS D43C22003120001.

Declaration on Generative AI

Generative AI tools such as Grammarly and ChatGPT 4o were utilized solely for proofreading and grammar refinement in the preparation of this manuscript. The authors retain full responsibility for the content presented in the final version.

References

[1] C. Miller, C. Valasek, Remote exploitation of an unaltered passenger vehicle, Black Hat USA 2015 (2015).

[2] S. Woo, H. J. Jo, I. Kim, D. H. Lee, A practical security architecture for in-vehicle CAN-FD, IEEE Trans. Intell. Transp. Syst. 17 (2016) 2248–2261. URL: https://doi.org/10.1109/TITS.2016.2519464. doi:10.1109/TITS.2016.2519464.

[3] B. Groza, P. Murvay, A. V. Herrewege, I. Verbauwhede, Libra-can: Lightweight broadcast authentication for controller area networks, ACM Trans. Embed. Comput. Syst. 16 (2017) 90:1–90:28. URL: https://doi.org/10.1145/3056506. doi:10.1145/3056506.

[4] A. Radu, F. D. Garcia, Leia: A lightweight authentication protocol for CAN, in: I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, C. A. Meadows (Eds.), Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II, volume 9879 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 283–300. URL: https://doi.org/10.1007/978-3-319-45741-3_15. doi:10.1007/978-3-319-45741-3_15.

[5] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, S. Zanero, Cannolo: An anomaly detection system based on LSTM autoencoders for controller area network, IEEE Trans. Netw. Serv. Manag. 18 (2021) 1913–1924. URL: https://doi.org/10.1109/TNSM.2020.3038991. doi:10.1109/TNSM.2020.3038991.

[6] B. Lampe, W. Meng, A survey of deep learning-based intrusion detection in automotive applications, Expert Systems with Applications (2023) 119771.

[7] A. Palanca, E. Evenchick, F. Maggi, S. Zanero, A stealth, selective, link-layer denial-of-service attack against automotive networks, in: M. Polychronakis, M. Meier (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment - 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings, volume 10327 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 185–206. URL: https://doi.org/10.1007/978-3-319-60876-1_9. doi:10.1007/978-3-319-60876-1_9.

[8] C. Miller, C. Valasek, Advanced can injection techniques for vehicle networks, Black Hat USA, Las Vegas, NV, USA 4 (2016).

[9] S. Longari, M. Penco, M. Carminati, S. Zanero, Copycan: An error-handling protocol based intrusion detection system for controller area network, in: L. Cavallaro, J. Kinder, T. Holz (Eds.), Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy, CPS-SPC@CCS 2019, London, UK, November 11, 2019, ACM, 2019, pp. 39–50. URL: https://doi.org/10.1145/3338499.3357362. doi:10.1145/3338499.3357362.

[10] S. Kulandaivel, S. Jain, J. Guajardo, V. Sekar, CANNON: reliable and stealthy remote shutdown attacks via unaltered automotive microcontrollers, in: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021, IEEE, 2021, pp. 195–210. URL: https://doi.org/10.1109/SP40001.2021.00122. doi:10.1109/SP40001.2021.00122.

[11] A. de Faveri Tron, S. Longari, M. Carminati, M. Polino, S. Zanero, Canflict: Exploiting peripheral conflicts for data-link layer attacks on automotive networks, in: H. Yin, A. Stavrou, C. Cremers, E. Shi (Eds.), Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022, ACM, 2022, pp. 711–723. URL: https://doi.org/10.1145/3548606.3560618. doi:10.1145/3548606.3560618.

[12] K. Serag, R. Bhatia, V. Kumar, Z. B. Celik, D. Xu, Exposing new vulnerabilities of error handling mechanism in CAN, in: M. Bailey, R. Greenstadt (Eds.), 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021, USENIX Association, 2021, pp. 4241–4258. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/serag.

[13] G. Bloom, Weepingcan: A stealthy can bus-off attack, in: Workshop on Automotive and Autonomous Vehicle Security, 2021.

[14] ISO Central Secretary, Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling, Standard ISO 11898-1:2015, International Organization for Standardization, Geneva, CH, 2015. URL: https://www.iso.org/standard/63648.html.

[15] C. C. in Automation, Controller area network extra long (can xl), [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-xl/, ????

[16] M.-J. Kang, J.-W. Kang, Intrusion detection system using deep neural network for in-vehicle network security, PloS one 11 (2016) e0155781.

[17] K. Cho, K. G. Shin, Error handling of in-vehicle networks makes them vulnerable, in: E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, S. Halevi (Eds.), Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, ACM, 2016, pp. 1044–1055. URL: https://doi.org/10.1145/2976749.2978302. doi:10.1145/2976749.2978302.

[18] M. Wolf, A. Weimerskirch, T. J. Wollinger, State of the art: Embedding security in vehicles, EURASIP J. Embed. Syst. 2007 (2007). URL: https://doi.org/10.1155/2007/74706. doi:10.1155/2007/74706.

