Using Machine Learning to Detect Anomalies in Embedded Networks in Heavy Vehicles

Hossein Shirazi, Indrakshi Ray(✉), and Charles Anderson(✉)

Colorado State University, Fort Collins, CO 80526, USA
{shirazi,iray,chuck.anderson}@colostate.edu

Abstract. Modern automobiles have more than 70 electronic control units (ECUs) and 100 million lines of code to improve safety, fuel economy, performance, durability, user experience, and to reduce emissions. Automobiles are becoming increasingly interconnected with the outside world. Consequently, modern day automobiles are becoming more prone to cyber security attacks. Towards this end, we present an approach that uses machine learning to detect abnormal behavior, including malicious ones, on embedded networks in heavy vehicles. Our modular algorithm uses machine learning approaches on the internal network traffic in heavy vehicles to generate warning alarms in real-time. We tested our hypothesis on five separate data logs that characterize the operations of heavy vehicles having different specifications under varying driving conditions. We report a malicious detection rate of 98–99% and a mean accuracy rate of 96–99% across all experiments using five-fold cross-validation. Our analysis also shows that with a small subset of hand-crafted features, the complex dynamic behavior of heavy vehicle ECUs can be predicted and classified as normal or abnormal.

Keywords: Anomaly detection · SAE-J1939 · Heavy vehicle security

1 Introduction

In the last few decades, complex electronic systems have been incorporated into vehicles in order to improve performance, efficiency, safety, and usability. In modern-day vehicles, mechanical systems rely heavily on advanced electrical and computational systems that include network, software, and hardware. Consequently, modern day vehicles have become vulnerable to both physical and cyber attacks. In the near future with the advancement of wireless fleet management technology, we anticipate that cyber attacks would increase and have catastrophic consequences.

Checkoway *et al.* analyzed multiple attack vectors in vehicles and showed that Electronic Control Units (ECUs) have a potential of being compromised [1]. A compromised ECU can be used by an attacker to inject malicious messages into the in-vehicle network through a physical connection. Subsequently, researchers

A. Benzekri et al. (Eds.): FPS 2019, LNCS 12056, pp. 39–55, 2020.
https://doi.org/10.1007/978-3-030-45371-8_3

remotely compromised an ECU and injected messages to stop a Jeep Cherokee running on a highway [2]. This triggered a recall of 1.4 million vehicles [3]. Attacks can be launched by devices connected to the physical bus or through a remote connection.

Our research focuses on embedded networks of heavy vehicles. Heavy vehicles are expensive and they constitute a critical infrastructure of a nation. In order to allow interoperability of various components manufactured by different Original Equipment Manufacturers (OEMs), heavy vehicles use a standardized protocol known as SAE-J1939 implemented over a Controller Area Network (CAN) bus, which allows ECUs to communicate with each other. The use of standardized protocols makes heavy vehicles susceptible to attacks, as the code for deciphering messages is easily accessible. The damage inflicted by heavy vehicles can also be catastrophic. Consequently, heavy vehicles are a much higher target for attackers compared to cars [4].

Two different categories have been introduced as countermeasures against these attacks: *proactive* and *reactive*. *Proactive* mechanisms protect against malicious injections and are implemented through Message Authentication Code (MAC). While message authentication ensures a certain level of security in Internet applications, there are limitations of applying it to the heavy-vehicle domain. First, there is limited space for embedding the MAC in the messages. Second, the messages are required to be processed in real-time, which may not be feasible [3]. Third, applying this approach requires modification of lower-level protocols.

Multiple Intrusion Detection Systems (IDS)s have been proposed as implementations of *reactive* approaches in the literature [3,5–7]. Similar to other network defense mechanisms, current IDS systems monitor exchanged messages and detect any abnormal behavior, but there are certain limitations. The CAN bus is a broadcast communication network, and it cannot verify the sender of messages. Consequently, it is extremely difficult for state-of-the-art IDS systems to detect hijacked ECUs. Moreover, to the best of our knowledge, there is no defense mechanism that attempts to detect malicious message injection in the messages that adhere to the SAE-J1939 standard. Such message injections can compromise the *integrity* and the *availability* of the system.

In this work, we address some of the limitations stated above. We scope our work to detect the injection of messages which adhere to the SAE-J1939 standard. In the context of this work, we use the term heavy vehicle as one that uses the SAE-J1939 standard. Our proposed approach uses machine learning to detect abnormal traffic in the embedded network. A few reasons motivated our choice. The most important reason is the high-dimensional nature of the data, and its complicated non-linear relationship to the operational state of the vehicle. High-dimensional data is often difficult to understand. For instance, a single packet in heavy vehicles has many dimensions. Thus, the data is quite difficult to analyze or understand. Furthermore, there is a space-time correlation among the data that needs to be modeled. There are not many computational models besides machine learning that can analyze this complex temporal data.

The use of SAE-J1939 makes it possible to convert raw transmitted messages on the CAN bus to specific parameters of the vehicle. Thus, we define a machine learning model based on low-level vehicular PARAmeters. While each message contains information about the current state of the vehicle, it does not give any information about the previous state. To solve this limitation, we added the history of previous values to each parameter value to leverage the learning model. In addition, some statistical derivative features have been added to give even deeper insights about the model.

