SAIDuCANT: Specification-Based Automotive Intrusion Detection Using Controller Area Network (CAN) Timing

Habeeb Olufowobi, Clinton Young, Joseph Zambreno, *Senior Member, IEEE*, and Gedare Bloom, *Senior Member, IEEE*

***Abstract*—The proliferation of embedded devices in modern vehicles has opened the traditionally-closed vehicular system to the risk of cybersecurity attacks through physical and remote access to the in-vehicle network such as the controller area network (CAN). The CAN bus does not implement a security protocol that can protect the vehicle against the increasing cyber and physical attacks. To address this risk, we introduce a novel algorithm to extract the real-time model parameters of the CAN bus and develop SAIDuCANT, a specification-based intrusion detection system (IDS) using anomaly-based supervised learning with the real-time model as input. We evaluate the effectiveness of SAIDuCANT with real CAN logs collected from two passenger cars and on an open-source CAN dataset collected from real-world scenarios. Experimental results show that SAIDuCANT can effectively detect data injection attacks with low false positive rates. Over four real attack scenarios from the open-source dataset, SAIDuCANT observes at most one false positive before detecting an attack whereas other detection approaches using CAN timing features detect on average more than a hundred false positives before a real attack occurs.**

***Index Terms*—CAN bus, intrusion detection system, timing model, real-time systems.**

I. Introduction

THE connected car industry is quickly growing, and by some estimates will account for almost $40 billion in annual revenue by 2020 [1]. This growth is led by cyber-physical system (CPS) advancements in enhancing safety and automation, and by expanding use of Internet connectivity for in-vehicle infotainment, which brings connected cars into the Internet of Things (IoT). These applications have increased the cyber connectivity and complexity of vehicles, as demonstrated by the rising number of electronic control units (ECUs), wireless communication interfaces, and software lines of code in the modern vehicle. The dramatic increase in vehicle functionality however also makes the vehicular systems, including the safety-critical systems, more vulnerable to cybersecurity risks and attacks [2]. Vulnerabilities across the autonomous vehicles, vehicular adhoc networks, vehicle-to-vehicle, vehicle-to-infrastructure, connected car, intelligent transportation system, and even traditional (non-connected) automobiles motivate adversaries to launch cyberattacks against vehicles [3]. Unfortunately, today's car lacks the necessary security mechanisms to protect the vehicular system from attack.

A critical asset to secure is the automotive in-vehicle network, which facilitates communication between ECUs over multiple physical networks and protocols, with the most prevalent being the controller area network (CAN). An adversary may subvert the in-vehicle network through attack surfaces that increase proportionally to new vehicle features. Physical access to the on-board diagnostics (OBD-II) port can be used to easily compromise the network, while remote access through a wireless or cellular connection can greatly increase an attack's scalability and reduce exposure of the attacker. Bluetooth attacks have been demonstrated by Checkoway *et al.* [4], while Miller and Valasek [5] accessed a Jeep Cherokee through its WiFi network by exploiting a weakness in its password generation protocol. Once access to the in-vehicle network is achieved, the attacker can manipulate and delete data, degrade vehicle functions, and even take over control of the vehicle. The limited computational, memory, and power resources of ECUs hinder the implementation of complex security mechanisms. Hence, lightweight and computationally efficient algorithms are an important requirement in implementing security mechanisms for the in-vehicle network.

We introduce Specification-based Automotive Intrusion Detection using Controller Area Network Timing (SAIDuCANT), a specification-based intrusion detection system (IDS) that uses the real-time model of the CAN bus to specify intended behavior, and then detects violations of the model as signs of a compromised network. Given an instance of a message, we aim to determine if its completion time aligns with the timing model specification of the message. Our approach to this problem is to infer the parameters of the real-time model of the CAN bus during normal operation. Using the schedulability analysis of the network, which guarantees that message deadlines will be met in the worst-case, we derive the timing model specification for a

Manuscript received March 2, 2019; revised July 1, 2019, September 20, 2019, and November 27, 2019; accepted November 27, 2019. Date of publication December 20, 2019; date of current version February 12, 2020. The work was supported by the National Science Foundation under Grants CNS 1646317 and CNS 1645987. The review of this article was coordinated by Dr. M. Kisacikoglu. *(Corresponding author: Gedare Bloom.)*

H. Olufowobi is with the Department of Electrical Engineering and Computer Science, Howard University, Washington, DC 20059 USA (e-mail: habeeb.olufowobi@scs.howard.edu).

C. Young and J. Zambreno are with the Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA (e-mail: cwyoung@iastate.edu; zambreno@iastate.edu).

G. Bloom is with the Department of Computer Science, University of Colorado, Colorado Springs, CO 80918 USA (e-mail: gbloom@uccs.edu).

Digital Object Identifier 10.1109/TVT.2019.2961344

set of messages and hypothesize that messages that do not fit into this timing model are anomalous. The timing model expresses the behavior of the CAN bus from which anomalous deviations indicate an attack is in progress. Although our focus is on the CAN bus as the in-vehicle network, we expect our results would apply well to any network that provides real-time behavior.

The contributions of this paper are:

1) A method for extracting real-time model parameters from observations of CAN bus message behavior without prior knowledge.
2) A specification-based IDS based on real-time schedulability response time analysis of the CAN bus.
3) Two new metrics for measuring the performance of automotive intrusion detection systems. These metrics provide essential and useful information that can be used in making a decision about the IDS more so than the traditional classifier metrics.
4) Prototype and evaluation of real-time model specification-based IDS using real CAN logs generated from passenger sedan vehicles. The evaluation shows that SAIDuCANT outperforms existing timing-based IDS for CAN and especially exhibits a low false positive rate in normal data prior to the start of an attack.

This paper extends our previous work [6] with modifications to the detection algorithm to improve its classification performance, expansion of the evaluation using additional metrics and data from real attacks, and comparison of our approach with other work that uses the timing features of CAN bus messages for intrusion detection.

The remainder of this paper is organized as follows. In Section II, we discuss the related work on specification-based intrusion detection systems for in-vehicle networks. Section III provides a primer on CAN response time analysis. Section IV describes the design of SAIDuCANT including the threat and attack model, timing model extraction, and anomaly detection. In Section V, we describe the experimental setup for evaluation, and Section VI presents the experiments and results. Section VII discusses SAIDuCANT's limitations and possible directions for future work. Section VIII concludes the paper.

II. Related Work

Security problems of in-vehicle networks have been studied over the years by several researchers [7]. Koscher et al. [8] were the first to demonstrate and perform practical attacks on vehicles. The authors demonstrated complete control of a wide range of automotive functions by sniffing the CAN bus and reverse engineering ECU code. Hoppe *et al.* [9] demonstrated practical attacks on the CAN bus, and demonstrated an anomaly detection method by looking at the frequency of messages transmitted on the bus. Existing works have applied cryptographic techniques to in-vehicle networks, such as digital signatures, encryption, and message authentication codes [10]–[12], but the communication overhead of these techniques is very high, making them unsuitable or at least difficult in practice for the CAN bus.

A plethora of automotive in-vehicle network IDSs have been developed over the years that explore methods of detecting anomalies as indicative of intrusions [13]–[28]. However, none of these works use a specification-based approach, instead relying on message properties such as frequency [9], [16], [24], [28], inter-arrival time [19], [22] and entropy [14], [20], or physical properties of ECUs such as their clock drift [18] or voltage [25]. SAIDuCANT captures the behavior and models the timing of the CAN messages to extract the specification of the network activities to detect intrusions. More precisely, we use the worst-case response time analysis of each message to build a set of specifications for message transmissions to compare with observed network activities to detect intrusions. In the following we contrast SAIDuCANT with prior work in specification-based IDSs.

