USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

A Formal Security Analysis of CAN XL

Zhaozhou Tang, *Georgia Institute of Technology;* Khaled Serag, *Qatar Computing Research Institute;* Z. Berkay Celik, *Purdue University;* Vijay Ganesh, Saman Zonouz, and Raheem Beyah, *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity26/presentation/tang-zhaozhou

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

A Formal Security Analysis of CAN XL

Zhaozhou Tang
Georgia Institute of Technology

Khaled Serag
Qatar Computing Research Institute

Z. Berkay Celik
Purdue University

Vijay Ganesh
Georgia Institute of Technology

Saman Zonouz
Georgia Institute of Technology

Raheem Beyah
Georgia Institute of Technology

Abstract

For decades, the Controller Area Network (CAN) has been the backbone of in-vehicle communication. As modern vehicles integrate cameras, LiDARs, and AI components, classic CAN (CAN CC) faces growing limitations in bandwidth, functionality, and security. To fill these gaps, CAN XL was introduced as the next generation of CAN, aiming to offer longer payloads, higher bandwidth, and enhanced security.

CAN XL makes significant standard-level changes across multiple stack layers. It also introduces several security features, but it is unclear whether these are mere add-on extensions or whether the standard redesign itself tackles CAN's chronic security weakness: the MAC sub-layer. This sub-layer governs frame formats and error handling and has historically enabled many CAN CC attacks. As the industry transitions to CAN XL, the security posture of its yet-unexplored MAC sub-layer must be understood before widespread deployment.

This paper presents the first security analysis of the CAN XL standard, focusing on its MAC sub-layer. We develop a bit-precise CAN XL formal model and release it to facilitate future research. We design a formal analysis workflow guided by CAN XL's field-oriented structure to uncover vulnerabilities. Contrary to expectations, our analysis shows that CAN XL remains vulnerable to all known CAN CC MAC sub-layer issues while introducing seven new vulnerabilities, arguably worsening security. We validate them using commercial CAN XL controllers and demonstrate exploitability via two multi-stage attacks on a testbed simulating real vehicle traffic. Finally, we propose mitigations including formally verifying standard revisions that could prevent several attacks.

1 Introduction

The Controller Area Network (CAN) has long been the most widely used in-vehicle network protocol, enabling Electronic Control Units (ECUs) to manage vehicle operations. As modern vehicles integrate advanced features from entertainment to autonomous driving, in-vehicle networks increasingly demand higher bandwidth, performance, functionality, and security. To keep up with these demands, CAN Extra Long (CAN XL) was introduced in May 2024 as a major upgrade of classic CAN (CAN CC). Its standard brings significant changes across protocol layers to offer longer payloads, higher bandwidth, and many more features. It is expected by many to become the primary CAN variant and the main bus for intelligent driver assistance systems in future vehicles [1, 4–6].

In its effort to enhance security, CAN XL introduces several features such as CANsec, tunneling, and virtual networks. However, it remains unclear whether the standard redesign itself happened with security in mind, or whether security is only confined to these add-on extensions. This question is especially acute for the MAC sub-layer—arguably CAN CC's weakest point—which governs bus access, frame formats, and error handling. It is the root cause of a major portion of CAN CC's vulnerabilities [30] and underlies many of the most prominent CAN attacks, such as full network DoS, injecting errors, mapping message sources, and shutting down target nodes [13, 24, 26–29, 33, 35]. CAN XL substantially revises this sub-layer: it introduces a new frame format, overhauls error handling (including new states and mechanisms), and makes parts of error handling configurable (enabled/disabled). To support incremental deployment, it also adds coexistence features for mixed CAN CC/FD/XL networks, increasing protocol complexity. Whether these far-reaching changes fix known MAC sub-layer vulnerabilities, maintain the status quo, or possibly introduce new ones remains an open question.

To close this gap, we conduct a security analysis focusing specifically on CAN XL's MAC sub-layer to examine whether CAN XL's standard changes eliminate any CAN CC vulnerabilities or introduce new ones. Through the analysis, we aim to answer an overarching question: *From a strict MAC sub-layer standpoint, have the changes made by CAN XL improved, maintained, or worsened security?* Conducting this analysis poses several challenges. (C1) Precision: The CAN XL MAC sub-layer operates on bits, not messages. An analysis must accurately capture bit-level protocol operations across more complex protocol fields and error handling mechanism un-

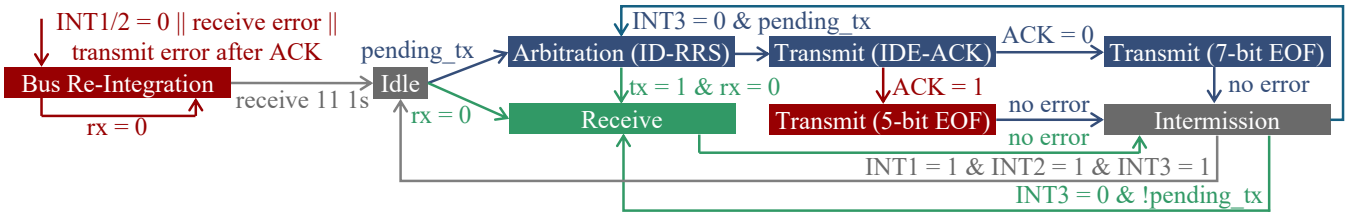


Figure 1: Simplified CAN XL error signalling disabled state diagram.

der all possible attacker actions. (C2) Generation coverage: As CAN XL is backward compatible, the analysis needs to thoroughly account for interactions between CAN XL and prior CAN generations. These include multiple frame formats, different node types, and error states. (C3) Tractability: Due to the previous two complexities, a CAN XL network has a large state space, making it difficult to perform a comprehensive exploration. Ad hoc approaches used by prior CAN CC security analysis, including manual standard reviews and testing [13, 28, 35], cannot handle these challenges.

To address them, we formally analyze the CAN XL MAC sub-layer by formally modeling a CAN XL network in the presence of an attacker. Using the nuXmv model checker [11], we verify properties relevant to new CAN XL features and known CAN CC weaknesses to discover standard-level vulnerabilities. To the best of our knowledge, this is the first formal security analysis of any part of the CAN XL standard. To address (C1), our model is bit-precise to accurately reflect the standard and thoroughly cover attacker actions. To address (C2), we model all CAN generations (CAN CC/FD/XL) to analyze cross-generation interactions. To address (C3), we employ domain-specific modeling techniques, including sequentially decomposing CAN XL according to its field-oriented design and using data and behavior abstractions to over-approximate possible protocol operations. Building on this field-level decomposition, we design a formal analysis workflow that breaks down the global protocol state space into manageable portions and analyzes them separately.

Our analysis yields counterintuitive results: CAN XL preserves all previously known CAN CC vulnerabilities and introduces seven additional ones. Exploiting them, attackers can launch many new attacks, including intercepting messages, identifying node types, new stealthy spoofing attacks, and violating data consistency extensively among nodes. These vulnerabilities also allow attacks that achieve equivalent goals to all common CAN CC attacks, even though some protocol features exploited in CAN CC have been modified in CAN XL. Examples include DoS of the whole network, shutting down target nodes, and mapping message sources. Moreover, through a comparative assessment on attack effectiveness, we find that CAN XL's new vulnerabilities make several attacks faster and less detectable than similar CAN CC attacks. Thus, we conclude that despite CAN XL's extensive modifications and attempts to enhance security, *the security of its MAC sub-layer not only fails to improve, but arguably worsens.*

We validate the presence of all discovered vulnerabilities using off-the-shelf CAN XL protocol controllers. To demonstrate the vulnerabilities' exploitability, we use them to construct and evaluate two multi-stage attacks from reconnaissance to execution. Finally, we propose mitigations, including formally verifying standard revisions to prevent several attacks. To inform relevant stakeholders before CAN XL's widespread adoption, we reported these vulnerabilities to Bosch, the developer of the CAN XL standard.

In summary, we make the following contributions:

- We present the first formal model of the MAC sub-layer of all CAN generations (CAN CC, CAN FD, and CAN XL) and release it publicly to support future research.
- We formally analyze CAN XL's MAC sub-layer by designing a formal analysis workflow guided by its field-oriented structure. To the best of our knowledge, this is the first standard security analysis of CAN XL.
- We verify that CAN XL inherits all CAN CC's MAC sub-layer vulnerabilities and discover seven new standard vulnerabilities introduced uniquely by CAN XL.
- We conduct a comparative security analysis of CAN XL and CAN CC's MAC sub-layer by assessing the effectiveness of attacks enabled by CAN CC and CAN XL vulnerabilities.
- We validate all discovered vulnerabilities on off-the-shelf protocol controllers. To demonstrate exploitability, we devise and evaluate two multi-stage attacks comprising the discovered vulnerabilities.
- We propose mitigations, including formally verifying standard revisions that could prevent several attacks.

2 Background

CAN XL MAC Sub-Layer Operation. CAN XL is a multi-master broadcast bus protocol. Its MAC sub-layer can operate in two significantly different error signalling configurations: *Error Signalling Disabled:* This is newly introduced in CAN XL and can only be used when all nodes in a network support CAN XL. Fig. 1 shows its simplified state diagram. A node may start arbitration when the bus is idle, transmitting an XL frame (Fig. 2) bit by bit. When multiple nodes start arbitration concurrently, a node yields if it sends a 1 but receives a 0 when transmitting the ID. Otherwise, it continues trans-

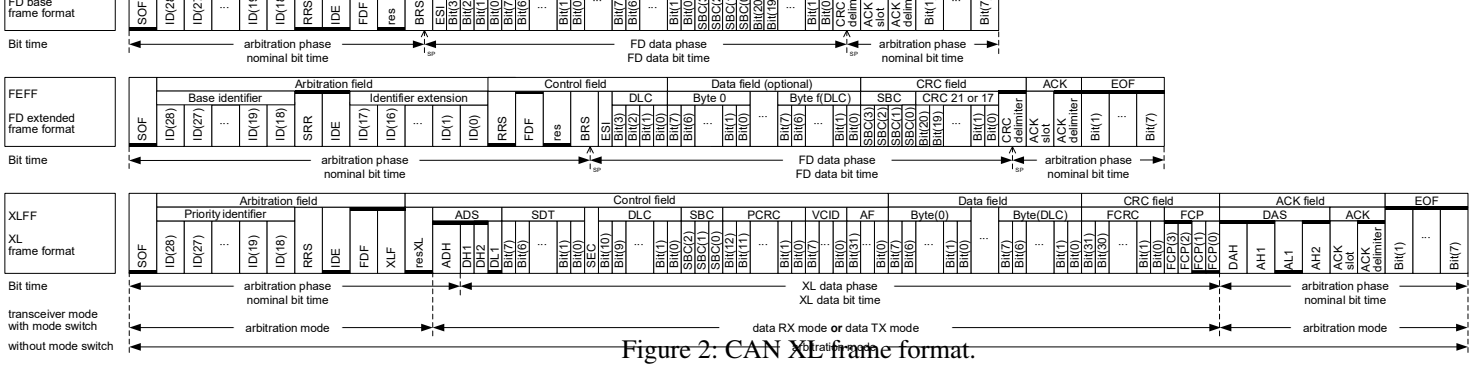


Figure 2: CAN XL frame format.

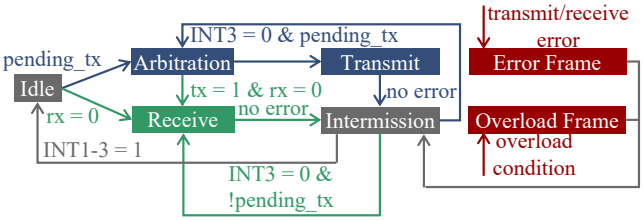


Figure 3: Simplified error signalling enabled state diagram.

Table 1: CAN node types and supported frame formats.

