Evaluation of the Architecture Alternatives for Real-Time Intrusion Detection Systems for Vehicles

Mubark Jedh [1], Jian Kai Lee[1], and Lot i ben Othmane[2]

[1]Iowa State University, Ames, IA, USA
[2]University of North Texas, TX, USA

***Abstract*—Attackers demonstrated the use of remote access to the in-vehicle network of connected vehicles to take control of these vehicles. Machine-learning-based Intrusion Detection Systems (IDSs) techniques have been proposed for the detection of such attacks. The evaluations of some of these IDSs showed their efficacy in terms of accuracy in detecting message injections but were performed offline, which limits the confidence in their use for real-time protection scenarios. This paper evaluates four architecture designs for real-time IDS for connected vehicles using Controller Area Network (CAN) datasets collected from a moving vehicle under malicious speed reading message injections. The evaluation shows that a real-time IDS for a connected vehicle designed as a separate process for CAN Bus monitoring and another one for anomaly detection engine is reliable (does not lose messages) and could be used for real-time resilience mechanisms as a response to cyber-attacks.**

***Keywords*— In-vehicle Network Security, Intrusion Detection, Real-time Intrusion Detection, CAN Bus**

I. Introduction

Modern Automobile contains 20 to 80 Electronic Control Units (ECUs) that control the functionalities of the vehicle, including engine, power steering, and seats. These ECUs communicate using the Controller Area Network (CAN) bus protocol [1]. The CAN protocol was designed more than 40 years ago and has still widespread use in automotive, aerospace, and many other industries because of its low cost, error detection capability, and reliability. The CAN protocol does not include, however, security measures such as authentication [2].

Several recent research works demonstrate that attackers can access the CAN Bus using a variety of interfaces such as telematics and OBD-II units to inject messages into the CAN Bus. Hoppe et al. [4] were the first researchers to point out the security weaknesses of the CAN Bus protocol. Their findings were later confirmed by Checkoway et al. [5], who performed a security analysis of attack surfaces, including physical and long-range wireless communication, and demonstrated the exploitation of the flaws that they have identified to fully take control of the vehicle's systems. Recently, Upstream's research team identified 367 publicly reported incidents for a decade long [3]. The analysis of these incidents shows an exponential growth of attacks, as depicted by Figure 1.

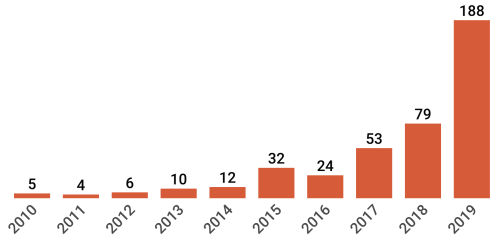

Fig. 1: Growth and distribution of cyber-attacks on connected vehicles between 2010 and 2019 [3].

Intrusion Detection Systems (IDSs) have been proposed as an alternative to the attack prevention approach on connected vehicles [6], [7]. For instance, Kim et al. proposed a backward-compatible CAN authentication system for CAN Bus that use the part of the CRC field of the CAN messages, which introduces delays in vehicle functionalities and hinder the error detection capability of the CAN bus [8]. Wu et al. [9] and Young et al. [10] provide comprehensive surveys on IDS for connected and autonomous cars. Most of the ML-based IDS for the connected vehicle are evaluated offline using datasets of CAN logs, including the ones that designed in [6], [7],

2693-9177/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS57517.2022.00091

TABLE I: Major requirements for IDS for vehicles.

	IDS Requirements
1	The response time of IDS must be small enough to trigger reactive safety mechanisms, such as braking.
2	The IDS must not lose CAN data.
3	The IDS must run on an ECUs that has limited capabilities in terms of processing speed and memory size.

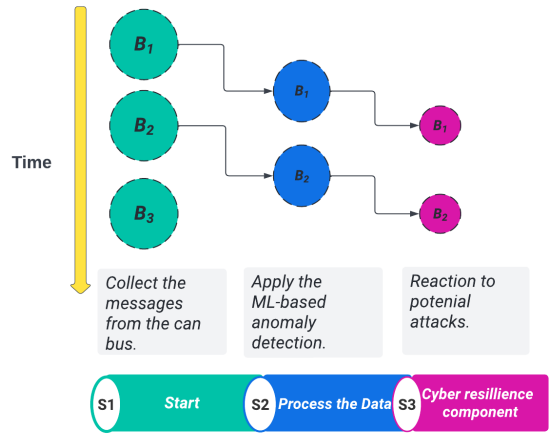

Fig. 2: Concurrency of the real-time IDS components in handling CAN Bus messages. The figure shows that message batches are processed in three stages. The components that implement the stages run concurrently.

which limits the confidence in the capabilities to use them for real-time IDS for vehicles.