A specification-based detection method relies on a specification that describes the behavior of the system components. This legitimate behavior of the system is described by its functionalities and the constraints of other interacting components. The monitoring of the system activities involves detecting deviations from the sequence of operations outside of the specification, which are considered intrusions. Expected behavior of the system components may be manually extracted and crafted as security specifications [29]. Manually-defined specifications can provide low false positive rates when compared with other anomaly-based detection methods [30]. An advantage of specification-based detection is that the IDS is effective immediately when the specification is defined, as there is no user or data profiling involved. However, the amount of work required in capturing and verifying the correctness of a specification is a major drawback.

Specification-based detection has been applied to several systems including network protocols, applications, and CPSs [31]–[35]. Mitchell and Chen [31], [32] proposed a behavior-rule specification-based IDS for medical CPS and unmanned aircraft systems. In their approach, they use a binary failure threshold to classify a node as normal or malicious based on the node's compliance threshold. Esquivel-Vargas *et al.* [35] proposed an approach to automatically deploy a specification-based IDS to monitor a building automation system using rules that represent valid device behavior in BACnet networks to detect violations in the network traffic. Fauri *et al.* [34] proposed an approach to combine formal specification with anomaly-based monitoring to overcome the semantic gap between network anomalies and actionable alerts by leveraging the lightweight logical system specification.

The concept of specification-based detection for CAN bus was first investigated by Larson *et al.* [36]. They described the application of a specification-based IDS for the CANopen protocol using the application protocol layer. They show that potential attacks can be detected from the trace of extracted information through theoretical simulation, and concluded that the most important ECU to protect is the gateway ECU.

Studnia *et al.* [37] proposed a language-based detection approach using language theory to develop a set of attack signatures from the behavioral model of CAN. The authors generate sets of forbidden sequences from the behavioral model that corresponds to the manifestation of possible attacks on the network that they seek to detect. Lee *et al.* [38] proposed an IDS called OTIDS

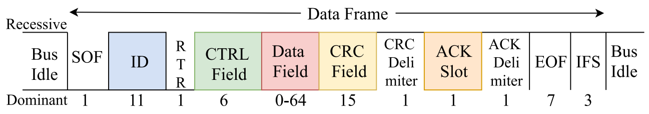


Fig. 1. CAN Message Structure.

that measures the response performance of network nodes based on the offset ratio and the time interval between request and response in CAN messages. The authors claim that each node has a fixed response offset ratio and time interval in a normal operation mode which varies significantly in attack modes. This difference in the offset ratio and time intervals is used to detect attacks in the network.

SAIDuCANT differs from the prior art because we leverage real-time schedulability analysis of messages to automate creating a specification. The novelty of our approach is in the close coupling we create between real-time theory and intrusion detection, and in the automation of parameter extraction.

III. Controller Area Network (CAN) Background and Response Time Analysis

CAN is a message-based protocol that uses a lossless bitwise arbitration to transmit binary signals over twisted pair cabling. Dominant bits represent the logical 0, and recessive bits the logical 1. As shown in Figure 1, data is transmitted between ECUs via frames that include an Identifier field, Control field, Data field, and a Cyclical Redundancy Check (CRC). The CAN protocol includes collision detection and avoidance, error detection, signaling, and fault confinement.

CAN efficiently implements static fixed priority non-preemptive scheduling of messages through bus arbitration. CAN messages may be periodic, sporadic, or aperiodic. Periodic message instances arrive at a regular interval with a fixed length called period. Sporadic messages recur with a minimum inter-arrival time between successive instances, while aperiodic message instances occur at arbitrary times.

Each transmitting message goes through the arbitration process to determine which wins the bus. When a message wins arbitration and starts transmission, it becomes non-preemptable. Messages win arbitration according to their priority, which is determined by the message identifier (ID): lower IDs have higher priority.

CAN bus is susceptible to faults due to electromagnetic interference (EMI). EMI errors can be modeled as a random single bit fault in CAN bus that, when detected, will cause a receiver to transmit an error frame and cause retransmission of the original message [39], [40]. If an error is detected either by the sending node or in the CRC field, the error is signaled directly to all the nodes on the bus. The receiving nodes will discard the received erroneous message, and the sending node, assuming only a transient fault on the wire, then enters arbitration to retransmit the message frame. The error recovery process transmits up to 31 bits in the worst case (error signaling and recovery time is typically between 17 to 31-bit times) in addition to the retransmission of the message.

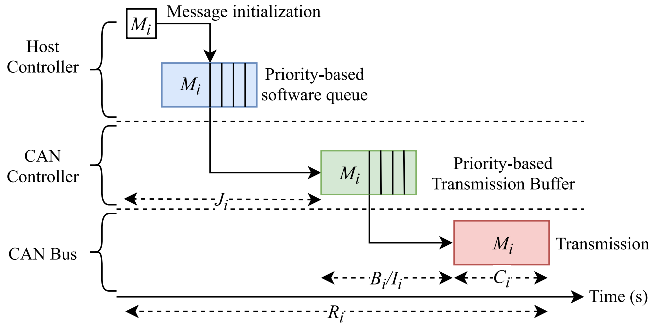


Fig. 2. CAN Message Transmission States.

Tindell *et al.* [40], [41] and Davis *et al.* [42] present a real-time model and worst case response time analysis of the CAN bus derived from fixed priority response time analysis (RTA) of CPU scheduling. We adopt their terminology and rely on some of their key results in developing our specification-based approach. For readers familiar with real-time schedulability, the key difference between task scheduling and CAN message scheduling is the use of messages in place of tasks, and each release of the message is a message instance rather than a job. A message is parameterized by its period and ID, which is a unique identifier and also the message's priority, with a lower ID having a higher priority. Every period units of time, a message releases another message instance. Each message instance has its own transmission time and queuing jitter with a data payload of 0 to 8 bytes. The length of the data payload is specified in the Data Length Code (DLC) frame field.

As illustrated in Figure 2, messages go through the steps of message release, queuing for transmission, arbitration, and, finally, transmission. The process involving a message release includes the preparation and storage in the software queue, which is considered part of the computation time of the node sending the message. A message release time is the time instant the message is ready to be written into the priority-based transmission buffer queue. When a message is released, it is written to an available transmission buffer, or if there is no available transmission buffer, it is stored in the host controller (CPU) priority-based software queue until a buffer is available for writing it. Once written to the transmission buffer, the message is ready for transmission. In the transmission buffer, messages go through an arbitration process, and the message with the highest priority gets to transmit in the bus.

Our notation is summarized in Table I. M denotes an ordered set of messages, and $M_i \in M$ is a message with ID i in the set. $M_{i,k}$ denotes the k^{th} instance of M_i, which has completion time $T_{i,k}$. If M_i is periodic, the time from 0 until the occurrence of the first instance i.e., $M_{i,1}$, is the message phase, denoted by ϕ_i. Concretely, the kth instance of M_i, denoted as $M_{i,k}$, is released at time $\phi_i + (k-1)P_i$ and should complete its transmission by time $\phi_i + k(P_i)$, where $(k = 1, 2, \ldots)$. A message may also have a deadline, however we assume a constrained, implicit deadline (equal to the period). Thus, M_i can be characterized by a 3 tuple (ϕ_i, C_i, P_i), representing the message phase, the message worst-case transmission time, and the period respectively.