A vehicle's parameters are categorized in particular groups in the SAE-J1939 protocol, for example, frequency and sender. Thus, we created multiple models based on each parameter group, referred to as Parameter Group Number (PGN) in the standard. The learning algorithms create a behavioral profile for each PGN that will be used later to compare with its current behavior to detect any deviation from the regular pattern. We used a wide range of learning algorithms to train models and studied the performance of each algorithm.

Multiple challenges had to be addressed in our solution. The first challenge is that behavior varies across vehicles. While the SAE-J1939 standard is common to all the vehicles, the parts, models, embedded ECUs, and software are all different. As a solution, we applied general purpose machine learning algorithms. Machine learning classifiers can learn local features of interest in a collection of data. The feature set has been enriched by adding derived statistical features. We believe this is sufficient to accurately predict whether or not a given SAE-J1939 message is normal.

The next challenge is deciding the number of previous messages that need to be considered for creating a behavioral profile. If this number is too small, even standard infrequent patterns will be classified as abnormal. If this number is too large, abnormal behavior will not be distinguishable from the normal behavior pattern. We make this trade-off based on some experiments.

The other challenge relates to the computational intensity of the learning model. Machine learning models, in general, are computationally expensive, taking a long time to produce a prediction while also using a lot of resources. In an operational heavy vehicle, the ECUs generate messages every hundred milliseconds; our proposed approach is required to have a response time in that range. Furthermore, the experimental setup and hardware need to be applicable for an in-vehicle configuration. To address this challenge, we ran experiments on lightweight Raspberry Pi Zero computers. We report our results in the experiments section.

Key Contributions. This paper makes the following contributions:

- Developing profiles of PGNs by employing machine learning techniques to analyze high-dimensional data generated by the ECUs.
- Detecting abnormal messages by employing machine learning techniques and generating real-time alarms;
- Our proposed method produces significant accuracy in different experiments in the range of 98–99% which asserts our initial hypothesis.

The rest of the paper is organized as follows: Sect. 2 discusses background and existing defense mechanisms. Section 3 models the adversaries capabilities, attack scenarios, and simulation. Section 4 details architecture design of our proposed approach, which is evaluated in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Related Works

2.1 Background on CAN and SAE-J1939

Controller Area Network. Development of the Controller Area Network (CAN) started at Robert Bosch GmbH and was released in 1986 [8]. CAN employs a two wired differential bus, which supports speeds of up to 1 Mbit/s [9]. The initial version, CAN 1.0, used 11 bits for the identifier fields, which has since been upgraded to 29 bits in CAN 2.0 called an extended version. The protocol establishes a priority mechanism to prevent message collision on a shared bus; a lower value of the identifier specifies a higher priority for the sender [9]. CAN 1.0 was proposed in a time that neither the Internet nor concepts of *virus* and *worm* were prevalent [6] and security was not a concern. This is evident in the fact that the CAN protocol alone does not address any security concerns.

SAE-J1939 Standard. The CAN bus forms the lower layer of a vehicle's network. SAE-J1939 comprises the higher level layer of a heavy vehicle's network. The SAE-J1939 protocol allows ECUs from multiple vendors to communicate. SAE-J1939 defines five layers in the seven-layer OSI network model including the CAN ISO 11898 specification and uses only extended frames with a 29-bit identifier for the physical and data-link layers. *Protocol Data Unit (PDU)* is a block of information transferred over the internal network. Each *PDU* in the SAE-J1939 protocol consists of seven fields: *priority (P)*, *extended data page (EDP)*, *data page (DP)*, *PDU format (PF)*, *PDU specific (PS)* (which can be a destination address, group extension, or proprietary), *source address (SA)*, and *data field*. There is also a reserved field *Reserved(R)* with one bit length for further usage. These fields are all packed in one data frame and sent over the physical media to other network devices [10]. The combination of the *PF* and the *PS* fields derives two important parameters: Parameter Group Number (PGN) and Destination Address (DA). PGN is used to group similar vehicular parameters together in one single message frame. Each of these parameters is named Suspect Parameter Number (SPNs). Each SPN defines how the application layer can interpret some portion of the data field using the Digital Annex [11]. DA specifies destination of the message.

2.2 Introduced Defense Mechanism

Many concepts from the IT networks, such as securing the network, cryptography, proposed attacks and defense mechanisms, have been adapted for the internal network of automobiles. The proposed countermeasures can be classified into *proactive* and *reactive* mechanisms [12].