Frame Type	Frame Field Values			Node Types			
	FDF	XLF	resXL	CC	FD Tolerant	FD Enabled	XL Enabled
CC	0	X	X	●	●	●	●
FD	1	0	X	○	◐	●	●
XL	1	1	0	○	◐	◐	●

mission. Thus, a lower ID has a higher priority. Transmitters ignore errors from SOF to AH2. At the ACK slot, they detect an ACK error and finish the message with a shortened End-of-Frame (EOF) of 5 1s when receiving a 1. When receiving a 0 in the ACK slot, they consider the transmission successful and transmit an EOF with 7 1s. An error after ACK causes the transmitter to detect a protocol exception event. If a node does not have a message to transmit and receives a Start of Frame (SOF, logical 0) when the bus is idle, it starts receiving by decoding the bus state bit by bit based on the frame format. When no errors occur until the ACK slot, receivers send a 0 to indicate valid reception. On errors, they detect a protocol exception event. After a frame completes nodes enter a three-bit intermission. If a node receives 0 in the third bit of intermission (INT3), it interprets this as an SOF and starts arbitration by transmitting the ID if it has a pending transmission. All nodes detect a protocol exception event on receiving a 0 in the first two bits of intermission. When nodes detect a protocol exception event, they enter the bus re-integration state, transmit 1s, monitor the bus, and resume normal operation after receiving 11 1s. They remain in bus re-integration and restart counting 11 1s on receiving a 0 or 1→0 edge.

Error Signalling Enabled: In this configuration, CAN XL's MAC sub-layer operates similarly to CAN CC/FD. Fig. 3 shows a simplified state diagram. Nodes can transmit CC/FD frames in addition to XL frames. It has two main differences with error signalling disabled. ① Error handling and states: Transmitters and receivers detect five types of errors throughout operation: bit, stuff, form, CRC, and ACK errors. They record errors using the transmit error counter (TEC) and receive error counter (REC). Nodes operate in three error states: the default error-active state, the error-passive state when TEC or REC exceeds 127, and the bus-off state when TEC exceeds 255 and nodes stop communicating. Nodes signal errors with error frames which have an error flag and an error delimiter of 8 1s. The error flag contains 6 0s/1s for error-active/error-passive nodes, respectively. ② Overload handling: Overload frames have the same format as active error frames, consisting of a flag and a delimiter. Under these overload conditions, nodes transmit overload frames to delay bus traffic without incrementing error counters: receiving a 0 in the first two bits of intermission, or the last bit of error delimiter/overload delimiter, or a receiver receiving a 0 in the last EOF bit.

Mixed CAN Generations. CAN defines several node types based on the CAN generations they support (Tbl. 1). In mixed networks with other node types, XL enabled nodes must enable error signalling. Frames of different generations are distinguished by three bits after ID (e.g., CC frames' FDF is 0, and values of XLF and resXL do not matter). A node type may transmit/receive a supported frame (●) or detect errors when receiving an unsupported frame (○). For compatibility, certain node types may tolerate a frame (◐). As soon as they receive a bit indicating an unsupported frame (e.g., FD tolerant nodes receive a 1 in FDF), they detect a protocol exception event, enter bus re-integration, and exit after receiving 11 1s.

Frame Coding and Bit Rates. A CAN XL frame's (Fig. 2) arbitration field is coded with dynamic bit stuffing. After five consecutive identical bits, an inverse stuff bit is inserted. From DL1 to the end of FCRC, fixed bit stuffing is used. A stuff bit is inserted after every ten bits and has the inverse value of the preceding bit. The arbitration phase is transmitted with a nominal bit rate. The XL data phase must use a faster XL data bit rate, at least twice the nominal.

Physical Layer Modes. Bits are encoded with the differential voltage between CANH and CANL wires. Nodes interpret bits by reading the bus state at a configurable sample point within a bit. Two physical layer modes are defined. In SIC mode (same as CAN CC/FD), a high/zero voltage represents a 0/1 (dominant/recessive bit). Collisions between a 1 and a 0 result in a 0. In fast mode (new mode introduced in CAN XL), a positive/negative voltage represents a 0/1 (level_0/1). Fast mode can only be used from ADH to the end of FCP in XL frames when error signalling is disabled. Researchers have a limited understanding of this mode, except for the fact that collisions between level_0 and level_1 result in an unpredictable bus state, as stated in the standard [2].

3 Related Work

CAN CC Vulnerabilities. Researchers have discovered many vulnerabilities in CAN CC's MAC sub-layer:

(Va) *Error Handling:* CAN CC defines error states (Sec. 2) whose distinct behaviors are exploitable by attackers. The passive error regeneration vulnerability allows an error-passive node to encounter further errors when sending passive error frames [28]. Attackers can exploit these vulnerabilities by injecting errors using simultaneous transmission, where they transmit a message with the same ID as the victim at the same time with a different payload [13]. Both messages gain bus access and an error occurs when the payload first differs.

(Vb) *Data Inconsistency:* An error in the last EOF bit causes inconsistent frame validation outcomes, as only the transmitter validates a frame but receivers do not [35].

(Vc) *Bus Access Denial:* Overload frames (Sec. 2) after messages prevent the bus from returning to idle without generating visible messages or errors, and deny all nodes bus access.

CAN CC Attacks. Exploiting *Va*-*Vc*, researchers have proposed various attacks on CAN CC:

Confidentiality Attacks: Attackers can exploit *Va* to push nodes into error-passive and *map all IDs* they transmit [28].

Integrity Attacks: *Va* enables *advanced spoofing* attacks that push a node into bus-off/error-passive to spoof/hijack its messages [33]. Attackers can exploit *Vb* to *violate data consistency* by injecting errors in the last EOF bit [35].

Availability Attacks: Attackers can exploit *Vc* to launch stealthy *full network DoS* attacks by continuously sending overload frames [35]. They can exploit *Va* to *shut down target nodes* by pushing them into bus-off [13, 24, 27–29].

CAN CC Security Analysis. Security analysis of CAN CC primarily uses ad hoc approaches. Most vulnerabilities are identified by manual standard reviews [8, 13, 15, 19, 24, 32, 33, 35]. Alternatively, CANOX performs systematic testing by mutating various parameters in a physical CAN CC network, such as traffic load, priority, and error states [28]. Despite discovering impactful vulnerabilities, these approaches cannot handle CAN XL's complexity. Analyzing CAN XL requires considering all protocol operations over more complex fields and versatile attacker actions. It also requires accounting for all interactions with prior CAN generations which involve multiple node types, frame formats, and error states. Manual reviews or testing cannot account for them comprehensively.

4 Formal Analysis Methodology

4.1 Property Specification

We verify properties specifying the design goals of new CAN XL features to determine if they introduce vulnerabilities and properties that CAN CC vulnerabilities could compromise to assess if CAN XL suffers from similar security problems.

Protocol Exception. Protocol exception events and bus re-integration states are central to two major new CAN XL features: error signalling disabled operation and mixed CAN generations compatibility. The standard specifies that nodes enter the bus re-integration state on detecting a protocol exception event and exit after receiving 11 1s. 11 1s match the format between consecutive frames (1 ACK delimiter + 7 EOF + 3 intermission–Fig. 2). Accordingly, we derive these properties the protocol exception mechanism should fulfill:

- $\mathcal{P}_1$: After a node enters the bus re-integration state, it eventually leaves this state.
- $\mathcal{P}_2$: If a node enters the bus re-integration state during a frame, it stays in this state until the frame completes.
- $\mathcal{P}_3$: A node leaves the bus re-integration state before the next frame arrives.

Violating $\mathcal{P}_1$-$\mathcal{P}_3$ has many adverse consequences. Nodes may stay in bus re-integration persistently, unable to take part in bus communication ($\mathcal{P}_1$). They may prematurely exit bus re-integration in the middle of a frame only to encounter further errors or disrupt the ongoing frame ($\mathcal{P}_2$). Finally, not leaving bus re-integration early enough can make nodes lose additional messages due to a single error ($\mathcal{P}_3$).

Data Consistency and Availability. CAN CC attacks could compromise network-wide data consistency, a fundamental property CAN aims to offer. Specifically, attackers can exploit *Vb* to let only the transmitter validate a frame but receivers do not. Therefore, we verify $\mathcal{P}_4$-$\mathcal{P}_5$ to identify whether CAN XL prevents similar attacks. $\mathcal{P}_4$ is a property exactly outlined in the standard which ensures that all nodes agree on what messages are sent [2]. $\mathcal{P}_5$ is a stronger requirement we derive, which ensures nodes agree on messages' content:

- $\mathcal{P}_4$: A frame can be simultaneously accepted as a valid frame either by all nodes or by no node.
- $\mathcal{P}_5$: If a message is corrupted (i.e., receivers read different bits from what the transmitter sent), no nodes validate it.

Exploiting *Vc*, CAN CC attacks can send overload frames to prevent the bus from returning to idle after frames complete. This compromises availability and stops subsequent communication from making progress. To prevent similar attacks, we derive the following property:

- $\mathcal{P}_6$: The bus eventually returns to idle after a frame ends.

Classic Error Handling. Error handling is a historically vulnerable mechanism that underlies *Va*. From the standard, we derive these design goals of error handling and verify them for error signalling enabled operation only:

- $\mathcal{P}_7$: Transmitters send error frames on encountering errors.
- $\mathcal{P}_8$: Transmitters' error frames let receivers detect errors.

Violations could cause transmit errors to remain undetected ($\mathcal{P}_7$) or fail to propagate to receiver nodes ($\mathcal{P}_8$). Moreover, CAN CC's passive error regeneration vulnerability (*Va*) can trigger repeated errors rapidly and disrupt communication by letting nodes detect further errors when sending error frames. To prevent such attacks, we derive this property:

- $\mathcal{P}_9$: A node does not encounter further errors when sending

an error frame.

Formal Specifications. In App. A, we show the properties' formal definitions in computational tree logic (CTL).

4.2 Threat Model

We assume a remote attacker has compromised an ECU on the CAN bus and is able to execute arbitrary code. This is in line with the most common threat model assumed by CAN security literature [13, 24, 27, 28], and can be achieved through ECUs' external connectivity channels, as demonstrated by many real-world attacks [3, 10, 12, 17, 18, 23, 25]. To thoroughly discover vulnerabilities, we consider two possible levels of attacker control over the protocol layers under this model, following a prior systematization [30]. ① HL attackers: The attacker controls the higher layer only and can transmit or receive valid frames, abiding by all link-layer rules such as arbitration and error handling. ② LL attackers: The attacker additionally controls the link layer. Previous works assumed such attackers require strong privileges (e.g., physical access) and considered them less often. Recent research [15, 19, 32] shows this assumption is not needed and LL attackers require only the same privileges as HL attackers. Remote attackers can achieve link-layer control on a significant portion of commercial ECUs using various techniques, such as by programming the ECU's GPIO to directly access the bus (i.e., bit-banging, which is implemented by several open-source libraries [9, 34]). Attackers can read or write individual bits on the bus when it is idle or busy, but must abide by physical layer rules. Thus, they cannot modify bits on the bus arbitrarily and can only modify the bus state from 1 to 0 in SIC mode. In many attack scenarios, CAN mechanisms such as error handling can maintain key properties. For example, network-wide data consistency may be maintained by making errors visible to all nodes. However, corner cases could still lead to successful attacks, which we aim to find.

4.3 Model Construction

We model a CAN XL network in nuXmv, with benign nodes and an attacker attached to the channel (Fig. 4). A network may contain an arbitrary number of nodes. However, all core protocol interactions except for arbitration are between a single transmitter and multiple receivers that behave identically. All receivers have the same protocol state machine and receive the same bits. Therefore, they can be modeled as a single receiver node. In the case of arbitration, its outcome is decided by a single winner and a contender with the closest priority. Therefore, two nodes are sufficient to capture core protocol interactions in both cases, and we adopt in our model.

MAC Sub-Layer Modeling. The node models the MAC sub-layer based on the standard. We model it as a finite state machine (FSM) synchronized to a clock whose period equals one bit duration. Nodes read/write a bit from/to the channel every clock cycle. We make this choice because CAN XL's lower layers provide mechanisms to ensure nodes are synchronized to the bit stream on the bus [2]. We model it using the nuXmv tool as it natively supports this synchronous FSM formalism. Formally, the MAC sub-layer is an FSM $M = (S, I, O, S_0, \delta)$. I/O is a finite set of inputs/outputs. For CAN XL, they are the bits received from/transmitted to the channel, with possible valuations $\{0, 1\}$. S is a finite set of states. In our case, it tracks a node's transmit/receive status and current protocol field. S_0 is the set of initial states (idle for CAN XL), and $\delta : S \times I \to S \times O$ represents the state transition function. After receiving a bit, depending on its value and the current state, the node transitions to the next state and outputs a bit. CAN XL has different states and transitions depending on the error signalling configurations. Thus, the node model has two variants: error signalling disabled and error signalling enabled. They model the transitions in Fig. 1 and 3 respectively at the bit level, with transitions from intermission back to idle to model unbounded message sequences. An error signalling enabled node is nondeterministically initialized to a node type (Tbl. 1), allowing interactions between any CAN generations to be analyzed. For error signalling enabled, we do not model error states transitions (Sec. 2). Instead, a node is nondeterministically initialized to and stays in an error state. This avoids long traces to reach the error-passive state. It is sufficient for the properties we analyze because error state transitions only occur instantaneously at frame boundaries, and nodes' behaviors during frame transmission/reception only depend on their current error states.