TABLE I
TABLE OF NOTATIONS FOR RESPONSE TIME ANALYSIS

Variable	Definition
M	set of messages $M = (M_1, M_2, \ldots, M_n)$
$M_i \in M$	the ith message
C_i	transmission time
P_i	message period
$\widetilde{P_i}$	estimated period
R_i	worst case response time
J_i	the queuing jitter
w_i	the queuing delay
B_i	the blocking time
$f_{i,min}$	lower bound on completion time relative to release
$f_{i,max}$	upper bound on completion time relative to release
$M_{i,k}$	the kth instance of message m_i
ϕ_i	phase of M_i
$T_{i,k}$	completion time of $M_{i,k}$ (CAN message time stamp)
τ_{bit}	the transmission time of a single bit
E_i	the error overhead

Davis et al. [42] determine a message worst-case response time (WCRT) by taking the maximum response time over the instances of the message in a busy period,

$$R_i = \max_{q \in [0, Q_i - 1]} (R_i(q)) \tag{1}$$

where Q_i is the number of instances of message M_i that become ready for transmission before the end of the busy period, and $R_i(q)$ is the WCRT of instance q. $R_i(q)$ and Q_i are given by

$$R_i(q) = J_i + w_i(q) - qP_i + C_i \tag{2}$$

$$Q_i = \left\lceil \frac{t_i + J_i}{P_i} \right\rceil \tag{3}$$

where J_i, the queuing jitter of the frame, corresponds to the maximum time variation between the release of a message instance and queuing the message for transmission; w_i, the queuing delay under faults, corresponds to the maximum time a message can remain queued before successfully transmitting. This delay may be due to other higher and lower priority messages using the bus. C_i, is the transmission time, which corresponds to the maximum time a message can take to be transmitted. t_i is the length of the *priority level-i busy period* during which only messages with higher priority to i get transmitted. The busy period of the message ends at the earliest time that the bus becomes idle or when messages of lower priority get transmitted. t_i is found by solving the following recurrence relation with a starting value of $t_i^0 = C_i$ and ending when $t_i^{n+1} = t_i^n$:

$$t_i^{n+1} = B_i + E_i(t_i^n) + \sum_{k \le i} \left\lceil \frac{t_i^n + J_k}{P_k} \right\rceil C_k \tag{4}$$

where B_i is the blocking time, which is the longest time that any lower priority message can occupy the bus while message M_i is queued, and is given by

$$B_i = \max_{k>i} (C_k). \tag{5}$$

The worst case overhead caused by the error recovery mechanism that can occur for a given time interval is

$$E_i(t_i) = \left(31\tau_{bit} + \max_{k \ge i}(C_k)\right) F(t_i) \tag{6}$$

where there can be 31 overhead bits for error signaling, and τ_{bit} is the transmission time of a single bit (determined by the bus speed). $F(t_i)$ is a step function that yields the maximum number of errors on the bus for a time interval and must be a monotonic non-decreasing function. According to Broster et al. [43], the expected number of errors for the fault model in an aggressive environment is 30 faults per seconds.

The queuing delay w_i is composed of two elements: B_i, the blocking time as given in Equation 5, and I_i, the interference time, which is the longest time that all higher priority messages can occupy the bus before the message i is finally transmitted, given by

$$I_i = \sum_{k<i} \left\lceil \frac{w_i + J_k + \tau_{bit}}{T_k} \right\rceil C_k. \tag{7}$$

Therefore, the queuing delay w_i is given by:

$$w_i = B_i + I_i \tag{8}$$

The worst case queuing delay w_i given an error model to account for random errors on the bus is determined by calculating the delay for each of the Q_i instances, and is given by the following recurrence relation:

$$w_i^{n+1}(q) = B_i + E(w_i^n + C_i) + qC_i + I_i \tag{9}$$

with starting value $w_i^0(q) = B_i + qC_i$ and terminating when $w_i^{n+1}(q) = w_i^n(q)$. This analysis adds a degree of pessimism as it includes the 3-bit inter-frame space in the computed queuing delay, which can be removed by subtracting $3\tau_{bit}$ from the calculated response time values.

IV. Real-Time Specification-Based IDS Design

Expected regularity of messages in the CAN bus motivates a supervised learning approach to create the specification-based IDS. In a supervised learning approach, a classifier is trained to differentiate between normal and anomalous behavior. Supervised learning uses training and detection phases. In the training phase, the IDS collects CAN traces that represent the normal behavior of the network and extracts real-time parameters as the features that compose the specification. In the detection phase, the behavior of each message observed on the bus is checked whether or not it conforms with the specification. In our current analysis we restrict to checking only the periodic and sporadic messages. In this section, we present the design of SAIDuCANT starting with the assumed threat model. Then, we describe the method used in the training phase to extract real-time model parameters from observations of CAN bus messages before explaining the detection phase's algorithm using those parameters.

A. Threat and Attack Model

In this paper, we focus on impersonation attacks (masquerade, replay, or injection), in which the goal of the adversary is to control the vehicle. ECUs on the CAN bus take action based on the most recently received data field of specific IDs that they are programmed to monitor. By transmitting an injected message soon after the authentic message of the same ID is transmitted, the attacker's injected message will be acted on by the ECUs on the bus instead of the authentic message.

We assume an adversary is able to receive and send messages on the CAN bus. A receive operation involves eavesdropping messages, and a send operation involves transmitting injected (forged or replayed) messages. We assume the adversary does not modify any regular transmission of messages, but observes the network traffic to learn about the transmission pattern and properties of the data packets of a particular node and then impersonates that node. This assumption fits with the known attacks such as replay and masquerade attacks that penetrate the CAN bus by first subverting a non-critical ECU, and then eavesdrop and inject messages targeting the critical ECUs, but do so while behaving according to the bus protocol. SAIDuCANT aims at detecting the active transmission of an attacker, and is unable to detect passive tools that eavesdrop and record network traffic, since they do not interfere with the timing of messages transmitted on the bus.

B. Timing Model Extraction

Although we expect messages within a CAN bus to be schedulable according to some real-time model, we do not expect to know the actual model or its parameters for a given system. The exact timing model and its parameters, especially precise message periods, are difficult to obtain—they are not normally disclosed by manufacturers. Thus, we assume the RTA-based model described in Section IV and derive its real-time parameters from observations of the CAN bus. The model and parameters comprise the IDS specification. Once the specification is learned, it does not change over time unless features are added to the vehicle, for example by reflashing an ECU with a software update, in which case the RTA model would need to be relearned.

Algorithm 1 infers bounds at which the period of each message could occur by reconstructing the steps the message will go through before transmission. Bounded parameter estimates are derived from CAN bus activity by calculating upper and lower bounds for each message's period (inter-arrival time). The algorithm extracts for each distinct message M_i a bounded period estimate, $f_{i,min}, f_{i,max}$, and the transmission time C_i.

Algorithm 1 takes as input a CAN log and message ID i. It returns the estimate $\widetilde{P}_i$ of the period by iteratively calculating upper and lower bounds on the release and inter-arrival times of successive message instances. The release time of the first message instance of a given message cannot be inferred directly, because the system state prior to the first observed message is unknown. Thus, the first instance of each message is ignored. In line 4, the algorithm scans backward to find the timestamp of the previous message with lower priority or the time the bus is in an idle state. We are uncertain of the release time of $M_{i,k}$: it may have occurred at any point during recent higher-priority messages that may have interfered with its transmission until the most recent lower-priority message or an idle bus. Thus, the algorithm pessimistically selects the earliest and latest possible release times of the current message, denoted L_{cur} and H_{cur}.

