01 Jan 2022

Cansafe: An Mtd based Approach for Providing Resiliency Against Dos Attack within In-Vehicle Networks

Ayan Roy

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

Recommended Citation

A. Roy and S. K. Madria, "Cansafe: An Mtd based Approach for Providing Resiliency Against Dos Attack within In-Vehicle Networks," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pp. 3243 - 3250, Institute of Electrical and Electronics Engineers, Jan 2022.
The definitive version is available at https://doi.org/10.1109/ITSC55140.2022.9922025

2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)
October 8-12, 2022, Macau, China

CanSafe: An MTD based approach for providing resiliency against DoS attack within in-vehicle networks

Ayan Roy
Department of Computer Science
Missouri University of Science and Technology
USA
Email: ar3g3@mst.edu

Sanjay Kumar Madria
Department of Computer Science
Missouri University of Science and Technology
USA
Email: madrias@mst.edu

***Abstract*— Trending towards autonomous transportation systems, modern vehicles are equipped with hundreds of sensors and actuators that increase the intelligence of the vehicles with a higher level of autonomy, as well as facilitate increased communication with entities outside the in-vehicle network.However, increase in a contact point with the outside world has exposed the controller area network (CAN) of a vehicle to remote security vulnerabilities. In particular, an attacker can inject fake high priority messages within the CAN through the contact points, while preventing legitimate messages from controlling the CAN (Denial-of-Service (DoS) attack). In this paper, we propose a Moving Target Defense (MTD) based mechanism to provide resiliency against DoS attack, where we shuffle the message priorities at different communication cycles, opposed to the state-of-the-art message priority setup, to nullify the attacker's knowledge of message priorities for a given time. The performance and efficacy of the proposed shuffling algorithm has been analyzed under different configuration, and compared against the state-of-the-art solutions. It is observed that the proposed mechanism is successful in denying DoS attack when the attacker is able to bypass preemptive strategies and inject messages within the in-vehicle network.**

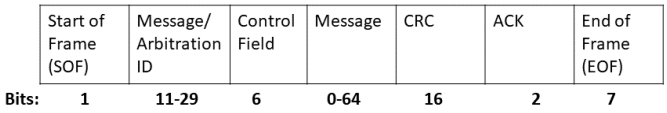


Fig. 1: General Structure of CAN message.

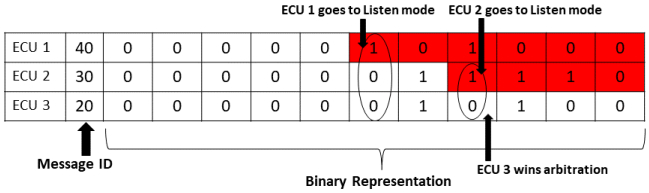


Fig. 2: Message Arbitration mechanism between message IDs 40,20 and 30. Red box indicates that the ECU goes to listening mode as it has a higher ID than the other messages in the arbitration.

I. INTRODUCTION

Modern-day vehicles are equipped with hundreds of electronic control units (ECUs) that manage different functionality of the vehicles, such as the brakes, steering, and throttle. The ECUs within a vehicle form a network called the in-vehicle network (IVN), where they share different kinds of information that determines the state of the vehicle or the next action of the vehicle for a given state. Multiple IVNs are known for in-vehicle communications, such as controller area network (CAN), local interconnected network (LIN), and FlexRay. However, CAN is widely accepted as a defacto standard for inter-vehicle communication due to its stability and cost-effectiveness [1]. CAN is a serial communication system where multiple ECUs connected to it share the same physical communication bus. As a result, to avoid message collision, CAN specify an inverse relationship between the message IDs (a CAN message structure is shown in Fig 1) and the message priorities. Thus, at any given time if more than one ECU participates in the message arbitration, the ECU with the lowest message ID wins the arbitration and sends the message through the CAN (as illustrated in Fig 2). Meanwhile, the other ECUs switch to listening mode and wait for the next arbitration, thus providing a non-destructive way of message communication. Furthermore, the modern vehicles are connected to various other entities outside the in-vehicle environment using Bluetooth and 3G/4G networks, such as through the infotainment system [2]. Such remote communication with the outside world aims to enhance the driver's comfort and safety, as well as facilitates in keeping the various software components of the vehicle up-to-date. regular over-the-air (OTA) updates.

However, CAN is highly vulnerable to remote security attacks, as it does not have any authentication mechanism in place. As a result, it is not difficult for an attacker to connect an external device and snoop into the data packets of the CAN. Furthermore, it can reverse engineer the information collected [3] (more commonly called reconnaissance) and inject information within the CAN. The practicability of remote injection inside CAN has been demonstrated in [4], in which the authors were able to control the braking and steering mechanism of a vehicle through remote injection of data packets. A remote attacker can exploit the strict relationship specification between the message ID and the priority in CAN and effectively perform a DoS attack by always injecting a message with the lowest ID at every time interval, thereby gaining control of the CAN at every time. This can deny legitimate ECUs from sending important information to other ECUs.