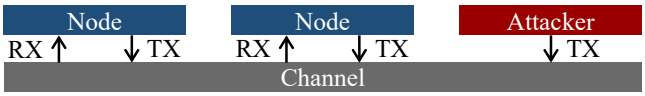


Figure 4: Model architecture.

Other Protocol Layers. Since we focus on the MAC sub-layer, we do not assume specific message schedules, payload semantics, or protocols at higher layers. Instead, nodes may have a pending transmission at any arbitrary time, whose payload is nondeterministically chosen (i.e., we consider all possible messages). The physical layer is modeled using a channel that receives the outputs from all nodes and the attacker and sends the resulting bit to nodes based on physical layer rules. We assume it operates in SIC mode (0s override 1s) due to its compatibility with all frame formats, node types, and error signalling options. In Sec. 8 and 9, we discuss other layers' impacts on MAC sub-layer vulnerabilities.

Attacker. We model an LL attacker (Sec. 4.2) as it possesses all abilities of an HL attacker, allowing us to discover vulnerabilities exploitable by either attacker type. We then manually separate the discovered vulnerabilities into those exploitable by LL attackers only or by either type. The attacker model nondeterministically outputs a 1 or 0 every clock cycle. This captures all possible attacker behaviors, including injecting

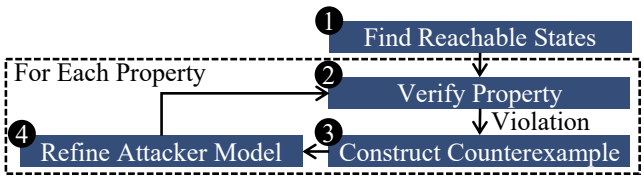


Figure 5: Analysis workflow.

valid messages or bits in any protocol fields.

Addressing Tractability. Since CAN XL's MAC sub-layer operates on bits, it must be modeled at the bit level to accurately preserve behavioral details. As CAN XL frames have long and diverse fields, and we consider all possible messages with a versatile LL attacker that can manipulate any protocol fields, the protocol has a large state space. Complexity grows further in mixed networks, as different node types across CAN generations interact, each transmitting multiple frame formats and having distinct error states. These complexities lead to state explosion and make formal analysis intractable. To address this, we devise two domain-specific modeling techniques based on CAN XL's structure:

Sequential Decomposition: Since frames comprise fields processed sequentially, we decompose the protocol model based on its fields. This breaks down the protocol's state space into smaller portions without losing precision. It allows us to analyze them separately, which is more tractable than exploring all states as a whole. Formally, we decompose the protocol FSM $M = (S, I, O, S_0, \delta)$ into $\{M_0, M_1, ..., M_n\}$, where $M_i = (S_i, I, O, S_{0,i}, F_i, \delta_i)$ is the FSM of the field i. S_i/δ_i denotes the states/transition function restricted to i, and $S_{0,i}/F_i$ denotes the starting/reachable ending states of i. To connect fields, a field's starting states must equal the union of all its immediate predecessor fields' ending states:

$$\forall j.\ S_{0,j} = \bigcup_{i:\mathsf{next}(i,j)} F_i \tag{1}$$

where $\mathsf{next}(i, j)$ means field i is j's immediate predecessor. This ensures the decomposition is sound: any partial execution trace of a field is contained in a full protocol execution. As Eq. 1 ensures every starting state of each field is reachable via a trace in one of its immediate predecessor fields, a trace in field i can be extended by a trace in field $i-1$. Inductively, the trace can be extended to field $i-2$, etc, until the protocol's initial state, forming a full protocol execution.

Concretely, we decompose CAN XL into: ① a precise model of the common arbitration field shared by all frame formats; for each frame format, ② an abstracted (explained next) model of the variable-value control to CRC fields coded with bit stuffing and ③ a precise model of the fixed-format ACK to intermission fields that do not use bit stuffing; and finally, ④ precise models for special frame formats (e.g., error frames) that may be transmitted from any field.

Algorithm 1 Find Reachable States

```
1:  F ← {∅, ∅, ..., ∅}
2:  S_0 ← {idle, ∅, ..., ∅}
3:  repeat
4:      for each field j do
5:          S_0[j] ← ⋃_{i:next(i,j)} F[i]
6:          repeat
7:              Check AF F[j] with nuXmv
8:              if Found counterexample σ then
9:                  F[j] ← F[j] ∪ σ(|σ| − 1)
10:             end if
11:         until No more counterexample
12:     end for
13: until F does not change
```

Abstraction: We abstract away CRC computation by choosing the values of the CRC fields nondeterministically (i.e., unconstrained by the payload), which over-approximates the set of possible messages. To model CRC error detection, we over-approximate receivers' behaviors by allowing them to nondeterministically choose whether to detect a CRC error at protocol fields where CRC checks occur. Although these abstractions limit our ability to prove CRC-algorithm-specific properties (e.g., what payload modifications cause CRC collisions), they are suitable for discovering MAC sub-layer vulnerabilities as they do not rule out any MAC sub-layer behaviors. They may introduce spurious counterexamples, but we experimentally validate all discovered vulnerabilities (Sec. 7.1) to ensure all findings correspond to real protocol behaviors.

4.4 Analysis Workflow

We design a formal analysis workflow leveraging CAN XL's sequential field-oriented structure (Fig. 5).

Find Reachable States. We first compute the protocol's reachable states (Fig. 5-①) using the decomposed models by solving the least fixpoint of Eq. 1. This is done once before verifying any properties. We use an automated script to perform this computation using Algorithm 1. We initialize the arbitration model's starting states to idle and all other fields' starting and ending states to the empty set (lines 1-2). For each model, we set its starting states to the union of all its immediate predecessor fields' ending states (line 5). To model unbounded message exchange, the starting states of the arbitration model include ending states of the ACK-intermission models. We use nuXmv to verify the property **AF** F, which states the model always finally reaches a state within the set F. If F does not cover all ending states, nuXmv generates a counterexample. We extract its ending states, add them to F, and repeatedly verify **AF** F until no counterexamples are found (lines 6-11). We repeat this for all models until no model's ending states change, at which point the decomposed models collectively capture all reachable states of the protocol.

Table 2: Discovered new CAN XL standard vulnerabilities and exploits.

Error Signalling	Vulnerability	Violated Property	Similar CAN CC Vulnerability	Exploit	Impact
Disabled	*V1.1*. Undetected Tampering	$\mathcal{P}_4$	None	Message Interception	Availability
	V1.2. Undetected Tampering	$\mathcal{P}_5$		Message Overwriting	Integrity
	V2.1-V2.3. Tx/Rx State Mismatch	$\mathcal{P}_3$, $\mathcal{P}_4$	None	ID Mapping	Confidentiality
				Target Node Shutdown	Availability
				Message Replacement	Integrity
				Data Consistency Violation	Integrity
Enabled	*V3.1*. Mixed Network State Mismatch	$\mathcal{P}_3$, $\mathcal{P}_4$	None	Inconsistent Frames	Integrity
				Advanced Spoofing	Integrity
				Data Consistency Violation	Integrity
	V3.2. Mixed Network State Mismatch			Node Type Identification	Confidentiality
				Target Node Shutdown	Availability
	V4. Premature Re-Integration Exit	$\mathcal{P}_2$, $\mathcal{P}_4$, $\mathcal{P}_9$	None	Data Consistency Violation	Integrity
Both	*V5*. Forced Re-Integration	$\mathcal{P}_1$, $\mathcal{P}_6$	None	Full Network DoS	Availability
Enabled	*V6*. Rx-Only Validation	$\mathcal{P}_4$, $\mathcal{P}_8$, $\mathcal{P}_9$	*Vb*	Double Receive	Integrity
	V7. Error Regeneration	$\mathcal{P}_9$	*Va*	Set Victim's Error Counter	Availability

Algorithm 2 Construct Counterexample

Input: nuXmv's counterexample (σ), Current field (j)
Output: The full counterexample

```
1:  procedure CONSTRUCT(σ, j)
2:      if j is arbitration then
3:          Check idle → ¬EF σ(0) with nuXmv
4:          if Found counterexample σ' then
5:              return σ' · σ
6:          end if
7:      end if
8:      for i : next(i, j) do
9:          Check ¬EF σ(0) with nuXmv
10:         if Found counterexample σ' then
11:             σ ← σ' · σ
12:             return CONSTRUCT(σ, i)
13:         end if
14:     end for
15: end procedure
```

Verify Properties and Construct Counterexamples. We verify the properties with nuXmv (Fig. 5-②). Since we have previously computed the protocol's reachable states exhaustively, we use binary decision diagram (BDD) based algorithms to verify properties over the entire protocol state space. Each property is verified using relevant protocol field models. For example, $\mathcal{P}_2$ is verified using all models as it must hold for all fields, while $\mathcal{P}_6$ is only verified using the ACK-intermission models as it applies only to a frame's ending fields.

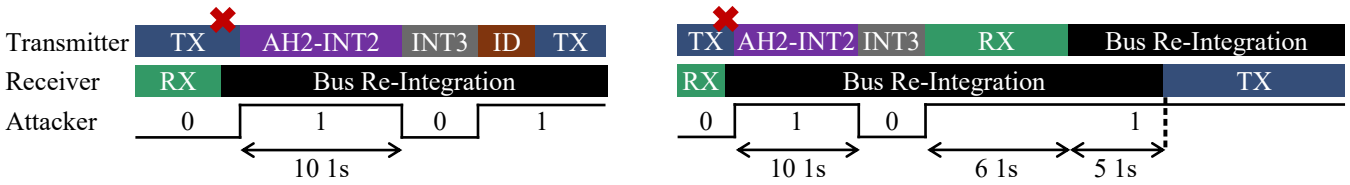
If a property is violated, nuXmv generates a counterexample of a protocol execution over the modeled fields. To facilitate root cause investigation, we use Algorithm 2 to automatically construct full traces where nodes reach the property violation starting from the idle state (Fig. 5-③). We extract the starting state $\sigma(0)$ from nuXmv's counterexample σ for the field j. We check the property $\neg$**EF** $\sigma(0)$ on every field i of j's immediate predecessor fields to find if an execution trace leads to $\sigma(0)$. If one is found, we prepend it to σ and repeat prior steps on i's immediate predecessor fields (lines 8-14) until we reach arbitration. We check $idle \rightarrow \neg$**EF** $\sigma(0)$ to find a counterexample from the idle state to $\sigma(0)$. If one is found, a full counterexample is constructed (lines 2-7). Otherwise, it means the violation is only reached after multiple consecutive messages and we repeat the steps in lines 8-14.

Refine Attacker Model. If a property is violated, we manually analyze its counterexample to find the attacker actions causing it. We refine the attacker model (Fig. 5-④) so it does not perform the same actions and verify the property again to discover other violating scenarios. This is repeated until no more violations are found.

Computational Performance. On a laptop with an Intel Core i7-12700H processor and 32GB RAM, computing reachable states takes 4min for the error signalling disabled configuration and 13min for error signalling enabled. To show the necessity of our model decomposition and abstraction techniques for ensuring analysis tractability, we construct a monolithic model for error signalling disabled with the CRC modeled precisely. Reachable state computation times out after one hour. Any subsequent analysis becomes infeasible.

5 Analysis Results

We discover that CAN XL inherits all CAN CC vulnerabilities (*Va*–*Vc*) when error signalling is enabled. This not only happens in mixed networks due to backward compatibility, but also in pure CAN XL settings. Moreover, we discover seven new standard vulnerabilities (Tbl. 2). When error signalling is enabled, CAN XL introduces *V6* and *V7*. They violate similar properties as known CAN CC vulnerabilities (*Vb* and *Va*) but exploit CAN XL's new frame formats. *V3* and *V4* arise due to mixed CAN generation interactions. When error signalling is disabled, CAN XL introduces *V1* and *V2*. Finally, *V5* applies to both error signalling configurations. We detail *V1*-*V5* in this section and *V6*-*V7* in App. B.



(a) Receivers in bus re-integration.

(b) Transmitter in bus re-integration.