To construct a bounds on the period, the algorithm subtracts the latest and earliest release of the previous instance of the same message from the earliest and latest release of the current instance, respectively, to obtain Δ_L and Δ_H. These Δ values represent the smallest and largest possible inter-arrival time between the previous and current instance. $f_{i,min}$ and $f_{i,max}$ are, eventually, the Δ_L and Δ_H that are closest to each other.

The final value of $f_{i,min}$ is taken as the estimated period $\widetilde{P}_i$, which, assuming a constant actual period and non-negative release jitter, is no greater than the actual period. The release jitter is the difference between $f_{i,max}$ and $f_{i,min}$, which describes the maximum error in the estimated $\widetilde{P}_i$ because the actual period is no greater than $f_{i,max}$.

Since vehicles of the same make, model, and even trim can offer different features, two seemingly identical cars may have distinct RTA specifications. Thus, the specification of a particular car must be obtained by running Algorithm 1 to extract the node IDs and their timing characteristics. Figure 3 shows two distinct messages for Car X and Car Y with the inferred period bounds from Algorithm 1. The difference between the lower and upper bound represents the tightness in the minimum and maximum timestamp that a message can assume. There is a variation in this tightness as seen in Figure 3 which is indicative of the performance of the algorithm on the messages in a CAN bus. Car Y shows a slightly loose bound compared to Car X. Algorithm 1 obtains upper and lower bounds for each

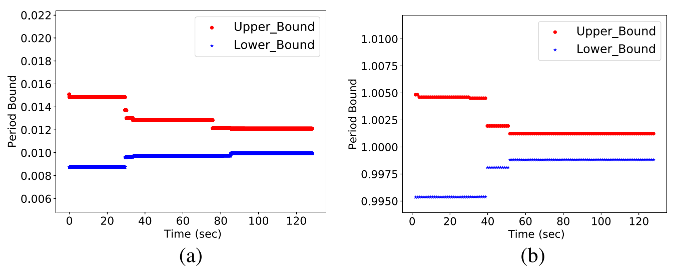


Fig. 3. Variation of inferred lower and upper bound of the period for consecutive instances of different message IDs. (a) Message A Car X. (b) Message A Car Y.

Algorithm 1: Estimate the Period and Release Jitter of a Message M_i Given a Partial Log and ID i.

```
1:  function DerivePeriodicParameters ($Log, i$)
2:      $f_{i,\min}, f_{i,\max} \leftarrow 0, \infty$
3:      for $M_{i,k} \in Log, k \geq 1$ do
4:          $T_{l,m} \leftarrow$ FindPreviousTimestamp()
5:          $L_{cur} \leftarrow T_{l,m} - C_{l,m}$
6:          $H_{cur} \leftarrow T_{i,k} - C_{i,k}$
7:          if $k > 2$ then
8:              $\Delta_L \leftarrow L_{cur} - H_{past}$
9:              $\Delta_H \leftarrow H_{cur} - L_{past}$
10:             if $\Delta_L > f_{i,min}$ and $\Delta_H < f_{i,max}$ then
11:                 $f_{i,min}, f_{i,max} \leftarrow \Delta_L, \Delta_H$
12:         $L_{past}, H_{past} \leftarrow L_{cur}, H_{cur}$
13:     $\widetilde{P}_i = f_{i,min}$
14:     $J_i = f_{i,max} - f_{i,min}$
15:     return $(\widetilde{P}_i, J_i)$
```

ID that are within approximately 5 ms of each other; thus an attacker cannot successfully inject a message without violating the expected period of the next authentic message, because the inferred period converges within ± 2 ms of the real period. Since the real period of automotive CAN messages is on the order of 10, 100, or 1000 ms, an adversary can only inject additional messages without being detected if the range between upper and lower bounds is greater than 5, 50, or 500 ms, respectively. With SAIDuCANT, the tightness on the estimate of the period makes detection of message injection attacks possible.

C. Anomaly Detection

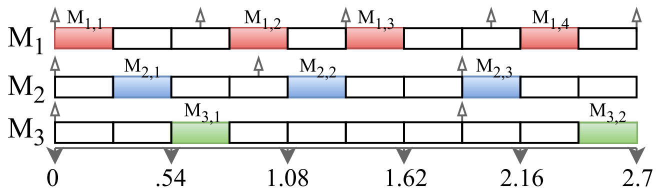
SAIDuCANT monitors the bus and calculates an interval of possible values that bounds the valid completion time of each message instance. This calculation relies on the learned parameters and the RTA model as a specification, and on the history of observations of messages that have been transmitted on the bus since the time the bus was last idle. This history contains each message's priority, transmission time, and the data payload, which are necessary to account for blocking and interference factors that delay the time between a message instance release and its transmission. A message is considered anomalous if its completion time violates the acceptable interval defined by the specification of its real-time parameters.

We obtain the response time of each message using Equation 1 with the estimated $\widetilde{P_i}$ and J_i determined by Algorithm 1. We use this response time in a supervised learning algorithm to classify messages as normal or anomalous. Algorithm 2 takes as input a message instance's completion time, the estimated period, response time, phase, and the instance count. Note that we estimate the phase ϕ_i as T_i minus C_i of the first instance. Algorithm 2 calculates the minimum timestamp that a message instance can assume by adding the phase to the instance multiplied by the period. The maximum timestamp represents the minimum timestamp plus the WCRT. The algorithm classifies the message instance as normal if its actual timestamp falls between the calculated minimum and maximum timestamps. Algorithm 2 is $O(1)$ for each message received from the bus.

We call a message instance *delayed* if it does not arrive by the expected maximum timestamp, and *dropped* if it does not arrive by the minimum timestamp of the next instance. Algorithm 2 classifies as normal the first message instance after dropped messages (Lines 6–8), and classifies delayed messages as normal (Lines 11–12).

D. Example

Consider the message log and schedule in Figure 4, composed of messages $M_1(0, 0.27, 0.675)$, $M_2(0, 0.27, 0.945)$, and $M_3(0, 0.27, 1.89)$ with M_1 having the highest priority (of 1) and M_3 having the least priority (of 3), and with time in milliseconds. The busy period starts at time $t = 0$ with the release of all the first message instances, $M_{1,1}, M_{2,1}, M_{3,1}$, and $M_{1,1}$ wins arbitration. Thus, $M_{1,1}$ causes interference for both $M_{2,1}$ and $M_{3,1}$. At $t = 0.675$, M_1 releases instance $M_{1,2}$ while $M_{3,1}$ is in transmission, thus blocking $M_{1,2}$ until $M_{3,1}$ completes. The bus is idle from $t = 1.62$ to 1.89.



$M_{i,k}$	DLC	Data	$T_{i,k}$
$M_{1,1}$	8	FF FE 7E F0 86 0B 30 00	0.27
$M_{2,1}$	8	6F 9F 6F 94 0F A0 EE 0B	0.54
$M_{3,1}$	8	01 F4 02 4D 04 18 82 B6	0.81
$M_{1,2}$	8	FF FE 7E F0 86 0B 30 00	1.08
$M_{2,2}$	8	6F 9F 6F 94 0F A0 EE 0B	1.35
$M_{1,3}$	8	FF FE 7E F0 86 0B 30 00	1.62
$M_{2,3}$	8	6F 9F 6F 94 0F A0 EE 0B	2.16
$M_{1,4}$	8	FF FE 7E F0 86 0B 30 00	2.43
$M_{3,2}$	8	01 F4 02 4D 04 18 82 B6	2.70

Fig. 4. Example of periodic message behavior in CAN bus (Time in ms.)