Proactive Mechanism. Proactive mechanisms focus on improving protocols, applications, systems, *etc.* to prevent the occurrence of any attacks. These mechanisms are not foolproof [12], but can be remarkably effective. The CAN and SAE-J1939 protocols do not support authentication and encryption so a wide range of attacks can be launched on them. Towards this end, Murvay *et al.* [9] proposed a mechanism to include message authentication on the protocol and evaluated the overall overhead on network communication. In addition, they evaluated their enhancement from a computational point of view and showed that low-end automotive-grade controllers are capable of symmetric cryptography and high-end cores can handle asymmetric (public-private cryptography) algorithms. However, even with an authentication mechanism on the CAN bus, the maximum payload length is only 8 bytes, so the available space for an encrypted Message Authentication Code (MAC) is very limited [3]. Multiple solutions have been proposed to address this limitation. These include sending MAC in multiple data frames, using multiple CRC fields, or exploiting an out-of-bound channel for message authentication [3,13–15]. Although proactive mechanisms are capable of preventing attacks, they would require changing the protocols, applications, and/or hardware. These types of solutions are unrealistic as they do not take into account vehicles that are currently operational.

Reactive Mechanism. Reactive mechanisms detect an attack or an impending attack and reduce its impact on the victim's vehicle at the earliest and provides a response mechanism to either block the attack or alert other systems [12] while trying to minimize the number of *false-alarms*[1].

The physical signal characteristics have been recently used to fingerprint ECUs connected to the CAN bus using voltage or time. Murvay *et al.* [16] have authenticated messages on the CAN bus using physical characteristics of ECUs. The approach requires measuring the voltage, filtering the signals, and calculating mean square errors to uniquely identify ECUs. The signals from different ECUs showed minor variations in, for example, how fast rising edge is set up or how stable a signal is. Although these characteristics remain unchanged over a period of several months, there are certain limitations. Examples include results varying with changes in temperature [17]. Furthermore, their method was evaluated on the low-speed bus with trivial ECUs, not on the high-speed bus with critical ECUs. In addition, the authors did not consider collision situations that would impact the identification mechanism [18].

Cho *et al.* [3] proposed a time-based IDS. The clock offsets and skews of asynchronous nodes depend solely on their local clocks, thus being distinct from others. Their method detects anomalies based on clock skew of different ECUs connected to the bus. The approach measured intervals of periodic messages for different ECUs to measure unique clock skews for each ECU. However, this method can be defeated in certain ways. For example, if an adversary can match the clock skews of a tampered device with that of the actual one, this approach does not work [19]. Moreover, this approach does not work for non-periodic messages.

[1] A false-alarm occurs when an alert is not due to an actual attack.

Machine learning algorithms are widely used for levels higher than the physical layer. Learning algorithms have the ability to learn patterns and detect any deviation outside of an accepted threshold. Hence, these algorithms have been widely employed to create detection mechanisms. Kang *et al.* [5] proposed an intrusion detection model that discriminates between regular and abnormal packets in an embedded vehicles network by using a Deep Neural Network (DNN). They generate features directly from a bit-stream over the network by counting the occurrence of '1' in the binary data field of messages. Consequently, the features have no semantics associated with them. For simulating the anomalies, they showed a wrong value of the Tire Pressure Monitoring System (TPMS) on the panel and achieved an accuracy of 99%. As this algorithm tested only one type of attack, it is unclear how this mechanism will detect more complex attacks. Furthermore, it authenticates targeted fields with other current field values but does not take into account previous values of the fields. An attack message that is consistent with other feature values can bypass this algorithm.

Chockalingam *et al.* [6] investigated the use of different machine learning algorithms in detecting anomalies in CAN packets or packet sequences and compared the algorithms. They used Long Short-Term Memory (LSTM) to consider the sequence of inputs for labeling the dataset. While they did not report the accuracy, the Area Under Curve (AUC) for Receiver Operating Characteristic (ROC) curve varies from 82% to 100% in different situations.

Narayanan *et al.* [7] introduced OBD Secure-Alert which detects abnormal behavior in vehicles as a plug-n-play device on the vehicle. They used Hidden Markov Model (HMM) to decide whether a vector is normal or not. HMM considers just a single previous vector so Narayanan *et al.* added more previous messages to each learning vector.

Mukherjee *et al.* [20] used a Report Precedence Graph-based (RPG) which describes how graphs modeling the state transitions can be used to distinguish between safety and security-critical events in real-time.

3 Threat Model

3.1 Adversary Capabilities

Attackers can compromise in-vehicle ECUs physically or remotely using known attack surfaces or exploiting new vulnerabilities. For example, they can exploit telematic devices to gain remote access to the CAN bus.

In this study, we assume that an adversary has access to the CAN bus. In addition, we assume that the adversary receives all messages on the CAN bus and is capable of generating SAE-J1939 compatible messages with the desired frequency, and has full control of all fields including the priority and data. Message priority helps the adversary to set a higher priority for their messages and block messages with lower priorities on the bus. This violates both the availability of some functionalities and the integrity of the system.

3.2 Attack Scenarios

We discuss the following attack scenarios that are feasible given the adversary capabilities. Certain attacks target vulnerabilities on the CAN level and are applicable to both regular and heavy vehicles. One of the most commonly studied attacks against the CAN network is a DoS attack. In this attack, the adversary will send unauthorized messages with the highest priority and frequency to dominate the bus. Thus, sending or receiving messages will be delayed or even impossible.

