Vulnerability of Controller Area Network to Schedule-Based Attacks

Sena Hounsinou*, Mark Stidd*, Uchenna Ezeobi*, Habeeb Olufowobi†, Mitra Nasri‡, Gedare Bloom*
*University of Colorado Colorado Springs, Colorado Springs, CO, USA
†University of Texas at Arlington, Arlington, TX, USA
‡Eindhoven University of Technology, Eindhoven, Netherlands
{shoueto, mstidd2, uezeobi, gbloom}@uccs.edu, habeeb.olufowobi@uta.edu, m.nasri@tue.nl

***Abstract*—The secure functioning of automotive systems is vital to the safety of their passengers and other roadway users. One of the critical functions for safety is the controller area network (CAN), which interconnects the safety-critical electronic control units (ECUs) in the majority of ground vehicles. Unfortunately CAN is known to be vulnerable to several attacks. One such attack is the bus-off attack, which can be used to cause a victim ECU to disconnect itself from the CAN bus and, subsequently, for an attacker to masquerade as that ECU. A limitation of the bus-off attack is that it requires the attacker to achieve tight synchronization between the transmission of the victim and the attacker's injected message. In this paper, we introduce a schedule-based attack framework for the CAN bus-off attack that uses the real-time schedule of the CAN bus to predict more attack opportunities than previously known. We describe a ranking method for an attacker to select and optimize its attack injections with respect to criteria such as attack success rate, bus perturbation, or attack latency. The results show that vulnerabilities of the CAN bus can be enhanced by schedule-based attacks.**

***Index Terms*—Controller Area Network, Schedule-Based Attack, Bus-off Attack**

I. Introduction

The evolution of electronic control units (ECUs) and networks has made the vehicle a complex cyber-physical system. The modern vehicle now embeds up to 100 ECUs with millions of lines of software code that enable their communication with the outside environment and the Internet. The dramatic increase in vehicle functionality has exposed their safety-critical systems to cyber-physical risks and attacks. In-vehicle networks, such as the controller area network (CAN) bus, have several vulnerabilities [1]–[3]. The CAN bus is an attractive target for cyber attackers because it does not implement security mechanisms despite being widely used in automotive and avionics domains [4].

The CAN bus presents a predictable behavior to satisfy the transmission requirements of each message. While the schedulability analysis of the CAN bus guarantees messages will meet their deadline in the worst-case [5], the determinism of its timing model can expose the scheduling information of the safety-critical messages [6,7]. In this paper, we show that this information enables an adversary to initiate schedule-based attacks to compromise the automotive system more effectively than prior attack methods.

A schedule is a sequence of resource allocations to entities, and a successful schedule-based attack uses the schedule to determine when an attack should happen with respect to the timing and ordering of the targeted entity's resource access. We introduce, formalize, and evaluate the efficacy of a schedule-based attack on the CAN bus by leveraging the transmission schedule.

The main contributions of this paper are:

1) A comprehensive investigation of the vulnerabilities of the CAN bus to schedule-based bus-off attacks. This investigation includes analysis of the traditional bus-off attack strategy for synchronizing the attack injection, and demonstration that the schedule-based approach is more effective.
2) A framework for schedule-based attacks that includes schedule analysis, prediction, and exploitation.
3) An optimization strategy to maximize the attacker's criteria (e.g., short response time, high success ratio) within the exploitation stage of the schedule-based attack framework.

The remainder of the paper is as follows: first, we provide some background about CAN and the traditional bus-off attack, followed by the motivation for this work in Section II. In Section III-A we describe the system model and in Section III-B the threat model used throughout this paper. The schedule-based attack and our approach to implement it are presented in Section IV. In Section V we describe our experimental evaluation. The related work is reviewed in Section VII and Section VIII concludes the paper.

II. Background and Motivation

In this section, we provide some preliminaries about CAN bus and the bus-off attack.

A. Controller Area Network (CAN)

CAN is a broadcast serial communication protocol with carrier-sense multiple access and collision detection (CSMA/CD) that uses a lossless bitwise arbitration to transmit binary signals over twisted pair cabling. The bus facilitates

This work is partially supported by NSF CNS-2046705, NSF CNS-2011620, NSF OAC-2001789, and Colorado State Bill 18-086.

reliable communication with no central controller. CAN transfers messages according to a fixed-priority non-preemptive scheduling policy and uses non-return to zero bit encoding. CAN bus signifies bits with logic value of 0 as dominant and bits with logic value of 1 as recessive. It distinguishes 4 different message frame formats: data, error, remote, and overload frames. Data is transmitted in the bus through the data frame that includes the identifier field (ID) used in the arbitration process. A data frame contains up to 8 bytes of data field and its length is specified in the *data length code* (DLC).

CAN applies a non-destructive arbitration process and, when a message wins arbitration and starts transmission, it can not be preempted. Messages win arbitration according to their priority determined by the message ID; the smaller the ID the higher the priority. Any node may start transmission when it has detected that the bus is idle, beyond the *interframe space* (IFS). Transmission begins by sending a dominant *start of frame* (SOF) bit, followed by ID bits, one at the time, starting with the most-significant bit. When multiple nodes start transmitting concurrently, the bus is recessive only if all nodes float a recessive bit, and a transmitting node monitors the bus to ensure that it does not read a dominant bit when it sent a recessive bit, thereby losing arbitration.

Figure 1 shows an example of three nodes during arbitration. At bit time T_0, *Node 1*, *Node 2*, and *Node 3* attempt to start transmission by introducing a dominant SOF bit on the CAN bus. At time T_1, all three nodes send a dominant bit as their first identifier bit. During the following bit time, *Node 3* reads that a dominant bit has been introduced on the bus (as both *Node 1* and *Node 2* sent a dominant bit) while it fails to transmit a recessive bit. At that point, *Node 3* loses arbitration and stops transmitting. After T_2, only *Node 1* and *Node 2* remain in arbitration until time T_3 when *Node 1* fails to transmit a recessive bit. *Node 2* therefore wins arbitration and continues transmission beyond T_3. *Node 1* and *Node 3* will automatically attempt to retransmit at T_4 after *Node 2* has completed transmitting and the IFS has elapsed.

B. CAN Fault Confinement

The CAN protocol implements robust error management that is essential for fault tolerance and aimed at detecting errors caused by disturbances or hardware faults in the bus. When an error occurs in the bus communication, an error flag is raised that is signaled to all nodes on the bus. The node that detects the error, depending on its state—error-active or error-passive—transmits either an error-active flag of six dominant bits or an error-passive flag of six recessive bits, and the erroneous message is discarded by the receiving nodes. The frame in transmission is then queued for retransmission by the sending node.

Each node implements two error counters with initial values of 0: *transmit error counter* (TEC) and *receive error counter* (REC). The transitions between the error states depend on the counters. When an error is observed by a non-transmitting node, its REC is increased by 1, and when observed by a transmitting node, then the TEC is increased by 8. If the TEC or REC exceeds 127, the node transitions from error-active to error-passive state, and from passive to bus-off when TEC reaches 255. An error-active state is restored when the TEC and REC of a node are below 128.

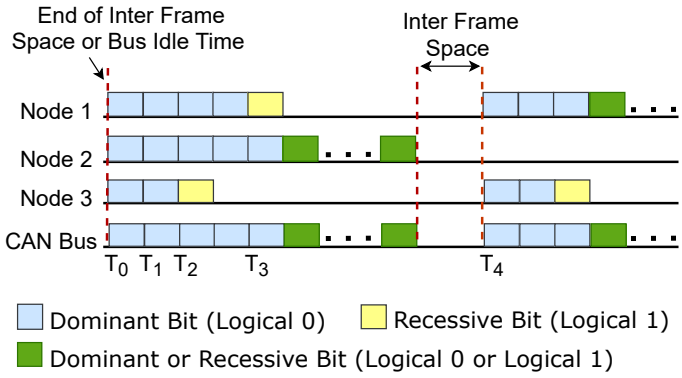


Fig. 1: CAN Arbitration Example. Node 1-3 arbitrate simultaneously. Node 2 wins the bus at time T3 and sends its remaining bits (in green). During the next transmission attempt at time T4, Nodes 1 wins arbitration against Node 3.