Algorithm 2: Anomaly Detection From Timing Specification.

```
 1: function DETECT ($T_{i,k}, \widetilde{P_i}, R_i, \phi_i, k$)
 2:     $min_{ts} \leftarrow \phi_i + (\widetilde{P_i} * k)$
 3:     $max_{ts} \leftarrow min_{ts} + R_i$
 4:     $next_{min_{ts}} \leftarrow min_{ts} + P_i$
 5:     $next_{max_{ts}} \leftarrow max_{ts} + P_i$
 6:     if $T_{i,k} > next_{max_{ts}}$ then
 7:         $k \leftarrow \left\lceil \frac{T_{i,k} - \phi_i}{P_i} \right\rceil$
 8:         return 0
 9:     if $min_{ts} \le T_{i,k} \le max_{ts}$ then
10:         return 0 $\leftarrow$ $normal$
11:     if $max_{ts} \le T_{i,k} < next_{min_{ts}}$ then
12:         return 0 $\leftarrow$ $normal$
13:     else
14:         return 1 $\leftarrow$ $anomalous$
```

To better understand how the $f_{i,min}$ and $f_{i,max}$ are calculated, consider M_1. The first instance $M_{1,1}$ is ignored. For $M_{1,2}$, scanning backward finds that the preceding message is of lower priority, which implies that the release of this message occurs during or immediately after the transmission of $M_{3,1}$. Therefore, a lower bound on the release time is given by subtracting the transmission time from the timestamp of the preceding message, i.e., $L_{cur} = T_{3,1} - C_{3,1} = 0.81 - 0.27 = 0.54$. The upper bound is always calculated directly from the message instance, e.g., $H_{cur} = T_{1,2} - C_{1,2} = 1.08 - 0.27 = 0.81$. The range from $[(T_{3,1} - C_{3,1}), (T_{1,2} - C_{1,2})] = [0.54, 0.81]$ describes the maximal time interval that $M_{1,2}$ could have spent waiting for transmission. As expected, $M_{1,2}$'s actual release time $0.675 \in [0.54, 0.81]$. Because the first instance does not calculate an upper and lower bound, the second instance is not able to calculate a valid Δ_L or Δ_H, so the algorithm stops processing this instance, stores the calculated L_{cur} and H_{cur} as L_{past} and H_{past}, and moves on to $M_{1,3}$. Scanning backward from $M_{1,3}$ find that the previous message $M_{2,2}$ has lower priority, so $L_{cur} = T_{2,2} - C_{2,2} = 1.35 - 0.27 = 1.08$. Again, the upper bound is calculated as $H_{cur} = T_{1,3} - C_{1,3} = 1.62 - 0.27 = 1.35$. Now $\Delta_L = L_{cur} - H_{past} = 1.08 - 0.81 = 0.27$ and $\Delta_H = H_{cur} - L_{past} = 1.35 - 0.54 = 0.81$. These calculated bounds are used as the first estimates for the period, so $f_{1,min} = 0.27$ and $f_{1,max} = 0.81$ after processing $M_{1,3}$. The actual period of $M_1 = 0.675 \in [0.27, 0.81]$. For $M_{1,4}$, the algorithm calculates $\Delta_L = 1.89 - 1.35 = 0.54$ and $\Delta_H = 2.16 - 1.08 = 1.08$. Although the new Δ_L improves on $f_{1,min}$, the new Δ_H is worse than the $f_{1,max}$ so the bounds are not updated. As the log ends with no more instance of M_1, its estimated period and jitter are $\widetilde{P_1} = 0.27$ and $J_1 = 0.81$.

V. Experimental Setup

We evaluate SAIDuCANT using data we collected and with published datasets. We collected data from two different sedan vehicles, Car X and Car Y, which are the same make but different model and year. The vehicles are operated in a controlled setting on a dynamometer in the Cyber Security Laboratory of the National Transportation Research Center managed by Oak Ridge National Lab, and CAN log data are collected through the OBD-II ports. The vehicles have a medium speed CAN bus and high speed CAN bus. Initial test data was recorded for the vehicle state comprising ignition key turn (handbrake on), acceleration, maintaining a constant speed, braking, and reverse. We performed attacks by injecting malicious messages at high frequency to override normal vehicle operations. These malicious messages were constructed by spoofing legitimate messages. Messages are injected at different intervals through the OBD-II port for about 60 seconds at a frequency higher than normal to cause a malfunction in the vehicle.

Furthermore, we evaluated the performance of SAIDuCANT using CAN data from Hacking and Countermeasure Research Lab made available for research purposes.[1] The dataset contains a standard vehicle operation and attack datasets comprising fuzzy, RPM spoofing, gear spoofing, and DoS attacks. These datasets were recorded from a real vehicle through the OBD-II port. The ground truth about the dataset is known as it contains information about regular and injected messages. For the gear and RPM spoofing attacks, the respective IDs are injected every 1 millisecond. The fuzzy attack dataset contains randomly injected messages IDs performed every 0.5 milliseconds while DoS attack dataset contains attacks where the dominant message ID 0000 is injected every 0.3 milliseconds to disrupt the vehicle functions.

We observed messages that appear just once in a log. These messages appeared mostly at the beginning of the log, and we suspect they relate to the initial startup of the vehicle. We have ignored these one-time messages in our results.

To evaluate IDS performance we use traditional classification metrics by collecting the number of true negatives (TN), true positives (TP), false negatives (FN), and false positives (FP), and calculating the accuracy, recall, precision, and F1 score in the usual way:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$Precision = \frac{TP}{FP + TP} \tag{12}$$

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{13}$$

We also introduce two new metrics for characterizing performance of an automotive IDS, the time to detection (TTD) and false positives before attack (FPBA), that we define as

$$TTD = T_D - T_A \tag{14}$$

$$FPBA = \sum_{m \in Log[0:T_A]} isFP(m) \tag{15}$$

where T_D and T_A denote the detection time and completion time of the first instance of an injected attack message, respectively, $Log[x : y]$ is a subsequence of messages observed on the network from time x until y, and $isFP(m)$ is a binary valued function that returns 1 if message m is a false positive, and 0 otherwise. The TTD measures the time after an attack happens before it is detected, hence it is a latency indicator of IDS performance. FPBA captures the classifier performance prior to the existence of an attack.

These metrics provide a more meaningful measure of performance than the traditional classifier metrics. The information provided by these metrics relate the classifier accuracy with the timeliness of detection. Often, traditional classifier metrics are used for measuring model or algorithm performance, but they may give a false sense of performance by achieving high accuracy and low false positive rates that still translate to an impractical solution. For example, even 0.01% false positive rate implies one false positive per second in a 1 Mbps bus. We introduce TTD and FPBA, which have never been used for evaluating automotive IDSs, to better classify IDS performance with respect to timely, accurate detection.

VI. Experiments

We conducted five experiments using the two datasets. The first experiment evaluates SAIDuCANT in the absence of attacks, and the second evaluates with attacks, both using the dataset collected at ORNL. In the third experiment, we validate the performance of SAIDuCANT with synthetic attacks derived from that dataset. The fourth and fifth experiments evaluate SAIDuCANT using the open research data, and compare SAIDuCANT with interval- and frequency-based IDSs.