Figure 6: Tx/Rx state mismatch by injecting 0 in INT3 (*V2.3*).

5.1 Error Signalling Disabled Vulnerabilities

V1. Undetected Tampering. We discover two violations of data consistency properties. ***(V1.1)***: If an attacker injects an error in a message and sends an ACK, the transmitter validates the message but receivers do not, violating $\mathcal{P}_4$. ***(V1.2)***: If an attacker overwrites 1s with 0s in a message's data and CRC to alter its content while keeping the CRC valid, both the transmitter and receivers validate the corrupted message, violating $\mathcal{P}_5$. Their root cause is the standard specifying that receivers shall detect errors, but the transmitter shall not detect or signal errors from SOF to AH2 and shall consider a transmission successful if it receives a valid ACK. This specification stems from the physical layer restrictions of the fast mode (Sec. 2), which enables higher bit rates but makes it difficult for transmitters to detect errors [16]. While well-motivated, it inadvertently enables the new exploits below.

Exploit 1: Message Interception. LL attackers can directly inject an error in a message and then send an ACK to exploit *V1.1*. HL attackers can perform simultaneous transmission using a longer message as explained in App. C–*V1*.

Impact. This can be used to launch targeted DoS attacks on a safety-critical message as receivers do not receive it, and can bypass retransmission-based recovery as the transmitter believes the transmission is successful. It is not possible for CAN CC, where the transmitter also detects errors. It incurs only one error and an intrusion detection system (IDS) monitoring higher-layer events can hardly distinguish it from a normal transient error.

Exploit 2: Message Overwriting. Exploiting *V1.2*, an HL attacker can overwrite a message by simultaneously transmitting another message with the same length that differs from it only by 1 to 0 bit flips in the data and CRC.

Impact. This allows silent message integrity corruption, as the transmitter believes the original message is successfully transmitted, but receivers get a different payload. It does not cause errors and is undetectable by a higher-layer IDS.

V2. Tx/Rx State Mismatch. We discover that in certain scenarios, if a message contains errors before its ACK, some nodes are in the bus re-integration state while others are not when a new message arrives, violating $\mathcal{P}_3$. This also violates $\mathcal{P}_4$ as the new message is only validated by nodes not in bus re-integration. This has two root causes. First, the standard specifies that errors before ACK shall let receivers enter bus re-integration but the transmitter shall remain in normal operation, introducing a mismatch between their states. Second, to let receivers exit bus re-integration, the next message's SOF must be separated by at least 11 1s. However, this condition is not met in the scenarios below:

(V2.1) *Short Pulses Between Messages:* Injecting short 0 pulses between a message's ACK and the next SOF forces receivers to stay in bus re-integration, as the standard specifies they shall restart counting 11 1s after receiving a 0 pulse. If such 0 pulses do not overlap with the transmitter's sample point, they will not cause errors or impact the transmitter.

(V2.2) *0 in AH2:* When an attacker injects a 0 in AH2, the next SOF may only be separated by 10 1s (ACK + ACK delimiter + 5 bit EOF + 3 bit intermission).

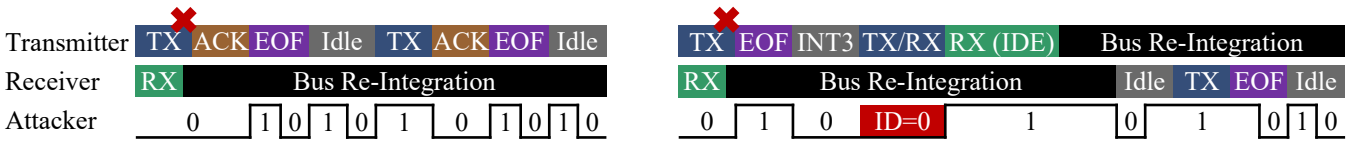
(V2.3) *0 in INT3:* If an attacker injects a 0 in INT3, the transmitter shall treat it as the SOF of a new message, which is separated from the previous by 10 1s (AH2 + ACK + ACK delimiter + 5 bit EOF + INT1 + INT2–Fig. 6a).

An attacker can control whether only the transmitter or only the receivers remain in bus re-integration during the next message. He can let the transmitter transmit a message after the new SOF (Fig. 6a) so receivers stay in bus re-integration. Alternatively, the attacker can transmit a message and let the previous transmitter detect a receive error after one or more 1s and enter bus re-integration. An example is transmitting 6 1s after the new SOF (Fig. 6b) to cause a stuff error which violates frame coding (Sec. 2). At this time, the receivers in bus re-integration have received one or more 1s. They will resume operation earlier than the transmitter and may transmit a new message while the transmitter is in bus re-integration.

Exploit 1: ID Mapping. *V2* enables LL attackers to disable all but one node (Fig. 6a), and *V2.1* allows them to keep nodes in bus re-integration when the bus is idle by injecting short pulses. Exploiting them, an attacker can persistently disable all but one node to identify all messages this node transmits. Repeating this on all nodes, he can map all IDs each node transmits. Fig. 7a shows the detailed steps:

Step ①: The attacker listens to bus traffic and records all periodic IDs and their periods.

Step ②: The attacker picks a random ID, injects an error to push receivers into bus re-integration, and sends an ACK so its transmitter believes the transmission is successful. The attacker then sends a small pulse in the message's EOF to



(a) ID mapping.

(b) Target node shutdown.

Figure 7: Tx/Rx state mismatch (*V2*) exploits.

keep receivers in bus re-integration.

Step ③: The attacker keeps sending small pulses when the bus is idle to keep receivers in bus re-integration. If he receives an ID, it must be from the same transmitter as the initially chosen ID. The attacker records this and repeats step ② on the message, but without injecting an error in it.

Step ④: The attacker continues step ③ for a duration longer than the longest message period recorded in step ① to ensure he can observe all periodic IDs. The attacker then stops sending pulses to bring all nodes back into normal operation.

Step ⑤: The attacker picks an unmapped ID, performs steps ② to ④, and repeats until all periodic IDs are mapped.

Step ⑥: To map aperiodic IDs, an attacker uses step ② to shut down all nodes except its transmitter when he sees an aperiodic ID. He performs step ③ until he sees a mapped periodic ID and maps the aperiodic ID to the same transmitter.

Impact. ID mapping is useful for reverse engineering, as information on what messages each node transmits is proprietary. It can help attackers identify each node's functionality to launch targeted attacks. For CAN CC, the same goal can be achieved by exploiting error states (*Va*), which are removed from CAN XL's error signalling disabled specifications. However, CAN XL's new features make it vulnerable to the same attack.

Exploit 2: Target Node Shutdown. *V2* also allows an LL attacker to disable a single node (Fig. 6b). With *V2.1*, it enables persistent target node shutdown using these steps (Fig. 7b):

Step ①: The attacker injects an error in the victim's message and sends a 0 in INT3 to keep receivers in bus re-integration.

Step ②: After INT3, the attacker sends a partial message with ID 0 ending with a recessive IDE (Fig. 2) to win arbitration. As a receiver, the victim detects a form error at the recessive IDE and enters bus re-integration.

Step ③: The attacker waits for 10 bits until other nodes resume normal operation. The attacker then sends short pulses in every message's EOF and whenever the bus is idle to keep the victim in bus re-integration.

Impact. An attacker can use this to launch targeted DoS attacks by shutting down a node controlling critical system functions. Defenses can use it to disable an attacker node and prevent it from impacting the system's operation. The same can be achieved for CAN CC exploiting error states (*Va*), which are not present in CAN XL when error signalling is disabled.

Exploit 3: Message Replacement. Exploiting *V2* with *V1*, LL attackers can launch a new attack not possible for CAN CC. An attacker first intercepts (*V1*) a message, then exploits *V2.3* to send a 0 in INT3 after the intercepted message and push its transmitter into bus re-integration. As soon as receivers resume operation while the transmitter is still in bus re-integration, the attacker injects a fake message with the same ID as the transmitter's message.

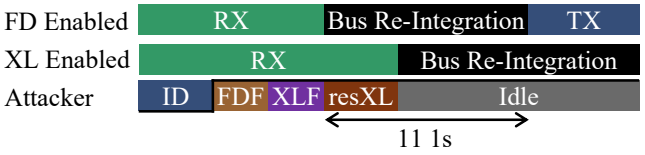


Figure 8: Mixed network state mismatch (*V3.1*).

Impact. This is a stealthy spoofing attack that can evade several types of defenses, such as those monitoring message timings [31,38] or letting each ECU monitor for its ID [14,22]. This is because the attack effectively replaces the original message with a fake one as receivers only see the fake message transmitted at almost the expected time. Moreover, the transmitter in bus re-integration cannot observe the fake message.

Exploit 4: Data Consistency Violation. *V2* allows LL attackers to compromise network-wide data consistency in many ways, e.g., disabling selected nodes for arbitrary durations to prevent them from receiving certain messages or inject messages that only the remaining ones can receive.

Impact. This violates CAN's fundamental network-wide data consistency property. It undermines higher-layer control and protocols that rely on consistency at the lower layer, causing inconsistent system states and control decisions.

5.2 Error Signalling Enabled Vulnerabilities

V3. Mixed Network State Mismatch. In a mixed network, we discover scenarios violating $\mathcal{P}_3$, where some nodes are in bus re-integration while others are not when a message arrives, even if they support the frame. This also violates $\mathcal{P}_4$ as only nodes not in bus re-integration validate the message.

***(V3.1)** Partial Frames:* The standard specifies that different node types detect unsupported frames at distinct protocol fields (Sec. 2). Partial unsupported frames ending with 1s cause them to enter and leave bus re-integration at different times (e.g., FD enabled nodes enter and leave bus re-integration earlier than XL enabled nodes by one bit when receiving a partial frame ending with 1 in resXL–Fig. 8). This timing mismatch may cause some node types to be in bus

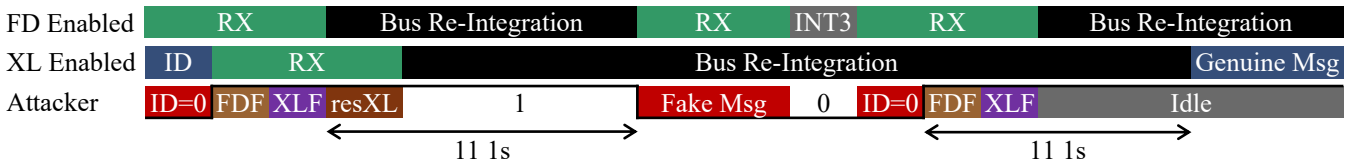


Figure 9: Mixed network state mismatch (*V3*) exploit: inconsistent frames.

re-integration while others are not when a message is transmitted. An attacker can exploit this to push selected node types into bus re-integration.

***(V3.2)** 0 in INT3 or Short Pulses Between Messages:* Similar to *V2.1* and *V2.3*, injecting 0s in INT3 or short pulses between messages forces nodes that do not support the frame to stay in bus re-integration when a new message arrives.

Exploit 1: Node Type Identification. An LL attacker can exploit *V3.2* to test whether a node has a specific type or another type supporting more frame formats. When the attacker sees a message transmitted by the node, he wins arbitration with a higher priority ID and sends a potentially unsupported frame (e.g., XL frame for FD enabled nodes) and a 0 in INT3. He then checks if the node retransmits the message after INT3. The absence of a retransmission indicates that the node has the expected type, does not support the frame, and has entered bus re-integration. Otherwise, it supports the frame and has another type. Interference from other traffic during the test is handled as described in App. C–*V3*.

Impact. Node type identification reveals proprietary network configuration information remotely. It is the prerequisite for targeted attacks on specific node types as described below.

Exploit 2: Inconsistent Frames. LL attackers can exploit *V3.1* to launch a new attack that lets nodes of the same type as a message's transmitter receive the genuine message while nodes of other types receive a fake one. To illustrate its steps, Fig. 9 shows an example where an attacker lets all XL enabled nodes receive a genuine message while other nodes receive a fake one when an XL enabled node sends a message:

Step ①: The attacker wins arbitration over the target message and pushes all nodes into bus re-integration (e.g., by sending a partial XL frame with 1 in resXL).

Step ②: When nodes of a type different from the transmitter's (i.e., FD enabled) resume operation and nodes of the same type are in bus re-integration, the attacker sends a fake message with the same ID so different-type nodes receive it.

Step ③: The attacker sends a 0 in the fake message's INT3 and pushes different-type nodes into bus re-integration (e.g., by sending a partial XL frame with ID 0 ending with XLF).