As Figure 2 depicts, a *simulated real-time IDS* (1) collects CAN messages as they are injected into the in-vehicle network and (2) analyzes them using an ML-based techniques and reports the results to the cyber-resilience response component within specified time constraints, e.g., the time to perform (1) and (2) shall be less than the time a driver takes to react to danger. Besides the time constraints, an effective real-time IDS shall not lose data and run on an ECU that has limited memory and processing capabilities–see Table I. This paper analyzes the performance of four architecture alternatives for real-time IDS for connected vehicles.[1] It reports the assessment of their satisfaction of the requirements by evaluating their (1) anomaly evaluation time and (2) reliability in terms of losing CAN messages. The findings demonstrate the possibility of deploying effective ML-based IDSs for connected vehicles.

This paper is organized as follows. Section II describes the related works, Section III describes the architecture alternatives of the real-time IDS for connected vehicle, Section IV describes the evaluation of the architecture alternatives, and Section V concludes the paper.

II. Related work

Several preventive security countermeasures have been developed to defend and enhance in-vehicle network security against cyber-attacks, such as authentication protocols [11], [12], [13]. The main issue with these mechanisms is that these address only a subset of the attacks on the connected vehicles and require modification of the CAN protocol, which cannot be used for aftermarket vehicles. In addition, most of the remote attacks exploit software vulnerabilities in the protection mechanisms, such as in [14], [15], [16], [17].

IDSs have been proposed as an alternative to prevention mechanisms from attacks on connected vehicles. The concept of In-vehicle IDS was first introduced in [18], in which the characteristics of onboard intrusion detection are proposed for the first time. Wu et al. [9] and Young et al. [10] provide surveys on IDS for connected and autonomous cars. These mechanisms discriminate messages associated with attacks, with acceptable accuracy and false positive [19]. Neural Networks (NN) has been the commonly used ML-based approach for designing IDSs for the CAN bus, e.g., [20], [21], [22], [23].

Valasek and Miller are among the pioneer to propose real-time IDS for connected vehicles [24]. They developed a small device that reads data from the CAN Bus through OBD-II port, learns the traffic pattern to detect anomalies, and shorts the circuit disabling all CAN messages when anomalous detected. Matsumoto et al. [25] proposed a real-time IDS that prevents authorized messages from reaching the receiver ECU. The system monitors the traffic of the CAN Bus and transmits Error Frame to override the unauthorized messages when it detects them. The technique requires, however, modification of the CAN Bus protocol.

The design of security enhancement for vehicles needs to meet multiple design metrics, such as reliability and performance which conflict with the safety-critical requirement for vehicle communication. Boddupalli and

[1]We focus on CAN message injection attacks. Other attacks on connected vehicles, e.g., on V2V could be considered in the future.

TABLE II: IDS architecture constraints.

Constraint	Value
Recommended maximum rate of injection of CAN messages	1908/sec
Rate of injection of CAN messages	1000/sec [6]
Reaction time constraint	2.5 sec
Detection speed of 1000 messages using the similarity threshold technique	0.094 seconds

- The CAN bus is designated for a maximum signaling of 1 Mbits/s [28] but 250kb/s is the recommended rate by the Society of Automotive Engineers (SAE) in J1939 standards [29], which transports up to 1908 CAN frame per second – 1908=250000/(128+3) where the data payload is of 8 bytes.
- The brake reaction time is less than 2.5 second for 90% of the drivers.[30]
- We postulate that the IDS needs to process the CAN messages batch file in less than 2.5 seconds, which is the upper bound of braking reaction time [30].

Ray assessed the requirements for real-time attack detection and mitigation in connected and autonomous vehicles and emphasized the importance of the basic safety of such a mechanism. [26], [27]. They trained a NN-based IDS and proposed an architecture that addresses the requirements for the case of collision avoidance using vehicle-to-vehicle mechanisms. The main components of the architecture are: (1) a predictor of abnormal behavior that uses the trained IDS, (2) machine learning models for computing the application decision trained from driving a vehicle in different weather conditions (windy, raining, snowing, and clear) and road types (city, suburban, and highway) which are used to estimate the response of the module, and (3) a plausibility check module that checks the safety of using the output of the response estimator. When the system detects an anomaly, it replaces the response computed by the collision application with the output of the response estimator when the plausibility check is positive; that is, the estimated response is safe. The system triggers service degradation if it detects an anomaly and the plausibility checks of the output of the estimated response is negative.

III. Real-time architecture alternatives of IDS for Connected Vehicles

A. Problem description

In a typical IDS environment, a set of ECUs inject CAN messages into the CAN bus, a CAN Bus monitor captures the messages exchanged in the bus, and an anomaly detection engine analyzes these messages to identify malicious message injections, as depicted in Figure 3. The *CAN bus monitor* reads the messages available in the CAN bus continuously and sends them to the anomaly detection engine in batches of, e.g., 1000 messages.