Moving Target Defense (MTD) is a potential security

2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC) | 978-1-6654-6880-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ITSC55140.2022.9922025

solution to many static systems [5], in which the system parameters are constantly changed to nullify the information gathered during the reconnaissance phase of an attacker. The primary steps for MTD based techniques involve: **What** to dynamically change in the system configurations, **When** to make the change into the new state from the current state, and **How** to change the moving attributes to increase the unpredictability of the attack surface. MTD-based security techniques involve using an existing mechanism that is triggered either during certain time intervals (time-based) or when certain events occur (event-based).

In this paper, we propose an MTD-based solution to prevent the DoS attack within the in-vehicle network. We reshuffle the relationship between the message/arbitration ID generated in a separate module, referred to as the Shuffling Module, and their priorities at every instant of time to confuse the attacker about the message priorities (as opposed to the specification of the CAN bus). As the specifications of the CAN bus are static, a separate security module/hardware (refer to as SMOD throughout the paper) is needed to handles the message arbitration phase when one or more ECUs wants to send information through the CAN. The detailed specification of SMOD is outside the scope of this paper. However, once completed, the message is sent from the SMOD to the CAN bus. The primary contribution of the paper is as follows:

- We propose a new technique that does not necessarily follow a strict relationship between the arbitration ID and their priorities as described in the CAN specifications. The proposed shuffling algorithm is executed at the Shuffling Module that generates a new arbitration ID for every requesting ECU at a given communication cycle while keeping the relationship between the priorities of the ECUs intact as the original CAN specification.
- We provide a detailed analysis of the proposed shuffling algorithm under different threats from an attacker. Furthermore, we also compare the various scenarios of the proposed mechanism under different arrangement of the parameters (such as how often to perform the shuffling operation) to determine the impact of the shuffling on the overall in-vehicle network.

II. RELATED WORK

Over the past few years, a significant amount of research has been devoted to the security of the CAN in general.

Pre-emptive solutions aims to prevent any attack into the CAN of a vehicle.For instance, the authors in [6] have analyzed the detection performance of Pearson correlation, k-means clustering and Hidden Markov Model in an in-motion vehicle with injected speed and revolutions per minute (RPM) readings. The authors concluded that the use of fabricated (or simulated) dataset incorrectly indicates the performance of an intrusion detection technique. The authors in [7][8][9] have proposed different supervised/unsupervised machine learning solutions for detecting intrusion as a pre-emptive measure within the CAN.

Various authors have analyzed the feasibility of graph based solutions [10][11] for detection injection attacks. However, the proposed solutions relies heavily on an assumption that the normal driving behavior remains constant and any abnormal driving behavior (which may not be injected but different driving condition) is an attacked state. Many researchers [12][13] have leveraged the frequency of arrival of messages in CAN as a potential pre-emptive solution where they assume that every message within the CAN arrive after periodic intervals, that is unknown to the attacker. However, as the behavior of the ECUs (or the messages originating from the ECUs) is quite inconsistent with respect to their arrival time, the efficacy of the proposed models reduces. Furthermore, the efficacy of the model reduces if the attacker is able to mimic the original message frequency.

An IP-shuffling based MTD defense mechanism has been introduced by the authors in [14] where the authors dynamically changes the default IP address of the ECUs to introduce uncertainty or nullify any reconnaissance of an attacker. However, it is to be noted that the proposed approach serves the purpose of preserving the confidentiality of the ECU from the attacker but does not stop the attacker from performing a DoS (noted by authors also). The authors in [15] have constructed a deterministic finite automation (DFA) for the bit-by-bit arbitration process where a firewall deployed within the system rejects any fake ID injected by an attacker if it is not contained within the whitelist of the ECU IDs maintained inside the firewall. However, the efficacy of the proposed DFA fails if an attacker gathers information through CAN sniffing to reverse engineer and gather some ECU IDs from the whitelist as the IDs remain fixed. Such a captured ID when replayed in separate communication cycle, will be considered as an authentic state by the proposed DFA.

To summarize, most of the preemption mechanism can be used effectively to deny message injection within the network. However, many solutions proposed are more content-based, meaning that it relies on the content of the message to detect if the message is an injected message or not. Such solutions are considered reactive mechanism, which may serve the purpose of detecting an injection attack. but does not guarantee resiliency against DoS attack. This is because for injecting a message, as per the CAN specification, the attacker has to win the arbitration among the contending ECUs in a communication cycle. Thus, if the attacker is able to inject the payload, it means that it has successfully won the arbitration and prevented other authenticate ECUs from sending information during that communication cycle.

III. PRELIMINARIES, ASSUMPTIONS, AND THREATS

A. Preliminaries

- Shuffling Module : The purpose of the Shuffling Module in the proposed mechanism is to randomly generate the IDs corresponding to different ECUs at different communication cycle. The shuffling algorithm leverages the network parameters (elaborated in section IV.A) which are loaded onto it during the bootstrapping phase. The Shuffling Module works independently of any other

module. This means that the Priority List generated by the Shuffling Module need not be synchronized with any other module within the vehicle. The Shuffling Module can be implemented as separate hardware module that can be installed in vehicles, the details of which is outside the scope of this paper.

- SMOD : SMOD is a separate module that performs the arbitration mechanism instead of the CAN. The SMOD is connected to the Shuffling Module, from which it receives the updated priority list during every communication cycle. The priority list received is leveraged to perform the arbitration process of the different ECUs in contention. Furthermore, the SMOD is connected to the CAN that relays the dominant bits obtained from the ECUs and broadcast into the entire in-vehicle network. In other words, SMOD acts as an interface between the ECUs and the CAN for sending the bits.

