Thinking Beyond Bus-off: Targeted Control Falsification in CAN

Ipsita Koley
Dept. of CSE, IIT Kharagpur
ipsitakoley@iitkgp.ac.in

Sunandan Adhikary
Dept. of CSE, IIT Kharagpur
mesunandan@kgpian.iitkgp.ac.in

Soumyajit Dey
Dept. of CSE, IIT Kharagpur
soumya@cse.iitkgp.ac.in

***Abstract*—Controller Area Network (CAN) is considered to be the de facto standard for intra-vehicular communication of modern automobiles. Due to the lack of strong authentication and confidentiality schemes, CAN has been the popular target of a multitude of attacks in the last two decades. Among such attacks, the Bus-off attack (BoA) is considered an important attack vector since it helps the attacker impersonate a trusted safety-critical controller while actually sending false messages.**

In this paper, we uncover the limitations of classical BoA along with the recent variants in the context of targeting specific automotive safety-critical control loops. In particular, we show that state-of-the-art system-agnostic BoAs are actually futile in the presence of rudimentary range and gradient-based signal monitors that filter out signals exhibiting drastic changes. Instead, we propose a new attack model $FalCAN$ that maximally impacts a given automotive closed-loop while maintaining stealth against monitors. Given the control and bus network specifications, the attack first figures out maximal delays that can be introduced for targeted control loops while attempting BoA. The attack then identifies suitable false data sequences for such delayed actuations that maximize the state error while staying undetected. We implement this attack model on automotive-grade ECUs with benchmark CAN traffic and visualize the impact by real-time emulation of the plant using Hardware-in-loop (HIL) simulation.

***Index Terms*—Automotive, Security, CAN, False data injection**

I. Introduction

Technological advancements have taken automotive engineering to a new level. Modern-day vehicles represent a collection of feature-rich intelligent cyber-physical control loops that not only act as a mode of transportation anymore but also take care of safety, comfort, and resource-aware performance guarantees. Various control functionalities are implemented as real-time tasks mapped to a number of electronic control units (ECUs) that process sensor measurements and compute control signals for the actuator units. Real-time communication among the ECUs is achieved through a heterogeneous intra-vehicular network consisting of lightweight network protocols like Controller Area Network (CAN), FlexRay, Local Interconnect Network (LIN), and Media-oriented systems transport (MOST). These protocols focus on the timeliness of message exchanges and basic integrity checks to ensure safe and secure operability. However such basic security measures are not resistant to insider attacks where the attacker compromises one of the authenticated hardware components that it finds accessible. With various software-based decision systems and wireless connectivity features being introduced in contemporary vehicles, possible attack surfaces have drastically increased. Since most of the real-time safety-critical control loops communicate via CAN and CAN lacks strong authentication and confidentiality schemes, it has naturally become quite a predominant victim of different kinds of in-vehicle attacks [3], [9], [12], [14].

Among these attacks, the bus-off attack (BoA), originally proposed in [4], is regarded as the most significant one. In BoA, an attacker sends a target ECU off the bus, and during this *bus-off period*, the attacker can impersonate any trusted controller task that is mapped to the target ECU, and inject falsified messages on the CAN bus. Due to the stealthiness of BoA, it has been widely explored and a number of its refinements have been proposed in the last few years [2], [7], [8], [16]. We start with a discussion of our observations about the practical limitations and actual effectiveness of BoA.

(L1) The state-of-the-art BoAs [2], [7], [8], [16] do not consider the presence of safety monitors, like range-based and gradient-based ones [11] for on-board sanity checking of ECU messages, a feature that is commonly used in automotive ECUs. Such monitors filter out signals that go beyond pre-designated value ranges or exhibit drastic changes. Hence, once a target ECU is sent to Bus-off mode, the spurious data injected into the CAN bus by any attacker needs to be stealthy against such monitoring tasks. Otherwise, an easily detectable attack during the bus-off period simply serves no purpose.

(L2) The primary objective of BoA is to keep the victim ECU off the network for some time. As a consequence, the controllers implemented on the victim ECU cannot receive/transmit sense/actuation messages through CAN. However, if the recovery time of the target ECU from the bus-off state is substantially less, BoA hardly affects the performance of the related control loop. This is because the delay-aware design of such control loops ensures tolerance of a bounded number of control actuation misses anyway. The work in [16] proposes a method to elongate the bus-off period of the victim ECU. This method demands accurate and repeated synchronization with the victim ECU's messages every time the victim node attempts to come out of the bus-off mode. Such a brute-force repetitive bus-off attempt to keep the victim ECU disconnected for a longer duration is very difficult to achieve for a time-triggered bus-off attack as the re-synchronization in subsequent attempts is highly likely to fail due to platform-level uncertainties.

(L3) BoA methods work by targeting some message originating from a victim ECU. Existing literature suggests choosing such messages that allow a rapid increase in ECU error counts, aiming for faster BoA [16]. Thereafter, suitable instances of this target message need to be chosen, which have longer time intervals preceding them to attempt BoA [7]. However, jitters in the heavy CAN traffic can cause erroneous victim instance selection. Instead, we should search for a structured attack methodology that smartly utilizes the adverse effects during BoA attempts and falsifies the control data, maintaining stealth.

In this work, we propose a real-time, data falsification attack $FalCAN$ on automotive control loops. Unlike state-of-the-art BoAs, simply disconnecting a legitimate ECU from the CAN bus is not the only and primary objective of *FalCAN*. Rather, we focus on two important aspects while designing $FalCAN. First$, the attack must be a targeted one that is intended to destabilize a given safety-critical control loop while maintaining its stealth against the

The authors acknowledge the generous grants received from "IHUB NTI-HAC Foundation - IIT Kanpur" and "Meity Grant No. AAA.22/8/2021-CSRD-Meity" for partially supporting this work.

safety monitors. $Second$, the attack model must consider platform-level uncertainties and determine stealthy and optimal control data falsification dynamically in real time.

A number of research works can be found in the literature that provide mathematical models of stealthy false data injection attacks [15], [19] along with model-based approaches to synthesize the same [10], [11]. However, those works are confined to the theoretical formulation of the attack rather than considering the uncertainties faced while implementing them on a real-time platform. On the other hand, the works in [4], [7], [17] that propose hardware-level attacks, are either denial-of-service types or random false data injection types. They are not designed to target a safety-critical control loop and the associated safety monitors. To the best of our knowledge, this is the first work that aims to bridge the gap between theoretical attack models and implementation challenges on real-time platforms. We summarize the primary contributions of this work as follows.

(1) We present a methodology $FalCAN$ that considers an intra-vehicular network structure, associated ECUs, and control task mappings to determine which message instances of which control loop exhibit the most BoA success probability.

(2) We model the target safety-critical control loop under attack-induced delays and falsified control inputs. Our modeling analyzes the impact of such attack-induced uncertainties on the state deviation error in a platform-sensitive manner.

(3) We present a real-time attack strategy that takes the best possible action based on the current CAN traffic. It aims to elongate the delay in the actuation of the true control signal. It then designs the optimal and stealthy false data dynamically at run-time based on the current response of the target control loop.

(4) We evaluate the potential of the proposed attack model $FalCAN$ in sustaining significant damage to the performance of the targeted system on a hardware-in-loop (HIL) setup consisting of automotive-grade ECUs using benchmark CAN traffic data [16].

II. Background

A. Intra-vehicular Network

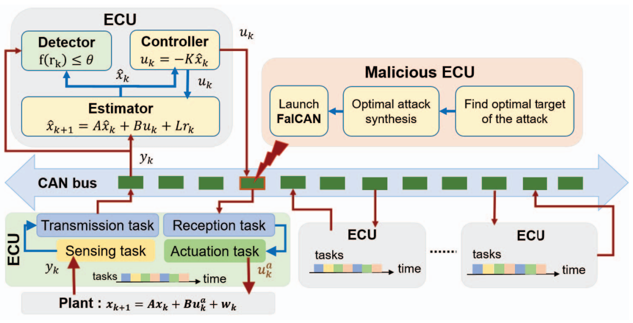


Fig. 1: Intra-vehicular network

A typical intra-vehicular network setup is demonstrated in Fig. 1. The automotive CPSs are implemented using i) electronic control units (ECUs) and ii) intra-vehicular network protocols. Each ECU has multiple cores and a number of tasks (including safety-critical ones) are running on them following certain static schedules. The safety-critical control loops communicate via CAN protocol. Each ECU has multiple CAN controllers and CAN transceivers that facilitate the transmission and reception of its messages through CAN. The ECUs process sensor measurements received from the CAN bus, compute required control input and transmit that to the actuator unit via the CAN bus. We consider linear time-invariant (LTI) discrete state-space model to mathematically represent these control loops as:

$$x_{k+1} = Ax_k + Bu_k + w_k;\ y_k = Cx_k + v_k;\ r_k = y_k - C\hat{x}_k;$$
$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + Lr_k;\ u_k = -K\hat{x}_k \tag{1}$$

Here, $x_k \in \mathbb{R}^n$ and $y_k \in \mathbb{R}^m$ denote the system state vector and measurement vector, respectively. A, B, C are the system matrices. The initial state $x_0 \in \mathcal{N}(\bar{x}_0, \Sigma)$, the process noise $w_k \in \mathbb{R}^n \sim \mathcal{N}(0, \Sigma_w)$ and the measurement noise $v_k \in \mathbb{R}^m \sim \mathcal{N}(0, \Sigma_v)$ are considered to be independent Gaussian random variables. The observable system state $\hat{x}_k$ is estimated using system output y_k in every k-th sampling instant while minimizing the effect of noise. The estimated system state $\hat{x}_k$ is used for computing the control input $u_k \in \mathbb{R}^p$. The system residue i.e. the difference between the measured and the estimated outputs, is denoted as r_k. The estimator gain L and controller gain K are designed in such a way that stability of both $(A - LC)$ and $(A - BK)$ is ensured.

B. Safety Monitors

We consider that every ECU is equipped with a monitor that verifies each received signal against a set of safety rules as mentioned below.

a) *Range monitor*: It verifies whether sensor measurements and control signals fall within their stipulated limits. This implies the following conditions must hold: $y_k \in [-\theta^y_{val},\ \theta^y_{val}]$ and $u_k \in [-\theta^u_{val},\ \theta^u_{val}]\ \forall k$ where $\theta^y_{val} \in \mathbb{R}^m$, $\theta^u_{val} \in \mathbb{R}^p$.

b) *Gradient monitor*: The change in sensor measurements and control signal must be contained within specified limits i.e. $\nabla y_k = (y_k - y_{k-1}) \in [-\theta^y_{grad},\ \theta^y_{grad}]$ and $\nabla u_k = (u_k - u_{k-1}) \in [-\theta^u_{grad},\ \theta^u_{grad}]\ \forall k$ where $\theta^y_{grad} \in \mathbb{R}^m$, $\theta^u_{grad} \in \mathbb{R}^p$.