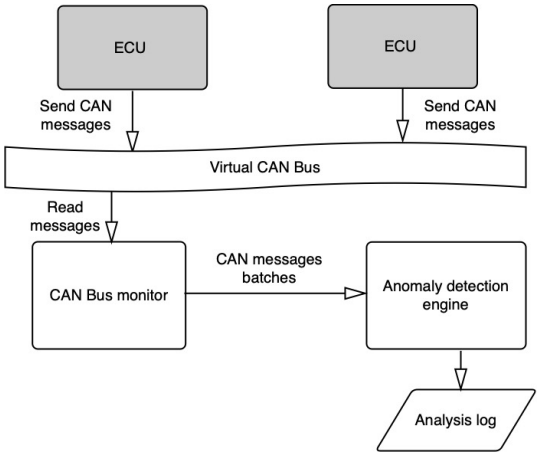

Fig. 3: High-level architecture of the IDS simulation environment.

The real-time IDS must address three main requirements depicted in Table I. First, the response time of the IDS must be less than the expected braking reaction time; satisfying this requirement makes the IDS a good candidate for e.g., a safety resilience mechanism that activates the brakes in case of detection of cyber-attacks. Second, the loss of CAN messages is not allowed, which is important for the reliability of IDS. Third, The IDS must run on a cheap ECU that has limited capabilities in terms of processor speed and memory size. In addition, Table II enumerates a set of constraints that the architecture shall satisfy.

Formally, let B be the component of injecting CAN messages onto the CAN Bus (which represents the ECUs of the given vehicle), C be the CAN Bus monitoring component, P be anomalous detection engine the component that applies machine learning on the messages it receives from C to detect attacks, and R be the cyber-resilience response that activates the mitigation actions, such as, breaking.

$$\forall B(m_{ij}) \longrightarrow C(m_{ij}) \tag{1}$$

Equation 1 states that C captures all the messages injected by B. Note that B and C communicate through

TABLE III: Practical concurrency scenarios of the IDS main components.

Architecture scenario	Concurrency technique of CAN Bus monitor	Concurrency technique of the anomaly detection agent	Use of a queue
Scenario 1 - The two components run in a single process	main process	main process	no
Scenario 2 -The two components run in a single process that includes one subprocess s	main process	child-process	no
Scenario 3 - The two components run in a single process with two threads	thread	thread	yes
Scenario 4 - The two components run in two processes	main process	main process	yes

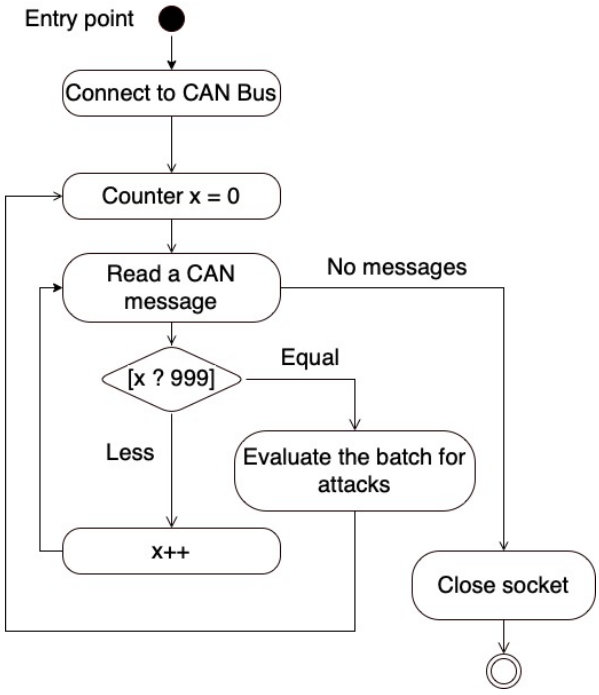

Fig. 4: IDS flowchart diagram.

the CAN Bus, which does not have a buffer.[2] Component C sends the messages it receives from B to P in blocks of n messages. Let m_{ij} be a CAN message that has the order j in messages block i, where i indicates the order of messages blocks sent by C to P. Equation 1 states that C must receive all the messages that B injects in the CAN Bus; allowing losses of CAN messages implies preventing potentially the ML-based IDS from indicators of attacks.

Component P serves messages blocks in sequence as stated by Equation 2. Equation 3 states that C collects a block of n messages and then sends them to component P, which applies the ML-based IDS and outputs whether it detects an attack or not. Equation 4 states that Component R uses the output of P to trigger a cyber-resilience response action if needed. The behavior of component R is outside the scope of this paper.

$$P(B_i) \longrightarrow P(B_{i+1}) \tag{2}$$

$$C(B_i) \longrightarrow P(B_i) \tag{3}$$

$$P(B_i) \longrightarrow R(B_i) \tag{4}$$

Typically, each ECU services/reacts to each message it reads from the CAN bus (process or ignore) at a rate higher than the sending messages rate, to avoid loss of messages. In contrast, the ML-based IDS techniques process the CAN messages in batches, which takes much longer than the time to send a CAN message. Thus, the IDS would lose data if the CAN bus monitor and anomaly detection engine operate sequentially, which violates the third requirements that we set for our real-time IDS: loss of CAN messages is not allowed. This problem could potentially be addressed by running the CAN Bus monitor and anomaly detection engine concurrently. Thus, the solution is to support the execution of B, C P, and R in concurrent processes as stated by Equation 5.

$$B|||C|||P|||R \tag{5}$$

[2]That is, let B injects a message m in the Bus. The message m is lost if B starts sending the next message $m+1$ before C reads m.