The bus-off state is an error state of a CAN controller in which the node is disconnected from the bus communications, i.e., it can neither transmit nor acknowledge frames. A node that is in the bus-off state can only rejoin the network after observing 128 occurrences of the bus-free signal of 11 consecutive recessive bits. When a node recovers from bus-off, it resets its counter and starts from the initial error-active state.

C. CAN Bus-Off Attack

In 2016, Cho and Shin [8] introduced the bus-off attack, which exploits the fault tolerance feature of the CAN protocol. The authors identified three conditions that need to be satisfied for a successful attack: (i) the attack message must have the same ID as the victim message; (ii) it must be synchronized with the victim message, and (iii) it must have a dominant bit in at least one position where the victim message has a recessive bit. Under the software-based (remote) threat model, the dominant bit must occur in either the control or data fields of the CAN frame since the ID field of both messages have to match, and some fields (SOF, EOF, and IFS) have a fixed format while other fields (CRC and ACK) are generated by the controller hardware automatically.

The success of the attack depends on the adversary's ability to synchronize roughly 16 transmissions of the victim ID with a guaranteed *unique* preceded message. We revisit discussion of the existence and effectiveness of these preceded messages in Section II-D. However, some of these messages could be skipped or may require more than 16 transmissions if the victim ECU sends multiple message IDs. (Successful transmission of other messages from the same CAN controller on the ECU would decrease the TEC, thereby lengthening the attack success time.)

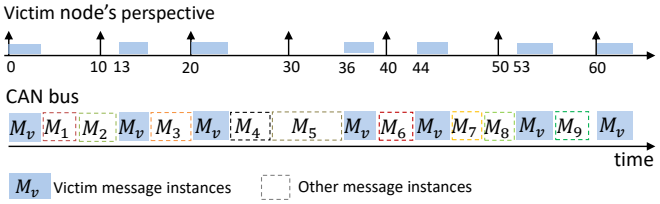


Fig. 2: Example of CAN schedule. The victim message shown in light gray is preceded by idle time and seven different messages in a hyperperiod.

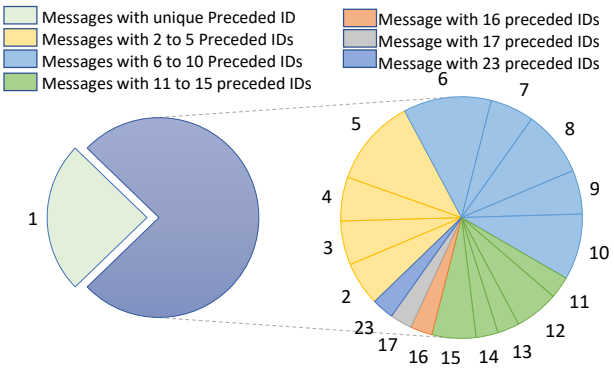


Fig. 3: Real CAN Traffic: 34 out of 45 message IDs have non-unique preceded IDs. The size of each slice represents the number of messages having the labeled number of different preceded IDs in a hyperperiod.

D. Motivation for the Schedule-Based Attack

To succeed in the traditional bus-off attack [8], the victim and attack messages must be synchronized, as specified by condition (ii) (see Section II-C). This condition is generally met by identifying a *unique* message that precedes the victim and is used to synchronize the victim and attack messages. The synchronization is accomplished by enqueueing the attack message M_a when the victim message M_v is expected to encounter blocking (from a preceded message of lower priority) or interference (from one or multiple higher-priority preceded messages): because of the blocking or interference, M_a and M_v will arbitrate and transmit synchronously.

Since a software-based attacker can only observe the completion of messages, the attacker watches for when the identified preceded ID finishes transmission and enqueues M_a before the arbitration phase for the victim message begins. However, when no unique preceded ID can be identified, it becomes more challenging to predict the arrival of the victim message and launch the attack.

For example, consider the hyperperiod of the schedule depicted in Figure 2 consisting of ten periodic messages. The victim message has a period of $P_v = 10$ and the remaining seven messages have a period of 70. During the hyperperiod, 7 instances of M_v are released. As it can be seen in this example, five of the instances of the victim message are preceded with five distinct message instances (in this example, by M_2, M_3, M_5, M_6, and M_8 and two message instances have no preceded message (the first and the last messages of M_v). Since the preceding message IDs are unique, the attack model introduced by Cho and Shin [8] can, at best, target only one of the instances of the victim message and hence their bus-off attack will fail because then only one attack message can be injected in a hyperperiod and the remaining six instances of M_v will therefore still transmit successfully and decrement the TEC to 0 before the next injection of the attack message (which occurs in the next hyperperiod).

To overcome this challenge, Cho and Shin [8] elaborated on how to fabricate and inject unique preceded IDs on the CAN bus to interfere with the transmission of M_v. To succeed in mounting the bus-off attack with this approach, an adversary fabricates messages to synchronize its transmission timing and inject the message right before the target message. Their evaluation shows that one injection of fabricated preceded ID message per attack can be sufficient for a bus-off attack. This approach requires that the adversary chooses an acceptable value for the ID field of the fabricated message. By periodically sending fabricated attack messages, the bus load is artificially inflated prior to the beginning of the attack, and may be a detectable feature that intrusion detection systems can use to mitigate the bus-off attack. For this approach to work well requires the attacker to know the available bus idle times and fraction of available bandwidth, hence would itself require the kind of schedule-based reverse engineering necessary for our work while still potentially leading to the bus going into overload or causing lower priority messages to miss their deadlines.

The schedule-based bus-off attack does not rely on the uniqueness of the preceded ID to achieve the message synchronization required by condition (ii). Namely, a CAN message can be a target of a schedule-based bus-off attack even if messages preceding its transmissions are *not* of the same ID every time. By relaxing this requirement, it becomes possible to launch an attack on any instance of the victim message with which an attack message can be synchronized.

We show that with a schedule-based bus-off attack, an opportunity to attack appears for any instance of the victim message that incurs blocking or interference, *regardless of the message causing it*. For example, in Figure 2, although M_v does not have a unique preceded ID, its instances can be a target of a schedule-based bus-off attack as they encounter interference from other CAN messages.

To identify those instances of a victim message that the adversary can synchronize the attack message with, the adversary observes the CAN bus to gain knowledge about the message sets prior to launching the attack. Such knowledge includes the sequence of distinct message instance transmissions including their IDs and completion times. The attacker can leverage this information to identify opportunities that can result in a successful synchronization of the attack message with the victim message.

To see the prevalence of messages with non-unique preceded IDs in real vehicle data, we analyzed a real CAN

TABLE I Summary of Notations

Variable	Definition
M	Set of Messages $M = (M_1, M_2, ..., M_n)$
M_i	Message with ID i, where $1 \leq i \leq n$
$M_{i,k}$	The kth instance of M_i
P_i	Period of M_i
C_i	Transmission time of M_i
J_i	Jitter of M_i
u_i	Utilization of M_i
U	System utilization
ϕ_i	Offset of M_i
$T_{i,k}$	Completion time of $M_{i,k}$
M_a	Attack message
v	ID of victim message
$M_{v,x}$	Instance x of the victim message to be attacked

traffic data set that was captured for approximately 30 minutes through the OBD-II port [9]. The data set contains 2,369,868 attack-free messages from 45 distinct IDs. Figure 3 shows the breakdown of the ratio of message IDs with different numbers of preceded IDs. Eleven message IDs have a unique preceded ID, while the other 34 message IDs have between 2 and 23 distinct preceded IDs. In our analysis, 10 messages in total have two to five preceded IDs, 14 messages have six to ten preceded IDs, 7 message IDs have between 11 to 15 preceded message IDs, and finally, the three remaining IDs in the message set each have 16, 17, and 23 non-unique preceded IDs. Thus, more IDs may be vulnerable to the schedule-based bus-off attack than to the traditional bus-off attack relying on preceded messages.

III. System and Threat Models

In this section, we present the CAN timing model and how it is favorable for schedule-based attacks. We also discuss the capabilities and objectives of the adversary.