In a different attack, an adversary may monitor the CAN bus and target a specific action of the vehicle. Whenever the adversary sees a message related to that particular action, it sends a counter message to make the previous action ineffective. Hoppe *et al.* [21] implemented an application that monitors the bus, and every time there is a message to open the window, a counter message is sent to make the requested action ineffective. Such an attack is also plausible in a heavy vehicle because of easy access to the message documentation in the SAE-J1939.

Recently, however, attacks have been designed to take advantage of the SAE-J1939 protocol that are only applicable to heavy vehicles. Burakova *et al.* in [22] proposed the first special attack against heavy vehicles in 2016.

The adversary we modeled can inject SAE-J1939 crafted messages through a compromised ECU. ECUs frequently report an associated vehicular parameter. For example, the engine's ECU reports the value of RPM every 100 ms. In an attack scenario, an adversary can send incorrect parameter values, like RPM, with a higher priority to override original messages. In this case, an attacker can either dominate the original engine's ECUs with a higher priority message or can send an incorrect value for a specific parameter after seeing it on the bus.

For example, in the so-called fuzzing attack, the attacker changes the values of a specific parameter with random values and injects generated messages into the bus. In CyberTruck Challenge 2018[2], we conducted this attack on a real heavy vehicle. In that experiment, we successfully overwrote speed and RPM values on the CAN bus using SAE-J1939. We showed a speed of 50 mph on the dashboard even when the truck was stationary[3].

There is not any attack data publicly available to be used as a benchmark. Thus, we simulated new attack messages and injected them into the logged file to check whether our detection mechanism could find them. During our proposed attack, we maliciously changed the vehicle's parameters (such as current speed) multiple times.

Figure 1 shows one example of an attack where we changed the vehicles reported speed. It depicts the actual speed values of the vehicle and maliciously injected values. This attack in practice may violate the integrity of the system, can severely damage the vehicle, and give the driver incorrect information which may lead to an accident. While this experiment shows the manipulation

[2] https://www.cybertruckchallenge.org/.

[3] Due to a Non-Disclosure Agreement (NDA), we cannot release the make and model of the vehicle.

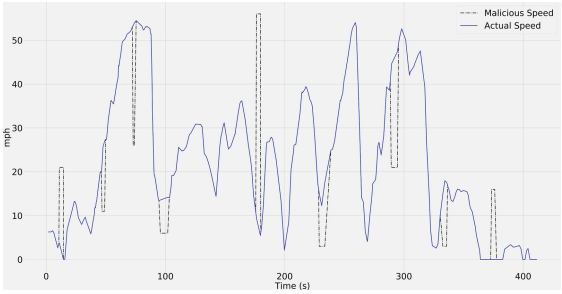


Fig. 1. Test data for speed where the speed value has been maliciously manipulated with other random values.

of speed, this attack is not simply limited to speed; we have tested a wide range of parameters and report our results in Sect. 5.

4 Proposed Defense Mechanism

4.1 Feature Engineering and Fingerprinting

The performance of machine learning is very much dependent on the choice of our features and how we represent them. In our proposed model, we define three types of features: `SPN values`, `History values`, and `Derivative features`.

SPN Values: Features based on SPN values are obtained by deciphering the sniffed messages on the CAN bus. We convert the raw messages to the SPN values.

History of Values: The value of each SPN depends on both the current vehicle's parameters and their previous values. Note that the classifier would need to use previous samples to make a more precise decision. Towards this end, we include n previous SPN values of each vector to overcome this challenge. As such, each vector will now have values of the current state and will also include the last n reported values for each SPN. We consider consecutive packets, *i.e.* $p_i \rightarrow p_{i-1} \rightarrow p_{i-2} \rightarrow p_{i-3} \rightarrow p_{i-4}$, where p_i denotes the current value, and p_{i-j} denotes previous values for $j >= 1$.

Derivative Features: To give more insight into how each SPN value changes over the last n history windows, we added multiple derivative features to the vector. For each *measured* SPN, we added *average*, *standard deviation*, and *slope* of the last n values to the feature set. In addition, we added history for these derived features as well. For example, the feature vector includes the current slope and the last n values of the slope as well. These new derivative features help the classifiers to better model the current behavior of ECUs and the change of the parameters over time; this leads to more precise predictions.

The other feature that we added was distance. We observed the changes between two consecutive messages of each PGN would be limited. This observation can be mathematically represented by the Euclidean distance between two feature vectors. If p and q are two consecutive vectors with the size of n, the Euclidean distance will be calculated by the formula given below which is used as a feature: $d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$.

4.2 Detection Mechanism Architecture

The architecture of the proposed detection mechanism consists of four separate modules: `BusSniffer`, `Message Decoder`, `AttackDetector`, and `AlarmGenerator`. Figure 2 highlights the architecture of the proposed detection mechanism.

`BusSniffer` connects to the CAN bus using an access point like the OBD-II port. This port connects directly to the CAN bus and captures all transmitted messages.

`Message Decoder` utilizes the SAE-J1939 standard to convert the raw messages on the bus to the SPN values and creates an initial vector of the vehicle's parameters. This module adds other meta-data fields including time-stamp, length of the data field, source address, destination address, and previously defined features such as derivative features and history of feature values.