B. Assumptions

- No insider attack: This means that none of the ECUs within the vehicle is compromised/manipulated.
- Safe Network Shuffling Parameters: The network shuffling parameters used for shuffling the mssage priorities is not disclosed to the attacker through any ECUs within the vehicle or the car manufacturing company.
- Untampered Security Module: The proposed security module introduced/installed within the in-vehicle network cannot be tampered. Furthermore, the shuffling technique is used as a black box within the ECU and the SMOD that is used to generate the new IDs of the ECUs while preserving the relationship between the message priorities originating from the different ECUs.
- No remote connection: Only ECUs and SMOD can connect to the Shuffling Module. Attacker remotely cannot access the Shuffling Module.
- Single Injection Attack: We assume that at any given communication cycle, only one bit can be remotely injected for different arbitration ID bit positions during the arbitration mechanism. In other words, any one message ID can be injected in the CAN for the arbitration mechanism in a given communication cycle.

C. Threats

- Denial of Service: The intention of an attacker is to generate the highest priority message at any given communication cycle to take control of the CAN and deny other legitimate ECUs from sending vital information. The attacker wants to disrupt the normal functioning of the vehicles by forcing the ECU to transit to listening mode by injecting high priority message at every communication cycle.
- Message Snooping: The attacker can snoop into the dominant bits from the CAN at every communication cycle. This is leveraged for the reconnaissance phase, where the attacker gathers the highest arbitration ID at each communication cycle and perform reverse engineering to determine the highest priority arbitration ID at the next communication cycle. The attacker can also perform replay attack by injecting the highest arbitration ID gathered at one communication cycle in a different cycle. The attacker can also snoop into the message payload sent by an ECU. However, such an attack is beyond the scope of this paper.
- Replay Attack: The attacker can replay the information gathered by snooping the CAN (i.e. the winning arbitration ID) from one communication cycle in other subsequent communication cycles. The replay attack in our threat model does not involve replaying the payload captured from an ECU by an attacker.

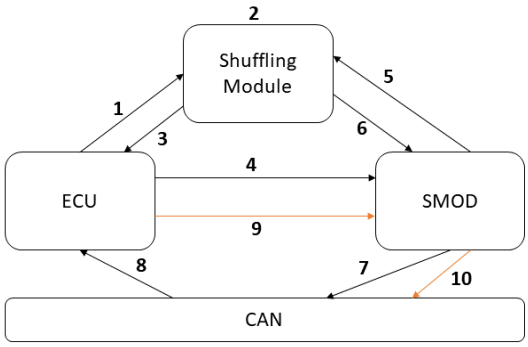


Fig. 3: Workflow: **1)** ECU requests for new ID at timestamp, t, **2)** Shuffling Module generates the new IDs of all the ECUs using the shuffling algorithm , **3)** Shuffling Module sends the new ID corresponding to the requested ECU ID, **4)** ECU sends the arbitration ID (new ID obtained in Step 3) to SMOD, **5)** SMOD requests for the current priority List stored in Shuffling Module, **6)** Shuffling Module sends the priority list to SMOD, **7)** SMOD sends the arbitration ID in the CAN (1-bit at a time), **8)** ECU hears the current dominant arbitration bit to decide whether to change in listening mode, **9)** ECU that wins arbitration at timestamp, t sends payload to SMOD, **10)** SMOD forwards payload to CAN that is broadcast within the in-vehicle network

IV. Proposed Mechanism

Fig.3 is an overview of *CanSafe* where arbitration of the messages originating from the ECUs is delegated to SMOD rather than the CAN. The SMOD is responsible for deciding the ECU that wins the arbitration at a communication cycle, and forwards the payload to the CAN, which is further broadcast within the in-vehicle network.

A. Network Parameter Generation

During this phase, the Shuffling Module of every vehicle is loaded with the network parameters that is responsible for generating the arbitration ID of the ECU at every communication cycle, T. As network parameters, 2 prime numbers, p and q, are chosen at random along with an evolving factor, α. The purpose of α is to arbitrarily change or evolve the prime numbers that are leveraged for generating the *seed* value, that brings more randomness to the generation of the arbitration ID at a given communication cycle.

Algorithm 1 Arbtration ID shuffling algorithm

Input: α, $p_{\text{T-1}}$, $q_{\text{T-1}}$, $\text{ECU}_{\text{T-1}}$ (optional)
Output: *Priority List* or ECU_{T}
1: *Priority List*: IDs in decreasing order of priority
2: $p_{\text{T}}, q_{\text{T}}, totalB = \alpha(p_{\text{T-1}}), \alpha(q_{\text{T}}), 11$ or 29
3: bitS = random number between 1 and $totalB$
4: $\lambda(n) = lcm(p_{\text{T}} - 1, q_{\text{T}} - 1)$
5: $seed = T * \lambda(n)$
6: priority bits $= sample(seed, bitS, totalB)$
7: ECU_IDs = Unique ECU IDs in vehicle
8: highest priority = Set 1 in priority bits, rest 0
9: *Priority List*.append(highest priority)
10: **for** i in ECU_IDs **do**
11: *Priority List*.append(highest priority $\oplus$ i)
12: Map $\text{ECU}_{\text{T-1}} \rightarrow \text{ECU}_{\text{T}}$ from *Priority List*
13: **return** *Priority List* or ECU_{T}

