Work-in-Progress: Real-Time Modeling for Intrusion Detection in Automotive Controller Area Network

Habeeb Olufowobi, Gedare Bloom
Electrical Engineering and Computer Science
Howard University
Email: {habeeb.olufowobi, gedare.bloom}@howard.edu

Clinton Young, Joseph Zambreno
Electrical and Computer Engineering
Iowa State University
Email: {cwyoung, zambreno}@iastate.edu

***Abstract*—Security of vehicular networks has often been an afterthought since they are designed traditionally to be a closed system. An attack could lead to catastrophic effect which may include loss of human life or severe injury to the driver and passengers of the vehicle. In this paper, we propose a novel algorithm to extract the real-time model of the controller area network (CAN) and develop a specification-based intrusion detection system (IDS) using anomaly-based supervised learning with the real-time model as input. We evaluate IDS performance with real CAN logs collected from a sedan car.**

I. Introduction

Vehicles may contain over 100 embedded control systems, or electronic control units (ECUs), interconnected through the in-vehicle network that facilitates their communications. ECUs perform distinct operations and function individually as a node on the vehicle network. Common networks include the controller area network (CAN), local interconnect network (LIN), and FlexRay. Specifically, our focus is on CAN which is the most commonly used bus system as the automotive network. In CAN topology, each ECU is connected to the same channel for communication through protocols specific to CAN bus, and messages are broadcast to the entire network.

The lack of inherent security features such as message encryption and authentication have paved the way for adversaries to exploit the vehicular network [1]–[3]. Observing messages sent through the network can reveal information that an adversary can use to subvert and infiltrate the vehicle operations. Limitations in the computational, memory, and power resources of ECUs has been a hindrance to implementing complex security mechanisms. Hence, an important requirement in implementing security mechanisms for vehicular networks is a lightweight and computationally efficient algorithm.

In this paper, we propose a specification-based intrusion detection system (IDS) using a real-time model of CAN. In this approach, the intended behaviors of CAN are analyzed and modeled using schedulability analysis derived from a message trace. We propose an algorithm that can effectively reconstruct the timing model of the messages based on the trace. This algorithm uses response time analysis to characterize bus operations, formulates the timing properties, and develops a specification of the normal activities that can be used to identify violations. In contrast to prior works, our approach leverages the real-time schedulability analysis of the CAN bus to build a set of timing specifications for the network activities to compare with the observed activities of the network to detect anomalous behavior. We validate our approach with real CAN data.

II. Background and System Model

In this section, we introduce notation and describe the timing model of the CAN bus. Tindell et al. [4], [5] and Davis et al. [6] present a real-time model and worst case response time analysis of the CAN derived from fixed priority response time analysis 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. Note that the CAN bus is formulated as a non-preemptive, fixed priority scheduler that may support periodic, sporadic, and aperiodic messages; we currently limit our analysis to periodic messages.

CAN consists of a set of nodes called ECUs interconnected by a broadcast channel. CAN Messages can be transmitted periodically, sporadically, or aperiodically, but in our current analysis we restrict to periodic messages. Each message has a data of up to 8 bytes specified by the data length code (DLC) that determines the messages transmission time. CAN efficiently implements static fixed priority non-preemptive scheduling of messages. The CAN protocol includes collision detection and avoidance, error detection, signaling, and fault confinement.

Scheduling decisions occur through bus arbitration. Each transmitting message goes through the arbitration process to determine which message 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). A message with a lower ID has 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. A fault can lead to overhead in the error frame and cause retransmission. CAN implements an efficient error handling mechanism in which an error detected in the bus is signaled to the sending node [4], [7]. The

receiving nodes will discard the received erroneous message, and the sending node retransmits the message. When an error is detected, the recovery process transmits up to 31 bits in the worst case in addition to the retransmission of the message.

A. CAN System Model and Response Time Analysis

Our notation is summarized in Table I. M denotes an ordered set of periodic 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. 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.