A. *Experiment 1: All Normal Data*

First, we recorded data for six representative datasets on Car X and five on Car Y. Each dataset is composed of data recorded for about 120 seconds of standard vehicle operations, i.e., *normal* data. One of the datasets *(training dataset)* is used to extract

[1]https://sites.google.com/a/hksecurity.net/ocslab/Datasets/CAN-intrusion-dataset

TABLE II
OUTCOME OF SAIDUCANT ON NORMAL DATA

Cars	Messages	TN	FP	Accuracy
Car X	486091	485476	615	0.9987
	323942	323673	269	0.9992
	241157	241061	96	0.9996
	246741	246650	91	0.9996
	239107	239047	60	0.9997
Car Y	345781	345451	330	0.9990
	327604	327310	294	0.9991
	381907	381383	524	0.9986
	337575	337086	489	0.9986

the timing model specifications of each message on the bus by applying Algorithm 1. The other datasets *(test datasets)* are used to validate the model by invoking Algorithm 2 for every message instance. A message instance is classified as anomalous if 1.) The message ID was not recorded during training, or 2.) Algorithm 2 returns *anomalous*.

For this experiment, which does not have attack data, any anomalous labels are false positives and normal labels are true negatives. Thus, the accuracy is simply the ratio of normal labels to total messages. Table II shows the classifier accuracy of SAIDuCANT (Algorithm 2) over each *test dataset*. Precision and recall are not calculated for this experiment because the dataset does not contain any attack messages which implies that there is only one relevant instance or data point of interest in each dataset. The message column indicates the total number of message instances present in each dataset.

About 48% of false positives we observed are from periodic messages with the same ID, but different phases. These messages exhibit the same behavior as a regular message, except they appear to either release multiple instances per period, or to transmit several messages with identical periods that are offset from each other. Our current detection algorithm is unable to classify these messages because we have assumed one periodic message per ID. We discuss other possible sources and mitigation for false positives in Section VII.

B. *Experiment 2: Real Attack*

This experiment considers the algorithm performance on a real attack dataset involving the vehicle backup light for Car X. We performed a message injection attack that activates the backup light every 700 microseconds. The injections are made in intervals of length 15 seconds, with 15 seconds of non-injected messages in between. Thus, the attack data contains a mix of normal and attack message instances during injection intervals [15, 30] and [45, 60] seconds, and normal message instances outside those intervals.

In this experiment, due to infrastructure limitations, we are not certain which logged messages are from our injection and which are from the vehicle's normal operations. Thus, we cannot calculate metrics of classifier performance for this experiment. In this experiment we injected 2,845 messages to Car X as it was being driven on the dynamometer. The attack log contains 154,564 message instances, with 3,767 of them labeled anomalous by Algorithm 2. Although we cannot distinguish our injected messages from authentic ones in the log, we can say that we did not observe any anomalous labels for message instances of the injected message ID outside of the injection intervals, so we have confidence that the injected messages are, mostly, correctly labeled anomalous.

C. *Experiment 3: Synthetic Attacks*

We simulate message injection attacks on the *test datasets* by injecting a particular ID 2 to 3 times faster when an idle bus time is observed. This attack is achieved by recreating the expected message trace and injecting message IDs during the idle time. The idle time is used to ensure that the simulated attacks are accurately spaced to avoid any overlap in the message timestamp. The injected message is not altered, thus maintaining the same field properties as a normal message but with a different timestamp. The timestamps of the injected messages are set to fit within the limit of the idle time.

In this experiment, we have both attack and benign messages, and we know the ground truth because we know which messages we injected. Thus, we present the classification FP, TP, FN, and TN. Table III shows the classifier performance of Algorithm 2 for the synthetically generated attack data. The message column shows the total number of messages in each dataset. The predictive value of our positive test indicates an approximation of 90 to 99 percent accuracy. An average 91 percent recall indicates that the algorithm mostly labels the injected anomalous data correctly.

D. *Experiment 4: Real Attacks (Open-Source Data)*

In this experiment, we consider the algorithm performance on the open-source attack data. Table IV shows the classifier performance on the four attack datasets. In spoofing the gear and RPM datasets, the injected IDs constitute 99.72% and 99.91% of the total number of false positives, respectively. We found that before the start of the message injection attack, SAIDuCANT detects no FP in both datasets. This implies that when the IDs are being injected, they contribute to the regular IDs missing their expected deadlines, which results in false positives. For the fuzzy attack dataset, the false positives are distributed across the injected IDs. The DoS attack dataset exhibits zero false negatives with a small number of false positives ($<$0.003%) in the whole dataset.

E. *Experiment 5: Comparison With Other Detection Approaches*

Using the same dataset from VI-D, we compare SAIDuCANT with interval- and frequency-based detection approaches. In the interval-based detection approach, the IDS reads the normal CAN frames to build a timing model for each message ID interval. The IDS checks each message ID and calculates the average time interval between subsequent messages in the attack-free dataset. The generated intervals are then used for detection against the attack datasets. If an interval in the attack datasets is less than half of the calculated average interval for the message ID, the IDS alerts for anomalous behavior. The frequency-based detection approach calculates the frequency of

TABLE III
OUTCOME OF SAIDUCANT WITH SYNTHETIC DATA INJECTION ALGORITHM

Cars	Messages	TN	FP	FN	TP	Accuracy	Precision	Recall	F1 Score	TTD (ms)	FPBA
Car X	493042	485467	624	620	6331	0.9975	0.9103	0.9108	0.9105	0	0
	325766	323666	276	152	1672	0.9987	0.8583	0.9167	0.8865	0	0
	243698	241056	101	95	2446	0.9992	0.9603	0.9626	0.9615	0	5
	248428	246650	91	308	1379	0.9984	0.9381	0.8174	0.8736	0	7
	241739	239047	60	208	2424	0.9989	0.9758	0.9210	0.9476	0	3
Car Y	346930	345451	330	130	1019	0.9987	0.7554	0.8869	0.8159	0	0
	327604	327310	294	122	2433	0.9987	0.8922	0.9523	0.9212	0	0
	381907	381383	524	230	1933	0.9980	0.7867	0.8937	0.8368	0	2
	338869	337086	489	1	1293	0.9986	0.7256	0.9992	0.8407	0	51

TABLE IV
OUTCOME OF SAIDUCANT WITH REAL ATTACK DATASET (OPEN SOURCE)

Attacks	Messages	TN	FP	FN	TP	Accuracy	Precision	Recall	F1 Score	TTD (ms)	FPBA
Gear Spoofing	4,443,142	499,934	674,784	97,318	3,171,105	0.8262	0.8245	0.9702	0.8915	10	0
RPM Spoofing	4,621,702	534,974	798,213	119,923	3,177,591	0.8033	0.8010	0.9636	0.8748	9	0
Fuzzy	3,838,860	479,781	455,447	12,066	2,891,565	0.8782	0.8639	0.9958	0.9252	0	1
DoS Attack	3,665,771	587,521	70,475	0	3,007,774	0.9808	0.9771	1.0	0.9884	0	0

TABLE V
COMPARISON OF THE SAIDUCANT WITH INTERVAL-BASED AND FREQUENCY-BASED DETECTION APPROACHES

Attacks	Detection Approach											
	Interval-based				Frequency-based				SAIDuCANT			
	Recall	TTD(ms)	FPBA	F1 Score	Recall	TTD(ms)	FPBA	F1 Score	Recall	TTD(ms)	FPBA	F1 Score
Gear Spoofing	0.9367	2	190	0.7185	0.8739	1585	793	0.8739	0.9702	10	0	0.8915
RPM Spoofing	0.9528	0	144	0.7332	0.9618	79	160	0.9231	0.9636	9	0	0.8748
Fuzzy	0.9787	0	133	0.7708	0.8845	133	65	0.8847	0.9958	0	1	0.9252
DoS Attack	0.9998	0	139	0.8176	0.9032	204	356	0.9032	1	0	0	0.9884

each message ID in the attack-free dataset. Frequency is the rate of messages observed in a set time interval. For this work, we used a time interval of one second. If the frequency of a message deviates at a rate greater than two times normal, the IDS indicates an anomaly.