As the initial setup of Shuffling Module, 2 large distinct prime numbers, $p_{(init)}$ and $q_{(init)}$ and an evolving factor α is set as communication parameters that is kept private to the Shuffling Module performing the Shuffling algorithm. At a given communication cycle, the priority list of the message IDs is generated using a seed value, *seed*, as obtained from equation 2 that leverage the *Carmichael's totient function* ($\lambda(n)$) calculated using equation 1.

$$\lambda(n) = lcm(p_{\text{T}} - 1, q_{\text{T}} - 1) \tag{1}$$

$$seed = T * \lambda(n) \tag{2}$$

where p_{T} and q_{T} is the prime number generated at timestamp, T using α.

Thereafter, the *seed* value is used to determine message ID with the highest priority by setting particular bits (b_0,b_1,...,b_{28}) to high/low, assuming we use 29 bit CAN 2.0B with extended identifiers. The algorithm to generate the priority list is shown in Algorithm 1.

B. ECU ID generation and sending arbitration ID

When certain actions are performed on the vehicle, such as when the brake is applied, the associated ECU sends a *ECUIDGeneration* request packet to the Shuffling Module consisting of the following:

$$ECUIDGeneration =< ECU_{\text{T-1}} >$$

where $ECU_{T\text{-}1}$ is the current ID of the ECU. On receiving the request, the Shuffling algorithm generates the new *Priority List* and maps the IDs ($\text{ECU}_{\text{T-1}}$) with the IDs of the new *Priority List* (ECU_{T}) as shown below. $\text{ECU}_{\text{T-1}}$ in *Priority List*$_{\text{T-1}}$ $\rightarrow$ ECU_{T} in *Priority List*$_{\text{T}}$

where *Priority List*$_{\text{T-1}}$ and *Priority List*$_{\text{T+1}}$ is the *Priority List* generated at communication cycle $T-1$ (in the previous iteration) and T (at the current iteration) respectively. The Shuffling Algorithm module returns the ECU_{T} to the requesting ECU. Furthermore, the Shufflng module stores the current Priority List generated at communication cycle, T, for sending to the SMOD that indicates the initiation of an arbitration mechanism. This is to be noted that the Priority List is only generated when an ECU sends the *ECUIDGeneration* to the Shuffling Module, which indicates the start of a new communication cycle.

In order to generate the new IDs of the ECUs for the Priority List, the $seed$ is generated using $\lambda(n)$ as shown in equation 1 and 2 (Algorithm 1 lines 2-5). Thereafter it is leveraged to rearrange the priorities of the message IDs at a communication cycle T as shown in Algorithm 1 (lines 7-11). Furthermore, it is important to note that we do not change the priorities of the messages originating from the ECUs. In other words, as commonly agreed by the SAE, if a message originating from ECU_i has higher priority than that originating from ECU_j, it remains so at every communication cycle in our proposed mechanism. However, the arbitration ID and the message priority does not follow a strict relationship (lower message ID means higher priority) unlike the state-of-the-art solutions.

It is also to be noted that the Shuffling Module is the sole module in the proposed mechanism that generates the ECU IDs at every communication cycle. Therefore, α can be designed in any suitable way which is not the main focus of the paper. As there is no synchronization involved between the Shuffling Module and any other module, it does not impact the efficacy of our proposed mechanism.

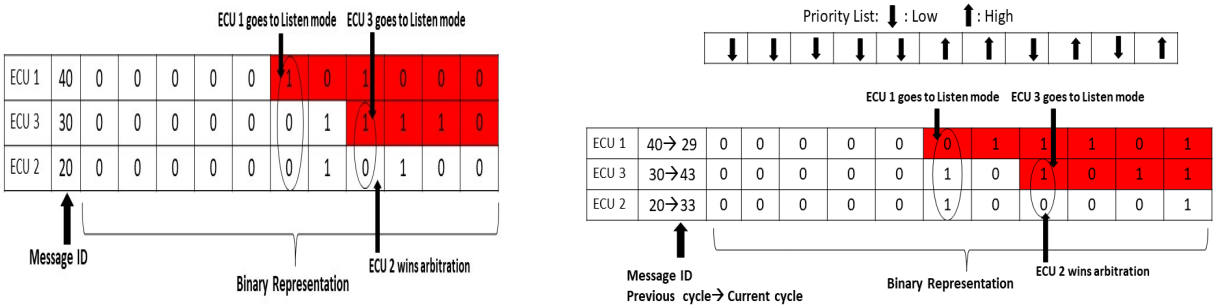
C. Arbitration Phase

We follow the same arbitration process as the state-of-the art standards. The SMOD receives the current *Priority List* from Shuffling Module, when it receives a bit of the arbitration ID from one or more ECUs that are in the arbitration in a communication cycle. If the SMOD receives bits from only one ECUs it directly gets to use the CAN. This is because there is no other ECUs in contention. As a result the bits from the only requested ECU is broadcast into the CAN, which eventually wins the arbitration and sends the payload. However, if the SMOD receives arbitration bits from multiple ECUs, the arbitration process is based on the IDs in the *Priority List*. Therefore, if *n* number of ECUs want to send messages into the CAN, the dominant bit during the arbitration phase is decided as shown in equation 3.