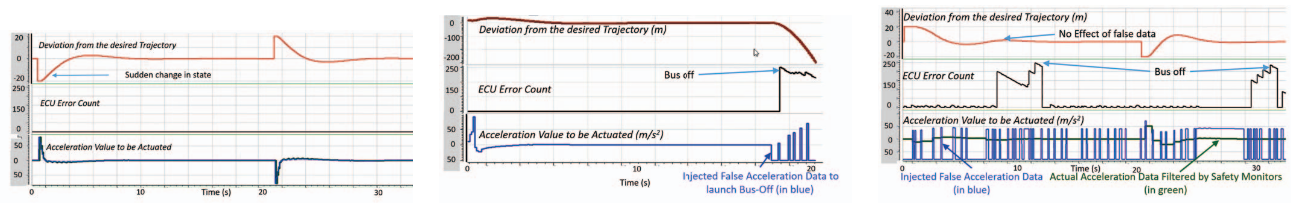
The certain sensor measurements y_k that violate these monitoring rules are filtered out and not sent to the estimator unit. As a consequence, the residue r_k will not be computed during those cycles. This implies $r_k = 0$ and the estimated state is evolved as $\hat{x}_{k+1} = A\hat{x}_k + Bu_k$ instead of Eq. 1. Similarly, if the safety monitor does not allow a control signal u_k received from CAN to pass through the actuator, the plant is actuated with the last valid control signal i.e. $x_{k+1} = Ax_k + Bu_{k-1} + w_k$.

C. Anomaly Detection

An intelligent attacker can smartly falsify the sensor measurements and control signals that bypass the safety monitors but induce enough damage to an automotive closed loop control system [11]. Therefore, we also consider the following two types of anomaly detectors implemented on each control unit along with the safety monitors.

a) *Residue-based detectors*: Residue-based detectors are widely used in CPS [6]. Such detectors compute a function $f(r_k)$ and compare it with a threshold θ_{res}. If $f(r_k) > \theta_{res}$, the detector flags an anomalous behavior of the system.

b) *Frequency-based detectors*: Frequency-based detectors are used in communication medium to detect unusual transmission rates. This kind of detector is especially suitable for a medium like CAN as it is mostly static i.e. no new node is added to CAN once it is deployed [9]. Consider that the desired number of times the target message is transmitted through CAN over an observation window of length N is l_{des}. In a certain observation window, if we notice the target packet l times which is more than the desired value, the situation is flagged. However, jitters in CAN affect the transmissions of CAN packets. Sometimes we can see the target packet in an observation window less than or more than



(a) Under no attack (b) Under BoA in absence of safety monitors (c) Under BoA in presence of safety monitor

Fig. 2: Effectiveness of bus-off attack in the presence and absence of safety monitors

the desired number of times. Therefore, to avoid false alarms, the frequency-based detector alarms an anomaly only when frequency violations occur (i.e. $l > l_{des}$) more than a predefined number of times θ_{freq}.

D. Classical Bus-off Attack

The bus-off attack (BoA), first proposed in [4], is fundamentally a denial-of-service attack that cuts a legitimate ECU from the CAN bus by exploiting CAN's error-handling strategies. Each ECU can be of either of the three states based on its transmission and reception error counts (TEC and REC): error active ($0 \leq TEC, REC < 128$), error passive ($128 \leq TEC, REC < 256$), and bus-off state ($TEC, REC \geq 256$). In a bus-off state, the ECU is sent off the bus for some duration. In BoA, an attacker targets a message m_v, originated from a victim ECU ECU_{victim}, and fabricates an attack message m_a with the same ID as m_v. It attempts to transmit m_a synchronously at the same time when m_v is transmitted. Repeated synchronizations increase the error count of the victim ECU ECU_{victim} and gradually send it to the bus-off state. The attacker then impersonates ECU_{victim} while actually sending false data to CAN. The detailed description of BoA and the necessary primers on CAN protocol to understand the attack approach are given in Appendix B and A respectively.

III. Motivation

In this section, we discuss the shortcomings of the classical bus-off attack [4] and its more efficient versions [7], [16] in inducing substantial damage to a target safety-critical control loop while being stealthy. From these limitations, we motivate the current work.

L1. Stealthiness of a bus-off attack against safety monitors: One of the requirements for BoA to be successful is to have the value in the data field of the attack message less than that of the victim message (see Appendix B). In state-of-the-art BoAs, the attacker does not give importance to what should be the attack value. Instead, in most cases, a 0 is considered an attack value. This may cause an abrupt change in the gradient of the target message signal, and thereby, filtered out by the safety monitors (Sec. II-B). We demonstrate such a scenario with an example of a trajectory tracking control (TTC) system. The TTC system has two states, deviation from the reference trajectory and the velocity, that are controlled by acceleration. The measured output of the system signifies the deviation from the reference trajectory. We implement a TTC system [10] in our lab setup consisting of LABCAR, a real-time system where the plant is implemented, an Infenion ECU TC397 where the control logic is implemented, and a CAN network that facilitates the communication between LABCAR and the ECU. We also attach an attacker ECU to the same network from where we launch BoA. The TTC system is simulated under a periodically varying reference. With the same setup, we analyze the response of TTC under no attack, under BoA without safety monitors, and under BoA with safety monitors in Fig. 2. The red (top), black (middle), and blue (bottom) plots in Fig. 2 denote the system response, error count of the ECU, and control input of TTC. The response of the TTC system under normal conditions i.e. no attack is launched, is given in Fig. 2a. We can see that the ECU error count remains 0. The response of the TTC system under BoA in the absence and presence of safety monitors are illustrated in Fig. 2b and Fig. 2c respectively. In both cases, the attacker targets the control signal of TTC and fabricates attack messages with the value set as 0. In Fig. 2b, we can observe that the plant response starts degrading once the error count goes beyond 256 i.e. the ECU is in a bus-off state. During this period attacker node impersonates the victim ECU and sends falsified acceleration (i.e. 0) which is the control input of TTC. This results in further degradation of TTC's response, thereby destabilizing the TTC system. On the other hand, in Fig. 2c, we can see that the performance of the TTC system is preserved as the safety monitor rejects the spurious control signals. Instead, the last valid control input is actuated (Sec. II-B). The response of TTC in Fig. 2c is almost the same as that of in Fig. 2a. Therefore, for an attack to be successful, it must be stealthy against such safety monitors. Otherwise, the whole purpose of the attack will be of no use.

L2. Impact of the bus-off attack on the performance of safety-critical automotive CPS: The objective of BoA is to disconnect the target ECU from the bus. If the attacker targets the message corresponding to the control signal of a safety-critical automotive CPS, the plant is not supposed to receive any control input during the time the source ECU stays in the bus-off state. Generally, an ECU recovers from a bus-off state in two ways: manual recovery and automatic recovery. A manual recovery scheme needs human intervention to re-activate the ECU. And, the automatic recovery scheme needs 128 instances of 11 recessive bits (= 1 bit of acknowledgment delimiter + 7 end-of-frame bits + 3 bits of IFS) to be transmitted through the CAN bus to bring the affected node back from bus-off to error-active state. This implies that in the worst case when the bus load is high, 128 CAN frames will be transmitted with an IFS consisting of 3 recessive bits between every two consecutive CAN frames. Therefore, during the bus-off recovery period, $(128 \times 111) = 14208$ bits are transmitted. Hence, the maximum time taken for recovery from the bus-off state is $\frac{14100}{250}$ ms = $\simeq$ 57 ms considering the bandwidth is 250 Kbps. Generally, the sampling period of the safety-critical control loops is less than 100 ms. For example, the TTC is sampled at every 80 ms which is higher than 57 ms. Therefore, the actuator will not receive control input for a maximum of $\lceil \frac{57}{80} \rceil = 1$ sample. During this time, the plant will update its states with the last received control input until it receives a new one. In general, control theorists design mathematical control laws while keeping environmental and platform delays in mind. This generally implies the control loop will work with the desired performance guarantee up to a bounded number of control signal misses. We demonstrate the effect of control signal misses on the performance of TTC in Fig. 2c. Whenever the ECU goes to the bus-off state, the plant does not receive control input i.e. acceleration until the ECU gets active again. We can see that the performance degradation due to such sporadic control misses is negligible as compared to the system under no attack (Fig. 2a). Therefore, the existing BoAs that focus only on sending a legitimate ECU off the bus, will not have much detrimental effect on the target control loop.

L3. The target of the bus-off attack: Usually, the ECUs do not execute only a single task. Rather, multiple mixed-critical tasks are co-scheduled on each ECU. One or more of the co-scheduled tasks transmit and receive messages to and from the CAN bus. To send a victim ECU off the CAN bus, consecutive 35 successful synchronizations of the attack and the target messages are required. However, transmissions of other messages from the victim ECU in between the synchronizations reduce the error count, and in the worst case, it may never let the error count of the victim ECU go beyond 256. To deal with this, existing BoA methodologies select the message with the smallest periodicity scheduled on the victim ECU as the target [16]. This ensures the minimum number of message transmissions in between the successful synchronizations. However, the selection of the target message based on only the sampling period does not guarantee attack success.

For BoA to be successful, the synchronization of the attack message with the target message has to be perfect. A perfect synchronization requires at least one higher-priority message preceding the target message. During the transmission of the higher-priority message, if the attack message is transmitted, it is forced to wait and then synchronized with the victim message (see CAN arbitration process explained in Appendix A). Existing literature [7] suggests considering only those message instances as targets that have at least one preceding higher-priority message. Their time-triggered BoA approach computes the time window before the target message instance during which only higher priority messages are transmitted and initiates the synchronization during this window only. However, in real-life CAN traffic, jitters affect the transmissions of the messages which may lead to incorrect computation of higher-priority time windows. Therefore, such time-triggered attack approaches tend to fail. Instead, we need a structured event-triggered attack approach that ensures a higher attack success rate.

In this work, we intend to design an attack on the CAN bus that addresses all the above limitations. The proposed attack model $FalCAN$ is not confined to only disconnecting the victim ECU from the bus. Rather, the philosophy of such an attack model is to degrade the performance of the target safety-critical automotive control loop maximally through controlled false data injection and by inducing delays during control actuation. In the following section, we formally present the attack model, and in Sec. V, we elaborate on the methodology to carry out the proposed attack.

IV. Problem Formulation

In the previous section, we pointed out why state-of-the-art BoA approaches are not enough to impose significant damage to the target safety-critical automotive control loop. In Fig. 3, we study how delays and false data are introduced during the stages of BoA. Let us assume, that both victim and attacker are initially in an error active state with TEC = 0. Once the attacker initiates BoA, both victim and attacker attempt re-transmissions until both enter the error passive state. This results in the transmission of attack message m_a with a delay of d^a_{ep} followed by the transmission of victim message m_v with a further delay of d^a_{ep}. While both victim and attacker are in the error passive state, every successful synchronization attempt by the attacker delays the transmission of m_v by d^a_{ep}. Note that, depending upon the TEC value of the victim at the time of attack commencement, bus load, bus speed, and jitters, these delays vary over time. Let us now study how these delays and false data injections affect the performance of the system.