A. System Model

We consider a CAN bus with a (ordered) set $M = \{M_1, \ldots, M_n\}$ of n periodic messages and a hyperperiod h that is the least-common multiple of their periods. Without loss of generality, every message $M_i \in M$ has ID i equal to its priority and is characterized by its period P_i, worst-case transmission time C_i, and maximum jitter J_i. A message M_i produces multiple message instances during the lifetime of the system. $n_i = h/P_i$ is the number of instances of M_i in one hyperperiod. We enumerate those instances as $M_{i,k} \in \{M_{i,1}, \ldots, M_{i,n_i}\}$. We denote the total bus utilization as $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$. The notation is summarized in Table I.

Because the CAN bus is a broadcast system, the adversary can compute C_i, and also record the ID and the frequency of the messages in transmission. However, delays incurred inside an ECU prior to transmission are unknown to CAN observers. For example, as shown in Figure 4, in addition to J_i, messages may incur other queuing delays such as *blocking* and *interference*. *Blocking* is the delay encountered when a message must wait for another lower-priority message to complete transmission before attempting to transmit. *Interference* refers to the time a message must wait while messages of higher priority transmit. Tindell et al. [10] and Davis et al. [5] provide a model used to estimate the value of the queuing delays for a set of CAN bus messages.

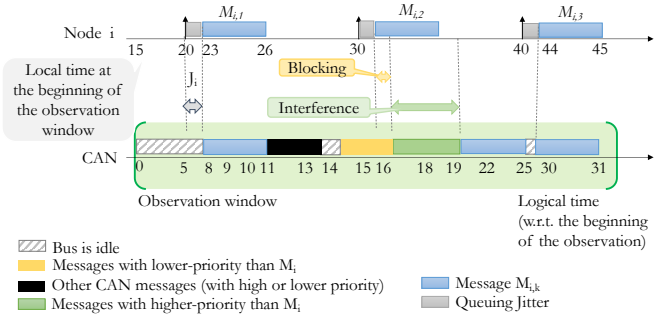


Fig. 4: CAN Timing Model.

In a typical safety-critical system, nodes related to safety-critical functionality have the highest priority and are sent the most frequently. This work does not put assumptions for the priority ordering and deadline of the messages transmitted on the CAN bus.

B. Threat Model

In this paper we assume an adversary that can use a software application to control an ECU that is communicating in the bus by exploiting a vulnerability. Here, the control of one ECU limits the impact the attacker may have on the vehicle, as they can only manipulate that ECU's behavior. For example, if the attacker's goal is (say) to crash the car, then the control of the Wi-Fi or cellular connection of the vehicle is not that helpful, but if the cellular connection ECU is on the CAN bus or connected through the gateway, then the attacker can use that foothold to achieve its goal [11] by causing safety-critical ECUs to go into the bus-off state. We do not examine how the adversary gained control of the ECU as this has been thoroughly demonstrated [3,12], but we assume that access is gained remotely, as described in prior work [1,3,8,12,13].

We assume that the attacker has supervisor mode access and is able to read/write the message buffers (used to enqueue transmissions and read received messages) and the configuration registers of the CAN controller, which includes message acceptance filters. Similar to prior work in this area [1,3,8,14]–[16], we also assume that the attacker can inject any data frame on the CAN bus and modify the acceptance filters of the compromised ECU through software commands to eavesdrop on the CAN bus traffic [8].

With the given capabilities, the adversary can collect logs of messages sent through the CAN bus, and analyze the log to understand the schedule of messages sent through the network. Based on this analysis, the attacker will be able to launch an attack on a particular message ID, called the *victim message* hereafter.

It is feasible and typical to assume that an attacker knows the make, model, or even trim of the target automobile, and can analyze it offline to derive the message parameters, i.e., their IDs, and relationship between data payloads and functionality. Partial schedule knowledge is possible because of the dependencies between some messages (i.e., receipt of certain messages cause the transmission of others). Problematically, the vehicle system comprises complex, distributed, independently functioning ECUs, and therefore the schedule of messages and their functionality cannot typically be determined statically [17]. Also, uncertainty introduced by jitter, non-deterministic startup times, and physical effects caused by clock drift and power fluctuation make it impossible to reconstruct one precise system schedule offline.

The goal of the attacker is to send a predetermined node to the bus-off state by increasing its TEC to 255. To achieve this, the adversary enqueues attack messages (denoted as M_a) at predefined times to transmit at the same time as the transmission of a specific victim message (denoted as M_v).

We also suppose that the adversary aims to cause the least disturbance (i.e., missed deadlines) to the CAN bus. Akesson et al. [4] show that an overwhelming majority of systems have a mechanism in place to respond (e.g., reporting the event, rebooting the system, and restarting the task) in case of a missed deadline. In addition, intrusion detection systems may be triggered in such situations [18,19]. Thus, the adversary's aim is to succeed in the attack with the minimum number of message injections to reduce perturbing the behavior of non-victim message transmissions, by transmitting the attack messages only on the instances of the victim when the chance of a successful message injection is high, and using a sequence of instances which result in increasing the TEC of the victim in the shortest time frame.

IV. The Schedule-Based Attack

The schedule-based attacker aims to drive a target node to bus-off state, as in the work of Cho and Shin [8]. However, it fundamentally differs from the previous approach in the following ways:

- **a1:** the attacker does not fabricate preceded IDs;
- **a2:** the attacker does not rely on the uniqueness of preceded IDs.

To satisfy **a1**, the adversary must only rely on normal message transmissions to synchronize the attack message with the victim. In the original bus-off attack, the adversary simply observes the bus traffic until the unique preceded ID transmits. It indicates to the adversary that the following message is the victim. Thus, instances of M_v that are transmitted after messages other than the known preceded ID are not exploitable by the attacker. For **a2**, the schedule-based attacker can also make use of those instances to drive the victim node's TEC to 255. This increases the number of instances that can be targeted in a single hyperperiod to "any instance of M_v that incurs blocking or interference" regardless of the message causing it.

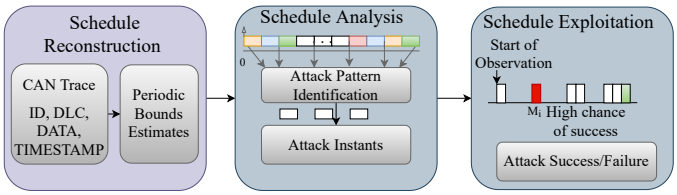


Fig. 5: Stages of Schedule-Based Attack in CAN Bus.

To succeed in the schedule-based attack, the adversary needs to learn when instances of interest are expected (since they cannot simply rely on the transmission of a particular preceded ID) and then send M_a at the appropriate time on the bus. Specifically, the attacker must:

1) examine the sequence in which messages are transmitted on the bus to estimate message parameters,
2) analyze the sequence to select and locate victim instances of interest in a hyperperiod,
3) identify the first instance of M_v (M_{v,n_v^*}) to be attacked based on the attacker's strategy,
4) observe the bus and enqueue M_a prior to M_{v,n_v^*} arbitration,
5) repeat step 4 for instances of M_v selected in step 2 until the TEC of the victim node reaches 255.

We refer to these steps as *message parameter estimation* (step 1), *schedule analysis* (steps 2 and 3), and *schedule exploitation* (steps 4 and 5). The first stage starts once the attacker has collected message logs to estimate the parameters, such as the message periods and instances. In the next stage, the adversary records the beginning and compositions of successive message transmissions ending with the transmission of an instance of M_v. Each record indicates a potential time at which the adversary can inject an attack message during the following hyperperiod. It also specifies the priority of the message(s) preceding a particular instance of the victim. The adversary selects records that satisfy the attack strategy (as described in section III-B) from which M_{v,n_v^*} is identified. In the exploitation stage, the adversary uses the noted time related to the arrival of M_{v,n_v^*} to anticipate its arrival. Once M_{v,n_v^*} is expected to arbitrate, the adversary enqueues M_a to transmit with M_{v,n_v^*} and the remaining victim message instances that are of interest and identified in stage 2.

Figure 5 shows these different stages of the CAN schedule-based bus-off attack. In the following subsections, we describe these stages in further detail.

A. Message Parameter Estimation

Olufowobi et al. [6,7] have shown that a greedy algorithm can be used to reverse engineer (estimate) parameters of a set of periodic messages, such as the message period and jitter, from a CAN log file. Their work focuses on using these parameters to tune a specification-based IDS.