$$dominant = max(ECU_1, ECU_2, ..., ECU_{\text{n}}) \tag{3}$$

where ECU_{i} is the bit received from the arbitration ID sent by the i^{th} ECU in contention and dominant stores the bit value (0/1) that has the highest priority out of bits obtained from the ECUs. The highest priority is decided based on the Priority List obtained from the Shuffling Module

An ECU sends the next arbitration bits only if its previous arbitration bit resembles the value observed in the CAN (i.e dominant value). However, if it does not match, the ECU goes into the listening mode and waits for the next arbitration cycle for sending the message. The difference in arbitration between the proposed mechanism and the traditional (state-of-the-art) mechanism has been highlighted in Fig 4. In Fig. 4(b) we show the transformation of ECU IDs from previous communication cycle to current communication



(a) (b)

Fig. 4: Arbitration Mechanism: (a) Traditional mechanism and (b) Proposed Mechanism

cycle (Algorithm 1, line 12) based on current Priority List. It is to be noted that the Priority List generated for the communication cycle has high values (1) as dominant bits in positions $0, 2, 4-5$ and low value (0) as dominant bits in positions $1, 3, 6-10$. ECU1 (transformed to 29 from 40) goes into listening mode as it is not the dominant bit out of the contending ECUs at the bit position 5. Furthermore, unlike the traditional mechanism (Fig 4(a)), ECU3 (transformed from 30 to 43) transitions into listening mode because it does not have the dominant bit out of the contending ECUs in bit position 3. Thus, ECU2 (transformed to 33 from 20) wins the arbitration in the communication cycle. It is also to be noted that ECU1 has a lower ID that ECU2 but still loses the arbitration. This shows that we do not have any strict relationship unlike the traditional mechanism, where ECU1 would have won the arbitration as it has the lowest arbitration ID. However, we still keep the priorities same as the traditional mechanism (ECU2 has higher priority than ECU3, which has more priority than ECU1).

D. Message Sending

If the last bit send by an ECU is reflected as a dominant value in the CAN, that ECU wins the arbitration (meaning all other contending ECU has transitioned into listening mode). Under such circumstance, the winning ECU completes the message transfer during the communication cycle by sending the remainder of the message (including the payload) to the SMOD, which is eventually reflected/broadcasted into the CAN. On the other hand, the other contending ECUs that lost the arbitration during the communication cycle , T, waits for receiving the *EOF* bit from the sending ECU. Once received, it reconnects with the Shuffling Module to generate the arbitration ID for the next communication cycle. The Shuffling Module repeats the same process as described previously in section IV(b) and replace the Priority List generated in the previous communication cycle with the current Priority List. Thereafter the same steps are followed for the arbitration mechanism.

V. Evaluations and Experimental Results

The performance of the proposed mechanism has been tested in an Intel core i5-9300H CPU 2.40 GHz workstation, and coded in Python 3.0. We determine the performance of the Shuffling Module with both 11 (CAN 2.0A) and 29 (CAN 2.0B) bit setup while assuming varying number of ECUs, to determine the scalability of the shuffling algorithm.

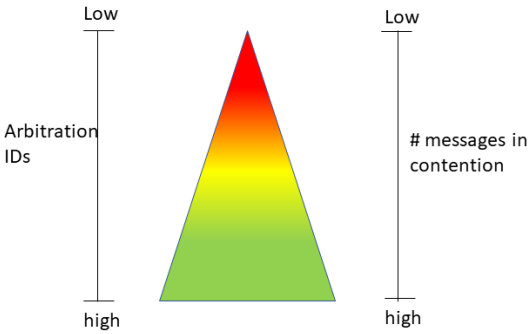


Fig. 5: Risk of an attack: **Red**: High risk, **Yellow**: Moderate Risk, **Green**: Low risk

As seen from Fig 5., the relationship between the number of messages (including the priority of those messages) and the success rate of the attack can be represented as a pyramidal structure as explained through an example below. As the number of message in the contention as well as the priority of those messages decrease, the success rate of the attack increase. For example, if m1 is the only message in contention during a communication cycle and has the least priority (the last element of the Priority List in Algorithm 1) among all the ECUs, the attacker has to simply complement one of the bit out of 11/29 bits. This is because the bits of all other arbitration IDs have a higher priority that m1. Therefore, if the attacker is able to inject one bit complement to m1, the ECU corresponding to m1 will go into listening mode. However, if the number of messages in the contention increases, then the other messages have a priority higher than that of m1. In order to successfully attack the system, the attacker has to generate the highest priority among all the messages in the contention, the difficulty of which increases as the number of messages in the contention increases.