In our proposed attack model $FalCAN$, we consider a safety-critical control signal to be the target. Therefore, m_a is fundamentally the falsified control signal to be injected by $FalCAN$ and m_v is the targeted control signal. Consider $FalCAN$ is initiated at k-th sample of the target control loop. The victim message m_{v_k} and the attack message m_{a_k} contain the actual control signal u^a_k that is to be targeted by $FalCAN$ and the falsified control signal $\tilde{u}^a_k$ that is to be injected by $FalCAN$ respectively at k-th sample. $\tilde{u}^a_k$ can be represented as $\tilde{u}^a_k = u^a_k + a_k$ where a_k denotes the attack value. From here onwards, the superscript a is used to symbolize system variables under attack. Assuming both victim and attacker ECUs are in error active state when $FalCAN$ is initiated, we study the following three cases.

Case 1 *(victim in error active)*: When both victim and attacker are in the error active state, the delays d_{ea}, d^a_{ep} and falsified control signal (see in Fig. 3) will instrument the actuation process as follows. (1) During d_{ea} no new control signal will be received, and the plant will use the last received control signal u^a_{k-1} for actuation. (2) Once $\tilde{u}^a_k$ (i.e. m_{a_k}) is transmitted successfully, the plant will be actuated with $\tilde{u}^a_k$ during d^a_{ep} until it receives u^a_k (i.e. m_{v_k}). (3) On the successful transmission of u^a_k, u^a_k will be used for actuation until new control input arrives i.e. during $d^v_{ep} = h^v - (d_{ea} + d^a_{ep})$ where h^v is the sampling period of target controller. Therefore, during the time window of $[k, k+1]$ samples, the states of the target system will be updated as,

$$\begin{aligned} x^a_{k+1} &= Ax^a_k + B_1 u^a_{k-1} + B_2 \tilde{u}^a_k + B_3 u^a_k + w_k \\ &= Ax_k + B_1 u^a_{k-1} + (B_2 + B_3) u^a_k + B_2 a_k + w_k \end{aligned} \tag{2}$$

where, $B_1 = B_c \int_0^{h^v - (d^a_{ep} + d^v_{ep})} e^{As} ds \simeq B_c d_{ea}$, $B_2 = B_c \int_{h^v - (d^a_{ep} + d^v_{ep})}^{h^v - d^v_{ep}} e^{As} ds \simeq B_c d^a_{ap}$, and $B_3 = B_c \int_{h^v - d^v_{ep}}^{h^v} e^{As} ds \simeq B_c d^v_{ep}$. Here, B_c denotes the continuous time input matrix. Since $FalCAN$'s target is the control signal, not sensor measurements of a safety-critical automotive CPS, there will be no change in estimation and control calculation i.e. they will follow Eq. 1.

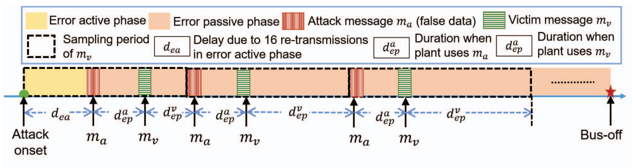


Fig. 3: Delay and false data introduced during $FalCAN$ stages

Case 2 *(victim in error passive)*: After the first successful synchronization, both attacker and victim enter error passive mode. At $(k+1)$-th sample the second synchronization is attempted by the attacker of the attack message $m_{a_{k+1}}$ i.e. $\tilde{u}^a_{k+1}$ with $m_{v_{k+1}}$ i.e. u^a_{k+1}. A successful synchronization transmits $\tilde{u}^a_{k+1}$ and delays u^a_{k+1} by d^a_{ep}. Therefore, the plant will be actuated with $\tilde{u}^a_{k+1}$ for d^a_{ep} until it receives u^a_{k+1}. After successful transmission of u^a_{k+1}, u^a_{k+1} will be used for actuation during a period of $d^v_{ep} = h^v - d^a_{ep}$ samples. Therefore, in the time window $[k+1, k+2]$, the states of the system are updated as,

$$\begin{aligned} x^a_{k+2} &= Ax^a_{k+1} + B_2 \tilde{u}^a_{k+1} + B_3 u^a_{k+1} + w_{k+1} \\ &= Ax^a_{k+1} + B_2 (u^a_{k+1} + a_{k+1}) + B_3 u^a_{k+1} + w_{k+1} \\ &= Ax^a_{k+1} + (B_2 + B_3) u^a_{k+1} + B_2 a_{k+1} + w_{k+1} \end{aligned} \tag{3}$$

In every subsequent instance of target control input, the successful synchronization with the attack message will update the state of the target safety-critical control loop following Eq. 3 until the victim ECU goes to the bus-off state from error passive mode.

***Case 3** (victim goes bus-off)*: Once the victim ECU enters the bus-off state, the target control signal will not be transmitted at all. Instead, the receiving ECU will get attack messages i.e. falsified control input. During the bus-off state, the victim ECU remains powered up, only its CAN controller is reset. In such a scenario, the estimator's state will be updated as $\hat{x}^a_{l+1} = A\hat{x}^a_l + Bu^a_l$ where l being the sampling instance when the victim enters the bus-off state. Therefore, during the bus-off state, the dynamics of the target CPS will change as,

$$\begin{aligned} x^a_{l+1} &= Ax^a_l + B\tilde{u}^a_l + w_l = Ax^a_l + B(u^a_l + a_l) + w_l \\ &= Ax^a_l + Bu^a_l + Ba_l + w_l; \\ \hat{x}^a_{l+1} &= A\hat{x}^a_l + Bu^a_l; \quad u^a_l = -K\hat{x}^a_l; \end{aligned} \tag{4}$$

In all of the above scenarios, we can observe that the states of the target safety-critical control loop are updated as the function of delays and false data induced by $FalCAN$. In the following claim, we show that this amplifies the state deviation.

Claim 1: The delayed actuation and control falsification increase the state deviation of the control loop under attack. □

Proof of the Claim 1 is given in Appendix C.

Remark 1: Instead of targeting to send the victim ECU to bus-off as early as possible, if we can introduce the delay in transmission of the target control signal repeatedly, the effects of the delays get accumulated in the actuation of the target control signal. This increases the state deviation. One can argue to have a delay-aware control design instead, that can tolerate these delays. However, the injection of falsified control signals in between the actual control signals will affect the performance of the system under attack even if the system is designed to be delay-tolerant. □

The above analysis serves as the background of our proposed attack model $FalCAN$. $FalCAN$ would intend to synchronize with the target control signal perfectly to induce delays in its actuation and inject falsified control signals in between for amplifying the state deviations. Now, we need to figure out a) $whom$ should $FalCAN$ select as the target that increases the probability of attack success, b) $what$ should be the optimal value of the attack a_k at every attack attempt that can maximally degrade the target system's performance while being undetected and c) how the attack can be carried out intelligently in real-time such that the effect of the delayed actuation and falsified control is maximized. We provide a thorough discussion of these topics in the next section.

V. Proposed Methodology

The objective of this work is to devise an attack model and thereby, an attack strategy, that would maximally degrade the targeted system's performance. Here, we bring up the assumptions we consider regarding $FalCAN$.

Assumption 1: In modern automotive systems, the AUTOSAR (AUTomotive Open System ARchitecture)-defined functional safety mandates temporal and spatial isolation of safety-critical tasks from other tasks [1]. Such design standards make manipulating the tasks corresponding to safety-critical controllers quite difficult. Therefore, lightweight in-vehicle communication protocols have become ideal targets for an attacker. Similar to classical BoA, we assume that the attacker is nothing but a piece of code implemented on a compromised ECU running no safety-critical tasks, like a telematic control unit [3]. Also, it targets the CAN bus, the de-facto intra-vehicular communication protocol that transmits most of the real-time safety-critical messages among the ECUs.

Assumption 2: The knowledge of the intra-vehicular network map is available to the attacker. The attacker knows the source of every periodic message in CAN traffic. This can be achieved via ECU fingerprinting techniques based on clock-skew, voltage, temperature, etc. [5], [13], [16], [20]. The attacker can log significant CAN traffic data remotely or physically accessing the vehicle's internal network [14].

Assumption 3: The objective of the attacker is to make the vehicle unsafe or steer it as much near to its safety boundaries as possible. Therefore, we consider the messages corresponding to the control signals of safety-critical automotive CPSs as the primary targets of the attacker.

Assumption 4: To impair the safety of a vehicle, targeting a single safety-critical control loop will suffice. We assume that the attacker is acquainted with the message IDs corresponding to the control signals and sensor measurements of the safety-critical CPSs. This can be achieved by analyzing the CAN traffic while driving the target vehicle in different scenarios, like, turning, speeding up, braking, etc.

Assumption 5: Attacker has the knowledge of the system parameters (A, B, C, K, and L matrices) of the safety-critical control loops. These can be learned by observing the sensor measurements and control inputs in CAN.

We now discuss in detail the target selection method of $FalCAN$, propose a methodology to synthesize optimal attack vectors dynamically at run time and present the steps to launch the $FalCAN$ in real-time.

A. *[Whom to attack]: Identification of target message*

A number of control tasks along with other mixed-critical tasks are co-scheduled on each ECU. The attacker would target the message corresponding to one of the control signals. Let us denote the victim ECU as ECU_{victim} and the IDs of the control signals transmitted from ECU_{victim} are stored in ID_{set}. Destabilizing one such control signal m_v will affect the vehicle's safety. We must select the target control loop in such a way that ensures higher attack success. $FalCAN$ needs to synchronize the attack message m_a with m_v perfectly to induce delay in actuation of m_v. For perfect synchronization, it is desired to have at least one higher-priority message transmitted just preceding the m_v in the CAN bus. Here, we define *attack window* as a sequence of consecutive messages of priority higher than m_v transmitted through CAN just before m_v. If an attacker attempts to transmit the attack message m_a during the attack window of m_v, it will be forced to wait by the CAN arbitration process (see Appendix A). Subsequently, both m_a and m_v will be synchronized. Therefore, the message that has the longest attack window, exhibits higher synchronization probability. Thereby, it is more vulnerable to $FalCAN$. In Algo. 1, we present a systematic way to identify such optimal target for $FalCAN$.