Our algorithm takes as input a CAN log and message ID i and returns as output the period estimate P_i. This approach follows the period estimation algorithm introduced

by Olufowobi et al. [6] with a slight modification to assume that a period is estimated when the difference between the upper and lower bound is less than the transmission time of a single bit. Specifically, for each transmission of M_i, the period estimation algorithm infers period bounds at which each message could occur by reasoning backward in the steps the message will go through before transmission. By taking into consideration the factors contributing to the message response time as well as completion time of successive instances of M_i, the algorithm tightens the bounds on the P_i.

Once the period of each message is estimated, the attacker can calculate the hyperperiod, i.e., the least common multiple of the periods. It is worth noting that since the period estimation step requires observing at least two instances of a message, the time at which it completes the calculation of periods will be larger than the offset of any individual message. Consequently, in any observation window of length h, the attacker will see h/P_i message instances of a message M_i, regardless of the start time of the observation window.

B. Schedule Analysis

As described in Section III-B, to succeed in the bus-off attack, the adversary intends to launch the schedule-based attack using the victim message instances which ensure that the attack message will be transmitted at the same time as the victim message. Such an assurance is provided by the adversary's ability to enqueue the attack message right before the victim message, during the transmission of the message(s) preceding the victim. In this stage of the schedule-based attack, the adversary identifies all those transmissions, when they will occur in the future and how they can be used to implement the attacker's strategy.

Definition 1: An *attack pattern* is a pattern that describes a sequence of message transmission(s) ending with a transmission of an instance of the victim message.

In the schedule-based attack, attack patterns can be represented as a regular expression on a language whose alphabets are $\alpha = \{H, L, V, O\}$, where H and L are symbols that represent any message ID with a higher-priority and lower-priority than the victim, V represents the victim message, and O represents any other message type or state of the bus (e.g., idle state, or error messages). Knowing the victim message ID enables the attacker to directly translate the status of the bus to any of the symbols in the alphabet.

Example 1: Consider an attacker that wants to perform a bus-off attack on M_1 in the example shown in Figure 6. The attacker's goal is to send its attack message (which has the same priority as the victim message) at the time that the victim message appears on the CAN bus.

The following regular expressions can be used to represent three attack patterns: (i) $\langle \{H\}^+ V \rangle$, where $\{H\}^+$ represents any non-empty sequence of higher-priority messages that precede a message of M_1 (the victim message) without any idle time in between, (ii) $\langle L\ V \rangle$, where L is any lower-priority message than the victim message that immediately precedes the victim message, and (iii) $\langle O\ V \rangle$, i.e., the bus was idle

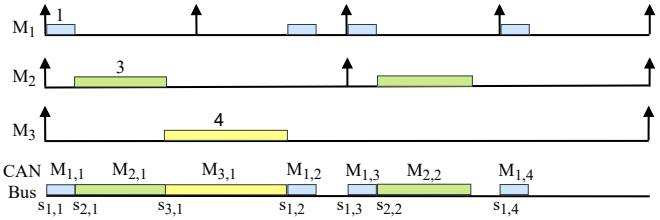


Fig. 6: CAN Schedule with three message IDs.

(or occupied by messages other than H and L types) prior to the transmission of the victim message. Every instance of a message will be subject to one and only one of these three attack patterns (because the alphabet defined for the regular expressions partition all four types of messages).

For pattern (i), the attacker can send its message anytime in the interval from the start of the sequence of higher-priority messages until the moment immediately before the victim starts arbitration. Hence, the attacker has a higher *attack surface* in this case. The attack surface for pattern (ii) is shorter because it is at most as long as the transmission time of a lower-priority message that precedes the victim. Finally, we assume that the attack surface in the last case is 0, i.e., the attacker can be successful only if it sends its message exactly at the time the victim sends its message. Hence, in terms of success when there are uncertainties in the system, the attacker would be more successful in case (i) than in case (ii) and (iii).

In the rest of this section, we focus on how attack patterns are used as the basis for the schedule-based attack. First we explain how attack patterns are identified in the schedule (Sec. IV-B1). Next, we describe how to find a robust set of *attack moments* (that increase the chance of a successful victim prediction) by proposing four heuristic methods (Sec. IV-B2) Finally, we discuss how to locate the best attack patterns to satisfy the adversary's strategy as well as M_{v,n_v^*} (Sec. IV-B3).

*1) **Identifying Attack Patterns**:* The attacker starts a timer right after it obtains the length of the hyperperiod and then observes messages being transmitted on the bus. It categorizes each observed message ID into H, L, or V and checks its observations against the attack pattern types. As soon as it spots an attack pattern, it calculates the potential start time and end time of the attack for the *next hyperperiod to come*. Note that this attack does not require the nodes to be time synchronized (messages can have any arbitrary offset). The attacker only needs to know the length of a hyperperiod. The example below explains how the attacker works.

Example 2. Consider the attack patterns used in the previous example and the system shown in Figure 6. Let us assume that the attacker starts observing the schedule at time $t_{init} = 100$ and the first message it sees is $M_{2,1}$, i.e., the attacker does not see the beginning of the hyperperiod. The attacker keeps track of the lower- and higher-priority messages that it observes on the bus using two counters: t_l and t_h. These counters store the start time of the transmission of a message with a higher and lower priority than M_1, respectively.

The attacker updates the counters as soon as it sees messages matching the counter description, i.e., a higher or lower priority message. For example, at time $s_{2,1}$, t_l is updated by storing the start time of M_2 as a relative time w.r.t. t_{init} (i.e., $t_l \leftarrow 0$). When the attacker observes another low-priority message (M_3), it updates $t_l \leftarrow C_2$ to point to the start time of M_3 w.r.t. t_{init}. As soon as the attacker observes M_1, it stores the current value of t_l (since a lower-priority message was transmitted prior to M_1) in its attack-pattern table as a potential attack opportunity which must start exactly at time $t_{init} + h + C_2$. The high-priority counter t_h is updated when a higher-priority message than M_1 is observed, but it will not be updated when another high-priority message is on the bus (to match pattern (i)). Both counters are invalidated as soon as the bus becomes idle. In this example, by the time $t_{init} + h$, the attacker has already identified one window to perform the attack, i.e., using pattern (ii) at time $t_{init} + h + C_2$.

We summarize the approach in Algorithm 1. Algorithm 1 is an online algorithm that processes the hyperperiod and for each instance of the victim message, stores the type of attack and the attack surface (starting moment and ending moment of a successful attack to that message in that hyperperiod). This algorithm produces a list $\Delta = \langle \delta_1, \delta_2, \ldots, \delta_{n_v} \rangle$ of the attack opportunities that happens for each of the $n_v = h/P_v$ message instances of message M_v in the observed hyperperiod. Each item $\delta_k \in \Delta$ is a tuple $\delta_k = ([s_k, e_k], \theta_k)$ that describes the attack type ($\theta_k \in \{T1, T2, T3\}$) and the attack surface (i.e., $[s_k, e_k]$) that appeared in the observed hyperperiod before the k^{th} message instance of message M_v. For example, in Figure 6, when M_2 is the victim message, we have $\Delta = \langle ([s_{1,1}, s_{2,1}], T1), ([s_{1,3}, s_{2,2}], T1) \rangle$.

This algorithm uses t_h and t_l timers to keep track of the latest observed high- and low-priority messages on the bus, respectively. Because the adversary has configured the filters of the attack node to receive all CAN messages transmitted, the contents of the messages can be retrieved at the attack node. Thus, every time a message completes transmission, the attacker reads its ID i and compute the start time and end time of its transmission on the bus, denoted by t_i and t_e, respectively (lines 4-6). Using t_i and the completion time of the previous message received before M_i, the attacker determines whether the bus was idle before M_i, and invalidates t_h and t_l if true (lines 7-8). Otherwise, t_h and t_l are updated based on the priority of M_i as follows (line 10-21):

- M_i has a lower priority than M_v: t_i represents the beginning of the transmission time of a lower-priority message (and potentially, the beginning of a type (ii) attack pattern). Thus, t_h is invalidated and $t_l \leftarrow t_i$ (line 11).
- M_i has a higher priority than M_v: in this case, if t_h is valid, i.e., the previous message(s) received had also a higher priority than the victim, no change to t_h is necessary. In other words, the sequence of high-priority messages that has started with the previous higher-priority message(s) can be continued (to possibly form a type (i) attack pattern), and its start time remains the same. If, on the other hand, t_h is not set yet, then $t_h \leftarrow t_i$ to indicate that a new potential attack pattern has started (line 13).