Step ④: The transmitter and nodes of the same type (i.e., XL enabled) exit bus re-integration earlier. It retransmits the genuine message and all nodes of its type receive it.

Impact. This is a targeted spoofing attack that deceives selected nodes. Victims develop an inconsistent view of the system state relative to other nodes, leading to conflicting control decisions. Moreover, it is useful for evading payload-inspecting IDSs [21,36] by letting the IDS receive the genuine message while nodes of other types receive a fake one.

Target Node Shutdown, Advanced Spoofing, and Data Consistency Violation. LL attackers can exploit *V3.2* to shut down target nodes by keeping them in bus re-integration. However, the victim may not be a single node, but all nodes of a specific type. This is still useful in many scenarios. For example, if a network has only one node of a particular type. Similar to message replacement (*V2*), LL attackers can exploit *V3.1* to spoof a message after pushing its transmitter into bus re-integration. Although it cannot let the transmitter believe it has transmitted the message successfully, it is still useful for evading defenses that let transmitters monitor for their own IDs. LL attackers can also exploit *V3.1* to violate data consistency among different types of nodes in various ways (e.g., drop messages only for specific types or inject messages only specific types can receive).

V4. Premature Re-Integration Exit. In a mixed network, we discover that when a message is interrupted with errors and all nodes in normal operation are error-passive (Sec. 2), nodes that do not support the frame exit bus re-integration before the passive error frame ends, violating $\mathcal{P}_2$. This is because passive error frames contain 14 1s, more than what is needed to exit bus re-integration (11 1s). If a node transmits a message right after exiting bus re-integration, it collides with error-passive nodes' error delimiter and error-passive nodes detect another error, violating $\mathcal{P}_9$. Due to the error, error-passive nodes cannot receive the message. They transmit another passive error frame which is invisible to other nodes. Thus, other nodes can still receive the message, violating $\mathcal{P}_4$.

Exploit: Data Consistency Violation. After pushing XL enabled nodes with protocol exception disabled (Sec. 2) into error-passive, LL attackers can exploit *V4* to violate data consistency between them and other nodes in many ways. Fig. 10 shows an example. When a node with protocol exception enabled transmits a message, the attacker can let nodes with protocol exception disabled receive a fake message, but other nodes receive the genuine one:

Step ①: The attacker wins arbitration with a higher-priority ID after seeing the target message's ID.

Step ②: The attacker sends a partial XL frame ending with a 1 in resXL to push nodes with protocol exception enabled into bus re-integration. Nodes with protocol exception disabled transmit a passive error frame.

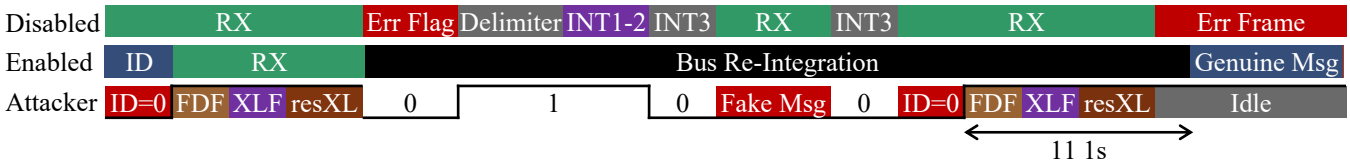


Figure 10: *V4* exploit: inconsistent frames targeting XL enabled nodes with different protocol exception configurations.

Step ③: The attacker overwrites the passive error flag with 0s, sends a 0 in INT3, and the fake message. Nodes with protocol exception enabled stay in bus re-integration. Nodes with protocol exception disabled receive the fake message.

Step ④: The attacker sends a 0 in the fake message's INT3 and a partial XL frame ending with a 1 in resXL. Nodes with protocol exception disabled transmit a passive error frame.

Step ⑤: During the passive error frame, nodes with protocol exception enabled exit bus re-integration and retransmit the original message. Nodes with protocol exception disabled cannot receive it but other nodes can.

Impact. Attacks to violate data consistency exploiting *V3.1* require pushing nodes into bus re-integration. They cannot target XL enabled nodes with protocol exception disabled as they do not enter bus re-integration. However, *V4* exposes these nodes to similar attacks after an attacker pushes them into the error-passive state. Such attacks significantly undermine the robustness of higher-layer control and protocols that require network-wide data consistency provided by CAN XL.

5.3 Error Signalling Agnostic Vulnerability

V5. Forced Re-Integration. We find that an attacker can send 0s when nodes are in bus re-integration to force them to stay in this state indefinitely, violating $\mathcal{P}_1$ and $\mathcal{P}_6$. This is because the condition to exit bus re-integration as specified in the standard (receiving 11 consecutive 1s) is never met.

Exploit: Full Network DoS. When error signalling is disabled, LL attackers can push all nodes into bus re-integration (e.g., by injecting errors in the EOF) and send a stream of 0s to keep them in bus re-integration. The attackers can send 0s perpetually to stop all communication indefinitely or for a finite duration to delay messages so they miss their deadlines. HL attackers can delay bus traffic for finite durations using simultaneous transmission (App. C–*V5*).

Impact. This compromises the availability of the network, rendering it non-operational. It is stealthy as keeping nodes in bus re-integration does not generate messages or errors detectable by an IDS. For CAN CC or CAN XL with error signalling enabled, attackers can achieve the same by sending overload frames (*Vc*). Thus, *V5* is most useful when error signalling is disabled, where overload frames are not used.

Table 3: Comparing CAN CC/XL attack effectiveness.

Impact	Attack Category	Possible in CAN CC?	Protocol	Attacker Control	Swiftness	Persistence	Coverage	Stealth
C	ID Mapping	✓	①	LL (↓)	↗	≈	≈	↗
			②	HL (≈)	≈	≈	≈	≈
	Sniffing	✓	①/②	HL (≈)	≈	≈	≈	≈
	Type Identification	X	②	LL				
I	Advanced Spoofing	✓	①	LL (↓)	↗	≈	≈	↗
			②	HL (≈)	↗	≈	≈	↗
	Data Consistency Violation	✓	①	LL (≈)	≈	≈	↑	≈
			②	LL (≈)	≈	≈	↑	≈
	Injection	✓	①/②	HL (≈)	≈	≈	≈	≈
A	Full Network DoS	✓	①	HL/LL (↑)	≈	≈	≈	≈
			②	LL (≈)	≈	≈	≈	≈
	Target Node Shutdown	✓	①	LL (↓)	↗	↓	≈	↗
			②	HL (≈)	↗	≈	≈	↗
	Flooding	✓	①/②	HL (≈)	≈	≈	≈	≈
	Error Injection	✓	①/②	HL (≈)	≈	≈	≈	≈
	Interception	X	①	HL				

①: CAN XL error signalling disabled ②: CAN XL error signalling enabled

6 Comparative Security Analysis

To compare MAC sub-layer security, we compare how effectively an attacker can launch different attack categories on CAN CC and CAN XL networks exploiting their standard-level vulnerabilities. Tbl. 3 shows the considered attack categories. If a category is possible for both protocols, we compare its minimum required attacker control (HL or LL), swiftness, persistence, coverage (how many different attack variants within the same category that achieve different goals are possible), and stealth. For stealth comparison, we consider higher-layer anomalies (i.e., errors) caused by the attack, since most CAN IDS today monitor the higher layer [30]. A ↑/↓ means CAN XL improves/degrades relative to CAN CC for a criterion. A ↗ means a criterion can be improved with stronger (LL vs. HL) attacker control.

ID Mapping. Exploiting *Va*, HL attackers can push nodes to the error-passive state and map their transmitted IDs for CAN CC and CAN XL with error signalling enabled. With error signalling disabled, it can be achieved by exploiting *V2*. Despite requiring stronger LL control, ID mapping is faster for CAN XL. Attackers can isolate a transmitter using

a single message and do not need to inject repeated collisions to keep nodes in the error-passive state. Furthermore, the time to map a CAN CC ECU is proportional to the number of IDs it transmits [28]. However, the time to map a CAN XL ECU depends only on its longest message period. The attack is also less detectable, as an IDS monitoring the higher layer only observes a single error instead of repeated collisions.

Node Type Identification. A unique attack for mixed networks, it is possible by exploiting *V3.2*.

Advanced Spoofing. For CAN CC and CAN XL with error signalling enabled, attackers can exploit *Va* to launch advanced spoofing attacks by pushing a node to bus-off/error-passive and spoofing/hijacking its messages. However, pushing nodes into different error states takes time and causes multiple errors, making the attacks detectable. Contrastingly, when error signalling is disabled, attackers can overwrite (*V1.2*) or replace (*V2*) messages. They are faster and less detectable as they do not require additional setup steps and cause no or only one error. In mixed networks, attackers can spoof messages after pushing the transmitter into bus re-integration (*V3.2*) or launch inconsistent frame attacks (*V3.1*), without requiring additional setup time or generating errors.

Data Consistency Violation. By violating $\mathcal{P}_4$ only, *Vb* allows one form of data consistency violation in CAN CC: letting receivers validate a message but the transmitter does not. This can be achieved for CAN XL when error signalling is enabled as it inherits *Vb*. In contrast, many more forms of consistency violations are possible among arbitrary nodes when error signalling is disabled by exploiting *V2*, or among nodes of different types in mixed networks by exploiting *V3.1* and *V4*.

Full Network DoS. *Vc* allows LL attackers to launch full network DoS attacks on CAN CC and CAN XL with error signalling enabled. With error signalling disabled, the same can be achieved by exploiting *V5*. In contrast, it only requires HL control to introduce controlled, finite bus delays (App. C–*V5*).

Target Node Shutdown. *Va* allows HL attackers to shut down target nodes by pushing them into the bus-off state for CAN CC and CAN XL when error signalling is enabled. When error signalling is disabled or in mixed networks, LL attackers can shut down nodes by exploiting *V2* or *V3*. This is often faster than CAN CC. The fastest technique for CAN CC needs 255 nominal bit durations [27]. *V2* or *V3* allows attackers to disable a node in one message duration plus 16 bits (SOF-IDE in a message with ID 0, Fig. 7b), which is often faster. It may be less persistent as pulses must be injected in every message's EOF and whenever the bus is idle to keep nodes in bus re-integration. However, in Sec. 7.2.1, we achieve a high suppression rate experimentally, making the attack highly effective in practice. Finally, this attack is less detectable as it causes fewer errors: one error when error signalling is disabled or no errors in mixed networks. Contrastingly, its CAN CC counterpart incurs at least 32 errors.

Interception. This attack is only possible for CAN XL networks with error signalling disabled by exploiting *V1.1*.

Sniffing, Injection, Flooding, and Error Injection. For completeness, we note that HL attackers can launch these attacks for both CAN CC and CAN XL using the same techniques (by sniffing/injecting messages directly or errors with simultaneous transmission) with the same effectiveness.

Conclusion. In both error signalling configurations, CAN XL's MAC sub-layer vulnerabilities enable all CAN CC attack categories. Moreover, when error signalling is disabled or in mixed networks, its new vulnerabilities enable new attacks and make several attacks faster and stealthier. Although some new or improved attacks require LL control, this is achievable remotely on many standard ECUs and has been demonstrated in practice (Sec. 4.2), and does not make them less feasible. Thus, we argue that *from a strict MAC sub-layer standpoint, changes by CAN XL worsen security*.

7 Evaluation

We perform two evaluations. First, we validate the presence of the discovered vulnerabilities in commercial CAN XL protocol controllers. Next, we show the vulnerabilities' exploitability by using them to construct and evaluate two multi-stage attacks from reconnaissance to execution.

7.1 Vulnerability Presence Validation

Setup. We use two commercial CAN XL interfaces: PEAK-System PCAN-USB XL and Vector VN1641. We set up a network with each device and connect an STM32 microcontroller as an attacker. Our formal analysis identifies one or more counterexamples per vulnerability, each triggered by different attacker actions. For each counterexample, we set up nodes to transmit/receive messages and the attacker to inject signals as specified, and verify whether the observed node behaviors match the counterexample.

Standard Vulnerabilities. Tbl. 4 shows their presence.

Table 4: Standard vulnerabilities in commercial devices.

Device	*V1*	*V2*	*V3*	*V4*	*V5*	*V6*	*V7*
PCAN XL	✓	✓	✓	✓	✓	✓	✓
VN1641	✓*	✓*	✓	✓	✓	✓	✓

PCAN XL: All vulnerabilities can be triggered by the same attacker actions as our formal analysis discovers.