Algo. 1 would analyze a given CAN traffic offline to find the optimal target ID ID_{target} from ID_{set}. Each message $ID \in ID_{set}$ has the following attributes: (i) *periodicity*, (ii) an array ins of instances, (iii) instance count $count$ over a CAN hyper-period, (iv) temporary variable $tWinLen$ to hold attack window length, and (v) average attack window $avgAtkWinLen$ which is the average of attack window lengths of ID's instances. Moreover, each instance $x \in ins$ has the following attribute: duration $atkWinLen$ of higher priority message transmission preceding x. We provide the following as input to Algo. 1: (i) ID_{set}, (ii) hyper-period h in CAN schedule, (iii) minimum length ϵ of attack window to ensure successful

```
Require: List of messages $ID_{set}$ transmitted from victim ECU, CAN hyper-period $h$,
    the minimum length of attack window $\epsilon$ to ensure synchronization, CAN traffic log
    $CAN_{traffic}$
Ensure: Target message ID $ID_{target}$
 1: function IDENTIFYTARGET($ID_{set}$, $h$, $\epsilon$, $CAN_{traffic}$)
 2:     for each $ID \in ID_{set}$ do $ID.count \leftarrow h/ID.period$;
 3:         $ID.tWinLen \leftarrow 0$;
 4:     ANALYZECANTRAFFIC($CAN_{traffic}, ID_{set}$)
 5:     for $ID \in ID_{set}$ do
 6:         $sum \leftarrow 0$;
 7:         for $i = 1$ to $ID.count$ do $sum+ \leftarrow ID.ins[i].atkWinLen$
 8:         $ID.avgAtkWinLen \leftarrow sum/ID.count$;
 9:         if $ID.avgAtkWinLen < \epsilon$ then $ID_{set}.Remove(ID)$;
10:     $SortedID_{Set} \leftarrow$ SORT($ID_{set}, periodicity, asc$);
11:     $ID_{target} \leftarrow SortedID_{set}[0]$;
12:     return $ID_{target}$
13: function ANALYZECANTRAFFIC($CAN_{traffic}, ID_{set}$)
14:     $j \leftarrow 0$                                        ▷ Initialization
15:     while $!EOF(CAN_{traffic})$ do
16:         $next \leftarrow$ CANREAD(); $j \leftarrow j + 1$;
17:         for each $x \in ID_{set}$ do $count \leftarrow x.count$;
18:             if $next.ID > x.ID$ then $x.tWinLen \leftarrow 0$;
19:             else if $next.ID < x.ID$ then $x.tWinLen+ \leftarrow next.Len$;
20:             else                                   ▷ When $next.ID == x.ID$
21:                 ID.ins[j%count].atkWinLen←min(ID.ins[j%count].atkWinLen,
    x.tWinLen);
22:                 $x.tWinLen \leftarrow 0$;
```

Algorithm 1: Optimal Target Identification

synchronization of attack message (m_a) with target message (m_v, see Sec. II-D) with ID same as ID_{target}, and (iv) a CAN traffic log $CAN_{traffic}$.

In line 2, Algo. 1 computes the number of instances of each message in ID_{set} over a CAN hyper-period h. We initialize $tWin$ with 0 (line 3). Algo. 1 next calls ANALYZECANTRAFFIC() in line 4 to analyze the CAN traffic offline (lines 15-22). ANALYZECANTRAFFIC() takes the stored $CAN_{traffic}$ (stored in a file) and ID_{set} as inputs and analyses each message until it reaches the end of the logged $CAN_{traffic}$ (line 15). In line 16, we read a CAN packet $next$ from $CAN_{traffic}$. For each candidate message x in ID_{set}, we first check whether the ID of $next$ is higher than the ID of x (line 18). If so, it signifies that $next$ is of lower priority than x, and we reset $tWinLen$ of x in line 18. Otherwise, the ID of $next$ being lower than the ID of x (line 19) indicates $next$ is of higher priority than x. Therefore, $next$ can contribute to the attack window of x. Subsequently, we update $tWinLen$ in line 19. When IDs of $next$ and x match, it means we have encountered an instance of x while reading CAN traffic (lines 20-22). In this case, we update the attack window length of the particular instance of x in line 21. Therefore, after going through the entire CAN log file, we know the possible attack window length of each instance of each ID in ID_{set}.

From line 5 to 9, Algo. 1 further processes this analysed information. We compute the average attack window length $avgAtkWinLen$ contributed by all the instances of each message ID in line 8. If $avgAtkWinLen$ of an ID is less than ϵ we discard this from consideration of possible target (line 9). It can only happen if a significant number of ID's instances are non-attackable. Next, we sort the pruned list ID_{set} with respect to the periodicity of the messages in ascending order (line 10). The topmost ID of the sorted list $SortedID_{set}$ is the target ID ID_{target} that has the longest average attack window (line 11). Algo. 1 can be used to identify the most vulnerable control signal of each ECU.

B. *[With what to attack]: Synthesis of optimal attack vector*

Once we know the optimal target, we need to form the attack messages. In the proposed attack model $FalCAN$, the attack message is the falsified target control signal only. To compute the false data a_k at each time step such that (i) stealthiness of $\tilde{u}^a_k$ is ensured against the safety monitors (Sec. II-B) and anomaly detectors (Sec. II-C) and (ii) performance of the target system can be degraded maximally, $FalCAN$ determines an estimation $\tilde{x}^a$ of system states as follows.

$$\tilde{x}^a_{k+1} = A\tilde{x}^a_k + B\hat{u}^a_k + L\{y^a_k - C(A\tilde{x}^a_k + B\hat{u}^a_k)\} \qquad (5)$$

In Eq. 5, $\hat{u}^a_k$ and y^a_k denote the most recently received control signal and sensor measurement respectively from CAN. The above consideration is justified as any node connected to CAN can read every packet, and the plant has been last actuated by $\hat{u}^a_k$ only. Using $\tilde{x}^a_{k+1}$, the falsified control signal is computed as $\tilde{u}^a_{k+1} = -K\tilde{x}^a_{k+1} + a_{k+1}$. $FalCAN$ follows two strategies while attacking: i) synchronization with a target control signal to induce delay, and ii) deliberate injection of the falsified control signal to amplify the effect of delayed actuation. In the former case, the attacker must ensure that $\tilde{u}^a_{k+1}$ must be less than the control signal expected (i.e. $-K\tilde{x}^a_{k+1}$) to be transmitted in CAN (refer to CAN arbitration process in Sec. A). Therefore, during synchronization attempts, the attack value a_{k+1} must be negative. On the other hand, while attempting to inject a falsified control signal to CAN, the following conditions must hold for $\tilde{u}^a_{k+1}$ to be stealthy.

(a) *Range constraint on control signal:* The falsified control signal $\tilde{u}^a_{k+1}$ must belong to $[-\theta^u_{val}, \theta^u_{val}]$ i.e. $|\tilde{u}^a_{k+1}| \leq \theta^u_{val} \Rightarrow |-K[(A-LCA)\tilde{x}^a_k + (B-LCB)\hat{u}^a_k + Ly^a_k] + a_{k+1}| \leq \theta^u_{val} \Rightarrow |a_{k+1}| \leq \theta^u_{val} + |K[(A-LCA)\tilde{x}^a_k + (B-LCB)\hat{u}^a_k + Ly^a_k]| \Rightarrow |a_{k+1}| \leq \mathbf{UB^1_{k+1}}$. Here, $\mathbf{UB^1_{k+1}} = \theta^u_{val} + |K[(A-LCA)\tilde{x}^a_k + (B-LCB)\hat{u}^a_k + Ly^a_k]|$.

(b) *Gradient constraint on control signal:* The change in control signal due to addition of a_{k+1} must not drastically change $\tilde{u}^a_{k+1}$ from the last actuated control input which we denote as $\hat{u}^a_k$. This can be ensured if the following relation holds: $|\nabla\tilde{u}^a_{k+1}| \leq \theta^u_{grad} \Rightarrow |\tilde{u}^a_{k+1} - \hat{u}^a_k| \leq \theta^u_{grad} \Rightarrow |-[K\{(A-LCA)\tilde{x}^a_k + (B-LCB)\hat{u}^a_k + Ly^a_k\} + \hat{u}^a_k] + a_{k+1}| \leq \theta^a_{grad} \Rightarrow |a_{k+1}| \leq \theta^a_{grad} + |K[(A-LCA)\tilde{x}^a_k + (B-LCB)\hat{u}^a_k + Ly^a_k] + \hat{u}^a_k| \Rightarrow |a_{k+1}| \leq \mathbf{UB^2_{k+1}}$. Here, $\mathbf{UB^2_{k+1}} = \theta^a_{grad} + |K[(A-LCA)\tilde{x}^a_k + (B-LCB)\hat{u}^a_k + Ly^a_k] + \hat{u}^a_k|$.

(c) *Range constraint on sensor measurements:* The change in the plant state x_{k+2} owing to the actuation of falsified control signal $\tilde{u}^a_{k+1}$ may result in undesired change in sensor measurements y^a_{k+2}. Therefore, for the falsified control signal $\tilde{u}^a_{k+1}$ to introduce error in plant state x_{k+2} stealthily from the range monitor of sensor measurements, the following condition must hold:$|\tilde{y}^a_{k+2}| \leq \theta^y_{val} \Rightarrow |C(A-BK)\tilde{x}^a_{k+1} + CBa_{k+1}| \leq \theta^y_{val} \Rightarrow |a_{k+1}| \leq |B^{-1}C^{-1}|(\theta^y_{val} - |C(A-BK)\tilde{x}^a_{k+1})| \Rightarrow |a_{k+1}| \leq \mathbf{UB^3_{k+1}}$. Here, $\mathbf{UB^3_{k+1}} = |B^{-1}C^{-1}|(\theta^y_{val} - |C(A-BK)\tilde{x}^a_{k+1})|$ and $\tilde{y}^a$ denotes estimation of sensor measurements. Since the attacker has the information of state estimation rather than actual state and the range monitoring rule has to be carried out for future sensor measurements, we compute the estimated sensor measurements to have a bound on a^u_{k+1}. Note that, here B^{-1} or C^{-1} are not the conventional inverse matrices of B, C as these are not square matrices. They are defined in such a way that $B.B^{-1} = I_{n\times n},\ C.C^{-1} = I_{m\times m}$

(d) *Gradient constraint on sensor measurements:* Similar to the last constraint, the change in sensor measurements due to a^u_{k+1} must be within specified limit $[-\theta^y_{grad}, \theta^y_{grad}]$. Therefore, the following must be satisfied to ensure stealthiness of u^a_{k+1}: $|\nabla\tilde{y}^a_{k+2}| \leq \theta^y_{grad} \Rightarrow |\tilde{y}^a_{k+2} - y^a_{k+1}| \leq \theta^y_{grad} \Rightarrow |C(A-BK)\tilde{x}^a_{k+1} - y^a_{k+1} + CBa_{k+1}| \leq \theta^y_{grad} \Rightarrow |a_{k+1}| \leq |B^{-1}C^{-1}|(\theta^y_{grad} - |C(A-BK)\tilde{x}^a_{k+1} - y^a_{k+1}|) \Rightarrow |a^u_{k+1}| \leq \mathbf{UB^4_{k+1}}$. Here, $\mathbf{UB^4_{k+1}} = |B^{-1}C^{-1}|(\theta^y_{grad} - |C(A-BK)\tilde{x}^a_{k+1} - y^a_{k+1}|)$ and y^a_{k+1} denotes the actual sensor measurement that the attacker can read from CAN.