Algorithm 1 [online] Identification of Attack Patterns

Inputs: h, v
Outputs: Δ

```
1:  t_l ← −∞, t_h ← −∞
2:  Δ ← ⟨⟩
3:  while length of observation window ≤ h do
4:      i ← the latest observed message ID
5:      t_i ← the start time of transmission of message i
6:      t_e ← the end time of transmission of message i
7:      if bus is idle then
8:          t_h ← −∞, t_l ← −∞
9:      else
10:         if i is lower priority than v then
11:             t_l ← t_i, t_h ← −∞
12:         else if i is higher priority than v and (t_h < 0) then
13:             t_h ← t_i, t_l ← −∞
14:     if (i = v) then
15:         if (t_h ≥ 0) then
16:             append ([t_h, t_e], T1) to Δ
17:         else if (t_l ≥ 0) then
18:             append ([t_l, t_e], T2) to Δ
19:         else
20:             append ([t_e, t_e], T3) to Δ
21:         t_l ← −∞, t_h ← −∞
22: return Δ
```

When $i = v$ (i.e., the message received is an instance of the victim message), the algorithm appends one attack opportunity to the final output Δ depending on the type of attack (type (i), (ii), or (iii)). If the attack is not type (i) or (ii), then it is categorized as type (iii) with an attack surface with zero length, i.e., from $[t_e, t_e]$ (line 20). After storing the attack, both timers t_h and t_l are reset (line 21).

*2) **Dealing with Uncertainties**:* Although Algorithm 1 produces the list Δ that include potential attack instants, the attack's success can be affected if there are uncertainties in the start time and transmission time of messages on the bus for the future hyperperiods. To address this challenge, we propose to obtain statistically robust attack opportunities (and attack surface) by observing the schedule over N hyperperiods (instead of one), identify attack patterns for each hyperperiod, and extract attack opportunities that repeated more frequently (and hence are more robust to uncertainties). In this paper, we propose three statistical methods to obtain a robust set of attack opportunities per message instance of the victim message, i.e., per $M_{v,k}$. The output of this step, denoted by $\phi = (\phi_1, \phi_2, \ldots, \phi_{n_v})$, where ϕ_k is a robust start time for an attack on the k^{th} instance of the victim message. Since the time values stored in ϕ are relative w.r.t. one observation window with length h, the attacker can reuse them every h units of time to continue performing an attack over multiple hyperperiods in the future.

To explain our three methods, we use a superscribe i ($1 \leq i \leq N$) for Δ, i.e., Δ^i, to represent the attack opportunity vector obtained from Algorithm 1 during the i^{th} observed hyperperiod. For simplicity, we use the superscript i to refer to the relevant data in the attack opportunity vector Δ^i too.

Method 1: Average Starting Time. For each attack opportunity identified by Algorithm 1, we compute the average starting time of that attack opportunity over N hyperperiods (regardless of the attack types). Hence, the final attack moment for the k^{th} instance of the victim message will be

$$\phi_k = \frac{1}{N} \sum_{i=1}^{N} s_k^i. \tag{1}$$

Method 2: Intersection of Transmission Times. In this method, regardless of the attack type, we calculate the intersection between the attack surfaces obtained over N hyperperiods for the k^{th} instance of the victim message as follows

$$\phi_k = \max\{s_k^i\}_{\forall i}. \tag{2}$$

In this method, if the interval I_k (see below) is empty, namely, there is no intersection between the attack surfaces of the N observations, then the attacker will not consider attacking that instance of the victim message and will treat that instance as an attack type (iii) (with zero attack surface). I_k is obtained as follows $I_k = [\max\{s_k^i\}_{\forall i}\ ,\ \min\{e_k^i\}_{\forall i}]$.

Method 3: Average of Mid-Transmission Times. In this method, for each attack pattern of victim instance k, we compute the midpoint of the transmission time for the preceding message(s) in the attack pattern. Then the average of is taken over the N hyperperiods observed (regardless of variations in the attack types) as follows:

$$\phi_k = \frac{1}{2N} \sum_{i=1}^{N} (e_k^i - s_k^i). \tag{3}$$

In Section V-C3 we demonstrate that method 3 allows the attacker to predict the arrival of attack opportunities over a longer period of time. Thus, using the list of robust attack opportunities, the attacker can select which victim instance n_v^* (and consequently, which attack opportunity) to start the attack from.

*3) **Identifying** M_{v,n_v^*}:* One of the strengths of the schedule-based attack is that it allows the adversary to develop the attack based on the robust attack opportunities generated. Moreover, when many attack opportunities exist in the schedule, the adversary can choose the approach that has the highest chance to result in a successful attack. For example, the adversary can choose to only use attack patterns that have never been classified as type (iii) during the N hyperperiods analyzed.

In this case, as described in Sec. III-B, the adversary wants to launch the attack on the M_v instances that can be synchronized with (in a robust way) to minimize bus disturbances while increasing the victim ECU's TEC to 255 very quickly. The shortest possible attack requires that the adversary uses consecutive instances so as to prevent the victim node from decrementing its TEC due to successful transmissions. Thus, to decrease the attack time, the adversary should identify sufficiently long sequences of attack opportunities in Δ with no type (iii) attack patterns and launch the attack from the first element n_v^* of that sequence.

Algorithm 2 Identifying n_v^*

Inputs: $\bar{\Delta} = (\Delta^1, \ldots, \Delta^N)$, h, v
Outputs: ${n_v}^*$

```
1:  n_v ← h/P_v
2:  n_v* ← 1, i ← 1
3:  maxcount ← 0
4:  while i ≤ n_v do
5:      j ← i, count ← 1
6:      while count ≤ n_v do
7:          if j > n_v then
8:              j ← 1
9:          if ∄θ_j^y (for 1 ≤ y ≤ N) ∧ θ_j^y = T3 then
10:             count ← count + 1, j ← j + 1
11:         else
12:             if count > maxcount then
13:                 maxcount ← count, n_v* ← j
14:                 break
15:     i ← i + 1
16: return n_v*
```

We describe the process of finding n_v^* in Algorithm 2. We start by selecting an instance of the victim message in Δ and counting the number of type (i) or type (ii) attack patterns starting from that element in Δ until a type (iii) pattern is identified (lines 6 - 11). If the total number of patterns counted is the maximum obtained thus far, it is recorded along with the position of the attack opportunity from which the count has started as n_v* (line 13).

Once the attack opportunity n_v^* is known, the attacker needs to predict *when* that particular attack pattern will contend for the bus in order to schedule the first attack message at the right time. Each of the three methods can provide one such start time for the attack opportunity on the message instance n_v^* (i.e., by using the value $\phi_{n_v^*}$). Further details of this step will be discussed in the next section.

C. Schedule Exploitation

The attack starts by observing the CAN bus traffic until the time indicated by $\phi_{n_v^*}$ when M_a is queued to arbitrate with M_v. Then the attacker follows its pre-calculated vector of attack opportunities (i.e., ϕ) to continue injecting M_a until the victim's TEC reaches 255. Note that the times stored in ϕ are relative to the start time of the first observation window. The attacker needs to adjust these values by h units of time whenever it reaches a new full cycle of n_v messages to continue its attack over multiple hyperperiods.

The success in the exploitation stage depends on how robustly the attack framework can withstand inaccuracies in message parameter estimations, jitter, message ordering.

and sporadic messages transmission. We briefly discuss their impact on the schedule-based attack.

Period Estimation Inaccuracies. Period estimation inaccuracies may lead to an error in the message sequence and hence even change the attack patterns. For example, when the period of a message M_i is underestimated by 10%, it leads to a false expectation of an additional transmission of M_i after every nine genuine transmissions of M_i. This error also inaccurately inflates the number of messages in a hyperperiod, and possibly the value of $n_v = h/P_v$. Finally, such an inaccuracy in the period estimation may result in an erroneous value for the system's hyperperiod.