Davis et al. [6] 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.

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

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

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; and t_i is the length of the *priority level-i busy period*.

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

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

and $E_i(t_i)$ is the worst case overhead caused by the error recovery mechanism that can occur for a given time interval,

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

where there can be 31 overhead bits for error signaling, and τ_{bit} is 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 t_i and must be a monotonic non-decreasing function. According to Broster et al. [8], the expected number of errors for the fault model in an aggressive environment is 30 faults per seconds.

TABLE I: Table of Notations

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

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 by solving the following recurrence relation:

$$w_i^{n+1}(q) = B_i + E(w_i^n + C_i) + qC_i + \sum_{k<i} \left\lceil \frac{w_i^n + J_k + \tau_{bit}}{P_k} \right\rceil C_k \tag{7}$$

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 interframe space in the computed WCRT which can be removed by subtracting $3\tau_{bit}$ from the calculated response time values.

III. CAN Timing Model Reconstruction and Anomaly Detection

The exact timing model parameters, especially precise message periods, are difficult to obtain—they are not normally disclosed by manufacturers. Thus, we reconstruct the real-time model parameters of the periodic messages by observing the message behavior on the CAN. Algorithm 1 infers bounds at which the period of each message could occur by reconstructing the steps the message will go through before transmission. These bounds are derived using the analysis described in Section II applied to information available globally on the CAN. The algorithm extracts 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 start of the log is unknown; indeed, the release of the first instance may occur prior to the start of the log. 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 $M_{i,k}$, which may have

Algorithm 1 Estimate the period and release jitter of a message M_i given a Log and ID i.

function DERIVEPERIODICPARAMETERS(Log, i)
  $f_{i,min}, f_{i,max} \leftarrow 0, \infty$
  for $M_{i,k} \in Log, k \geq 1$ **do**
    $T_{l,m} \leftarrow$ FindPreviousTimestamp()
    $L_{cur} \leftarrow T_{l,m} - C_{l,m}$
    $H_{cur} \leftarrow T_{i,k} - C_{i,k}$
    if $k > 2$ **then**
      $\Delta_L \leftarrow L_{cur} - H_{past}$
      $\Delta_H \leftarrow H_{cur} - L_{past}$
      if $\Delta_L > f_{i,min}$ **and** $\Delta_H < f_{i,max}$ **then**
        $f_{i,min}, f_{i,max} \leftarrow \Delta_L, \Delta_H$
    $L_{past}, H_{past} \leftarrow L_{cur}, H_{cur}$
  $\widetilde{P}_i = f_{i,min}$
  $J_i = f_{i,max} - f_{i,min}$
  return $(\widetilde{P}_i, J_i)$

occurred at any point between the first message with lower priority that could have blocked it or an idle bus, and until the end of the intervening messages of higher priority that may have interfered with transmission. 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 of the message. $f_{i,min}$ and $f_{i,max}$ update when Δ_L and Δ_H are closer.

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 $f_{i,max}$.

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, the estimated period, response time, and the message phase. Note that the phase ϕ_i estimate is 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 the maximum timestamps.

A. Example

Consider the schedule in Figure 1, 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

Algorithm 2 Anomaly detection from timing specification.

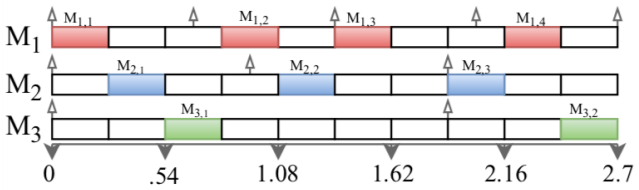
function DETECT($M_{i,k}, \widetilde{P}_i, R_i, \phi_i$)
  $min_{ts} \leftarrow \phi_i + (\widetilde{P}_i * k)$
  $max_{ts} \leftarrow min_{ts} + R_i$
  if $min_{ts} \leq T_{i,k} \leq max_{ts}$ **then**
    return 0 $\leftarrow normal$
  else
    return 1 $\leftarrow anomalous$

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, which therefore blocks $M_{1,2}$ until $M_{3,1}$ finishes transmission. The bus is idle from *t=1.62 to 1.89*. The embedded table shows the corresponding log for these messages with sample data, DLC, and completion time $T_{i,k}$.