(e) *Residue constraint*: The change in sensor measurements due to a_{k+1} must not let the residue-based detector raise an alarm. This

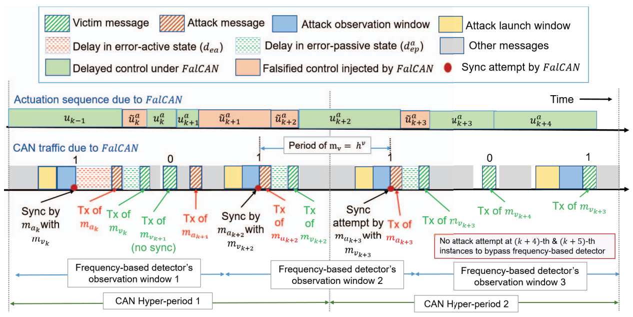

Victim message
Attack message
Attack observation window
Attack launch window
Delay in error-active state (d_{ea})
Delay in error-passive state (d^a_{ep})
Other messages
Delayed control under $FalCAN$
Falsified control injected by $FalCAN$
Sync attempt by $FalCAN$
Actuation sequence due to $FalCAN$
Time
u_{k-1}
$\tilde{u}^a_k$
u^a_k
u^a_{k+1}
$\tilde{u}^a_{k+1}$
$\tilde{u}^a_{k+2}$
u^a_{k+2}
$\tilde{u}^a_{k+3}$
u^a_{k+3}
u^a_{k+4}
CAN traffic due to $FalCAN$
Period of $m_v = h^v$
1
0
1
1
0
1
Sync by m_{a_k} with m_{v_k}
Tx of m_{a_k}
Tx of m_{v_k}
Tx of $m_{v_{k+1}}$ (no sync)
Tx of $m_{a_{k+1}}$
Sync by $m_{a_{k+2}}$ with $m_{v_{k+2}}$
Tx of $m_{a_{k+2}}$
Tx of $m_{v_{k+2}}$
Sync attempt by $m_{a_{k+3}}$ with $m_{v_{k+3}}$
Tx of $m_{a_{k+3}}$
Tx of $m_{v_{k+3}}$
Tx of $m_{v_{k+4}}$
Tx of $m_{v_{k+3}}$
No attack attempt at $(k+4)$-th & $(k+5)$-th instances to bypass frequency-based detector
Frequency-based detector's observation window 1
Frequency-based detector's observation window 2
Frequency-based detector's observation window 3
CAN Hyper-period 1
CAN Hyper-period 2

Fig. 4: An example of the proposed attack method

implies $|r^a_{k+2}| \leq \theta_{res} \Rightarrow |\tilde{y}_{k+2} - C(A\tilde{x}^a_{k+1} + B\hat{u}^a_{k+1})| \leq \theta_{res} \Rightarrow |CBa_{k+1}| \leq \theta_{res} \Rightarrow |a_{k+1}| \leq |B^{-1}C^{-1}|\theta_{res} \Rightarrow |a_{k+1}| \leq \mathbf{UB^5_{k+1}}$. Here, $\mathbf{UB^5_{k+1}} = |B^{-1}C^{-1}|\theta_{res}$. At every time step, whenever $FalCAN$ decides to inject a falsified control signal into CAN, it computes the attack value as,

$$|a^u_{k+1}| = min(\mathbf{UB^1_{k+1}}, \mathbf{UB^2_{k+1}}, \mathbf{UB^3_{k+1}}, \mathbf{UB^4_{k+1}}, \mathbf{UB^5_{k+1}}) \quad (6)$$

This ensures that the $FalCAN$ computes the optimal false data based on the estimation of the current system state in runtime to exert maximal deviation in system states while being undetected by the safety monitors and anomaly detectors. In the next section, we shall discuss how the algorithmic framework of $FalCAN$ bypasses the frequency-based anomaly detection rules.

C. *[How to attack]:* *Attack method*

We have discussed in the previous sections what the ideal target of $FalCAN$ is and how to forge the target control signal optimally that deviates the states of the target control loop maximally evading the safety monitors and anomaly detectors. We now demonstrate the strategies to launch $FalCAN$ on the target with the fabricated attack message in real-time with the help of Fig. 4. We consider there are three instances of the target message arriving at every CAN hyper-period. An instance of the target is attackable only if its preceding attack window is of significant length. The attackable and non-attackable instances are marked with 1 and 0. We divide the attack window into two parts: attack observation (marked with a solid yellow box) and attack launch (marked with a solid blue box) windows. Note that we could have considered the attack observation window to contain messages with a priority lower than the target. However, the proposed *event-triggered* attack model $FalCAN$ attempts synchronization of the attack message m_a with the victim message m_v immediately after reading a message from CAN that falls into the attack observation window of the target. Considering lower priority messages in the observation window may increase attack difficulty and result in mis-synchronizations. Also, the minimum length of the attack launch window should be chosen judiciously such that the synchronization is perfect. The attack observation window of each instance will be the same across the hyper-periods.

Assuming both attacker and victim are initially in the error-active state, let us consider $FalCAN$ is initiated at k-th sample of the target control loop which is the first instance in the first hyper-period in Fig. 4. This results in the actuation of older control signal u_{k-1} for an extended period of d_{ea} samples. This is followed by falsified control $\tilde{u}^a$ actuation for d^a_{ep} samples. Therefore, the actual control signal u^a_k is delayed by overall $(d_{ea} + d^a_{ep})$ samples. The second instance of the target message i.e. $m_{v_{k+1}}$ is not attackable. Instead of a synchronization attempt, $FalCAN$ injects $m_{a_{k+1}}$ immediately following the successful transmission of $m_{v_{k+1}}$. This strategy actually results in the actuation of the falsified control signal for a longer duration (see $\tilde{u}^a_{k+1}$ and $\tilde{u}^a_{k+2}$ in Fig. 4) and thereby, delaying the true control input further. Now, let us consider $\theta_{freq} = 2$ and the desired number of transmissions of m_v during the observation window of the frequency-based detector is 2. In Fig. 4, we can see that the number of violations flagged by the frequency-based detector in its first two observation windows is two. Therefore, $FalCAN$ does not attempt to attack during the transmissions of $m_{v_{k+4}}$ and $m_{v_{k+5}}$. Also, note that $FalCAN$ fabricates the m_v following the method presented in Sec. V-B. $FalCAN$ repeatedly applies these strategies in loop. Next, we provide a formalized methodology to carry out $FalCAN$ in real-time in Algo. 2.

```
Require: List of messages $ID_{set}$ transmitted from victim ECU, CAN hyper-period $h$,
    minimum length of attack window $\epsilon$, minimum attack launch window $\epsilon_1$, frequency-
    based anomaly detector's observation window $N$, CAN traffic log $CAN_{traffic}$,
    online CAN traffic analysis period $timeout$ seconds
 1: function AttackMethod($ID_{set}$, $h$, $\epsilon$, $\epsilon_1$, $N$, $CAN_{traffic}$, $timeout$)
 2:     $k \leftarrow 0; l \leftarrow 0; m \leftarrow 0; flag \leftarrow 0;$                          ▷ Initialization
 3:     $ID_{target} \leftarrow$ IdentifyTarget($ID_{set}, h, \epsilon, CAN_{traffic}$);     ▷ Offline
 4:     $cnt \leftarrow ID_{target}.count$;
 5:     $A, B, C, K, L, Sensor_{target}, X_{safe} \leftarrow$ GetSysInfo($ID_{target}$);
 6:     $\theta_{val}, \theta_{grad}, \theta_{res}, \theta_{freq}, l_{des} \leftarrow$ GetMonitorParams($ID_{target}$);
 7:     $atkWin, atkable \leftarrow$ GetAtkWin($ID_{target}, \epsilon, \epsilon_1, cnt, timeout$);
 8:     while $true$ do
 9:         $CANPacket \leftarrow$ CANRead();                 ▷ Reads CAN packet from Bus
10:         if $CANPacket.ID \in Sensor_{target}$ then
11:             $y^a_k \leftarrow CANPacket.data$;
12:         if $CANPacket.ID == ID_{target}$ then $l++$;
13:             if $l > l_{des}$ then $flag++$;
14:         if $CANPacket.ID \in atkWin[k\%cnt]\&\&flag < \theta_{freq}$ then
15:             $\tilde{x}^a_k \leftarrow$ EstimateState($A, B, C, K, L, \tilde{u}^a_{k-1}, y^a_{k-1}$);
16:             if $atkable[k\%cnt]$ then $\tilde{u}^a_k \leftarrow -K\tilde{x}^a_k - \Delta$;
17:                 AttemptSync($\tilde{u}^a_k$);
18:             else
19:                 $\tilde{u}^a_k \leftarrow -K\tilde{x}^a_k +$ ComputeAtkMsg($A, B, C, K, L, y^a_k$);
20:                 while CANRead()$.ID \neq ID_{target}$ do
21:                     transmit attack message with data $\tilde{u}^a_k$;
22:             $k++$;
23:         $m++$;
24:         if $(m\%N) == 0$ then $l \leftarrow 0$; $flag \leftarrow 0$;
25: function GetAtkWin($ID_{target}$, $\epsilon$, $\epsilon_1$, $cnt$, $timeout$)
26:     $j \leftarrow 0; tWin \leftarrow Null; tWinLen \leftarrow 0;$                              ▷ Initialization
27:     while !$timeout$ do
28:         $next \leftarrow$ CANRead();
29:         if $next.ID > ID_{target}$ then $tWin \leftarrow Null$; $tWinLen \leftarrow 0$;
30:         else if $next.ID < ID_{target}$ then
31:             $tWin.add(next.ID)$; $tWinLen+ \leftarrow next.Len$;
32:         else                                                  ▷ When $next.ID == ID_{target}$
33:             $atkWin[j\%cnt] \leftarrow atkWin[j\%cnt] \cap tWin[\epsilon_1 : tWinLen]$;
34:             $atkWin[j\%cnt].len \leftarrow \min(atkWin[j\%cnt].len, tWinLen - \epsilon_1)$;
35:             if $atkWin[j\%cnt].len + \epsilon_1 > \epsilon$ then $atkable[j\%cnt] \leftarrow 1$;
36:             else $atkable[j\%cnt] \leftarrow 0$;
37:             $tWin \leftarrow Null; tWinLen \leftarrow 0; j \leftarrow j + 1$;
38:     return $atkWin, atkable$;
```

Algorithm 2: Attack Methodology

We provide the same input as Algo. 1 to Algo. 2 as well along with the following: minimum length of the attack launch window ϵ_1, length N of the frequency-based anomaly detector's observation window, and the time duration $timeout$ of online CAN traffic analysis. Algo. 2 starts with finding the optimal target ID_{target} by calling the function IdentifyTarget() from Algo. 1 (line 3). We retrieve the following information of the target in lines 4-6: number of instances cnt of the target in a CAN hyper-period, system matrices A, B, C of the target control loop, set of IDs of the messages with sensor measurements $Sensor_{target}$ of the target control loop, the thresholds θ_{val}, θ_{grad}, θ_{res}, θ_{freq} of the safety monitors and anomaly detectors, and the desired number of times l_{des}, ID_{target} is expected to be transmitted in the observation window of the frequency-based detector.