VN1641: All vulnerabilities are present but *V1* and *V2* require different attacker actions to trigger because its implementation deviates from the standard as explained below.

Deviations From the Standard. Apart from standard-level vulnerabilities, we also notice behavioral deviations between the devices and our standard model in some error corner cases of the protocol. These deviations are intentional by vendors or due to implementation bugs.

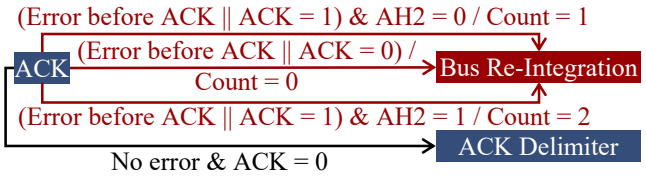


Figure 11: Modeling Vector VN1641.

PCAN XL: When validating *V2*, we notice that a node in bus re-integration with a pending transmission transmits the message when it receives a 0 after 10 1s instead of staying in bus re-integration. PEAK-System confirms this to be an implementation bug and has fixed it with a firmware update.

VN1641: When validating *V1*, we notice that a transmitter detects errors in the arbitration phase between SOF and AH2 instead of ignoring them. In this case, it does not validate a frame even if it is acknowledged. When validating *V2*, we notice a transmitter handles errors differently from the standard. ①: When it detects an error before ACK, it enters bus re-integration after ACK instead of remaining a transmitter. Moreover, if AH2 is 0, the next message is transmitted 11 1s after AH2 instead of the expected 10. ②: In some cases when it receives a 0 between ACK and INT3, it enters bus re-integration and exits after 9 1s instead of 11 1s. Vector informed us that behavior ① is intentionally implemented to prevent receivers from losing two consecutive frames in certain error scenarios. Behavior ② is an implementation bug.

VN1641 Formal Modeling and Analysis: To understand the impacts of its intentional standard deviations on *V1* and *V2*, we model VN1641, reflecting how it handles errors after ACK and counts the number of received 1s (Fig. 11). We find it still vulnerable to *V1* and *V2*. *V1* persists since a transmitter does not detect errors in the XL data phase. *V2* can be triggered by injecting an error, sending an ACK to prevent the transmitter from entering bus re-integration, and performing the actions in *V2.1* or *V2.3*. We validate both *V1* and *V2* with the device.

7.2 Multi-Stage Attacks

Since no real vehicles implement CAN XL yet, we evaluate attacks on a testbed replicating a real vehicle's CAN bus. To compare CAN XL attacks to similar CAN CC attacks, the testbed replicates the vehicle traffic used by [28] to evaluate several CAN CC attacks. The vehicle has 4 ECUs transmitting 50 periodic messages with periods from 9ms to 5s and a 34% busload. We use the four channels of VN1641 to simulate the ECUs, transmitting messages with an arbitration bit rate of 500kbit/s and XL data bit rate of 1Mbit/s. Messages have the same IDs and periods as in [28] and contain dummy data. We connect an STM32 microcontroller to the bus. It acts as an LL attacker by directly manipulating the bus using its GPIO.

7.2.1 Error Signalling Disabled Network

We show an attack on a pure CAN XL network with error signalling disabled, the de facto setup of future high-speed CAN XL networks. The attack proceeds in three steps:
Step ①: The attacker performs reconnaissance to determine what messages each ECU transmits.
Step ②: The attacker identifies each ECU's functionality by mapping the source of aperiodic response messages to diagnostic requests, similar to [28].
Step ③: The attacker executes targeted attacks, including disabling a safety-critical ECU he has identified, intercepting, and replacing its critical messages.
Step ①: ID Mapping. We exploit *V2* to identify every message's transmitter. Since the longest message period is 5s, to be conservative, we spend at least 30s mapping each ECU to ensure we can observe all its messages during this time. We map all 50 periodic IDs with 100% accuracy in about 3min (Tbl. 5). In [28], it takes an HL attacker 9min to map the same CAN CC network, with the majority of time spent on pushing ECUs into error-passive. CAN XL ID mapping does not require this, making it much faster. Although [28] does not evaluate ID mapping by LL attackers, we expect CAN XL's speed advantages to hold in this setting as well. This is because the need to repeatedly push the victim into error-passive is a fundamental limitation of CAN CC regardless of attacker type. Moreover, our reported time is not a lower bound. We can further reduce the time to map each ECU, provided it exceeds the longest message period.

Table 5: ID mapping performance.

	ECU1	ECU2	ECU3	ECU4
Accuracy	100%	100%	100%	100%
Time	41s	36s	37s	74s
Total Time	CAN XL (LL)	≈3min		
	CAN CC (HL)	≈9min		

Mapping an ECU requires injecting pulses in its messages' EOF and whenever the bus is idle to keep all other ECUs in bus re-integration. This sometimes causes errors if pulses accidentally align with the ECU's sample point due to clock jitter. The ECU enters and remains in bus re-integration and stops transmitting. To account for this, we pause the mapping after the bus is idle for twice the longest message period (10s), bring all ECUs back into normal operation, and resume mapping the same ECU. We have two such errors when mapping ECU4, so it takes much longer to map than the other ECUs.
Step ②: Aperiodic ID Mapping. We configure each ECU to respond to a specific ID and map each aperiodic response message to its transmitter exploiting *V2*. We map all responses with 100% accuracy (Tbl. 7). For each ECU, the mapping only takes the duration between the response and its next periodic ID, which is bounded by its minimum message period.
Step ③: Target Node Shutdown. We exploit *V2* to shut down an ECU. We conduct the attack on every ECU for

30min. Similar to [28], we measure the suppression rate $S_{rate} = (n_{normal} - n_{attack})/n_{normal}$. n_{attack} is the number of victim messages received during 30min when the attack is running. n_{normal} is the expected number of victim messages in 30min without an attack. As shown in Tbl. 6, we achieve above 99.9% S_{rate} for every ECU. This is higher than the S_{rate} achieved by HL attackers in the same CAN CC network in [28], making the attack highly impactful in practice. While LL attackers (not evaluated in [28]) can likely achieve a higher S_{rate} than HL attackers, the improvement over our CAN XL results would be minimal at best, as we already achieve close to the theoretical maximum of 100% S_{rate}.

Table 6: Target node shutdown performance.

Protocol	Attacker Control	S_{rate}				Errors/s
		ECU1	ECU2	ECU3	ECU4	
CAN XL	LL	99.99%	99.96%	99.96%	99.98%	0.05
CAN CC	HL	97.5%	91.4%	85%	83%	>6.4

This attack injects short pulses in every message's EOF to keep the victim in bus re-integration. Similar to ID mapping, it may cause accidental errors stopping all communication. We account for this by pausing the attack when the bus is idle for twice the shortest message period (18ms) and resuming when the victim's ID is received. All messages we fail to suppress are transmitted when the attack is paused. Such errors occur about once every 40 seconds. Since we need to pause and inject another error to push the victim into bus re-integration, the attack causes one error every 20 seconds (0.05 errors/s–Tbl. 6). Contrastingly, for both HL and LL attackers, disabling a CAN CC victim requires 32 errors to push it into bus-off every time it recovers from bus-off after a timeout. The timeout for the vehicle observed in [28] is shorter than 5s. Thus, it requires >6.4 errors/s to keep a victim disabled, making the attack more detectable.

Step ③: Message Interception and Replacement. We exploit *V1.1* to intercept messages or *V2* to replace them with a different one. We conduct both attacks on one ID from each ECU 2000 times and achieve 100% success (Tbl. 7).

Table 7: Error signalling disabled network attacks.

	Aperiodic ID Mapping	Msg Intercept.	Msg Replace.
Success Rate	100%	100%	100%

7.2.2 Mixed Network

We show a two-step attack on a mixed network, another major deployment scenario of CAN XL:

Step ①: The attacker performs reconnaissance to identify ECUs' node types and protocol exception configurations.

Step ②: Based on the information from step ①, the attacker executes targeted attacks on specific node types.

Setup. We set each ECU's node type, transmitted frame type, and protocol exception configuration as shown in Tbl. 8. This covers all node types and protocol exception configurations except FD tolerant, which VN1641 does not support.

Table 8: Testbed node type setup.

	ECU1	ECU2	ECU3	ECU4
Type	XL Enabled	XL Enabled	XL Enabled	FD Enabled
Tx Frame	FD	XL	XL	FD
Prot. Exp.	Enable	Enable	Disable	-

Step ①: Node Type Identification. We exploit *V3.2* and identify all ECUs' node types and protocol exception configurations correctly (Tbl. 9). For each ECU, the identification is completed within several message durations.

Step ②: Inconsistent Frames. We exploit *V3.1* to make different types of nodes receive inconsistent frames. We demonstrate it on one of ECU1's ID to let ECU2 receive a genuine message but ECU4 receives a fake one. Furthermore, we exploit *V4* by pushing ECU3 to the error-passive state so it cannot receive the message. We attempt the attack 2000 times and verify the reception status of every ECU is as we expect, achieving 100% success (Tbl. 9).

Step ②: Data Consistency Attacks. The attacker can launch many other data consistency attacks exploiting *V3.1* and *V4*. We show two attacks: injecting messages which only ECU3 can receive, and dropping a specific ID from ECU1 only for ECU4, while ECU2 and ECU3 can receive it. We attempt both attacks 2000 times and achieve 100% success (Tbl. 9).

Table 9: Mixed network attacks.

	Node Type Identification	Inconsistent Frames	Selective Inject	Selective Drop
Success Rate	100%	100%	100%	100%

8 Securing CAN XL

To identify the most critical research directions to secure CAN XL, we systematize each vulnerability exploit by its impact, difficulty, and possible defensive measures (Tbl. 10).

Attack Impact and Difficulty. Since attack impact depends heavily on the system in which CAN XL is deployed, a precise universal assessment is difficult. Instead, we classify attacks targeting availability and integrity as safety-critical, as these are the most important properties of vehicles and other cyber-physical systems where CAN XL will most likely be deployed. We consider an attack to have medium difficulty if it requires complex calculations or relies on a prior exploit. These include message overwriting (*V1*) which requires crafting the attack payload carefully to account for the CRC, and all attacks targeting mixed networks (*V3-V4*) which require identifying the victim's node type first.

Table 10: Systematizing CAN XL exploits' impacts, difficulty, and defensive measures.

Vulnerability	Exploit	Impact	Safety Critical?	Difficulty	Defensive Measures		
					Detection Signatures	Prevention	Higher-Layer Mitigations
V1.1	Message Interception	Availability	✓	Low	Invalid messages with an ACK	-	Acknowledgements
V1.2	Message Overwriting	Integrity	✓	Medium	-	-	CANsec authentication
V2	ID Mapping	Confidentiality	X	Low	0 in INT3 or AH2	Standard revision Limit LL attackers	-
	Target Node Shutdown	Availability	✓	Low			-
	Message Replacement	Integrity	✓	Low			CANsec authentication
	Data Consistency Violation	Integrity	✓	Low			-
V3.2	Node Type Identification	Confidentiality	X	Low	0 in INT3	Limit LL attackers	-
	Target Node Shutdown	Availability	✓	Medium			-
V3.1	Advanced Spoofing	Integrity	✓	Medium	Partial frames		CANsec authentication
	Inconsistent Frames	Integrity	✓	Medium			-
V4	Data Consistency Violation	Integrity	✓	Medium			-
V5	Full Network DoS	Availability	✓	Low	Continuous streams of 0s	-	-
V6	Double Receive	Integrity	✓	Low	Error in AH1	Limit LL attackers	Sequence numbers
V7	Set Victim's Error Counter	Availability	✓	Low	Error in specific CRC bits		-

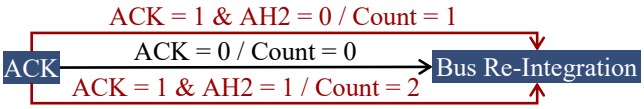


Figure 12: Revising transmitter behaviors to fix *V2*.

CAN XL Standard Revisions. Standard revisions can fix vulnerabilities at the source since they are rooted in the CAN XL standard. This may be difficult for the error signalling enabled configuration as changes may introduce incompatibility for mixed networks. Fortunately, mixed networks should be a transitional stage during CAN XL's adoption. New systems will switch to pure CAN XL networks with error signalling disabled, so revising this part of the standard will not risk incompatibility. Not all vulnerabilities can be eliminated. *V5* remains possible even if the bus re-integration exit condition changes, as attackers can always deliberately violate it. *V1* cannot be removed unless the transmitter detects errors, which is not always possible due to physical layer restrictions [16]. However, *V2* can be eliminated by these changes:

Transmitter Behavior: As shown in Fig. 12, the transmitter enters bus re-integration after ACK, irrespective of whether the message is acknowledged. It starts counting the number of received 1s so that it exits bus re-integration 11 1s after AL1 (including AH2 and ACK if they are recessive).