[19] S. Nürnberger, C. Rossow, - vatican - vetted, authenticated CAN bus, in: B. Gierlichs, A. Y. Poschmann (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings, volume 9813 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 106–124. URL: https://doi.org/10.1007/978-3-662-53140-2_6. doi:10.1007/978-3-662-53140-2_6.

[20] G. Kornaros, D. Bakoyiannis, O. Tomoutzoglou, M. Coppola, G. Gherardi, Trustnet: Ensuring normal-world and trusted-world can-bus networking, in: 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids, SmartGridComm 2019, Beijing, China, October 21-23, 2019, IEEE, 2019, pp. 1–6. URL: https://doi.org/10.1109/SmartGridComm.2019.8909715. doi:10.1109/SmartGridComm.2019.8909715.

[21] N. Nowdehi, A. Lautenbach, T. Olovsson, In-vehicle CAN message authentication: An evaluation based on industrial criteria, in: 86th IEEE Vehicular Technology Conference, VTC Fall 2017, Toronto, ON, Canada, September 24-27, 2017, IEEE, 2017, pp. 1–7. URL: https://doi.org/10.1109/VTCFall.2017.8288327. doi:10.1109/VTCFall.2017.8288327.

[22] M. Ring, D. Frkat, M. Schmiedecker, Cybersecurity evaluation of automotive e/e architectures, in: ACM Computer Science In Cars Symposium (CSCS 2018), 2018.

[23] M. Müter, N. Asaj, Entropy-based anomaly detection for in-vehicle networks, in: IEEE Intelligent Vehicles Symposium (IV), 2011, Baden-Baden, Germany, June 5-9, 2011, IEEE, 2011, pp. 1110–1115. URL: https://doi.org/10.1109/IVS.2011.5940552. doi:10.1109/IVS.2011.5940552.

[24] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, Y. Laarouchi, A language-based intrusion detection approach for automotive embedded networks, Int. J. Embed. Syst. 10 (2018) 1–12. URL: https://doi.org/10.1504/IJES.2018.10010488. doi:10.1504/IJES.2018.10010488.

[25] S. Longari, C. A. Pozzoli, A. Nichelini, M. Carminati, S. Zanero, Candito: Improving payload-based

detection of attacks on controller area networks, in: S. Dolev, E. Gudes, P. Paillier (Eds.), Cyber Security, Cryptology, and Machine Learning - 7th International Symposium, CSCML 2023, Be'er Sheva, Israel, June 29-30, 2023, Proceedings, volume 13914 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 135–150. URL: https://doi.org/10.1007/978-3-031-34671-2_10. doi:10.1007/978-3-031-34671-2_10.

[26] A. Nichelini, C. A. Pozzoli, S. Longari, M. Carminati, S. Zanero, Canova: a hybrid intrusion detection framework based on automatic signal classification for can, Computers & Security (2023) 103166.

[27] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, S. J. Prowell, Modeling inter-signal arrival times for accurate detection of CAN bus signal injection attacks: a data-driven approach to in-vehicle intrusion detection, in: J. P. Trien, S. J. Prowell, J. R. Goodall, J. M. Beaver, R. A. Bridges (Eds.), Proceedings of the 12th Annual Conference on Cyber and Information Security Research, CISRC 2017, Oak Ridge, TN, USA, April 4 - 6, 2017, ACM, 2017, pp. 11:1–11:4. URL: https://doi.org/10.1145/3064814.3064816. doi:10.1145/3064814.3064816.

[28] H. M. Song, H. R. Kim, H. K. Kim, Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network, in: 2016 International Conference on Information Networking, ICOIN 2016, Kota Kinabalu, Malaysia, January 13-15, 2016, IEEE Computer Society, 2016, pp. 63–68. URL: https://doi.org/10.1109/ICOIN.2016.7427089. doi:10.1109/ICOIN.2016.7427089.

[29] A. Taylor, N. Japkowicz, S. Leblanc, Frequency-based anomaly detection for the automotive CAN bus, in: 2015 World Congress on Industrial Control Systems Security, WCICSS 2015, London, United Kingdom, December 14-16, 2015, IEEE, 2015, pp. 45–49. URL: https://doi.org/10.1109/WCICSS.2015.7420322. doi:10.1109/WCICSS.2015.7420322.

[30] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, K. Oishi, A method of preventing unauthorized data transmission in controller area network, in: Proceedings of the 75th IEEE Vehicular Technology Conference, VTC Spring 2012, Yokohama, Japan, May 6-9, 2012, IEEE, 2012, pp. 1–5. URL: https://doi.org/10.1109/VETECS.2012.6240294. doi:10.1109/VETECS.2012.6240294.

[31] T. Dagan, A. Wool, Parrot, a software-only anti-spoofing defense system for the can bus, ESCAR EUROPE 34 (2016).

[32] M. Zago, S. Longari, A. Tricarico, M. Carminati, M. G. Pérez, G. M. Pérez, S. Zanero, Recan–dataset for reverse engineering of controller area networks, Data in brief 29 (2020) 105149.