Algo. 2 next calls GetAtkWin() to compute the attack observation window $atkWin$ and attackability index $atkable$ of each instance of ID_{target} in line 7. In GETATKWIN(), we maintain two variables $tWin$ and $tWinLen$ (line 26) to keep track of each instance's attack window and its length. Until $timeout$, GETATKWIN() reads each CAN packet $next$ and analyzes whether it belongs to attack window of ID_{target}'s next instance (line 27-37). Similar to function ANALYZECANTRAFFIC() in Algo. 1, GETATKWIN() checks if $next$'s ID is more than ID_{target}. If so, $next$ can not belong to ID_{target}'s attack window, and $tWin$ and $tWinLen$ are reset (see line 29). If $next$'s ID is less than ID_{target}, it is a probable member of ID_{target}'s attack window. We then add that in $tWin$ and increment $tWinLen$ by the length of $next$ (see lines 30-31). Once we read a CAN packet having ID same as ID_{target}, it implies we reach a new instance of ID_{target}. The attack observation window of this instance is updated with the common IDs between its last stored attack observation window and the observation window portion ($tWin[\epsilon_1 : tWinLen]$) of $tWin$. For example, while analyzing n-th CAN hyper-period, we derive the attack observation window of ID_{target}'s p-th instance as, say, $\{a, b, c, d\}$. During the analysis in the $(n+1)$-th CAN hyper-period, we store the entire attack window of ID_{target}'s p-th instance in $tWin$ as $\{a, g, e, d, j, f, c, b\}$. Assuming $\epsilon_1 = 4$, the updated attack observation window of ID_{target}'s p-th instance would be $\{a, b, c, d\} \cap \{j, f, c, b\} = \{b, c\}$. This is implemented in line 33. Similarly, we update the attack window length of the current instance of ID_{target} as the minimum of the previously stored ones and the length of $tWinLen$ (note that the entire attack window length, not the observation part) in line 34. The attackable index of the current instance is set to 1 if its attack window length is at least the lower bound (line 35), otherwise, it is set to 0 (line 36). We compute the attack observation windows and attackable index of ID_{target}'s instances online rather than offline because the relative order of the instances will differ in online and offline analysis, and the ordering is crucial. $FalCAN$ needs to know whether the next instance of ID_{target} is attackable and accordingly, it will take different attack strategies.

Following the above analysis, the attack is initiated (line 8). $FalCAN$ starts by reading each CAN packet $CANPacket$ (line 9). It then checks if the currently read message is one of the sensor messages of the target control loop and stores the data value in y_k^a (line 10-11). If it is an instance of ID_{target} only, we increment the counter l that keeps track of number of times ID_{target} appears in an observation window of the frequency-based detector (line 12). If it is more than l_{des}, a flag is raised (line 13). If $CANPacket$ belongs to the attack observation window of ID_{target}'s next instance and the number of times the flag has been raised by the frequency-based detector is less than θ_{freq}, $FalCAN$ aims to inject the falsified control signal (line 14). First, it estimates the state of the plant $\tilde{x}_k^a$ (line 15). Then, it checks if the next instance is attackable. If so, it would attempt synchronization to delay the target control signal. To do so, it computes the attack message as $\tilde{u}_k^a = -K\tilde{x}_k^a - \Delta$ where $\Delta > 0$ (refer to Sec. V-B). Next, $FalCAN$ synchronizes $\tilde{u}_k^a$ with u_k^a in line 17 as demonstrated in Fig. 4. Otherwise, $FalCAN$ generates $\tilde{u}_k^a$ using COMPUTEATKMSG() that follows Eq. 6 (line 19) and transmits $\tilde{u}_k^a$ after u_k^a (line 20-21). $FalCAN$ keeps applying these attack strategies in a loop (line 8-24). In the next section, we demonstrate the efficacy of $FalCAN$ on the performance of safety-critical control loops.

VI. RESULTS AND EVALUATION

Experimental Setup: For in-depth analysis of $FalCAN$'s effectiveness in real-time, a benchmark CAN network [16] is considered.

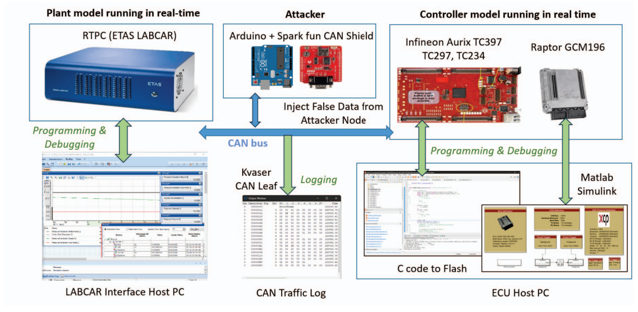


Fig. 5: Hardware-in-loop experimental setup

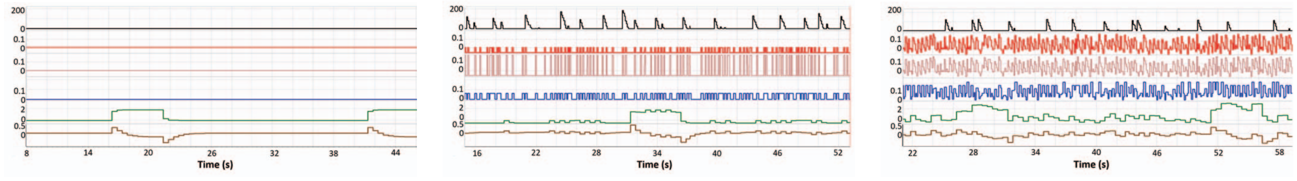
It consists of 4 ECUs, among which one is an emergency brake control module (EBCM) and another one is an engine control module (ECM). To realize this network, we built a lab setup as demonstrated in Fig. 5. It consists of i) ETAS LABCAR real-time PC (RTPC), where we implement the plant, ii) 3 Infineon ECUs (tri-core TC397, dual-core TC297 and 234), a Gannet ECU (Raptor GCM196) on which we schedule multiple mixed-critical tasks including controllers, iii) CAN network that connects these ECUs and RTPC, and iv) a node connected to the same CAN bus having Arduino Uno and Sparkfun CAN shield which is considered as attacker node. We implement a benchmark CAN traffic [16] from a 2011 Chevy-Impala on this CAN-enabled hardware-in-loop (HIL) setup. The integration of industry-grade ECUs enables us to replicate the timing intricacies of a real automotive network in our lab setup, which is important for modeling the delays and attacks following $FalCAN$. The electronic brake control module (EBCM) is one of the ECUs in this setup and is implemented on the Gannet ECU. The engine control module (ECM) is another significant part of our intra-vehicular network setup, which is implemented on the Infineon TC397 ECU. We emulate this network setup and log the CAN traffic by using the Kvaser CAN leaf (Fig. 5).

TABLE I: Monitoring and detection parameters

Monitoring and detection parameters	TTC	ESP
$\lvert\theta^u_{val}\rvert$	$30m/s^2$	10 rad
$\lvert\theta^y_{val}\rvert$	25 m	2 rad/s
$\lvert\theta^u_{grad}\rvert$	$10m/s^3$	1.5 rad/s
$\lvert\theta^y_{grad}\rvert$	30 m/s	$0.175rad/s^2$
$\lvert\theta_{freq}\rvert$	8	16
$\lvert\theta_{res}\rvert$	4.35	4.35

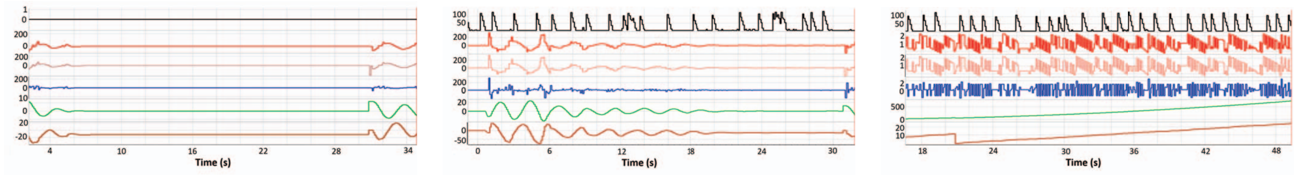
Target System Modeling: We analyze this collected CAN traffic data using the function IDENTIFYTARGET() defined in Algo. 1 to find out which messages that are transmitted from EBMC and ECM, are vulnerable against $FalCAN$. The function IDENTIFYTARGET()[1] returns 0x34A and 0x1C3 respectively for EBMC and ECM. The messages with IDs 0x34A and 0x1C3 correspond to the control signals of the electronic stability program (ESP) and trajectory tracking control (TTC) system respectively. The TTC System regulates the deviation (D) of a vehicle from a given trajectory and a reference velocity (V) by applying proper acceleration u_{acc} which is the control input (message ID 0x1C3) [10]. The measurable state of TTC is the distance D with message ID 0x121. ESP on the other hand stabilizes sideslip (S) and yaw rate (R) by applying correct steering input u_{st} i.e. the control signal (message ID 0x34A) [10]. The yaw rate is the measurable state of ESP with message ID 0x130. The plants of ESP and TTC are implemented in LABCAR. We have considered the system parameters (A, B, C, K, L matrices) of TTC and ESP from [10]. The systems are modeled to tolerate delay up to WCRT. The parameters of the safety monitors and anomaly

[1]Code available here: https://github.com/Ipsitakoley/FalCAN



(a) Under no attack (b) Under $FalCAN$ induced delay (c) Under FDI by $FalCAN$

Fig. 6: Evaluation of $FalCAN$ on ESP [From top to bottom: TEC, actuated steering angle (rad), received steering angle (rad), steering angle gradient (rad/s), yaw rate (rad/s), sideslip (rad/s)]



(a) Under no attack (b) Under $FalCAN$ induced delay (c) Under FDI by $FalCAN$

Fig. 7: Evaluation of $FalCAN$ on TTC [From top to bottom: TEC, actuated acceleration (m/s^2), received acceleration (m/s^2), acceleration gradient (m/s^3), deviation (m), velocity (m/s)]

detectors are given in Tab. I. We simulate both ESP and TTC under periodically varying references. With the same setup, we observe the systems' responses under normal conditions and various strategies of $FalCAN$ considering the bus speed as 500 Kbps and 50% bus load. We connect the attacker node on which we have implemented $FalCAN$ to the above CAN setup. It analyzes the CAN traffic online for a time period of 2 CAN hyper-periods to find out the attackability index and attack observation window of each instance of the target control loops (line 7 of Algo. 2). We have discussed in detail the limitations of the classical BoA approaches in Sec. III. Now, we report detailed observations regarding $FalCAN$. Here, note that we have only reported the outcome of the gradient monitor and frequency-based anomaly detector due to page limitation. However, the results are validated against the remaining monitors and detectors.

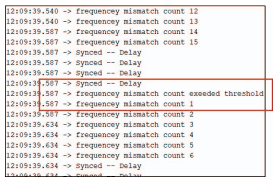
Case study 1: Electronic Stability Program (ESP): We analyze the effectiveness of our proposed attack model $FalCAN$ on ESP in Fig. 6. Fig. 6a represents the normal behavior of ESP when no attack is attempted. We can see the system states are stabilized, and the TEC of EBCM remains 0. Next, we initiate $FalCAN$. We observe the following two scenarios.