`AttackDetector` consists of two phases: *Training* and *Detecting*. The *training* phase requires preparing a dataset of regular and abnormal messages for every PGN. Multiple classifiers can be trained on the dataset, and the classifier that performs the best will be subsequently used. The *training* phase may take a long time; however, the trained classifiers can be used countless times without needing to re-train. Note that this phase can be done offline and outside of the vehicle before installing it.

In the *detection* phase, whenever a new vector comes in, the `AttackDetector` fetches it's PGN value and sends it to the designated classifier object, which tests whether it is a normal vector or not. If the classifier detects an abnormal message, it will trigger the `AlarmGenerator` module.

`AlarmGenerator` is responsible for preparing appropriate alarm messages using SAE-J1939 and transmits it over the CAN bus. The message will be generated in the form of a *Broadcast* message, and all connected nodes will be aware of this abnormal situation. This can also include turning on a warning light on the dashboard to notify the driver.

4.3 Detection Algorithm

Algorithm 1 discusses the steps to decide whether a transmitted message on the bus is normal or not. In the first step, the `BusSniffer` module adds messages to the processing queue named `mess`. Then, each added message is decoded to give the SPN values and aggregated for the derivative features. Based on the PGN, the appropriate classifier object will be fetched and used to predict whether the current message is normal or not. If it is abnormal, it will be sent to the `AlarmGenerator` module.

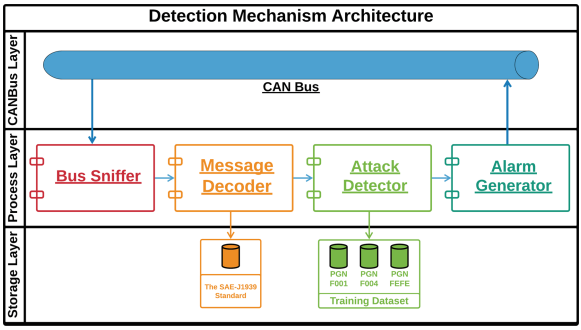


Fig. 2. Architecture of proposed detection mechanism.

Machine Learning Classifiers. We used seven classifiers: Decision Tree (DT), Gaussian Naïve Bayes (GNB), Gradient Boosting (GB), K-Nearest-Neighbors (KNN), Random Forrest (RF) and Support Vector Machine (SVM) with two kernels: Linear (SVCL(l)) and Gaussian (SVCR(r)). We trained the learning models and tested them in a five-fold cross-validation approach to avoid overfitting each classifier.

Result: Detecting Abnormal
$mess\_queue \leftarrow sniff\_bus()$;
while $mess\_queue \neq \{\}$ **do**
  $mess \leftarrow mess\_queue.pop()$;
  $vector \leftarrow decipher(mess)$;
  $add\_history \leftarrow add\_history\_length(mess)$;
  $fprint \leftarrow add\_derived\_features(added\_history)$;
  $cls \leftarrow get\_classifier\_Obj(fprint.PGN)$;
  $predict \leftarrow cls.predict(fprint)$;
  if $predict == Abnormal$ **then**
    $Generate\_Alarm(fprint)$
  else
    $Continue$;
  end
end

Algorithm 1. Detecting Abnormal Messages

5 Experiments and Evaluations

5.1 Creating Datasets from Previously Recorded Messages

We ran our experiments and modeled the problem using the log data. We use five previously captured CAN bus log messages that were generated at the University

of Tulsa[4]. For security reasons, however, we cannot release information about the model or manufacturer of the vehicles. We describe the driving conditions and log files which we have used.

Since the learning algorithms need adequate data to learn the pattern, we had to select PGNs which had at least 500 messages in the log file. After we created the final dataset, we only used a maximum of 5000 instances for training and testing purposes as an upper-bound.

In the first step, we convert each log file into multiple datasets where each one includes vectors of only one PGN. We then inject abnormal messages with an injection ratio of 50% with a history length of 10 for the following experiments. It is safe to assume that the rate of malicious messages during any real attack would be less than 50%. As such, without loss of generality, we used a higher injection ration to circumvent difficulties in training models with imbalanced datasets. In this experimental context, the training phase has a chance to learn from more malicious messages, however, in real attack far less malicious messages would be present. These choices do not affect the validity or generality of our approach. Table 1 summarizes experiment descriptions, number of used PGNs, and experiment duration.

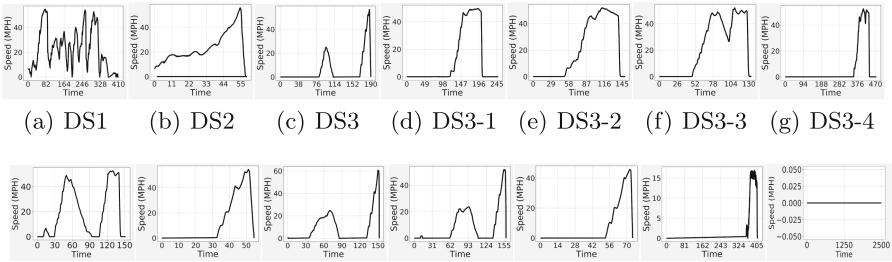
Table 1. Specification of each log file including log name, number of selected PGNs, experiment duration, and description of experiment.