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 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 $J_1 = 0.81$.

IV. PRELIMINARY RESULTS AND ANALYSIS

We illustrate and evaluate our model using a message set logged through the OBD-II port of a real sedan vehicle while driving on a dynamometer. Initial test data was recorded for the vehicle state comprising of ignition key turn (handbrake on), acceleration, maintaining a constant speed, braking, and reverse. We performed attacks by injecting malicious messages



$M_{i,k}$	**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. 1: Example of periodic message behavior. (Time in ms.)

at high frequency to override normal vehicle operations. These malicious messages were constructed by spoofing legitimate messages transmitting on the bus. We identified message IDs such as wheel speed and backup light while observing the recorded normal data to construct the attack. Messages are injected at different intervals through the OBD-II port for about 60 seconds at a frequency higher than the observed to cause a malfunction in the vehicle.

The test vehicle has a medium speed and high speed CAN bus, and our analysis currently focuses only on the medium speed bus. Through manual analysis, we deduced that most signals on this bus are periodic. We conduct two different experiments. First, we recorded data for five different standard vehicle operations, i.e., *normal* data, for about 220 seconds each. Two of these datasets are used to train the model by applying algorithm 1. We use the other three datasets to test 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*. Table II shows the performance of our approach measured by calculating the classifier accuracy of Algorithm 2 over at most 60 second time windows of the three test datasets. The message column indicates the total number of message instances present in each window. We observe a high false positive rate due to lines 2 and 3 of Algorithm 2. When a message is labeled anomalous, the IDS does not increment its counter for k, which, for a false positive, creates a scenario where the k gets stuck and we get a sequence of false positives until the end of the test window. We found that in each window there are only a few root cause false positives when a message instance transmits about a hundredth of a millisecond after the maximum predicted interval. In future work we will explore how to mitigate these trains of false positives.

We validate our detection method on a single attack dataset involving the vehicle backup light. We performed a signal injection attack that injects the message to activate the backup light every 700 microseconds. The injections are made in

TABLE II: Outcome of classification algorithm

Messages	TN	FP	Accuracy
85101	84549	552	99.3514
105674	104885	789	99.2534
105664	104815	849	99.1965
66909	66096	813	98.7849
105495	104954	541	99.4872
71974	71503	472	99.3442
105496	104969	527	99.5005
105666	104925	741	99.2987
105665	104846	819	99.2249

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. The attack log contains 106,877 message instances, with 3,365 of them labeled anomalous. Although we know that we injected 2,845 messages, 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, but 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 labelled anomalous.

V. Conclusion and Future Work

In this paper, we described a specification-based IDS using response time analysis of CAN that has promising preliminary results. Future work will evolve our approach to reduce false positives and to consider sporadic and aperiodic messages.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS 1646317 and CNS 1645987.

References

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

[2] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 528–533.

[3] 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.

[4] K. Tindell, A. Burns, and A. Wellings, "Calculating controller area network (can) message response times," in *Distributed Computer Control Systems 1994*. Elsevier, 1995, pp. 29–34.

[5] K. Tindell, H. Hanssmon, and A. J. Wellings, "Analysing real-time communications: Controller area network (can)." in *RTSS*, 1994.

[6] 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.

[7] S. Punnekkat, H. Hansson, and C. Norstrom, "Response time analysis under errors for can," in *Real-Time Technology and Applications Symposium, 2000. RTAS 2000. Proceedings. Sixth IEEE*. IEEE, 2000.

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