Scenario 1: We have already seen that the control execution misses caused by classical BoA does not affect the performance of the system in Fig. 2c (Sec. III). We now observe the effect of repeated actuation delays induced by $FalCAN$ on ESP. Therefore, we only attempt synchronization of the attack message with the target control signal to introduce delays in the control input's actuation (line 16 of Algo. 2) but do not inject any falsified control signal (line 21 of Algo. 2). Before actuating, the received control signal is passed through the safety monitors and anomaly detectors for the sanity check. If it is valid, then only it is actuated. Otherwise, the last valid control input is applied (Sec. II-B). Since $FalCAN$ designs the attack value following the method presented in Sec. V-B, it ensures stealthiness against the safety monitors and anomaly detectors. We can see in Fig. 6b that the received and actuated control signals are the same. This implies the control input received from CAN is considered valid and it is actuated. Also, note that the gradient and range of the received control signals are within their respective limits (given in Tab. I). When the number of times, the flag raised by the frequency-based monitor reaches θ_{freq} (which is 16 for ESP), $FalCAN$ decides not to attempt an attack. This is shown in the attacker's log file in Fig. 8a. In Fig. 8a, *Synced–Delay* implies $FalCAN$ is attempting an attack. Once $flag$ reaches $\theta_{freq} = 16$, it stops attacking ESP until the end of the current observation window of the frequency-based detector. This allows multiple uninterrupted transmissions of the target control signal. As an effect, the TEC of EBCM decreases and never reaches 256. However, the increase in TEC beyond 128 implies synchronization attempts were successful and the actuation is delayed (refer to Sec. IV). We can observe that there is a degradation in the system states, plotted in 5-th (yaw rate in green) and 6-th plot (sideslip in brown) starting from the top in Fig. 6b with respect to the same plots from Fig. 6a due to such attack-induced delayed actuation. The state deviation in ESP due to delay may not seem that significant. However, when the delayed actuation is combined with falsified control actuation, the effect can be detrimental as we see in the next scenario.

Scenario 2: In this scenario, $FalCAN$ follows line 16-21 of Algo. 2. If it observes the next instance of the target control signal is attackable, it attempts synchronization to induce delays in the actuation of the actual control signal. Otherwise, it injects a falsified control signal. Based on the estimation of the system state under delayed actuation, $FalCAN$ designs the stealthy and optimal control signal at run time. The effect of such false data injections (FDI) on ESP's states is demonstrated in the 5-th and 6-th plots from the top in Fig. 6c. In this case, as well, the attack ensures stealthiness against safety monitors (observe the gradient and value of the actuated control input in 2-nd and 4-th plots starting from the top in Fig. 6c respectively) and violation count of the frequency-based detector in Fig. 8a. Evidently, the deviations in the system's states are more significant than Fig. 6b.

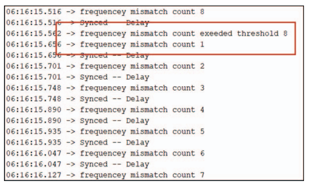
Case study 2: Trajectory Tracking Controller (TTC): The effectiveness of $FalCAN$ on TTC is demonstrated in Fig. 7. Fig. 7a demonstrates the response of TTC under normal conditions i.e. without attack. We can see the system stabilizes normally whenever control is activated due to a change in reference. And, TEC remains 0 always. Like ESP, we explore the following two scenarios in the case of TTC as well.

Scenario 1: $FalCAN$ first attempts to introduce only delays in actuation by repeated synchronization attempts (no injection of falsified control signals). The response of TTC under such attack-induced



```
12:09:39.540 -> frequencey mismatch count 12
12:09:39.540 -> frequencey mismatch count 13
12:09:39.587 -> frequencey mismatch count 14
12:09:39.587 -> frequencey mismatch count 15
12:09:39.587 -> Synced -- Delay
12:09:39.587 -> Synced -- Delay
12:09:39.587 -> Synced -- Delay
12:09:39.587 -> Synced -- Delay
12:09:39.587 -> frequencey mismatch count exeeded threshold
12:09:39.587 -> frequencey mismatch count 1
12:09:39.587 -> frequencey mismatch count 2
12:09:39.634 -> frequencey mismatch count 3
12:09:39.634 -> frequencey mismatch count 4
12:09:39.634 -> frequencey mismatch count 5
12:09:39.634 -> frequencey mismatch count 6
12:09:39.634 -> Synced -- Delay
12:09:39.634 -> Synced -- Delay
```

(a) ESP



```
06:16:15.516 -> frequencey mismatch count 8
06:16:15.516 -> Synced -- Delay
06:16:15.562 -> frequencey mismatch count exeeded threshold 8
06:16:15.656 -> frequencey mismatch count 1
06:16:15.656 -> Synced -- Delay
06:16:15.701 -> frequencey mismatch count 2
06:16:15.701 -> Synced -- Delay
06:16:15.748 -> frequencey mismatch count 3
06:16:15.748 -> Synced -- Delay
06:16:15.890 -> frequencey mismatch count 4
06:16:15.890 -> Synced -- Delay
06:16:15.935 -> frequencey mismatch count 5
06:16:15.935 -> Synced -- Delay
06:16:16.047 -> frequencey mismatch count 6
06:16:16.047 -> Synced -- Delay
06:16:16.127 -> frequencey mismatch count 7
```

(b) TTC

Fig. 8: Attack strategy under frequency-based detector

delayed actuation is presented in Fig. 7b. $FalCAN$ computes the attack message following the method mentioned in Sec. V-B to ensure stealthiness against safety monitors and anomaly detectors. This is evident in the gradient plot (blue plot) in Fig. 7b. Also, we can see in Fig. 8b that whenever a frequency-based anomaly detector raises a flag more than $\theta_{freq} = 8$, $FalCAN$ stops attacking. This lets uninterrupted transmissions of multiple target control instances. Therefore, TEC is not going beyond 256. However, the increase in TEC value signifies successful synchronizations and thereby, delayed actuation. We can see in Fig. 7b that the effect of the repeated delayed actuation on the states of TTC is quite significant with respect to Fig. 7a. This is because TTC is a more sensitive system than ESP. However, since TTC is designed as delay-aware, the system stabilizes after some time. In the next scenario, we discuss how these deviations in the TTC's states can be amplified.

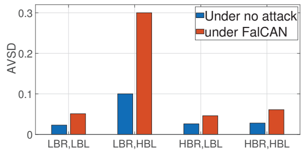
Scenario 2: In this scenario, $FalCAN$ injects falsified control input along with delaying the actual control signal (line 16-21 of Algo. 2). Fig. 7c shows the response of TTC under this attack strategy. Similar to the previous scenario, $FalCAN$ designs the attack following the method given in Sec. V-B. This ensures that the falsified control signal bypasses the safety monitors (see gradient plot in Fig. 7c) and the anomaly detectors (see Fig. 8b). However, we can see the system gradually destabilizes provided the attack maintains its stealth.

Attack Success Analysis: We showed that the proposed attack can incur substantial damage to the target control loops while being undetected by the safety monitors and anomaly detectors. We have also visualized the impact of $FalCAN$ on the dynamics of the vehicle using CarMaker. The videos can be found in GitHub[2]. We now introduce a metric *average state deviation* (AVSD) to provide an overall statistics of $FalCAN$'s efficacy. We define $AVSD$ as $\sqrt{\frac{\sum_{i=1}^{n}({y_i^a}^2 - y_{ref}^2)}{n}}$ where y_i^a is system's output under attack at i-th sample, y_{ref} is the reference output and n is the length of the statistical test window. We measure AVSD of ESP and TTC under normal conditions and under $FalCAN$ by varying bus load and bus rate and report them in Fig. 9. We have considered $n = 400$ and 200 for ESP and TTC respectively. We vary the bus load from 30%-38% (low) to 48%-80% (high), and the bus rate from 250 Kbps (low) to 500 Kbps (high). We can observe, that both systems exhibit higher AVSD under $FalCAN$ than normal conditions. When the bus speed is high, it is difficult to synchronize and AVSD should decrease. However, in the case of TTC, we can see AVSD remains high in high bus rates as well. This is because TTC destabilizes quickly under $FalCAN$ even with fewer FDIs.

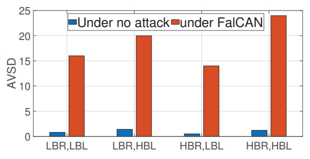
VII. Conclusion

In the above discussion, we point out that if we have a structured methodology for a stealthy and targeted attack that strategically delays the actual control signal and injects falsified control signals based on the estimation of the current system state, the effect of

[2]https://github.com/Ipsitakoley/FalCAN



(a) ESP



(b) TTC

Fig. 9: Average state deviation (AVSD) under $FalCAN$ [LBR: low bus rate, HBR: high bus rate, LBL: low bus load, HBL: high bus load]

the attack can be substantial. Whereas, the classical BoAs fail to incur much impact in the presence of the safety monitors. This work motivates us to synthesize system and platform-aware monitoring systems and anomaly detectors. In this work, we assumed that the closed-loop feedback control is linear and the proposed attack model needs knowledge of system matrices, detection parameters, and message type-to-ID mappings. To make the current work more practical and to implement it on both linear and non-linear system dynamics of a real vehicle, we intend to develop an end-to-end reverse engineering method to acquire this information. Moreover, we also aim to design adaptive monitoring and detection schemes to flag such attack attempts.

References

[1] Overview of Functional Safety Measures in AUTOSAR. https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_EXP_FunctionalSafetyMeasures.pdf, 2022.

[2] Gedare Bloom. Weepingcan: A stealthy can bus-off attack. In *AutoSec*, volume 2021, page 25.

[3] Stephen Checkoway et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX security*, volume 4, 2011.

[4] Kyong-Tak Cho et al. Error handling of in-vehicle networks makes them vulnerable. In *CCS*, 2016.

[5] Kyong-Tak Cho et al. Fingerprinting electronic control units for vehicle intrusion detection. In *USENIX Security)*, 2016.

[6] Jairo Giraldo et al. A survey of physics-based attack detection in cyber-physical systems. *ACM CSUR*, 51(4), 2018.

[7] Sena Hounsinou et al. Vulnerability of controller area network to schedule-based attacks. In *RTSS*. IEEE, 2021.

[8] Kazuki Iehira et al. Spoofing attack using bus-off attacks against a specific ecu of the can bus. In *2018 15th IEEE Annual Consumer Communications & Networking Conference*, pages 1–4. IEEE, 2018.

[9] Hyo Jin Jo et al. A survey of attacks on controller area networks and corresponding countermeasures. *T-ITS*, 23(7):6123–6141, 2021.

[10] Ipsita Koley et al. Catch me if you learn: Real-time attack detection and mitigation in learning enabled cps. In *RTSS*. IEEE, 2021.

[11] Ipsita Koley et al. Cad support for security and robustness analysis of safety-critical automotive software. *TCPS*, 7(1):1–26, 2023.

[12] Karl Koscher et al. Experimental security analysis of a modern automobile. In *S&P*. IEEE, 2010.

[13] Sekar Kulandaivel et al. {CANvas}: Fast and inexpensive automotive network mapping. In *USENIX Security*, 2019.

[14] Charlie Miller et al. Adventures in automotive networks and control units. *Def Con*, 21(260-264):15–31, 2013.