DLC and Transmission Time. Increase in a preceded ID's DLC (although rare) from an expected value may decrease the amount of time available to the attacker after the transmission of M_v to retransmit the sixteen messages in order to reach a high value of TEC and bus-off state.

Message Jitter and Message Ordering. Jitter can cause the order of messages in the CAN schedule to vary by introducing unexpected interference or blocking. Because of jitters, a message in a previously observed attack pattern may get transmitted later than its expected time. This can affect the order of the victim message and therefore lead to a failed prediction despite the statistical robustness of our methods to obtain the starting point of the attack.

Sporadic Messages: Messages not belonging to the periodic message set are considered sporadic messages when they transmit. Since they start when the bus is idle, even if they appear right before an instance of the victim message, our solution would not be impacted because we do not consider attacking instances of the victim message that are sent when the bus is idle. However, if the sporadic messages push a set of periodic messages, they may eventually change the sequence of messages sent on the bus. Our current solution relies only on fidelity of the observations we made during N hyperperiods. If the sporadic messages do not frequently show up in that observation window, then the accuracy of our attack will reduce eventually.

V. Evaluation

We have performed experiments to answer the following questions: (i) How accurate is our approach to estimating the period of CAN messages? (ii) How accurate is our solution in predicting the arrival of a victim message? (iii) How vulnerable is a CAN system to a schedule-based bus-off attack. We address the first question in Section V-C1. Question (ii) is evaluated in Section V-C2 using a set of random schedules and random victim messages to estimate the number of cases that result in an accurate schedule-based prediction. In Section V-C3 we compare our approach with the traditional bus-off attack approach. We evaluate the feasibility of each stage of the schedule-based attack with these experiments.

A. Experimental Setup

We used a software-only simulation of CAN and a real CAN dataset. Software for our setup is freely available[1]. The code is written in Python 3 on a computer with Ubuntu 18.04.5 LTS operating system, Intel Core i7-9700K CPU @ 3.60 GHz and 32 GB RAM

1) Simulator for Synthetic CAN Traces Generation: The software simulation uses CAN traces—real traces were collected from vehicles, and synthetic traces created using a custom Python trace generator.

We used a discrete-event simulation to generate the CAN logs. By examining the expected behavior of the CAN message protocol, we analyze the different states a message will go through before it can be transmitted on the bus. The three possible states for a message are pending, active, and the transmission state. The pending state represents the initial state of the message before it is released into the active queue. In the active state, the messages go through the arbitration process before they can transmit. If a message wins the arbitration, the message is moved into the transmission queue for processing. In conformity with the CAN protocol, only one message can occupy the transmission queue at a time, and message preemption is not allowed in the transmission queue. The messages' transmission times are computed from the data length code according to analysis in [5]. The completion time of the message transmissions scheduled through the transmission queue is recorded and denoted as the message trace timestamps.

B. Datasets

1) Synthetic Dataset Description: We assume that the relative deadline of each message instance is equal to its period, and priorities have been assigned based on the message ID as explained in II-A. Messages coming through to the active queue are guaranteed to be schedulable by the fixed-priority non-preemptive scheduling algorithm of the CAN bus. We simulate 50,000 ms to generate simulated data.

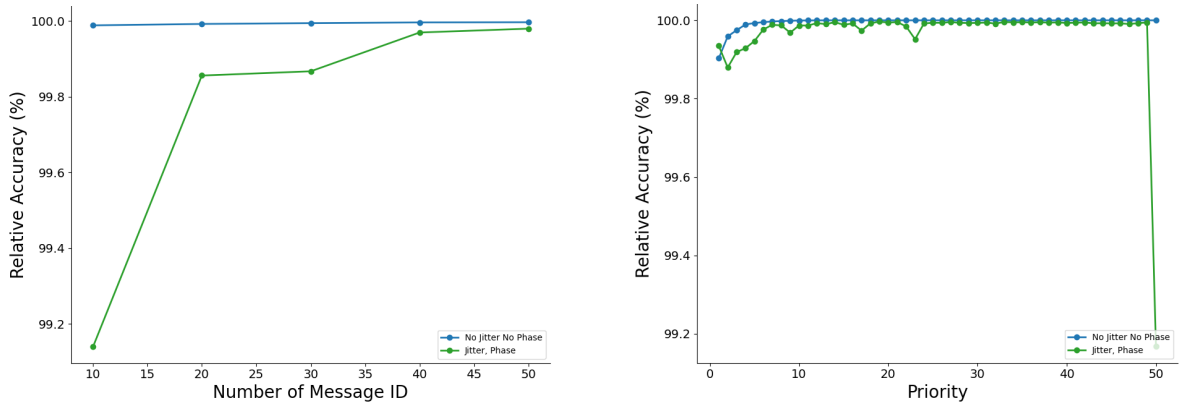
2) Real CAN Dataset Description: To evaluate the performance of our proposed schedule based attack, we used real vehicle CAN traffic captured for 30 to 40 minutes through the OBD-II port [9,20]. The datasets recorded are of normal vehicle operation and we used the Attack free data sets which has $988,987$ number of messages.

C. Experiments

1) Period Estimation with Uncertainty: The first experiment examines the accuracy of period estimation. We use the software simulation to explore the parameter space, we created 408 schedules by varying the number of messages and their parameters (period, data size, jitter, phase).

We created 17 message sets by varying the number of unique periodic messages from 10 to 40 and assigning harmonic periods to each message uniformly at random from the 16 divisors of 1000 ms (i.e., 1, 2, 4, 5, 8, 10, 20, 25, 40,

[1]https://github.com/Embedded-Systems-Security-Lab/sba-in-can/tree/rtss21



(a) Accuracy versus Number of Observable Message IDs

(b) Relative Accuracy versus Priority

Fig. 7: Accuracy of period estimation with and without jitter or phase.

50, 100, 125, 200, 250, 500, and 1000 ms). For each message set, we generate one schedule with no jitter and no offsets (i.e., simultaneous release at time 0), and 204 schedules by generating initial offsets (phase) at random from the uniform distribution over [[0,100000]] μs for each message in the set and jitter at random from the uniform distribution over [[0, 50]] μs for each message transmission. Thus, we created 204 schedules without jitter or phase, and 204 schedules with random phases and jitter. The former are used to demonstrate idealized behavior in the lack of uncertainty. We then attempt to estimate the message parameters of each schedule varying the number of messages (IDs) that can be observed from all messages to just 1 message to evaluate the affect of acceptance filters. We measured the accuracy of each reconstruction compared to the ground truth of its message set. The results in Figure 7 shows a high accuracy over 99%.

2) Schedule Prediction with Uncertainties: In this experiment we compare across the prediction heuristics described in Section IV-B2. We used the synthetic data to evaluate how practical the proposed prediction mechanism is by examining how many periods occur between the average time from the starting observation point to attack. We set to find out for how many periods we can accurately predict the arrival of a particular instance of the victim message. We also studied whether the accuracy of the prediction could be improved by increasing the number of hyperperiods (N) used in computing the predicted arrival in the subsequent hyperperiods.

Figure 8 shows the results obtained using the Average Starting Time (Method 1), the Intersection of Transmission Times (Method 2), and the Average of Mid-transmission Times (Method 3). For Method 1, we observe that as N increases, the number of hyperperiods for which the victim's arrival was accurately predicted gradually decreased from 21 hyperperiods for $N = 1$ to 16 hyperperiods for $N = 10$. Thus, the average starting times for the attack pattern becomes less accurate as N increases. In contrast, Methods 2 and 3 are less impacted by N. However, using Method 2 allows accurately predicting 21 hyperperiods while with Method 3 the adversary

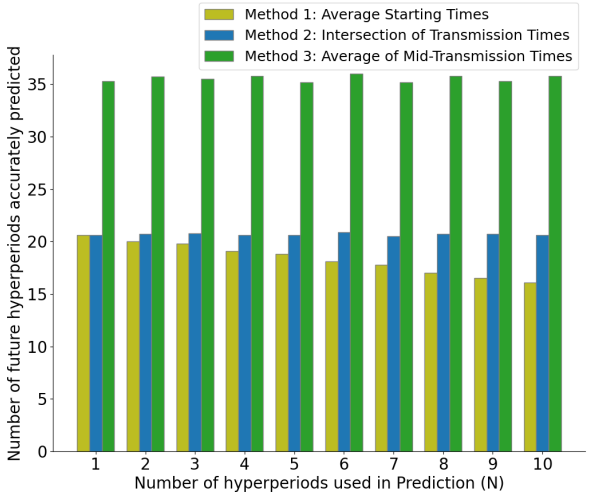


Fig. 8: Schedule Prediction

can accurately predict the arrival of the attack pattern for about 35 hyperperiods. Thus, of all three heuristics, Method 3 has the highest performance and provides the adversary with the best chance to succeed in the attack.