We experimented the time taken for Algorithm 1 to generate the Priority list at a give communication cycle, the results of which are shown in Fig. 6 . The Shuffling algorithm (Algorithm 1) was tested for both 11 and 29 bit arbitration IDs, and with varying number of ECUs (from 50-200, as the modern day vehicle consists of upto 100 ECUs [16]). It

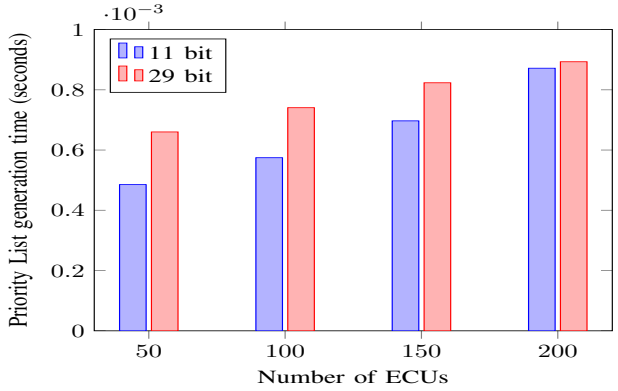


Fig. 6: Average time (out of 20 for each configuration) taken for the Priority list generation using Algorithm 1

was observed that the time taken for generating the Priority list increased by a small amount even if the number of ECUs in the vehicles increased, suggesting that the proposed mechanism is fairly scalable even if the number of ECUs in the future vehicles increases.

TABLE I: Attack success under different Shuffling strategy

Shuffling Strategy	**Attack Success**
Using Same Priority List	High
Generating priority List at regular time intervals	Moderate - High
Generating Priority List every communication cycle	Low

We also determined the frequency of the Shuffling algorithm for the effective working of the proposed mechanism. As shown in Table I, it can be seen that the success rate of an attack is high when the same Priority List is used at different communication cycle. Under such circumstance, the attacker has to gather the highest arbitration ID at a given communication channel and replay it in the next communication. However, it is to be noted that at a given communication channel, the attacker is able to gather the highest priority among the arbitration IDs in contention, which may or may not be the highest priority ID among all the ECUs. Nevertheless, the attacker can continue gathering information for a larger time, and eventually get the highest priority ID as the same Priority List is generated at every communication cycle. It was also observed that the success rate of an attack increases if the Priority List is generated at regular intervals (and not every communication cycle). To illustrate such scenario, let us assume that a Priority List is generated at time interval, T1, and is changed every δt seconds. Under such circumstance, the attacker can snoop into the arbitration IDs every δt seconds to reverse engineer the priority of the IDs. Thus the attack success can be **High** if the attacker finds the highest arbitration ID at a time $<< T1 + \delta$t seconds. However, as stated earlier, since the attacker can only know the highest arbitration ID among the IDs in contention in a given communication cycle, the attack success is **Moderate** if shuffled at regular intervals. Nevertheless, if the Priority List is generated at every communication cycle (i.e., when any ECU want to sent data into the CAN), it adds randomness into the time of generation of Priority List, which decreases the attack success within the system.

TABLE II: Performance Comparison against DoS. ✓- handles Dos Attack, ✗- Does not handle DoS attack

Katragadda et al. [17]	✗
Lee et al. [18]	✗
Humayed et al. [15]	✓
CanSafe	✓

We also analyzed the performance against DoS during the arbitration phase with some of the existing researches as shown in Table II. A major reason for being ineffective against DoS attack in [17] and [18] (as shown in Table II) is that it assumes that the attacker is either unaware of the sequence of the message within the CAN (as in [17]), or the attacker infinitely injects the lowest priority within the CAN (as in [18]). However, if the attacker gathers information of the message/arbitration IDs active in the CAN by snooping into the dominant bits in CAN, it can reverse engineer to gather the sequence of the message IDs, and inject it within the CAN to successfully execute a DoS. Also, instead of continuously injecting the high priority message in the CAN, if the attacker knows the high priority message and injects it at an acceptable frequency delay, it can still win the arbitration, as the messages originating from the ECUs may not follow a regular frequency. Under such a scenario, the attacker can perform DoS in a given communication cycle if it has the highest ID among the contending ECUs during the communication cycle. Furthermore, the existing models assumes that the entire arbitration ID is available during the detection phase, whereas in reality only a single bit at a time is available in the CAN. Therefore, the model has to wait for the entire arbitration bits to be sent to the CAN, which increases the active time of the ECUs (as the ECUs can transit to listening mode after seeing the dominant bit).

As our work closely resembles to the deterministic finite automation (DFA) based mechanism proposed in [15], We compared the probability of an attacker successfully winning the arbitration under different scenarios between *CanSafe* and [15]. In table III, Scenario 1 is the condition where the attacker has the ECU IDs in the white list that is maintained by the authors in [15]. Under such circumstances, the attacker can always win the arbitration by injecting the lowest ID from the white list at every communication cycle. However, the probability decreases to a maximum 0.5 if the attacker does not have the white list (i.e. the attacker injects the last arbitration bit which has a 50% chance of winning). It is to be noted that an attacker can easily perform reconnaissance, and reverse engineer the entire white list by collecting the winning arbitration IDs during a communication cycle. From the ECU IDs gathered from multiple communication cycles, the entire white list (which remains the same at all communication cycles) can be generated. In *CanSafe* the

TABLE III: Arbitration Attack Success

Humayed et al. [15] – Scenario-1	1
Humayed et al. [15] – Scenario-2	≤ 0.5
CanSafe	≤ 0.5

probability is also a maximum of 0.5, when the attacker has a 50% chance of injecting the last arbitration bit correctly. However, since in our mechanism, the ECU IDs and the priority list is changed randomly at each communication cycle, the attacker cannot attack with a probability of 1.