Commonly, a driver is expected to reach out to the brakes in 0.75 to 2.5 seconds–assuming they respect the safety distance rules. Assume that we want the system to trigger the brakes to avoid accidents when there are attacks. The real-time IDS can only be of use if it detects attacks within the reaction time, otherwise the accident is imminent. Let $T(C_i)$ be the time to collect the block of n messages i and $T(P_i)$ be the time to process the messages block i. The requirement is: $T(C_i) + T(P_i)$ must be less than the reaction-time threshold for the IDS to be as good as a human driver in reacting to dangers.

The research question is: *What is the best concurrency architecture alternative of deploying CAN Bus monitor (component C) and the anomaly detection engine (Component P) to provide real-time IDS for vehicles?*

Note that the three concurrency techniques that could be used are: using separate processes, using sub-processes, and using threads.

We describe in the following the experiment that we setup to answer the question empirically.

B. Empirical Setup

The *anomaly detection engine* applies the adopted ML-based anomaly detection technique and outputs the results of the evaluation. Jedh et al. developed an offline Machine Learning (ML)-based IDSs and evaluated them using CAN data extracted from a moving vehicle under malicious RPM and speed readings messages injections into the in-vehicle network of the vehicles [6], [7]. The technique constructs a Messages-Sequence Graph (MSG) from the CAN messages it receives from the CAN Bus monitor, computes the cosine similarity of the successive graphs, and reports message injection when one is detected. The technique has a detection accuracy of 97.32% and a detection speed of 2.5 milliseconds. Appendix A describes the technique in more details. Unlike most ML-based studies in the literature, the technique proposed in [7] does not depend on the brand or model of the car. We adopt this IDS in our evaluation because it has good performance and we have access to the implementation code and datasets.

To satisfy the first requirement, we implement the CAN Bus monitor and the anomaly detection engine using the C language. The anomaly detection engine uses PyObject library to call the data analysis module, passing it the CAN messages batches as a parameter, and parsing the analysis result.

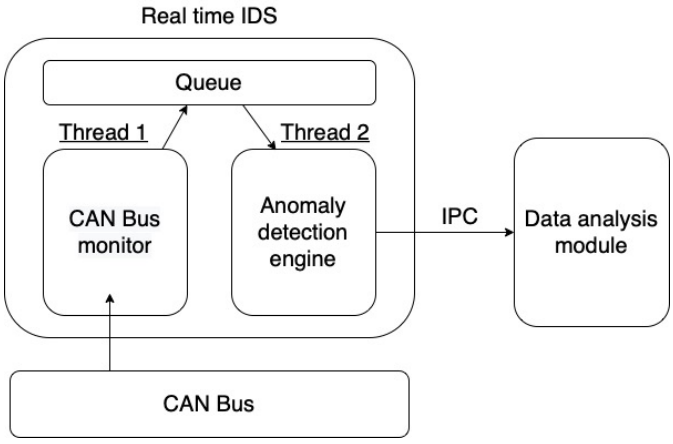

Fig. 5: Diagram of the IDS architecture of scenario 3.

Table III shows the four potential architecture scenarios for using concurrency techniques of the CAN Bus monitor and the anomaly detection engine components to address the second and third requirements discussed above. The descriptions of the architecture scenarios follows.

Scenario 1 - The two components run in a single process. In this scenario, the CAN Bus monitor and anomaly detection engine run sequentially as depicted by Figure 4. The CAN Bus monitor can potentially lose CAN messages that the ECUs send while the IDS busy analyzing the CAN messages batch it receives to identify potential attacks.

Scenario 2 - The two components run in a single process that includes one sub-process. The CAN bus monitor and the anomaly detection engine run in a single process. The main process continuously reads the CAN messages from the CAN bus and creates a sub-process, that evaluates the batch for attacks, that is executed when there are enough messages for the batch. Note that the sub-processes become zombies at the end of their executions, and it is complicated to shut them down from the main processes.

Scenario 3 - The two components run in a single process with two threads. The CAN bus monitor and the anomaly detection agent run in a single process but in separate threads, as depicted by Figure 5. We use a queue to pass data between the two components to prevent losses of CAN messages. The anomaly detection

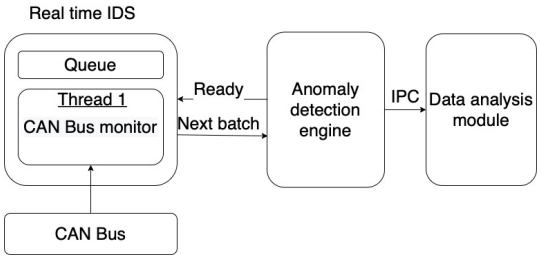

Fig. 6: Diagram of the IDS architecture of scenario 4.

engine uses Inter-Process Communication (IPC) technique to execute the data analysis module.