Receiver Behavior: If a message is valid, the receiver enters bus re-integration after ACK and exits after receiving 11 1s.

These changes ensure the transmitter and receiver are in the same state after ACK and need to receive the same number of 1s to exit bus re-integration. The message format remains the same as the original standard (11 1s after a valid ACK before next message, 9 1s after an invalid ACK).

Formal Verification: We model these changes and successfully verify $\mathcal{P}_2$ and $\mathcal{P}_3$. Thus, these changes eliminate *V2* and all attacks it enables without introducing new problems.

Attack Detection. Despite being less detectable from higher layers, many CAN XL attacks lead to unique link-layer events. These include invalid messages with an ACK (*V1*), 0 in specific protocol fields (*V2*-*V3*, *V6*-*V7*), partial frames (*V3*-*V4*), or continuous streams of 0s (*V5*). Tbl. 10 shows the signatures per attack. They can be detected by an IDS monitoring the link layer, similar to [20, 32] proposed for CAN CC.

Limiting LL Attackers. Many CAN XL attacks require LL control. Preventing remote attackers from gaining such control can fully stop these attacks (Tbl. 10). This requires ECU hardware changes, such as using ECUs that do not allow connecting other peripherals to the CAN bus. Fortunately, new CAN XL systems can incorporate these measures by design.

Higher Layer Mitigations. Security measures at the higher layers could mitigate some CAN XL attacks (Tbl. 10). By supporting authentication, CANsec [7] could limit attacks that compromise message content integrity (*V1*-*V3*). Requiring receivers to confirm message receipt with a separate acknowledgement message and having the transmitter verify it could mitigate message interception (*V1*). Appending sequence numbers to messages enables receivers to detect and discard duplicate messages, mitigating attacks that let them get the same message twice (*V6*). However, these measures do not remove underlying MAC sub-layer vulnerabilities, and their effectiveness will depend on deployment choices. As we focus on the MAC sub-layer, thoroughly analyzing the higher layers and assessing whether CAN XL is more secure overall constitutes a separate line of research.

Research Directions. Despite the defensive measures outlined above, we identify two low-difficulty and safety-critical attacks without a measure to fully prevent them (highlighted in red in Tbl. 10): message interception (*V1*) and full network DoS (*V5*). Defending against them could be the most critical research direction for securing CAN XL.

9 Discussion and Limitations

Standard vs. Implementation Vulnerabilities. Our main goal is to discover standard-level vulnerabilities by formally analyzing the CAN XL specifications then validating them on

off-the-shelf CAN controller implementations. Since implementations must comply with the standard to ensure interoperability, standard-level vulnerabilities will be present in all compliant implementations unless implementations deviate from the standard. Such deviations, which are expected with a new protocol, are more likely to introduce additional vulnerabilities than to avoid standard ones. Our formal model is proven effective at exposing such deviations, as it accurately captures protocol corner cases and helps discover bugs that vendors may not discover otherwise (Sec. 7.1). Formally analyzing these differences allows us to understand their security impacts, as demonstrated by the VN1641 example.

Analysis Approach Generalizability. While certain parts of our analysis approach are designed specifically for CAN XL, similar sequential decomposition techniques could apply to other protocols with sequential fields/phases (e.g., automotive Ethernet). Human expertise is required to identify appropriate boundaries to decompose the protocol into separate FSMs. Then, if Algorithm 1 is modified to accept the protocol's initial states and field/phase order as inputs, it can be reused to analyze other protocols given their decomposed FSMs.

Other CAN XL Layers. Since our analysis focuses on the MAC sub-layer of CAN XL compared to CAN CC, we must make assumptions about the operation of other layers and do not attempt to discover attacks exploiting other layers. For the physical coding sub-layer (PCS), which manages bit sampling and synchronization, we assume nodes are always synchronized to the bit stream on the bus (Sec. 4.3). As CAN XL modifies the PCS minimally, prior attacks exploiting CAN CC's PCS [24, 32] carry over to CAN XL. For the physical layer, we assume it operates in SIC mode, as it is common to all versions, mandated for critical frame fields, and is the only mode currently available in off-the-shelf transceivers. This makes it the natural baseline for meaningful comparison and evaluation. However, investigating the new fast mode, currently only available in non-commercial transceivers, offers an interesting future research direction. As it introduces new voltage symbols and unpredictable collision behavior, it will likely introduce vulnerabilities previously unseen in CAN CC. Regarding its impact on the vulnerabilities discovered in this paper, we expect it to only lessen the impact of *V1.2*, as *V1.2* requires deterministically overwriting level_1 with level_0. All other vulnerabilities remain intact, as they either relate solely to SIC mode or occur in frame fields that are required to always use SIC mode (e.g., INT).

Physical Access Attackers. Several real-world attacks have been launched by attackers who obtain physical access to the CAN bus and attach specialized equipment. These attackers can break any protocol rules and modify signals on the bus arbitrarily. To incorporate them in the analysis, we can modify the model (Fig. 4) to directly connect the attacker's outputs to nodes. We choose not to do so because CAN security literature still considers physical access a very strong privilege, and the attacker's strong capabilities make it unlikely any security properties can be guaranteed.

10 Conclusion

We present the first standard-level security analysis of CAN XL, the latest CAN generation. We focus specifically on its MAC sub-layer, a historically vulnerable part of the standard. We build the first formal model of the MAC sub-layer across all CAN generations (CC/FD/XL) that precisely captures its bit-oriented operation and thoroughly covers cross-generation interactions. We conduct its first formal analysis and address tractability challenges by designing an analysis workflow that decomposes the protocol into analyzable portions based on its field-oriented structure. Our analysis shows that CAN XL inherits all known CAN CC vulnerabilities and introduces seven new standard-level vulnerabilities. These weaknesses enable all classical CAN CC attacks as well as new attacks, several faster, more flexible, and less detectable than their CAN CC counterparts. We argue that despite its extensive modifications and attempts to improve security, from a strict MAC sub-layer standpoint, CAN XL's security has worsened. We validate all vulnerabilities on off-the-shelf CAN XL controllers and demonstrate their exploitability through two multi-stage attacks. Finally, we propose mitigations, including formally verifying standard revisions to prevent several attacks.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was supported in part by the National Science Foundation (NSF) under grants CNS-2144645, CNS-2333488, and CNS-2231651, Hyundai America Technical Center (HATCI), and Georgia Department of Transportation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

Ethical Considerations

Stakeholder Identification. A direct stakeholder of this research is the standardization body of CAN XL. Since CAN XL's main use case is in the automotive domain, another direct stakeholder is the automotive industry. Moreover, vehicle occupants and road users are indirect stakeholders.

Impacts. This research can benefit the CAN XL ecosystem by improving its security before widespread adoption. Our analysis helps the standardization body and automotive industry identify design and implementation weaknesses early and deploy more robust defenses in future products. Contrastingly, the research may reduce the effort required for adversaries to target CAN XL systems once they are deployed. For vehicle occupants and road users, successful exploitation could result in safety risks, operational disruption, or financial losses. For

the automotive industry, such exploits could impose incident-response costs or expose proprietary information.

Mitigations. In accordance with responsible disclosure practices, we have reported all discovered vulnerabilities in the CAN XL standard to the Robert Bosch Product Security Incident Response Team (PSIRT) since the CAN protocols are patented products by Bosch. PSIRT has acknowledged the receipt of our report and is investigating the issues internally. We document concrete countermeasures in the paper for the discovered vulnerabilities to further mitigate potential harm. All discovered bugs in commercial CAN XL controllers have been disclosed to and acknowledged by the vendors.

Decision. Since CAN XL has not yet been deployed in production systems, our findings will not cause immediate real-world harm. Instead, our goal is to proactively identify vulnerabilities to support the design of defensive measures before CAN XL's widespread adoption. Publishing our results at this stage gives relevant stakeholders time to assess exposure and implement countermeasures prior to large-scale deployment. Overall, we judge that such benefits outweigh potential risk.

Open Science

We provide the source code of our CAN XL protocol models and analysis scripts. We also provide the implementation of the attacks evaluated in Sec. 7.2. Our artifacts can be accessed at https://zenodo.org/records/20262062.

References

[1] Future of CAN networking in VW's passenger cars. https://tinyurl.com/54xhzd7w, 2021.

[2] *Road vehicles — Controller area network (CAN). Part 1: Data link layer and physical coding sublayer, ISO 11898-1:2024.* 2024.

[3] Apple carplay vulnerability exploited to gain root access. https://tinyurl.com/2r5rmybs, 2025.

[4] Application of CAN XL communication technology in automotive millimeter-wave radar (2). https://tinyurl.com/2rpp3e25, 2025.

[5] Application of CAN XL communication technology in vehicle-mounted millimeter-wave radar (1). https://tinyurl.com/5d8h83ch, 2025.

[6] CAN XL in the automotive industry: History, technical capabilities, and market outlook. https://tinyurl.com/84f66bnc, 2025.

[7] *CiA 613-2: CAN XL add-on services — Part 2: CANsec data plane*. 2025.

[8] Rohit Bhatia, Vireshwar Kumar, Khaled Serag, Z Berkay Celik, Mathias Payer, and Dongyan Xu. Evading voltage-based intrusion detection on automotive CAN. In *Network and Distributed System Security Symposium (NDSS)*, 2021.

[9] Tim Brom. Cant. https://github.com/bitbane/CANT, 2018.

[10] Zhiqiang Cai, Aohui Wang, Wenkai Zhang, Michael Gruffke, and Hendrick Schweppe. 0-days & mitigations: Roadways to exploit and secure connected bmw cars. *Black Hat USA*, 2019.

[11] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *Computer Aided Verification*, 2014.

[12] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, 2011.

[13] Kyong-Tak Cho and Kang G. Shin. Error handling of in-vehicle networks makes them vulnerable. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[14] Tsvika Dagan and Avishai Wool. Parrot, a software-only anti-spoofing defense system for the CAN bus. *ESCAR EUROPE*, 2016.

[15] Alvise de Faveri Tron, Stefano Longari, Michele Carminati, Mario Polino, and Stefano Zanero. Canflict: Exploiting peripheral conflicts for data-link layer attacks on automotive networks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[16] Magnus-Maria Hell. The physical layer in the CAN XL world. In *The international CAN Conference (iCC)*, 2020.

[17] Pengfei Jing, Zhiqiang Cai, Yingjie Cao, Le Yu, Yuefeng Du, Wenkai Zhang, Chenxiong Qian, Xiapu Luo, Sen Nie, and Shi Wu. Revisiting automotive attack surfaces: A practitioners' perspective. In *IEEE Symposium on Security and Privacy (SP)*, 2024.

[18] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *IEEE Symposium on Security and Privacy (SP)*, 2010.

[19] Sekar Kulandaivel, Shalabh Jain, Jorge Guajardo, and Vyas Sekar. CANNON: reliable and stealthy remote shutdown attacks via unaltered automotive microcontrollers. In *IEEE Symposium on Security and Privacy (SP)*, 2021.

[20] Stefano Longari, Matteo Penco, Michele Carminati, and Stefano Zanero. Copycan: An error-handling protocol based intrusion detection system for controller area network. In *ACM Workshop on Cyber-Physical Systems Security & Privacy*, 2019.

[21] Stefano Longari, Carlo Alberto Pozzoli, Alessandro Nichelini, Michele Carminati, and Stefano Zanero. Candito: improving payload-based detection of attacks on controller area networks. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, 2023.

[22] Tsutomu Matsumoto, Masato Hata, Masato Tanabe, Katsunari Yoshioka, and Kazuomi Oishi. A method of preventing unauthorized data transmission in controller area network. In *IEEE 75th Vehicular Technology Conference (VTC Spring)*, 2012.

[23] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015.

[24] Pal-Stefan Murvay and Bogdan Groza. Dos attacks on controller area networks by fault injections from the software layer. In *International Conference on Availability, Reliability and Security (ARES)*, 2017.

[25] Sen Nie, Ling Liu, Yuefeng Du, and Wenkai Zhang. Over-the-air: How we remotely compromised the gateway, bcm, and autopilot ecus of tesla cars. *Black Hat USA*, 2018.