Log	PGN	Time(s)	Description
DS1	10	411	The truck was driven around a block
			Three events of the vehicle braking suddenly (hard brake)
DS2	2	60	Log file does not have much data
			One hard braking event
DS3	10	270	We had 10 log files from 10 different driving conditions
			Selected one log file and used it to create in the DS3
			Used the DS3 dataset as a benchmark
			Highest recorded speed was 55 mph, followed by a hard brake
DS4	7	729	Vehicle was stopped for most of experiment
			Had a hard brake at the 400th second
DS5	9	2500	Vehicle was stopped during this experiment
			Had an active engine

5.2 Summarized Performance of Model

We selected the best classifier for each dataset and all of its log-PGN pairs. The number of messages related to each PGN was not equal in each dataset; as such

[4] The data is available at http://tucrrc.utulsa.edu/J1939Database.html.



(a) DS1 (b) DS2 (c) DS3 (d) DS3-1 (e) DS3-2 (f) DS3-3 (g) DS3-4

(h) DS3-5 (i) DS3-6 (j) DS3-7 (k) DS3-8 (l) DS3-9 (m) DS4 (n) DS5

Fig. 3. Wheel-based speed from different driving conditions.

we calculated the weighted average of scores based on the portion of instances of each PGN in the dataset to normalize the average. To display this fact, in the DS2 log file, we have two PGNs: $F004_{16}$ and $FEF0_{16}$ which constitute 83.96% and 16.04% percent of the messages respectively. We normalized learning scores based on the percentage of messages in the log file and report the results in Fig. 4. The total scores show good results. True Positive Rate (TPR) is more than 98% in all cases and total accuracy (ACC) is at least 99% (excluding DS2, which scored a 96%).

5.3 Different Driving Scenarios

The vehicle's parameters are constantly being changed by either external (*e.g.* road, weather, and driver's habits) or internal (*e.g.* increasing speed or torque) factors. These may degrade the accuracy of the model. To check whether these conditions can have any effect on the efficacy of the model, we gathered 9 log files from different driving scenarios of the vehicle from DS3. Figure 3 shows the wheel-based speed for each driving condition. We repeated the previous experiment for each of the 9 log files and then calculated the weighted average of the machine learning scores identical to Sect. 5.2. If the internal or external parameters can affect the model, it should be captured in this model. We compared the performance of the model in all nine other driving scenarios with initial DS3 results and looked for any differences.

The first column in the Fig. 5 shows the result from the original DS3 scores, and the rest show the other nine driving conditions. Among ten different driving conditions, including the DS3 initial test, we do not see any significant variation in learning scores and we get consistently high scores in different driving conditions. This deduces that the proposed model works well for different driving conditions.

5.4 Performance Across Multiple Classifiers

Although all of the previously discussed experiments were repeated across multiple machine learning classifiers, we selected and reported the best one. In this

experiment, we compared the performance of the different classifiers, We used Decision Tree (DT) with a max depth of 5, Gradient Boosting (GB), Gaussian Naïve Bayes (GNB), Random Forrest (RF), and SVM with two different kernels: *Linear* and *Gaussian (rbf)*.

Figure 6 compares the average performance of the classifiers in different datasets for two learning scores: TPR and ACC. The most striking results to emerge from Fig. 6 is that both GB and RF give the highest TPR on average for all of the datasets with more than 99%, which is significantly high. In other words, GB and RF detected more than 99% of injected malicious messages correctly. We achieved this detection rate along with an incredibly high accuracy rate simultaneously.

GB gives the best performance on average, and if there is a need for selecting one single classifier, it would be the best choice among the ones that we studied. Looking at two different kernels of SVM confirm that both perform almost identically in all cases. As a linear kernel gives us a very high accuracy, it deduces that data is linearly separable. Another benefit of this kernel is that it is much less complex than Gaussian (rbf) so linear kernel is a better choice.

For SVM with a linear classifier, the TPR and ACC are very close to 96.75% and 97.46% respectively. GNB and KNN do not generate good results and are not suitable for this problem. KNN is more suitable for clustering problems but the results show it does not work for this problem at all.

5.5 Timing Analysis

We want to detect any abnormal behavior in a real-time manner when the vehicle is moving. We need to demonstrate that our approach is efficient and can be installed on the vehicle. We selected Raspberry Pi Zero as a light-weight computer powered by a +5.1V micro USB supply and performed experiments on it. This device can read the CAN data via a connection to the OBD-II port.

Based on our proposed model, we can train the dataset before and save the trained model on the Raspberry Pi for online detection; therefore, we report the testing time only. For each dataset that includes multiple PGNs, we trained multiple classifiers for each PGN. Then we extracted the average testing time for each classifier-PGN. As the model can predict the next message in parallel (based on multiple trained classifiers), the longest testing time of each classifier-PGN will be the total testing time for each input. Consequently, we used the maximum testing time of classifier-PGNs in each dataset.