3) Case Study: SAE Benchmark: We developed a case study to further demonstrate the schedule-based attack approach using the modified SAE benchmark described by Tindell et al. [10]. This benchtop case study was performed on hardware representative of realistic ECUs on a live CAN bus. The benchmark includes both periodic and aperiodic tasks. Table II shows the parameters for the benchmark. The original benchmark is not specified for CAN, so we assign message priorities rate monotonically, with the exception of the sporadic messages, with ties broken by arbitrary ECU prioritization.

We implemented the benchmark using off-the-shelf hardware and in-house software[2]. The platform includes five

[2]https://github.com/gedare/tm4c129-canbus/tree/rtss21

TABLE II Modified SAE Benchmark. Type denotes periodic (P) or sporadic (S), with sporadic messages having a minimum interarrival time (IAT) in lieu of a period.

Sender	ID (hex)	Size (B)	Type	Period/IAT (ms)
VC	A0	1	P	5
	B0	6	P	10
	D0	1	P	1000
Brakes	A1	2	P	5
	B9	1	S	100
	C1	1	P	100
Battery	99	1	S	1000
	B2	1	P	10
	C2	4	P	100
	D2	3	P	1000
Driver	A3	1	P	5
	B3	2	P	10
IMC	A4	2	P	5
	B4	2	P	10
Trans	A5	1	P	5
	C5	1	P	100
	D5	1	P	1000

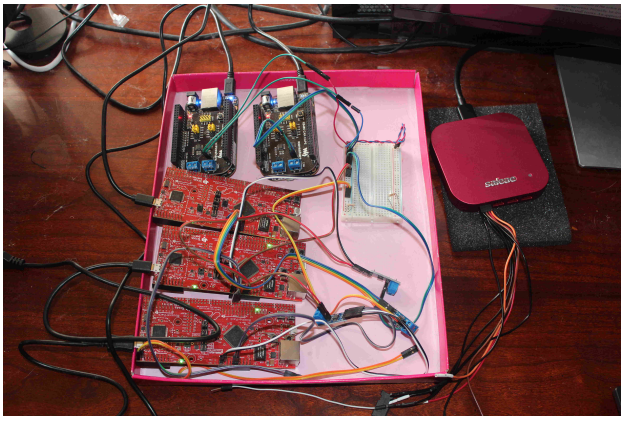


Fig. 9: Experimental Setup for SAE Benchmark Case Study

microcontrollers with hardware CAN controllers connected to 3.3v CAN transceivers that interface via a breadboard (see Figure 9). We ran the bus at 125 kbps, for which this benchmark has approximately 80% bus utilization. Note that when the subsystems synchronize with each other, every message except A0 has at least one genuine preceded ID. In general, they do not synchronize with each other except by happenstance. We simulate jitter in this setup by delaying transmission of messages by a small number of processor clock ticks (0-255) that introduces between 8 ns and 2 μs of jitter, which varies cyclically and relatively unpredictably throughout execution.

The Battery ECU is instrumented to be the attacker, and the Brakes ECU is targeted as the victim. The other ECUs just generate bus load and cause interference/blocking. We conducted the original bus-off attack with periodic synchronization as in prior work [8], and the schedule-based attack

TABLE III SAE Benchmark Case Study Results. The schedule-based attack (SBA) we introduce is evaluated with N hyperperiods for Schedule Analysis. We report the average number of attack message transmissions (M_a Avg Tx) and the success rate as the percent of the 50 attack trials that put the victim into the bus-off state. Each trial lasts for 10 hyperperiods.

Attack Method	M_a Avg Tx	Success Rate
Periodic	1,998.7	76%
SBA $N = 1$	278.4	80%
SBA Avg, $N = 2$	371.4	84%
SBA Avg, $N = 4$	80.9	36%

using Algorithm 1 for 1, 2, and 4 hyperperiods. For the trials using multiple hyperperiods we have used method 3 (average of mid-transmission times) to aggregate measurements across the hyperperiods. If any of the victim message instances are suspected to be preceded by an idle bus in any of the hyperperiods, then we do not use that attack opportunity.

Table III shows the results obtained from conducting the bus-off attacks with 50 attempted trials each lasting for 10 hyperperiods (10 seconds). Although the periodic approach achieves a reasonable success rate of 76%, it also generates a lot of extra bus traffic, because it attempts to inject the attack message at the same frequency as the authentic message. As mentioned in Sec. II-D, the addition of this much traffic would be trivially detected by even the simplest CAN intrusion detection systems. The schedule-based attack (SBA) using just one hyperperiod to make predictions achieves a better attack success rate than the periodic approach with many fewer attack injections, while the use of averaging the midpoint between the start and end of the preceded message is even more effective. Unfortunately, the performance drops off as more hyperperiods are included. We attribute this to the increased likelihood of encountering bus idle time prior to some victim instances, which the attack conservatively avoids.

VI. Discussion

The schedule-based attack does not rely on fabricated messages, achieves a better synchronization (by using attack patterns), lower detectability (by avoiding transmitting M_a in idle gaps), and reduced overall attack injection messages.

To perform the attack, the adversary must go through N hyperperiods and add a time instant for each item in the attack opportunity list. Only then the attacker can perform an attack according to any of the attack patterns that it selects (based on attack pattern type, for example). Hence, the preparation phase of this attack takes at least $N \cdot h$ units of time. Although N is a configuration parameter and can be a reasonable constant number to reduce the attack overhead, larger N values can increase the statistical robustness of the solution against system uncertainties.

Our solution stores one attack opportunity tuple per instance of the victim message during N hyperperiods. Hence, its

memory complexity is in the order of $O(n_v \cdot N)$, where $n_v = h/P_v$ is the number of message instances of the victim message in a hyperperiod. While N is a configuration parameter and can be a reasonable constant value, n_v will depend on the period of the victim message and the length of the hyperperiod. In automotive systems, for example, this can be between one and a thousand message instances. The response time of our attack is in $O(N \cdot h)$. As said earlier, here N is a reasonable constant number and h is the length of the hyperperiod and, for example, in automotive systems [21] is in the order of a few seconds. This means that a few seconds (or a minute) after being activated, the attacker can effectively start the bus-off attack.

Even when some nodes disconnect and reconnect to the network, the attacker can restart the attack and adjust with the new changes and after a few minutes, brings the victim message back to the bus-off mode.

On the negative side, despite using statistical robustness factor N to deal with system uncertainties, the proposed attack can still be affected by the uncertainties that have not been captured during our observation window. This could be the case when sporadic or aperiodic messages appear on the bus after the attack starts. Bit stuffing is another example of such uncertainty: Although the CAN timing model used in this work accounts for bit stuffing, the message data payloads we used were constant and therefore changes in transmission times were not included in the experimental evaluation.

VII. Related Work

Schedule-based attacks. The success of cyberattacks that depend on a particular ordering between the execution window of the attacker and its targeted task are called *schedule-based attacks*. Nasri et al. [22] introduced a taxonomy of such attacks and categorized them into four groups (anterior, posterior, pincer, and concurrent) based on the timing relation that makes the attack successful. They point out several challenges in analyzing schedule-based attacks.

Inferring task set parameters. Often, the success of a schedule-based attack depends on the ability of the attacker to *predict the future schedule* and possibly to *influence it* (so that it can execute the attack at the right moment). Chen et al. [23] proposed a parameter inference method for processor schedulers in a real-time operating system based on hijacking the idle task. The problem we consider, however, is different because the attacker can see all message transmissions on the bus and therefore has access to the actual schedule albeit delayed by the message transmission times.