[26] Khaled Serag. Proactive Vulnerability Identification and Defense Construction – The Case For CAN. 2023.

[27] Khaled Serag, Rohit Bhatia, Akram Faqih, Muslum Ozgur Ozmen, Vireshwar Kumar, Z. Berkay Celik, and Dongyan Xu. ZBCAN: A zero-byte CAN defense system. In *USENIX Security Symposium*, 2023.

[28] Khaled Serag, Rohit Bhatia, Vireshwar Kumar, Z. Berkay Celik, and Dongyan Xu. Exposing new vulnerabilities of error handling mechanism in CAN. In *USENIX Security Symposium*, 2021.

[29] Khaled Serag, Vireshwar Kumar, Z Berkay Celik, Rohit Bhatia, Mathias Payer, and Dongyan Xu. Attacks on can error handling mechanism. In *Workshop on Automotive and Autonomous Vehicle Security (AutoSec)*, 2022.

[30] Khaled Serag, Zhaozhou Tang, Sungwoo Kim, Vireshwar Kumar, Saman Zonouz, Raheem Beyah, Dongyan Xu, Z Berkay Celik, et al. SoK: Kicking CAN down the road. Systematizing CAN security knowledge. *arXiv preprint arXiv:2510.02960*, 2025.

[31] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network. In *International Conference on Information Networking (ICOIN)*, 2016.

[32] Zhaozhou Tang, Khaled Serag, Saman Zonouz, Z. Berkay Celik, Dongyan Xu, and Raheem Beyah. ERACAN: Defending Against an Emerging CAN Threat Model. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[33] Ken Tindell. CAN bus security: Attacks on CAN bus and their mitigations. Technical report, Canis Automotive Labs, February 2020.

[34] Ken Tindell. Canhack. https://github.com/kentindell/canhack, 2020.

[35] Ken Tindell. Three new can protocol hacks. https://kentindell.github.io/2020/01/20/new-can-hacks/, 2020.

[36] Armin Wasicek, Mert D Pesé, André Weimerskirch, Yelizaveta Burakova, and Karan Singh. Context-aware intrusion detection in automotive control systems. *ESCAR USA*, 2017.

[37] Fuyu Yang. A bus off case of can error passive transmitter. *EDN Technical paper*, 2009.

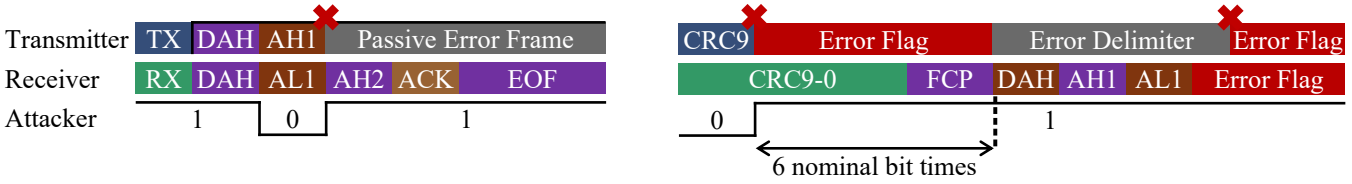
[38] Clinton Young, Habeeb Olufowobi, Gedare Bloom, and Joseph Zambreno. Automotive intrusion detection based on constant can message frequencies across vehicle driving modes. In *ACM Workshop on Automotive Cybersecurity*, 2019.

A Formal Property Specifications

Tbl. 11 shows the properties in Sec. 4.1 expressed in CTL using the syntax of nuXmv. We explain them in detail below:
$\mathcal{P}_1$: This states that at all times, if the node `n1` is in state `bus_reintegration`, then from this point, `n1` can always finally leave `bus_reintegration`. The same property, and other properties below, are defined symmetrically for `n2`, the second node we model (Fig. 4). We omit them for brevity.
$\mathcal{P}_2$: At all times, if `n1` is in `bus_reintegration` but `n2` is not in `int3` (the last field before a new frame), `idle`, or also in `bus_reintegration` , `n1` always stays in `bus_reintegration` in the next cycle. This ensures if `n1` enters `bus_reintegration` during a frame, it stays in this state until the frame completes.

Table 11: Formal property definitions.

Property	CTL Formula
$\mathcal{P}_1$	**AG** (n1.state = bus_reintegration -> **AF** n1.state != bus_reintegration)
$\mathcal{P}_2$	**AG** ((n1.state = bus_reintegration & !n2.state in {int3, idle, bus_reintegration}) -> **AX** n1.state = bus_reintegration)
$\mathcal{P}_3$	**AG** !(n1.state = bus_reintegration & (n2.state = tx_id | n2.state = rx_id))
$\mathcal{P}_4$	**AG** (n1.validate_frame -> **AF** n2.validate_frame)
$\mathcal{P}_5$	**AG** ((n1.tx != n2.rx & n1.state = tx_data) -> **AG** (!n1.validate_frame & !n2.validate_frame))
$\mathcal{P}_6$	**AG** (**AF** ((n1.state = idle) | (n1.state = tx_id) | (n1.state = rx_id)))
$\mathcal{P}_7$	**AG** ((n1.tx != n1.rx & n1.state = tx_data) -> **AX** n1.state = send_ef)
$\mathcal{P}_8$	**AG** (n1.state = send_ef -> **AF** n2.state = send_ef)
$\mathcal{P}_9$	**AG** !n1.increment_ec



(a) Rx-only validation (*V6*).

(b) Error regeneration (*V7*).

Figure 13: Error signalling enabled vulnerabilities.

6 nominal bit times

$\mathcal{P}_3$*:* Our definition in Sec. 4.1 is equivalent to the safety property in Tbl. 11 which states that n1 cannot be in bus_reintegration when n2 is transmitting or receiving a message's ID (the first message field) at any time.

$\mathcal{P}_4$*:* In the ACK-intermission model which models a frame's ACK field to the start of the next frame, we define a state variable validate_frame. It is initialized to false and set to true when the node validates this frame, based on frame validation rules in the standard. We verify $\mathcal{P}_4$ for the ACK-intermission model, which states that whenever n1 validates a frame, n2 always eventually validates this frame.

$\mathcal{P}_5$*:* We verify $\mathcal{P}_5$ for the ACK-intermission model. It states that whenever the bit transmitted by n1 does not equal the bit received by n2 when n1 is transmitting a frame's data (or other fields where consistency is required, omitted for brevity), both n1 and n2 can never validate the frame. In our implementation, since fields such as data are modeled outside the ACK-intermission model, we define a variable to track if the transmitted and received bits differ and propagate it across model boundaries using Algorithm 1.

$\mathcal{P}_6$*:* We verify $\mathcal{P}_6$ for the ACK-intermission model, which states that starting from any time, the bus always finally becomes idle, or a new frame transmission/reception starts.

$\mathcal{P}_7$*:* This states that whenever n1 detects a bit error (its transmitted bit does not equal its received bit) when transmitting a frame's data (and other fields where bit error detection is performed, omitted for brevity), it always sends an error frame (send_ef) in the next cycle.

$\mathcal{P}_8$*:* Whenever the transmitter n1 sends an error frame, the receiver n2 always eventually sends an error frame.

$\mathcal{P}_9$*:* We define a state variable increment_ec for the node which is set to true whenever it encounters an error when sending an error frame. Then we express $\mathcal{P}_9$ by specifying that this variable can never be true.

B Additional Analysis Results

V6. Rx-Only Validation. When error signalling is enabled, we discover a violation of $\mathcal{P}_4$ where a frame is only validated by receivers but not its transmitter. As shown in Fig. 13a, an attacker injects an error in an XL frame's AH1 when its transmitter is error-passive. Receivers do not detect an error because per the standard, they treat the bit after DAH as AL1 if a 0 is received. The transmitter detects an error and sends a passive error frame, and receivers receive 1s. Since expected values of all bits after AL1 are 1s (Fig. 2), receivers can complete the reception without errors. This also violates $\mathcal{P}_8$. If any receiver transmits a message immediately after this message, the transmitter encounters another error, violating $\mathcal{P}_9$. This is because for a node to terminate its passive error frame correctly, the bus must be idle for at least an additional 3 bit-periods between two consecutive frames [37]. This is not satisfied in this scenario. More back-to-back transmissions further increment the transmitter's error counter, possibly pushing it into bus-off.

Impact. LL attackers can exploit *V6* to make receivers get the same message twice since the transmitter detects an error and retransmits. This could cause inconsistent states and control decisions. Similar goals can be achieved by exploiting *Vb*. However, a crucial difference between *V6* and *Vb* is that *V6* may additionally cause rapid increments in transmitters' error counters and transmitter bus-off. Thus, in addition to introducing a new way to achieve similar goals as an existing attack, it significantly undermines the robustness of CAN XL communication, as a single genuine error at AH1 may cause many unexpected behaviors. CAN CC does not have an error

Figure 14: Message interception by HL attackers.

Figure 15: Full network DoS by HL attackers.

corner case with the same impacts.

V7. Error Regeneration. When error signalling is enabled, we discover a violation of $\mathcal{P}_9$ where a node encounters further errors when sending an error frame. It applies to XL frames that have 9 bits in FCRC after the last fixed stuff bit and an XL data bit rate equal to twice the nominal bit rate. As shown in Fig. 13b, an attacker injects an error in the first of these 9 bits. Depending on its error state, the transmitter sends an active or a passive error frame. The error flag starts at the next bit and ends at the bit corresponding to FCP0, followed by the recessive error delimiter. This violates the fixed FCP format and receivers detect an error. This is treated as a CRC error and not signalled immediately after FCP. Instead, receivers signal it with error frames after AL1. They collide with the transmitter's error delimiter, causing it to detect another error. If the transmitter is error-passive, back-to-back transmissions by receivers after the error frame further increment its error counter, possibly pushing it into bus-off.

Impact. Attackers can exploit *V7* to rapidly set a victim's error counter and achieve many purposes such as quick targeted DoS by pushing the victim to bus-off. It is exploitable only by LL attackers, not HL attackers. This is because to cause an error only in FCRC, the attacker needs to transmit a message with the same data but a different FCRC than the victim's, which is impossible with a standard CAN controller. Similar goals can be achieved by exploiting passive error regeneration (*Va*), but *Va* only applies to error-passive victims while *V7* applies regardless of victims' error states. Moreover, *V7* undermines the robustness of CAN XL communication, introducing a new corner case where a single genuine error leads to rapid error counter increments and possible transmitter bus-off.

C Additional Exploit Descriptions

V1. HL Message Interception. Exploiting *V1.1*, HL attackers can intercept messages using simultaneous transmission. As shown in Fig. 14, the attacker simultaneously transmits a message that is longer than a transmitter's message by 2 nominal bit durations. With disabled error signalling, both the attacker and the transmitter can transmit their messages until completion. Collisions between two different messages let receivers detect errors. When the transmitter is expecting the ACK, the attacker is transmitting the dominant AL1 bit, which the transmitter treats as a valid ACK.

V3. Node Type Identification. During node type identification, other nodes may also transmit and win arbitration over the target node's retransmission. Thus, for each consecutive message, the attacker records its ID and sends a 0 in INT3 until the bus becomes idle. If the target node's retransmission is observed, the node has the expected type. Otherwise, it has another type supporting more frame formats. If XL frames are observed when testing if a node is FD tolerant, the attacker cannot conclude that the node is FD tolerant if he does not see its retransmission. This may indicate that the node is FD tolerant and enters bus re-integration due to the attacker's FD frame, or is FD enabled and enters bus re-integration due to another node's XL frame. The test needs to be repeated until it is not interfered with by XL frames. To reduce such interference, the attacker should pick a high-priority ID that is not typically transmitted at a similar time as other nodes' messages to test. Exploiting *V3.2*, an attacker can also check if an XL enabled node enables or disables protocol exception (Sec. 2). Similarly, the attacker wins arbitration over the node's transmission and sends a partial XL frame with 1 in resXL, followed by an active error frame, and a 0 in INT3.

V5. HL Full Network DoS. Exploiting *V5*, HL attackers can delay bus traffic for finite durations by simultaneously transmitting a longer message with any node. As shown in Fig. 15, the collision causes the transmitter to detect errors in the EOF and receivers earlier, while the attacker continues transmission. Nodes stay in bus re-integration during the attacker's message (as verified by $\mathcal{P}_2$). The delay can be arbitrary up to the maximum length of a CAN XL message.