Figure 7 depicts testing time in a base-10 logarithmic scale in microseconds. DT has the best testing time among all classifiers. This classifier responds in less than six microseconds for all of the datasets. GB is the second best classifier which responds between 70–80 μs followed by GNB which takes around 200 μs. KNN did not perform well at all; it takes at least 16 ms for the best results. This is due to the way that this classifier predicts input samples and finds the k-closest to the input and labels it. This timing analysis shows that the best performing classifiers also act fast-enough to be employed in the heavy vehicle and run experiments in real-time.

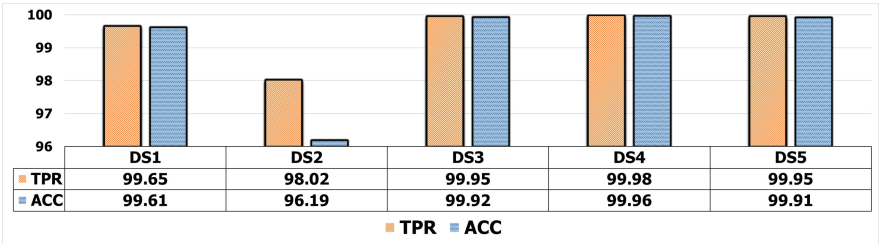


Fig. 4. The weighted average scores for each data log file

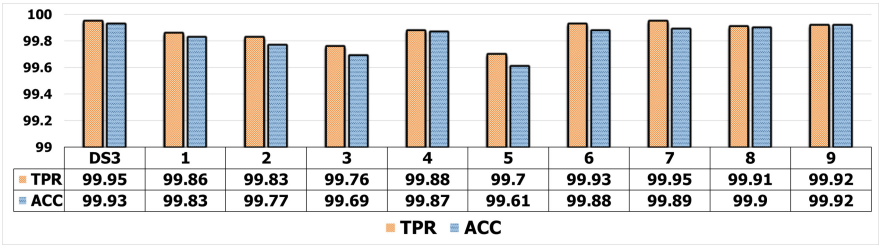


Fig. 5. TPR and ACC scores for different driving scenarios

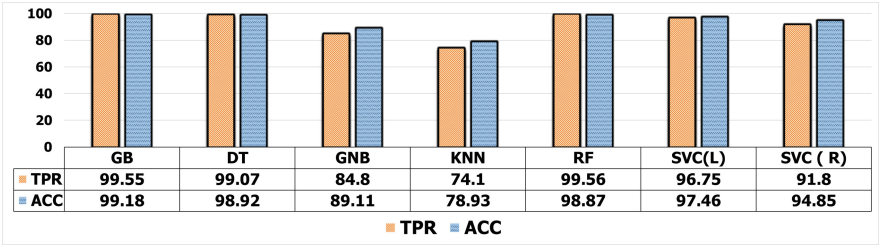


Fig. 6. TPR and ACC scorse for multiple classifiers on different datasets

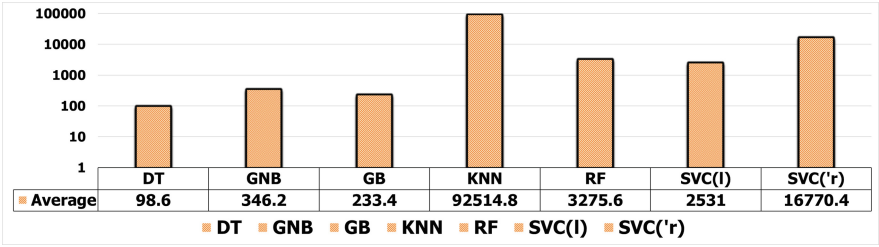


Fig. 7. Average computation time for each dataset for different classifiers

Table 2. Comparing discussed defense mechanisms from different perspectives

Authors	Description	ACC	Type	Techniques
Murvay *et al.* [9]	-Including message authentication to the protocol -Evaluated communication and computational overhead -It needs changes in existing technologies -Not useful for current vehicles	100%	Proactive	Message authentication
Cho *et al.* [3]	-Proposed an clock-based IDS -Detects based on clock skew of periodic messages	99.5%	Reactive	ML
Kang *et al.* [5]	-Intrusion detection technique using a DNN. -Packets are trained to extract features -Features have no semantic associated with them -May not be applicable for online detection due to the performance issue	99%	Reactive	ML
Chockalingam *et al.* [6]	-Studied at two different types of anomalies -Using LSTM helps to consider a sequence of inputs	82% to 100.0	Reactive	ML
Narayanan *et al.* [7]	-Introduced OBD Secure-Alert -Detecting abnormal behavior as a plug-n-play device -Added previous messages to each instance for detection	100%	Reactive	ML
Mukherjee *et al.* [20]	-Used a RPG anomaly detection technique -Visualized temporal relationships of human actions and functions and implemented by ECUs -Find out malicious intrusions in real time	N/A	Reactive	Precedence graph
Our Work	-Fingerprinting behaviour of embedded devices -Identifying regular behaviour of devices -Detecting anomalies using machine learning techniques -Carrying out experiments on five large datasets	96–99%	Reactive	ML

The lowest transmission rate of the message defined in the SAE-J1939 is 10 ms. Other messages are also not being generated faster than that rate. Our proposed mechanism can detect anomalies in order of 200 to 300 µs for two of

our best classifiers. Thus, our algorithm detects anomalies 40–50 times faster than transmission rates for messages on the bus.