VI. CanSafe Analysis

A. Brute-Force Attack

In this attack, the attacker constantly injects the lowest message ID (highest priority as per the state-of-the-art specification) #0000 to perform DoS within the in-vehicle network. As the proposed mecanism reshuffles the priority of the messages at every iteration, the probability of the brute-force attack to be successful is very low (represented below as P(success) in equation 4).

$$P(success) <= \frac{1}{2^{n}} \tag{4}$$

where n is the number of bits used to represent the arbitration ID (11 bits for CAN 2.0A and 29 bits for CAN 2.0B).

However, it is to be noted that even if #0000 is the highest priority message at a time, it may happen that the ECU with the actual #0000 sends a message. In the event of such a scenario, the arbitration mechanism fails due to collision. In the next arbitration cycle, #0000 is mapped to some other ID which is known by the authentic ECU in the contention but not by the attacker. Hence, the attack fails under such circumstance. Thus, it can be concluded that the intention of the attacker to inject the lowest priority message within the network is successful only if random shuffling yields #0000 as the highest priority message and the authentic ECU associated with the ID does not send any message

B. Attacker Reconnaissance and Replay Attack

In this attack, the attacker gathers the information at one cycle and leverage it for injecting message into the next cycle. The information that can be gathered by the attacker includes the arbitration ID that won the contention in the previous cycle, and the message sent by the ECU. At this stage the attacker may perform 2 kinds of attack : 1) Naive Attack 2) Reverse Engineering.

In the Naive attack, the attacker performs replay attack by injecting the highest arbitration ID gathered in the previous cycle. Under such circumstance the attack can be successful if one of the condition holds:

- Case 1: The arbitration ID gathered (which may not have been the highest priority ID at the previous cycle but was the highest among the IDs in contention) is the highest priority ID among all the ECU IDs in the vehicle or among the IDs in contention (which means that the random shuffling generates the same Priority List, which is highly unlikely) at the next cycle.
- Case 2: The attacker's injected message is the only message in contention.

For Case 1, the chances that the arbitration ID that won in the previous cycle becoming the highest priority among all the ECUs in the next cycle can be represented by equation 4, i.e., all the bits has to be exactly the same. Nevertheless, it is to be noted that such the attacks in Case 1 can be handled by simple check conditions that does not allow a ID that won the arbitration in a cycle to become the highest priority ID in the next cycle or no same priority list at consecutive communication cycles. As the priority list is generated only by the Shuffling Module, such a check condition does not affect the overall communication flow of the mechanism.

As for Case 2, this is a message injection attack, which is outside the scope of this paper.

In case of a Reverse Engineering attack, the attacker gathers the information of different cycle and tries to find out the random bits (Algorithm 1, line 6) that determines the priority list. However, at any given cycle, the attacker has to try out 2048 (for 11 bits) and $\approx$ 500 million (for 29 bits) to figure out the α. As Shuffling Module is the only module that is generating the priority list (with no synchronization with other modules), this proposed mechanism can exploit any relationship between the α's used in different communication cycles that need not be a linear relationship. Such an independence of the Shuffling Module obfuscates the random factor from an attacker at different communication cycle.

C. Bit inferring by an attacker

The connection between the ECU and the Shuffling Module is secure, meaning it cannot be intercepted by an attacker. Further, the attacker cannot connect to the Shuffling Module. As a result, the attacker cannot determine the ECU ID of an ECU at different communication cycle. As a single bit is sent by every ECU to the SMOD, that is reflected at the CAN, the attacker has no way to determine the bits sent by the ECUs in contention, in advance.Thus, the attacker cannot infer the dominant bit at a given time, without listening to it from the CAN during a communication cycle.

However, as it is mentioned in section 5(Fig 5), the attack success of an attacker increases if the number of messages in the contention is less and the priority of the message is also less. It may happen that the attacker does not inject any bit until the last bit of the arbitration ID. Under such circumstance the probability of an attack is maximum (=0.5). This is because the last bit of the arbitration ID can be 0 or 1. This means that the dominant bit can be either 0 or 1. So the attacker has to sent either 0 or 1 into the CAN. If the attacker sends 0, a contending ECU sends 1 and the dominant bit is 1, then the attacker loses the arbitration. However, if the attacker sends 1 and the contending ECU sends 0, then the attacker wins the arbitration and the the ECUs go into sleep mode. Thus, the attack success of an attacker is the maximum when only the last bit of the arbitration ID is injected into the CAN.

D. Dynamic ECU ID prevents tracking

As the ECUs use different dynamic arbitration ID which is allocated to it by the Shuffling Module, any attacker that wants to compromise an ECU, or masquerade to be some other legitimate ECU should be able to generate the arbitration ID of the ECU in a given communication cycle. As the number of ECUs, is around 100 in a modern day vehicle, the unique arbitration IDs in a given communication cycle is a subset of the number of possible arbitration IDs at every communication cycle (2^{11} for CAN 2.0A and 2^{29} in our proposed mechanism). Thus, it is difficult for an attacker to target a specific ECU as it has to correctly determine the ECU ID in the given communication cycle. Furthermore, as the proposed mechanism is also resilient to replay attack, it shows that the attacker cannot replay a captured ECU ID from an ECU at one communication cycle and use it to inject message in a different communication cycle. Thus, the proposed mechanism prevents masquerading attack against any ECU. Furthermore, as the ECU IDs are dynamic, it is also difficult for an attacker to associate a payload with any ECU, as it may happen that an arbitration/ECU ID belonging to one ECU in a communication cycle may be allocated to some other ECU in a different communication cycle. A possible solution to detect any aberration in a data is to design a simple anomaly detection system that determine if an ECU gives anomalous data in different communication cycle, which is outside the scope of this paper.