Olufowobi et al. [6,7] infer the real-time parameters of messages on the CAN bus using the worst-case response time analysis framework for CAN [5,10]. They extract the timing model of the message schedule to develop an intrusion detection system. We adapt and improve their approach to improve estimating the message period in the presence of jitter and offsets with the goal of launching an attack.

Other techniques such as *fast Fourier transform* and *circular auto-correlation* [24] have also been used to infer a task's period from execution traces [25,26]. Together with *Periodogram* [27], these techniques have been used to infer periodicity of a signal [24,27]–[34]. However, none of these techniques alone provide an accurate period estimate because they are affected drastically by the presence of preemption, sporadic or aperiodic tasks, missed jobs, and any uncertainty in the release and execution time of tasks [35].

To overcome these limitations, Vadineanu and Nasri [35] proposed a solution based on regression-based machine learning that adopts periodogram and circular auto-correlation to generate a feature set which was used to train 6 families of regression-based machine-learning (RBML) methods including rule-based decision-trees. Next, the result of RBML methods was used as a referee to chose between a set of candidate periods. The set was obtained by collecting the best 40 candidates from the periodogram and circular auto-correlation that pass a pruning rule which applies the rules of a work-conserving scheduling policy on the trace to derive an upper bound and a lower bound of the valid periods. Finally, using the ternary projection of the trace, the time intervals during which the system resource was idle is used to eliminate impossible candidate values. This framework performed well in period estimation in the presence of uncertainties (such as release jitters) and variations in the execution time.

Chen et al. [36] have shown a successful case of a schedule-based attack in which the attacker tries to infer the initial offset of a strictly periodic task (the victim) in order to predict its future releases. The work assumes that the attacker is in the same computer as the victim task, which does not hold in CAN and hence their solution cannot be applied directly to this problem.

Defenses against schedule-based attacks. To prevent information leakage, Völp et al. [37] modified the system's schedule by switching the execution of potentially leaky threads with the idle task. Mohan et al. [38] include security-oriented directives in the schedule to prevent information leakage. Schedule randomization techniques have been introduced [39,40] to reduce the success of schedule-based attacks. However, Nasri et al. [22] showed that regardless of the granularity of randomization, these techniques are ineffective against schedule-based attacks and may even increase the system's vulnerability.

VIII. Conclusion

In this paper we have introduced a novel optimization to improve the efficiency of the traditional CAN bus-off attack by leveraging the real-time nature of CAN to design a schedule-based attack. We have shown that the schedule-based attack is more widely applicable than the traditional approach that relies on a unique preceded message.

References

[1] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, vol. 4. San Francisco, 2011, pp. 447–462.

[2] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks–practical examples and selected short-term countermeasures," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2008, pp. 235–248.

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[4] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "An empirical survey-based study into industry practice in real-time systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[5] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[6] H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, "Saiducant: Specification-based automotive intrusion detection using controller area network (can) timing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 1484–1494, 2019.

[7] H. Olufowobi, G. Bloom, C. Young, and J. Zambreno, "Work-in-progress: Real-time modeling for intrusion detection in automotive controller area network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 161–164.

[8] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1044–1055.

[9] Hacking and C. R. Lab. Car-hacking dataset for the intrusion detection. [Online]. Available: https://sites.google.com/a/hksecurity.net/ocslab/Datasets/CAN-intrusion-dataset

[10] K. Tindell, A. Burns, and A. J. Wellings, "Calculating controller area network (can) message response times," *Control Engineering Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[11] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking tesla from wireless to can bus," *Briefing, Black Hat USA*, vol. 25, pp. 1–16, 2017.

[12] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, 2015.

[13] ——, "Adventures in automotive networks and control units," *Def Con*, vol. 21, pp. 260–264, 2013.

[14] M. Foruhandeh, Y. Man, R. Gerdes, M. Li, and T. Chantem, "Simple: Single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 229–244.

[15] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran, "Cloaking the clock: emulating clock skew in controller area networks," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 32–42.

[16] G. Bloom, "WeepingCAN: A Stealthy CAN Bus-off Attack," in *Workshop on Automotive and Autonomous Vehicle Security*. Internet Society, Feb. 2021.

[17] U. Ezeobi, H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, "Reverse Engineering Controller Area Network Messages using Unsupervised Machine Learning," *IEEE Consumer Electronics Magazine*, pp. 1–1, 2020, conference Name: IEEE Consumer Electronics Magazine.

[18] H. Olufowobi, S. Hounsinou, and G. Bloom, "Controller area network intrusion prevention system leveraging fault recovery," in *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy*, 2019, pp. 63–73.

[19] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom, "Survey of Automotive Controller Area Network Intrusion Detection Systems," *IEEE Design Test*, vol. 36, no. 6, pp. 48–55, Dec. 2019.

[20] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, Jan. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214209619302451

[23] C.-Y. Chen, A. Ghassami, S. Nagy, M.-K. Yoon, S. Mohan, N. Kiyavash, R. B. Bobba, and R. Pellizzoni, "Schedule-based side-channel attack in fixed-priority real-time systems," Tech. Rep., 2015.

[21] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[22] M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes, "On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 103–116.

[24] J. A. Gubner, *Probability and random processes for electrical and computer engineers*. Cambridge University Press, 2006.

[25] C. Young, H. Olufowobi, G. Bloom, and J. Zambreno, "Automotive intrusion detection based on constant can message frequencies across vehicle driving modes," in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, 2019, pp. 9–14.

[26] S. Liu and W. Yi, "Task parameters analysis in schedule-based timing side-channel attack," *IEEE Access*, vol. 8, pp. 157 103–157 115, 2020.

[27] A. Schuster, "On the investigation of hidden periodicities with application to a supposed 26 day period of meteorological phenomena," *Terrestrial Magnetism*, vol. 3, no. 1, pp. 13–41, 1898.

[28] C. Berberidis, W. G. Aref, M. Atallah, I. Vlahavas, and A. K. Elmagarmid, "Multiple and Partial Periodicity Mining in Time Series Databases," in *European Conference on Artificial Intelligence (ECAI)*, 2002, pp. 370–374.

[29] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *SIAM international conference on data mining*, 2005, pp. 449–460.

[30] T.-H. Li, "Detection and Estimation of Hidden Periodicity in Asymmetric Noise by Using Quantile Periodogram," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 3969–3972.

[31] R. G. McKilliam, I. V. L. Clarkson, and B. G. Quinn, "Fast Sparse Period Estimation," *IEEE Signal Processing Letters*, vol. 22, no. 1, pp. 62–66, 2014.

[32] Y. Malode, D. Khadse, and D. Jamthe, "Efficient Periodicity Mining Using Circular Autocorrelation in Time Series Data," *International Research Journal of Engineering and Technology (IRJET)*, vol. 2, no. 3, pp. 430–436, 2015.

[33] P. Unnikrishnan and V. Jothiprakash, "Daily Rainfall Forecasting for One Year in a Single Run Using Singular Spectrum Analysis," *Journal of Hydrology*, vol. 561, no. 1, pp. 609–621, 2018.

[34] T. Puech, M. Boussard, A. D'Amato, and G. Millerand, "A Fully Automated Periodicity Detection in Time Series," in *International Workshop on Advanced Analysis and Learning on Temporal Data (AALTD)*, 2019, pp. 43–54.

[35] S. Vadineanu and M. Nasri, "Robust and accurate period inference using regression-based techniques," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[36] C.-Y. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A novel side-channel in real-time schedulers," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 90–102.

[37] M. Völp, C.-J. Hamann, and H. Härtig, "Avoiding timing channels in fixed-priority schedulers," in *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, 2008, pp. 44–55.

[38] S. Mohan, M.-K. Yoon, R. Pellizzoni, and R. B. Bobba, "Integrating security constraints into fixed priority real-time schedulers," *Real-Time Systems*, vol. 52, no. 5, pp. 644–674, 2016.

[39] M. Yoon, S. Mohan, C. Chen, and L. Sha, "Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1–12.

[40] K. Krüger, M. Volp, and G. Fohler, "Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems," *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 106, p. 22, 2018.