For completeness, there are two other variants of using threads with a single process besides the one discussed above, which are: (1) implementing the CAN bus monitor in a thread and the anomaly detection engine in the main process and (2) implementing the anomaly detection engine in a thread and the CAN bus monitor in the main process. We do not report the evaluation of these variants because they use the same concurrency technique and do not outperform the two threads variant.

Scenario 4 - The two components run in two processes. The CAN bus monitor and the anomaly detection agent run in separate processes, as depicted by Figure 6. We use queue to pass data between the two components to reduce the losses of CAN messages.

IV. Evaluation of the four real-time IDS architecture scenarios

This section describes the evaluation environment and datasets, compares the anomaly evaluation times of the four architecture scenarios, and analyses the impact of CAN message injections on the message loss ratio of the four architecture scenarios.

A. Evaluation environment and datasets

We implemented the four architecture scenarios and deployed them to a Raspberry Pi that runs Raspbian 10, with four core processors of 1.2 MHz speed and 1GB of memory. Figure 3 shows the evaluation environment of the architecture scenarios. The environment uses an *ECUs emulator* that emulates car ECUs, which sends messages periodically into Linux virtual CAN bus. The code of the evaluation is available at [31].

TABLE IV: Speed of simulating injection of CAN messages and evaluating them for attacks using architecture scenario 1.

	Normal messages	messages with injection of speed readings	messages with injection of RPM readings
Time to send 1000 CAN messages in seconds			
Min	0.946	0.925	1.015
Max	1.120	1.217	1.056
Average	0.992	1.008	1.023
Time to evaluate 1000 CAN messages in seconds			
Min	0.115	0.116	0.131
Max	0.271	0.274	0.180
Average	0.154	0.143	0.151

In this evaluation, we use datasets [32] of CAN bus messages for (1) normal driving behavior, (2) injection of fabricated speed reading messages onto the CAN bus, and (3) injection of fabricated RPM reading messages onto the CAN bus collected from an in-motion Ford Transit 500 2017 [6].

The *ECUs emulator* reads the CAN messages stored in the dataset files and sends them through the virtual CAN bus. The messages are processed by the CAN bus monitor and anomaly detection engine in the four architecture scenarios. Table IV provides the time that the ECUs emulator[3] takes to send 1000 messages into the CAN Bus and the time that the IDS takes to evaluate 1000 CAN messages for the cases of normal messages dataset[4], messages with the injection of speed readings dataset, and messages with the injection of RPM readings dataset. We do not observe a big difference in processing a batch of normal CAN messages, messages with the injection of speed readings, and messages with the injection of RPM readings. We observe that the IDS takes an average 149 milliseconds to evaluate a batch of 1000 messages, while the time to send 1000 messages into the CAN bus is about 1.007 seconds. The IDS would lose about 149 CAN messages from each batch, which can impact the attack detection rate–i.e., ignoring 14.9% of the messages can impact the detection rate.

[3]The environment emulates the vehicle's ECUs network.
[4]An analysis of the messages batch size threshold is at [33].

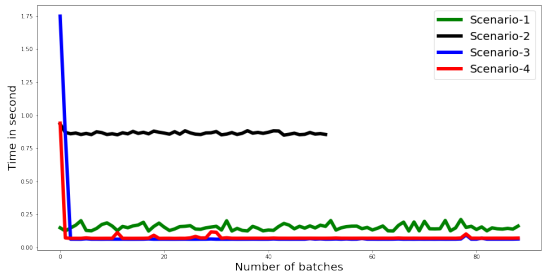

Fig. 7: Anomaly evaluation time of the four architecture scenarios.

TABLE V: Anomaly evaluation times and response times for the four architecture scenarios.

Architecture	Average time of sending 1000 CAN messages	Average evaluation time	Response time
Scenario 1 - Single process with no threads	998 ms	152 ms	1.15 sec.
Scenario 2 - Single process with one sub-process	944 ms	865 ms	1.809 sec
Scenario 3 - Single process with two threads	950 ms	90 ms	1.04 sec.
Scenario 4 - Two processes	945 ms	81 ms	1.026 sec.

B. Analysis of the anomaly evaluation time of the four architecture scenarios

We configured the ECU emulator to send the dataset "CAN Data with injection of "FFF" as the speed reading" with a rate of about 1000 messages through the virtual CAN Bus. Table V shows the average time of sending a batch of CAN messages, the average evaluation time, and the average response time (time taken from collecting the first CAN message of the batch to output the result of the evaluation of the batch) of the four IDS architecture scenarios. We consider the *anomaly evaluation time* as the difference between the end of evaluating the messages batch for attacks and the end of reading 1000 CAN messages from the virtual CAN Bus. The data shows that anomaly evaluation time is way below the batch messages sending time. Figure 7 shows the anomaly evaluation time of the four selected real-time IDS architecture scenarios. The diagram shows that