[15] Yilin Mo et al. False data injection attacks against state estimation in wireless sensor networks. In *CDC*, pages 5967–5972. IEEE, 2010.

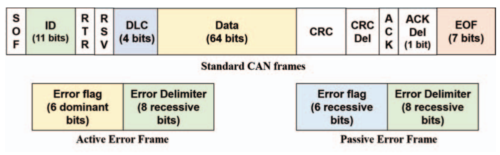
[16] Khaled Serag et al. Exposing new vulnerabilities of error handling mechanism in can. In *USENIX Security*, 2021.

[17] Yasser Shoukry et al. Non-invasive spoofing attacks for anti-lock braking systems. In *CHES*, pages 55–72. Springer, 2013.

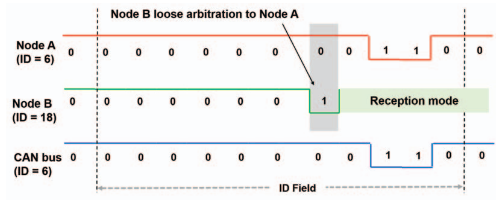
[18] CAN Specification. v2. 0. *Common public radio interface (CPRI)*, pages 1–75, 2004.

[19] Andre Teixeira et al. Secure control systems: A quantitative risk management approach. *IEEE Control Systems Magazine*, 2015.

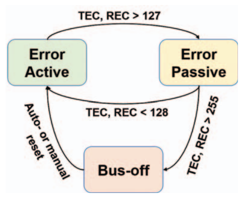
[20] Miaoqing Tian et al. Exploiting temperature-varied ecu fingerprints for source identification in in-vehicle network intrusion detection. In *IPCCC*. IEEE, 2019.



(a) CAN frames



(b) CAN arbitration



(c) Error states in CAN

Fig. 10: CAN Properties

Appendix A
Basics of CAN Protocol

CAN transmits data in the broadcast mode as a sequence of recessive (1) and dominant (0) bits [18]. CAN transmits both periodic and aperiodic messages. CAN hyper-periods are the time windows in CAN traffic in which the sequence of periodic CAN messages repeat. We highlight a few properties of CAN protocol here that are significant for our proposed attack model.

CAN Frames: There are four basic types of CAN frames: data frame for transmitting data, remote frame for requesting remote transmission of specific data, error frame for indicating transmission or reception error, and overload frame for introducing a delay between frames. The format of the data frame is laid out in Fig. 10a. CAN frames do not have any source or destination address. Rather, each frame has a unique identifier field ID that signifies its priority and type of data being carried in the frame. Data frames can contain maximum of 64 bits of data. Between transmissions of two consecutive frames in the CAN bus, there is an inter-frame space (IFS) of 3 bits. Therefore, at the end of each CAN frame, there is a sequence of 11 recessive bits (1 bit of acknowledgment delimiter + 7 end-of-frame bits + 3 bits of IFS).

Arbitration: Contention to the CAN bus by multiple messages at the same time is resolved through the *CAN arbitration* process [18]. Technically, CAN works as a *wired-AND gate*. An ECU sending a message with a higher ID value will see a *dominant bit i.e.* 0 in the bus if another ECU sends a message with a lower ID value at the same time, and thereby, retrieves itself from transmitting. Thus, *a message with a lower ID value is considered to be of higher priority over a message with a higher ID value*. An example of CAN arbitration is showcased in Fig. 10b.

Error Handling: CAN protocol defines 5 types of error [18], namely bit error, stuff error, acknowledgment error, and CRC error, which can occur during either transmission or reception of CAN frames. Each CAN node tracks the number of errors using transmission error count (TEC) and reception error count (REC). Error in a single transmission and reception increases TEC and REC by 8 and 1 respectively. Whereas, a successful transmission and reception reduce TEC and REC by 1. CAN define 3 error states based on the value of TEC and REC as demonstrated in Fig. 10c. An ECU (i) remains in *error active* mode if the TEC and REC are less than 128, (ii) enters an *error passive* mode when TEC or REC exceeds 127, but remains below 256, and (iii) goes to the *bus-off state* i.e. gets disconnected from the bus temporarily when TEC or REC exceeds 255 to notify some significant disruption in transmission. An ECU can come out of bus-off mode either manually or automatically. An automatic recovery needs 128 instances of IFS i.e. 11 recessive bits in the bus. During the error active state, a CAN node sends an active error flag containing 6 dominant bits followed by 8 recessive bits [18] on sensing any error. On the other hand, error frames in error passive mode contain 14 recessive bits (Fig. 10a) [18].

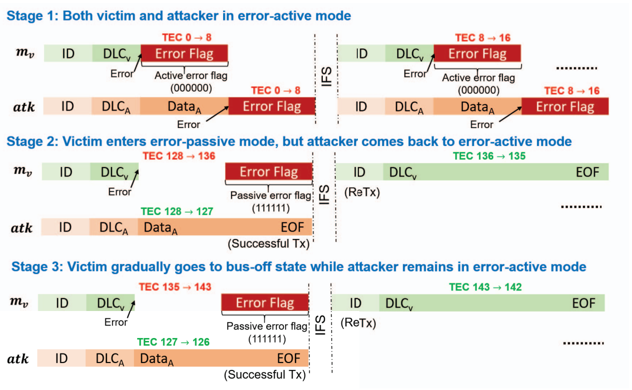

Stage 1: Both victim and attacker in error-active mode
m_v
atk
ID
DLC$_V$
DLC$_A$
Data$_A$
Error Flag
TEC 0 → 8
TEC 8 → 16
Error
Active error flag (000000)
IFS
Stage 2: Victim enters error-passive mode, but attacker comes back to error-active mode
TEC 128 → 136
TEC 136 → 135
TEC 128 → 127
Passive error flag (111111)
(ReTx)
EOF
(Successful Tx)
Stage 3: Victim gradually goes to bus-off state while attacker remains in error-active mode
TEC 135 → 143
TEC 143 → 142
TEC 127 → 126

Fig. 11: Different Stages Bus-Off Attack

Appendix B
Bus-off Attack

The bus-off attack (BoA) cuts a victim ECU from CAN bus by exploiting the error-handling strategy of the CAN protocol. It targets one of the messages m_v transmitted from the victim ECU. Before launching the BoA, the attacker, which is fundamentally a piece of code implemented on a compromised ECU, monitors the CAN bus traffic to compute the timing parameters (periodicity, arrival offsets, etc.) of m_v. It fabricates attack messages atk with the same ID as m_v and a value in the data field less than the one in m_v. It is assumed that initially both the victim node i.e. the ECU transmitting m_v, and the attacker node, i.e. the compromised/attacker ECU are in error active mode.

The 3 stages of BoA are demonstrated in Fig 11. At the first stage, the attacker injects atk in the CAN bus *synchronously* with m_v i.e. exactly at the same time. Since both atk and m_v have the same ID, both win the arbitration process at the ID field. However, during the transmission of the data field, the transmitting ECU of m_v senses the error first, and broadcasts an active error flag (containing 000000). As CAN works as a wired-AND gate, attacker ECU senses transmission error as well. This leads to an increase in TEC by 8 for both the victim and attacker ECUs. Due to the failed transmissions, both victim and attacker ECUs attempt retransmissions of their respective messages at the same time and the same transmission error occurs. After 16 consecutive retransmission attempts, the TEC of both attacker and victim node reach 128 i.e. both the nodes enter error-passive mode. In the second stage, synchronization of m_v and atk in error-passive mode causes the victim ECU to send a passive error flag (containing 111111) with a further increase in TEC by 8. As this generates no error in the attacker node, the attack message is transmitted successfully by reducing its TEC by 1. consequently, the victim ECU's subsequent re-transmission attempt becomes successful without any obstacle from the attacker and reduces its TEC by 1. Therefore, attacker's TEC is reduced by 1 while victim's TEC is increased by $8 - 1 = 7$. At this third stage of the attack, after subsequent synchronous transmissions in error passive mode, the attacker comes back to the error active mode but the victim eventually goes into the bus-off state.

Appendix C
Proof of Claim 1

In general, any control system is affected by delays generated by platform-level uncertainties. In the case of the CAN bus, this delay is the worst-case response time (WCRT) of a message due to jitters, blocking time caused by low-priority messages, and waiting time caused by high-priority messages. When the system is not under attack, because of WCRT, its states are updated (ignoring the process noise) as, $x_{k+1} = Ax_k + B'u_{k-1} + B''u_k = Ax_k + B_c d_{wcrt} u_{k-1} + B_c(h - d_{wcrt})u_k$. The control signal that is supposed to reach at $(k+1)$-th sample is delayed due to WCRT by d_{WCRT} samples during which the plant is actuated by the last received control signal u_{k-1}. Once u_k arrives, it is actuated during the remaining time window of $(h - d_{WCRT})$ samples. Now, assuming $x_k \simeq \hat{x}_k$ at steady state of the estimator, we get

$$x_{k+1} = [A - B_c(h - d_{wcrt})K]x_k - B_c d_{wcrt} K x_{k-1} \tag{7}$$

The above equation is a second-order linear homogeneous recurrence relation. The solutions of Eq. 7's characteristic equation are distinct values r_1 and r_2, and function of d_{wcrt}. Therefore, the solution of Eq. 7 is

$$x_{k+1} = \alpha r_1^{k+1} + \beta r_2^{k+1} \tag{8}$$

Here, α and β are arbitrary constants. Consider the *case 1* demonstrated in Sec. IV i.e. when $FalCAN$ is initiated and the victim is in the error active state. Assuming the steady state of estimation and ignoring the process noise, Eq. 2 can be written as the following second-order non-homogeneous recurrence relation,

$$x_{k+1}^a = [A - B_c(h - d_{ea})K]x_k^a - B_c d_{ea} K x_{k-1}^a + B_c d_{ep}^a a_k \tag{9}$$

The general solution of the associated homogeneous relation in Eq. 9 is $x_{k+1}^{a^{(h)}} = \alpha s_1^{k+1} + \beta s_2^{k+1}$. This is similar to Eq. 8 except that the distinct values s_1 and s_2 are functions of d_{ea} rather than functions of d_{wcrt} like r_1 and r_2. For ease of analysis, let us assume that $FalCAN$ considers a constant attack value at each time it attempts a false data injection. This implies the particular solution of Eq. 9 is $x_{k+1}^{a^{(p)}} = \gamma$ where γ is a constant. Therefore, the solution of Eq. 9 is

$$x_{k+1}^a = \alpha s_1^{k+1} + \beta s_2^{k+1} + \gamma \tag{10}$$

From Eq. 8 and 10, we can infer if the induced delay in error active state is more than the usual WCRT, the deviation in system state will be more i.e. $d_{ea} > d_{wcrt} \Rightarrow ||x_{k+1}^{a^{(h)}}||_2 > ||x_{k+1}||_2$. Moreover, the delay d_{ep}^a introduced by every synchronization attempt in the error passive state along with the false data of higher amplitude $|a_k|$ will further degrade the system states. Similar analysis can be drawn as well for *case 2* and *case 3* mentioned in Sec. IV.