Table 2 compares five different approaches in the literature. Some of the previous research leveraged machine learning techniques to find anomalies in an online [23] or offline manner [5,6] in the vehicle. [9] introduced prevention approaches which include introducing message authentication to the protocol. [20] suggested a graph-based approach that detects malicious messages in real time. We also added the results of this study to Table 2 where logs taken from vehicle driving have been injected with abnormal messages.

6 Conclusion and Future Work

The identification of normal behavior of embedded devices is a substantial step towards detecting any abnormal activities. However, as our detection mechanism shows, it is possible to fingerprint the behavior of embedded devices in the vehicles with high accuracy. In future, we want to combine SPN values from multiple PGNs into one single machine learning vector instead of looking at each PGN separately. To achieve this goal, we need to consider the time-series analysis and different types of classifiers.

In future, we plan to explore the use of unsupervised machine learning as labeled data may not be available. In our study, for four out of five datasets, the rate of false positives is less than 0.05%, which is significantly low. We plan to lower this even further.

Acknowledgement. We thank Landon Zweigle for reading through the manuscript. This work was supported in part by NSF Award Number CNS 1715458.

References

1. Checkoway, S., et al.: Comprehensive experimental analyses of automotive attack surfaces. In: USENIX Security Symposium (2011)
2. Miller, C., Valasek, C.: Remote exploitation of an unaltered passenger vehicle (2015)
3. Cho, K.-T., Shin, K.G.: Fingerprinting electronic control units for vehicle intrusion detection. In: USENIX Security Symposium (2016)
4. Wolf, M., Lambert, R.: Hacking trucks-cybersecurity risks and effective cybersecurity protection for heavy duty vehicles (2017)
5. Kang, M.-J., Kang, J.-W.: A novel intrusion detection method using deep neural network for in-vehicle network security. In: Vehicular Technology Conference (2016)
6. Chockalingam, V., Larson, I., Lin, D., Nofzinger, S.: Detecting attacks on the CAN protocol with machine learning. In: Annual EECS 588 Security Symposium (2016)
7. Narayanan, S.N., Mittal, S., Joshi, A.: OBD_SecureAlert: an anomaly detection system for vehicles. In: International Conference on Smart Computing (2016)
8. Bosch, R., et al.: CAN specification version 2.0 (1991)
9. Murvay, P.-S., Groza, B.: Security shortcomings and countermeasures for the SAE J1939 commercial vehicle bus protocol (2018)

10. SAE J1931, Data Link Layer (2016)
11. SAE J1939, Digital Annex (2015)
12. Mirkovic, J., Reiher, P.: A taxonomy of DDoS attack and DDoS defense mechanisms (2004)
13. Szilagyi, C.J.: Low cost multicast network authentication for embedded control systems. Ph.D. thesis (2012)
14. Nilsson, D.K., Larson, U.E., Jonsson, E.: Efficient in-vehicle delayed data authentication based on compound message authentication codes. In: Vehicular Technology Conference (2008)
15. Van Herrewege, A., Singelee, D., Verbauwhede, I.: CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus. In: Workshop on Lightweight Cryptography (2011)
16. Murvay, P.-S., Groza, B.: Source identification using signal characteristics in controller area networks. In: Signal Processing Letters (2014)
17. Choi, W., Joo, K., Jo, H.J., Park, M.C., Lee, D.H.: Voltageids: low-level communication characteristics for automotive intrusion detection system. Trans. Inf. Forensics Secur. **13**, 2114–2129 (2018)
18. Choi, W., Jo, H.J., Woo, S., Chun, J.Y., Park, J., Lee, D.H.: Identifying ECUs using inimitable characteristics of signals in controller area networks. Trans. Veh. Technol. **67**, 4757–4770 (2018)
19. Taylor, A., Leblanc, S., Japkowicz, N.: Anomaly detection in automobile control network data with long short-term memory networks. In: International Conference on Data Science and Advanced Analytics (2016)
20. Mukherjee, S., Walker, J., Ray, I., Daily, J.: A precedence graph-based approach to detect message injection attacks in J-1939 based networks. In: International Conference on Privacy, Security and Trust (2017)
21. Hoppe, T., Kiltz, S., Dittmann, J.: Security threats to automotive CAN networks – practical examples and selected short-term countermeasures. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 235–248. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87698-4_21
22. Burakova, Y., Hass, B., Millar, L., Weimerskirch, A.: Truck hacking: an experimental analysis of the SAE J1939 standard. In: Workshop on Offensive Technologies (2016)
23. John, G.H., Langley, P.: Estimating continuous distributions in Bayesian classifiers. In: Uncertainty in Artificial Intelligence (1995)