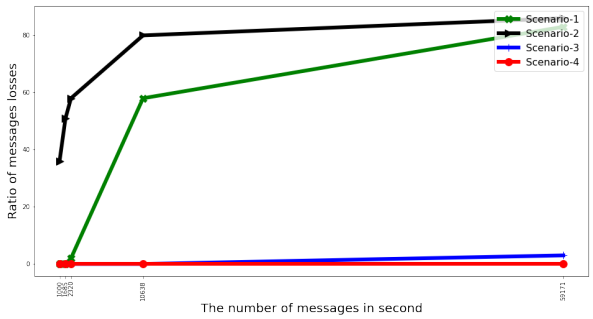

Fig. 8: Ratio of messages losses vs number of messages in second of sending 1000 CAN messages through the CAN bus in the four architecture scenarios.

anomaly evaluation times of a real-time IDS designed as a single process, single process with two threads, and as two processes are below the brakes reaction time (0.75 to 2.5 seconds), In addition, the anomaly evaluation times of the IDSs designed as a single process with two threads and as two processes are too close. Furthermore, scenario 2 shows the highest loss of messages compared to other scenarios. For each batch scenario 2, creates a sub-process that analyzes the batch using the machine learning algorithm, which increases the overhead for computation (switching from one process to another is usually time-consuming). During our evaluation we didn't address the problem of memory.

We conclude that the IDS response time of a real-time IDS designed as a single process with no threads, as a single process with two threads, and as two processes is below the brake reaction time (assuming the rate of sending messages through the CAN Bus is about 1000 messages/second) which makes them a good candidate for real-time IDS for connected vehicles.

C. Analysis of the reliability of the four architecture scenarios in terms of CAN message losses

Message loss ratio is the ratio of CAN messages that are sent through the CAN Bus by the ECUs emulator but are not received and processed by the anomaly detection engine. Theoretically, the CAN messages that the ECUs emulator sends through the CAN Bus while the single process IDS architecture is busy analyzing a batch of previously received messages are lost. Figure 8 shows the relationship between the messages loss ratio

and speed of sending messages into the virtual CAN bus by the ECUs emulator. The figure shows that the ratio of messages loss decreases as the rate of sending CAN messages decreases and it reaches 0% for the case of architecture scenario 1 and 2. It also shows that there are no messages losses for the case of IDS designed as two processes (architecture scenario 4) but using threading (as in architecture scenario 3) does not ensure no loss of messages, as it may be expected.

D. Generalization of the result

We reported in this paper on building a IDS for connected vehicles using the similarity of messages precedence graphs [7] and found that it complies with the real-time IDS properties. Besides, we reported two other results.

The first result is that the IDS response time of a real-time IDS designed as a single process with no threads, as a single process with two threads, and as two processes is below the brake reaction time, which makes them a good candidate for real-time IDS for connected vehicles is indeed dependent on the ML-based IDS that we have previously developed [7] and the use of a Raspberry Pi that runs Raspbian 10, with four core processors of 1.2 MHz speed and 1GB of memory, although we did not use parallelism. The average batch evaluation time of the anomaly detection engine can be higher than the one we report if an MCU with very low processing capability is used, which could make our IDS not suitable for real-time IDS for vehicles. In addition, other ML-based IDS may not be suitable for real-time IDS for connected vehicles if their average evaluation time of batches is high, e.g., higher than 1.5 seconds/1000 messages.

The second result is that there are no message losses for the case of IDS designed as two processes, but message losses are possible when threads are used to run the anomaly detection engine component when the rate of CAN messages is very high. This is a consequence of the overhead of starting a process versus the overhead of starting a thread. A two-process model of concurrency for the CAN Bus data collection and ML-Based Anomaly Detection engine should ensure no loss of messages as long as the rate of CAN messages that the Anomaly Detection Engine processes is higher than the rate of messages that the CAN Bus data collection component collects. Increasing the number of threads increases the required memory and processing needs.

TABLE VI: A comparison of IDS performance in offline, online, and simulated

Criteria	Offline	Online	Simulated
Dataset	Simulated	Real-time	Simulated real-time
Network	No network	Real	Simulated
Detection rate	0.97%	0.97%	0.97%
Detection latency	2.5ms	152 ms	152 ms
Response time	Indefinite	Unknown	1026 ms
ML testing environment	Typically, a laptop	Raspberry Pi	Raspberry Pi

Table VI shows a cross-comparison between offline, online, and simulated scenarios. In an offline scenario, typically, a laptop with high computing power but doesn't learn the traffic behavior and new attacks style.

The research is performed with one IDS and one dataset. A thorough evaluation using a set of other IDS, datasets, and hardware types could be performed to gain more confidence in the results.

V. Conclusion

ML-based IDSs techniques have been proposed for the detection of malicious injection of messages into the in-vehicle network of connected vehicles. The evaluations of such IDS have been performed offline, which limits the confidence in their use for real-time scenarios. We evaluated in this paper four concurrency architecture designs for real-time IDS for connected vehicles using CAN datasets collected from a moving vehicle under malicious message injections. The evaluation shows that a real-time IDS for a connected vehicle designed as two processes are reliable (no loss of messages) and have a low anomaly evaluation time that makes them a good candidate for real-time resilience mechanisms.