Table V shows the performance of SAIDuCANT compared to interval- and frequency-based detection. The table clearly shows that SAIDuCANT performs better—in terms of both the time it takes to detect attacks and the number of false positives before an attack happens—compared to other approaches. Over the four different attack scenarios, SAIDuCANT outperforms other timing-based approaches with negligible (at most one) FP prior to the start of an attack; in contrast, the interval- and frequency-based approaches on average detect over a hundred FP before an attack, and even in the best case detected 65 FP before the attack started. SAIDuCANT achieves these better results because the model specifications leverage the network semantics based on real-time theory. SAIDuCANT provides a significantly higher detection ratio for DoS and fuzzy attacks compared to the other methods; the F1 Score for SAIDuCANT algorithm is over 90 percent compared to 80 percent for interval-based and approximately 90 percent for frequency-based approaches, respectively.

VII. DISCUSSION AND FUTURE WORK

Due to the stochastic nature of driving, we obtained different results for each test dataset. The variability in different driving modes is one of the causes of the disparities in the results. Some of the data are recorded while the vehicles are in an accessory mode, drive to accelerate, drive to decelerate, accelerate in reverse, decelerate in reverse, maintaining a constant speed and braking operations. Also, the driver's actions and the underlying driving operations can be contributing factors to the difference in the presence of CAN messages and therefore the experimental results.

False positives can be reduced by manually tuning the upper bound of some of the IDs with arbitrary large periods (in the order of seconds) by 0.05 ms without increasing the attacker's chance of successful data injection. However, blindly applying a tuning number to the entire set of IDs increases the false negatives. 2The need for this tuning is a result of uncertainty in the RTA model, and future work could consider a more rigorous, systematic approach to tuning automatically or adaptively to accommodate for this uncertainty.

False positives may also be caused by hardware malfunctions that significantly disturb the timing behavior of messages. We did not observe any such scenarios in our experiments because the vehicular systems operated normally. In our approach, if such malfunctions cause an ECU to transmit too early or too often, then the behavior would be detected and treated as an attack. Note that other timing-based IDS, such as interval and frequency detection, would exhibit similar false positive behavior in the presence of hardware malfunctions.

In our analysis, we observe some messages with multiple periods, which Koyama et al. have described as Type-1 mixed

CAN messages [44]. These messages exhibit the same behavior as regular periodic signals but have extra instances that are triggered by events. These types of messages are common on the medium-speed CAN bus used for body electronics in some vehicles. For example, the door sensors of the car send their status periodically as sort of a heartbeat, but when a change occurs, such as someone opens the door, a status message is sent immediately. Our current detection algorithm does not perform very well in classifying these messages, because we have assumed one periodic message per ID. We aim to better classify such messages in future work.

Presently, our detection algorithm can detect attacks on periodic and sporadic messages, but not aperiodic messages or message IDs with several message instances per period. However, most of the significant information relating to the control of the safety systems in vehicles are transmitted periodically and sporadically with a single message instance from a single source ECU. Aperiodic messages are difficult to characterize because the timing of such a message cannot be ascertained at any given time and, to our knowledge, cannot be represented with a mathematical equation. In our analysis, messages that occur once on the bus are not labeled anomalous if their IDs are registered in the allowed list of nodes that can access and transmit on the bus.

In an advanced attack scenario, an adversary stops the transmission from the victim node before transmitting malicious frames. We consider two cases for this scenario. The first case is that the attacker compromises the victim node's software/firmware and sends the malicious messages from the victim ECU: SAIDuCANT cannot detect this case, and to our knowledge, neither can any IDS that only uses timing-based features nor any of the related work in network IDS–it remains an open problem, in a stronger threat model. The second case is that the attacker first launches a bus-off attack against the victim, and then masquerades as the victim after the bus-off is successful. SAIDuCANT currently does not consider this case, in which the attacker modifies messages on the bus to cause a bus-off state in the victim. For future work, we aim to modify SAIDuCANT to detect the bus-off attack as a prelude to the masquerade attack.

Plans for further study aim at reducing false positives, investigating other attack scenarios, and examining the recovery strategies for the in-vehicle network after an attack happens. We believe that the real-time model provides a solid theoretical foundation for such investigations.

VIII. Conclusion

In this paper, we present SAIDuCANT as an approach for detecting intrusions in in-vehicle networks using a specification-based IDS. The specification is developed through observations of message timing and worst case response time analysis of the CAN bus. We developed an efficient and straightforward algorithm to estimate the real-time parameters of the RTA-based model online in a black box approach. We evaluated SAIDuCANT experimentally on datasets from two different cars and open-source vehicle data. The IDS can detect message injection attacks on the CAN bus with high accuracy and low false positive rates. Compared to other detection approaches, SAIDuCANT exhibits a better F1 score compared with interval- and frequency-based approaches while reducing detection delay. We introduced two new metrics, TTD and FPBA, that measure the performance of an IDS respecting classifier accuracy and timeliness, for which SAIDuCANT yields better and consistent performance as compared to other detection algorithms. SAIDuCANT raises at most one false positive before an attack as opposed to interval- and frequency-based approaches that exhibit a minimum of 65 false positives prior to an attack. SAIDuCANT can be easily implemented on a vehicle's gateway ECU with limited computing power.

References

[1] "The connected car ecosystem: 2015–2030—Opportunities, challenges, strategies forecasts," SNS Research, Tech. Rep. 3300941, 2015. [Online]. Available: http://www.researchandmarkets.com/reports/3300941/

[2] "Cyber security in the connected vehicle report 2016," TU-Automotive, Tech. Rep. 351221, 2016. [Online]. Available: http://tu-auto.com/cybersecurityreport/

[3] H. Olufowobi and G. Bloom, "Connected cars: Automotive cybersecurity and privacy for smart cities," in *Smart Cities Cybersecurity and Privacy*. New York, NY, USA: Elsevier, 2019, pp. 227–240.

[4] S. Checkoway *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. USENIX Conf. Secur.*, 2011, pp. 323–338.

[5] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," BlackHat USA, 2015.

[6] H. Olufowobi, G. Bloom, C. Young, and J. Zambreno, "Work-in-progress: Real-time modeling for intrusion detection in automotive controller area network," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 161–164.

[7] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *Proc. IEEE Intell. Vehicles Symp.*, 2011, pp. 528–533.

[8] K. Koscher *et al.*, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 447–462.

[9] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks–practical examples and selected short-term countermeasures," in *Proc. Int. Conf. Comput. Safety, Rel., Secur.*, 2008, pp. 235–248.

[10] J. A. Bruton, "Securing CAN bus communication: An analysis of cryptographic approaches," *Nat. Univ. Ireland, Galway*, Ireland, pp. 1–5, 2014.

[11] W. A. Farag, "CANTrack: Enhancing automotive CAN bus security using intuitive encryption algorithms," in *Proc. 7th Int. Model., Simul., Appl. Optim., Conf.*, 2017, pp. 1–5.