E. Multiple Injection by Attacker/s

Our proposed mechanism is able to withstand DoS attack as long as there is a single injection inside the CAN bus, i.e. single message ID is injected at every communication cycle. However, it is to be noted, that the attacker can perform a more sophisticated attack by injecting 2 bits in a communication cycle. This may happen if a single attacker is able to perform 2 injection attacks within the CAN bus, or more than one attacker colludes to inject 0 and 1 simultaneously for each bit position. On the advent of such a scenario, the attacker can effectively effectively deny all legitimate ECUs by winning the arbitration if the 2 injected bits are 0 and 1 in the last bit position during the arbitration of a communication cycle. Under such circumstance, the attacker will either win the arbitration or result in a collision. Such an attack is not addressed in our proposed mechanism.

VII. Conclusion

In this paper, we have proposed a new shuffling algorithm for dynamically generating the priority IDs of the ECUs within in-vehicle networks. We leverage the MTD based shuffling technique to nullify the reconnaissance phase of an attacker and provides resiliency against DoS.We also showed that the proposed shuffling algorithm has a fairly constant time requirement for both 11 and 29 bits CAN architecture, and performed a detailed analysis of the proposed mechanism showing the possibilities of various conditions under which an attacker can to bypass/break the shuffling mechanism. The proposed mechanism enhances the security of the existing CAN specification and preserve the priority of the messages originating from the ECUs even if their arbitration IDs do not follow a strict relationship. For the future, we would like to analyze and enhance our proposed mechanism to handle multiple injection attacks by remote attackers.

References

[1] Hyun Min Song and Huy Kang Kim. Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data. *IEEE Transactions on Vehicular Technology*, 70(2):1098–1108, 2021.

[2] Shu Nakamura, Koh Takeuchi, Hisashi Kashima, Takeshi Kishikawa, Takashi Ushio, Tomoyuki Haga, and Takamitsu Sasaki. In-vehicle network attack detection across vehicle models: A supervised-unsupervised hybrid approach. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1286–1291. IEEE, 2021.

[3] Thomas Huybrechts, Yon Vanommeslaeghe, Dries Blontrock, Gregory Van Barel, and Peter Hellinckx. Automatic reverse engineering of can bus data using machine learning techniques. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 751–761. Springer, 2017.

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

[5] Rui Zhuang, Scott A DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40, 2014.

[6] Lotfi Ben Othmane, Lalitha Dhulipala, Moataz Abdelkhalek, Nicholas Multari, and Manimaran Govindarasu. On the performance of detecting injection of fabricated messages into the can bus. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[7] Harini Narasimhan, R Vinayakumar, and Nazeeruddin Mohammad. Unsupervised deep learning approach for in-vehicle intrusion detection system. *IEEE Consumer Electronics Magazine*, 2021.

[8] Hyun Min Song, Jiyoung Woo, and Huy Kang Kim. In-vehicle network intrusion detection using deep convolutional neural network. *Vehicular Communications*, 21:100198, 2020.

[9] Dongxian Shi, Ming Xu, Ting Wu, and Liang Kou. Intrusion detecting system based on temporal convolutional network for in-vehicle can networks. *Mobile Information Systems*, 2021, 2021.

[10] Mubark Jedh, Lotfi ben Othmane, Noor Ahmed, and Bharat Bhargava. Detection of message injection attacks onto the can bus using similarity of successive messages-sequence graphs. *arXiv preprint arXiv:2104.03763*, 2021.

[11] Riadul Islam, Rafi Ud Daula Refat, Sai Manikanta Yerram, and Hafiz Malik. Graph-based intrusion detection system for controller area networks. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[12] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 911–927, 2016.

[13] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*, pages 63–68. IEEE, 2016.

[14] Seunghyun Yoon, Jin-Hee Cho, and Dong Seong et al. Kim. Moving target defense for in-vehicle software-defined networking: Ip shuffling in network slicing with multiagent deep reinforcement learning. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II*, volume 11413, page 114131U. International Society for Optics and Photonics, 2020.

[15] Abdulmalik Humayed, Fengjun Li, Jingqiang Lin, and Bo Luo. Cansentry: Securing can-based cyber-physical systems against denial and spoofing attacks. In *European Symposium on Research in Computer Security*, pages 153–173. Springer, 2020.

[16] S Hamsini and M Kathiresh. Automotive safety systems. In *Automotive Embedded Systems*, pages 1–18. Springer, 2021.

[17] Satya Katragadda, Paul J Darby, Andrew Roche, and Raju Gottumukkala. Detecting low-rate replay-based injection attacks on in-vehicle networks. *IEEE Access*, 8:54979–54993, 2020.

[18] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 57–5709. IEEE, 2017.