References

[1] R. Bosch GmbH, "Can specification v2.0." http://esd.cs.ucr.edu/webres/can20.pdf, 1991.

[2] L. Ben Othmane, H. Weffers, M. M. Mohamad, and M. Wolf, *Wireless Sensor and Mobile Ad-Hoc Networks: Vehicular and Space Applications*, ch. A Survey of Security and Privacy in Connected Vehicles, pp. 217–247. Springer, 2015.

[3] Upstream Auto, "Upstream security's global automotive cybersecurity report." https://www.upstream.auto/upstream-security-global-automotive-cybersecurity-report-2020/, 2020. accessed on Feb 2020.

[4] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks – practical examples and selected short-term countermeasures," in *Computer Safety, Reliability, and Security* (M. D. Harrison and M.-A. Sujan, eds.), (Berlin, Heidelberg), pp. 235–248, Springer Berlin Heidelberg, 2008.

[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, SEC'11, (USA), p. 6, USENIX Association, 2011.

[6] L. b. Othmane, L. Dhulipala, M. Abdelkhalek, N. Multari, and M. Govindarasu, "On the performance of detecting injection of fabricated messages into the can bus," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 468–481, 2022.

[7] M. Jedh, L. Ben Othmane, N. Ahmed, and B. Bhargava, "Detection of message injection attacks onto the can bus using similarities of successive messages-sequence graphs," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4133–4146, 2021.

[8] S. Kim, G. Yeo, T. Kim, J. J. Rhee, Y. Jeon, A. Bianchi, D. Xu, and D. J. Tian, "Shadowauth: Backward-compatible automatic can authentication for legacy ecus," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, (New York, NY, USA), p. 534–545, Association for Computing Machinery, 2022.

[9] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, and K. Li, "A survey of intrusion detection for in-vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2020.

[10] 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, 2019.

[11] G. Carel, R. Isshiki, T. Kusaka, Y. Nogami, and S. Araki, "Design of a message authentication protocol for can fd based on chaskey lightweight mac," in *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 267–271, 2018.

[12] H. J. Jo, J. H. Kim, H.-Y. Choi, W. Choi, D. H. Lee, and I. Lee, "Mauth-can: Masquerade-attack-proof authentication for in-vehicle networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2204–2218, 2020.

[13] W. A. Farag, "Cantrack: Enhancing automotive can bus security using intuitive encryption algorithms," in *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, pp. 1–5, 2017.

[14] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle." https://www.youtube.com/watch?v=OobLb1McxnI, 2015.

[15] "Hacking tesla from wireless to can bus." https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf. accessed on Dec. 2021.

[16] "Experimental security assessment of bmw cars: A summary repor." https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Assessment_of_BMW_Cars_by_KeenLab.pdf. accessed in Dec. 2021.

[17] A. Greenberg, *Hackers Can Steal a Tesla Model S in Seconds by Cloning Its Key Fob.* wired., Sep 2018.

[18] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks–practical examples and selected short-term countermeasures," *Reliability Engineering and System Safety*, vol. 96, no. 1, pp. 11 – 25, 2011. Special Issue on Safecomp 2008.

[19] M. J. Kang and J. W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PLOS ONE*, vol. 11, pp. 1–17, 06 2016.

[20] F. Lokman, A. T. Othman, and M. H. Abu Bakar, "Intrusion detection system for automotive controller area network (can) bus system: a review," *EURASIP Journal on Wireless Communications and Networking*, vol. 184, no. 1, 2019.

[21] F. Lokman, A. T. Bin Othman, and M. Abu Bakar, "Optimised structure of convolutional neural networks for controller area network classification," in *2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pp. 475–481, July 2018.

[22] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, and S. Zanero, "Cannolo: An anomaly detection system based on lstm autoencoders for controller area network," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020.

[23] S. Stachowski, R. Gaynier, and D. J. LeBlanc, "An assessment method for automotive intrusion detection system performance." https://rosap.ntl.bts.gov/view/dot/41006, April 2019.

[24] C. Valasek and C. Miller, *A Survey of Remote Automotive Attack Surfaces.* Black Hat USA 2014, Aug 2014.

[25] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi, "A method of preventing unauthorized data transmission in controller area network," in *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, 2012.

[26] S. Boddupalli and S. Ray, "Redem: Real-time detection and mitigation of communication attacks in connected autonomous vehicle applications," *IFIP advances in information and communication technology*, vol. 574, pp. 105–122.

[27] S. Boddupalli, A. S. Rao, and S. Ray, "Resilient cooperative adaptive cruise control for autonomous vehicles using machine learning," 2021.

[28] Texas instrument, "Introduction to the controller area network (can)." https://www.ti.com/lit/an/sloa101b/sloa101b.pdf. Application Report SLOA101B–August 2002–Revised May 2016.