[12] L. Yu, J. Deng, R. R. Brooks, and S. B. Yun, "Automobile ecu design to avoid data tampering," in *Proc. 10th Annu. Cyber Inf. Secur. Res. Conf.*, 2015, pp. 92–98.

[13] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *Proc. 6th Int. Inf. Assurance Secur., Conf.*, 2010, pp. 92–98.

[14] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Proc. Intell. Vehicles Symp.*, 2011, pp. 1110–1115.

[15] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, "Survey on security threats and protection mechanisms in embedded automotive networks," in *Proc. 43rd Annu. IEEE/IFIP Dependable Syst. Netw. Workshop, Conf.*, 2013, pp. 1–12.

[16] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *Proc. World Congr. Ind. Control Syst. Secur.*, 2015, pp. 45–49.

[17] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS One*, vol. 11, no. 6, 2016 Art no. e0155781.

[18] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *Proc. USENIX Secur. Symp.*, 2016, pp. 911–927.

[19] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *Proc. Int. Conf. Inf. Netw.*, 2016, pp. 63–68.

[20] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *Proc. IEEE 2nd Int. Forum Res. Technol. Soc. Ind. Leveraging Better Tomorrow*, Sep. 2016, pp. 1–6.

[21] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom, "Survey of automotive controller area network intrusion detection systems," *IEEE Des. Test*, vol. 36, no. 6, pp. 48–55, Dec. 2019.

[22] M. Salem, M. Crowley, and S. Fischmeister, "Anomaly detection using inter-arrival curves for real-time systems," in *Proc. IEEE 28th Euromicro Conf. Real-Time Syst.,* 2016, pp. 97–106.

[23] A. Boudguiga, W. Klaudel, A. Boulanger, and P. Chiron, "A simple intrusion detection method for controller area network," in *Proc. IEEE Commun., Int. Conf.*, 2016, pp. 1–7.

[24] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell, "Modeling inter-signal arrival times for accurate detection of CAN bus signal injection attacks: A data-driven approach to in-vehicle intrusion detection," in *Proc. 12th Annu. Conf. Cyber Inf. Secur. Res.*, 2017, Art. no. 11.

[25] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "Voltageids: Low-level communication characteristics for automotive intrusion detection system," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 8, pp. 2114–2129, Aug. 2018.

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

[27] H. Olufowobi *et al.*, "Anomaly detection approach using adaptive cumulative sum algorithm for controller area network," in *Proc. ACM Workshop Automot. Cybersecurity*, 2019, pp. 25–30.

[28] T. Kuwahara *et al.*, "Supervised and unsupervised intrusion detection based on CAN message frequencies for in-vehicle network," *J. Inf. Process.*, vol. 26, pp. 306–313, 2018.

[29] C.-Y. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. Levitt, "A specification-based intrusion detection system for AODV," in *Proc. 1st ACM Workshop Secur. Ad Hoc Sensor Netw.*, 2003, pp. 125–134.

[30] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE Commun. Surv. Tut.*, vol. 16, no. 1, pp. 266–282, Jan.–Mar. 2014.

[31] R. Mitchell and R. Chen, "Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 1, pp. 16–30, Jan./Feb. 2015.

[32] R. Mitchell and I.-R. Chen, "Specification based intrusion detection for unmanned aircraft systems," in *Proc. 1st ACM MobiHoc Workshop Airborne Netw. Commun.*, 2012, pp. 31–36.

[33] P. Uppuluri and R. Sekar, "Experiences with specification-based intrusion detection," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2001, pp. 172–189.

[34] D. Fauri, D. R. dos Santos, E. Costante, J. den Hartog, S. Etalle, and S. Tonetta, "From system specification to anomaly detection (and back)," in *Proc. Workshop Cyber-Phys. Syst. Secur. PrivaCy*, 2017, pp. 13–24.

[35] H. Esquivel-Vargas, M. Caselli, and A. Peter, "Automatic deployment of specification-based intrusion detection in the BACnet protocol," in *Proc. Workshop Cyber-Phys. Syst. Secur. PrivaCy.*, 2017, pp. 25–36.

[36] U. E. Larson, D. K. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *Proc. Intell. Vehicles Symp.*, 2008, pp. 220–225.

[37] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks," *Int. J. Embedded Syst.*, vol. 10, no. 1, pp. 1–12, 2018.

[38] H. Lee, S. H. Jeong, and H. K. Kim, "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame," in *Proc. Privacy, Secur. Trust*, 2017, pp. 161–164.

[39] S. Punnekkat, H. Hansson, and C. Norstrom, "Response time analysis under errors for CAN," in *Proc. IEEE 6th Real-Time Technol. Appl. Symp., Proc.*, 2000, pp. 258–265.

[40] K. Tindell, A. Burns, and A. Wellings, "Calculating controller area network (CAN) message response times," in *Proc. Distrib. Comput. Control Syst.*, 1995, pp. 29–34.

[41] K. Tindell, H. Hanssmon, and A. J. Wellings, "Analysing real-time communications: Controller area network (CAN)," in *Proc. Real-Time Syst. Symp.*, 1994, pp. 259–263.

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

[43] I. Broster, A. Burns, and G. Rodriguez-Navas, "Timing analysis of real-time communication under electromagnetic interference," *Real-Time Syst.*, vol. 30, no. 1-2, pp. 55–81, 2005.

[44] T. Koyama, T. Shibahara, K. Hasegawa, Y. Okano, M. Tanaka, and Y. Oshima, "Anomaly detection for mixed transmission CAN messages using quantized intervals and absolute difference of payloads," in *Proc. ACM Workshop Automot. Cybersecurity*, 2019, pp. 19–24.

Habeeb Olufowobi received the B.Sc. degree in computer science from Fountain University, Osogbo, Nigeria, the M.Sc. degree in electrical engineering specializing in computer systems from California State Polytechnic University, Pomona, CA, USA, in 2014, and the Ph.D. degree in computer science from Howard University, Washington, DC, USA, in 2019. He is currently a Lecturer with the Department of Electrical Engineering and Computer Science, Howard University, Washington, DC, USA. His research interests include embedded systems security and network security, with particular interest in vehicular networks, and sensor data provenance.

Clinton Young received the B.S and master's degree in electrical engineering from the Iowa State University, Ames, IA, USA. He is currently working toward the Ph.D. degree in computer engineering. He is currently a Research Assistant working on automotive security with Prof. Zambreno and Gedare Bloom, Howard University, Washington, DC, USA. His research interests include embedded systems, software and hardware security.

Joseph Zambreno received the B.S. degree (*summa cum laude*) in electrical and computer engineering from Northwestern University, Evanston, IL, USA, in 2001, the M.S. degree in electrical and computer engineering from Northwestern University, Evanston, IL, USA, in 2002, and the Ph.D. degree in electrical and computer engineering from Northwestern University, Evanston, IL, USA, in 2006. He is a Professor of electrical and computer engineering with Iowa State University, Ames, IA, USA, which he joined in 2006. His research interests include computer architecture, compilers, embedded systems, reconfigurable computing, and hardware/software codesign, with a focus on run-time reconfigurable architectures and compiler techniques for software protection.

Gedare Bloom (SM'19) received the Ph.D. degree in computer science from The George Washington University, Washington, DC, USA, in 2013. He joined the University of Colorado Colorado Springs as an Assistant Professor of computer science in 2019. He was an Assistant Professor of computer science with Howard University from 2015 to 2019. His expertise is computer system security with a focus on real-time embedded systems. He has authored more than 20 peer reviewed articles, and served as a Program Committee Member and Technical Referee for flagship conferences and journals in these areas.