[29] Society of Automotive Engineers, "J1939 recommended practice for a serial control and communications vehicle network." https://www.sae.org/publications/collections/content/j1939_dl/.

[30] "Transportation engineering online lab manual - brake reaction time." https://www.webpages.uidaho.edu/niatt_labmanual/chapters/geometricdesign/theoryandconcepts/BrakeReactionTime.htm. accessed on Dec 2021.

[31] "Real-time-ids-scenarios." https://github.com/lbenothmane/Real-time-IDS-Scenarios.

[32] L. Ben Othmane and L. Dhulipala, "Injection of rpm and speed reading messages onto the can bus of a moving vehicle." https://dx.doi.org/10.21227/s1jy-h433, 2020. IEEE Dataport.

[33] "Using messages precedence similarity to detect message injection in in-vehicle network." https://lib.dr.iastate.edu/creativecomponents/651/, 2020.

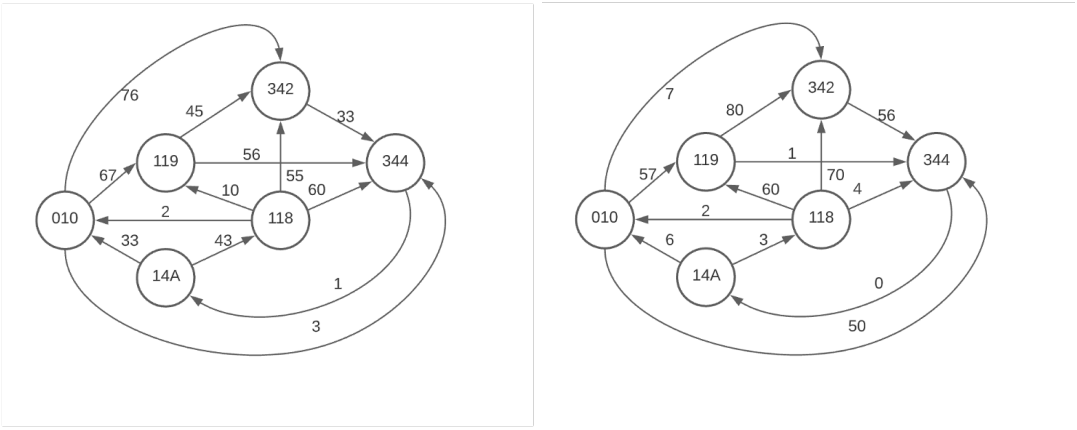

Fig. 9: Similarity of two messages-sequence graphs at successive time-window t (left) and $t+1$ (right). The labels of the nodes are the CAN ID and the labels of the edges are number of times a message with the CAN ID source of the edge is followed by a message with the CAN ID destination of the same edge during the time-window. For example, 33 messages and 56 messages with ID 344 followed messages with ID 342 at resp. time-windows t and $t+1$, which indicate a possible change of the behavior of the vehicle [7].

Appendix

ECUs collaborate to perform tasks such as increasing speed, braking, etc., by sending messages through the CAN bus [6]. Attackers take control of a connected vehicle by injecting messages into its CAN bus. To mitigate such attacks, Jedh et al. developed an ML-based IDS that captures the pattern of the sequences of the CAN messages and represent them with a directed graph, which they call *Messages-Sequence Graph (MSG)*, where the nodes represent the CAN IDs of the messages and the edges represent the sequences order of the messages, as depicted by Figure 9 [7].

To construct the MSG, the technique first creates a dictionary of the CAN IDs exchanged in the CAN bus. Then, it labels the nodes of the MSG with the CAN IDs and the edges with value "0". Next, it loops over the batch of the CAN messages of size w (e.g., 1000 successive messages) that were exchanged from time t. For each of the messages, it increases the label of the edge linking the node representing the CAN ID of the message to the node repressing the CAN ID of the previously processed message. Equation 6 represents the distribution of the messages-sequences at time t.

$$D(t) = \{E(N_i, N_j)(t)\} \tag{6}$$

where N_k is for node k and $E(a, b)$ is for the edge from node a to node b.

The authors consider that a MSG representing the w messages exchanged in a CAN bus at time t is *similar* to the MSG representing the w CAN messages exchanged during the following time slot $t+1$ in the case of normal driving behavior and that injection of messages into the CAN bus disrupts this pattern [7]–see Figure 9 [7]. Equation 7 formulates the *Similarity* concept for the IDS. That is, the similarity *Sim* at time *t+1* is the similarity of the distributions of the messages-sequences D at time t and at time *t+1*.

$$Sim(t+1) = Similarity(D(t), D(t+1)) \tag{7}$$

The technique uses the cosine similarity to measure the similarity between two MSGs of two successive time steps t and $t+1$. The metric measures the angle between two vectors, where the closer the value is to 1, the more similar the two vectors are. Then, it uses *similarity threshold* technique to identify CAN message injections. The technique provides an accuracy of 97.32%.