Computers & Security 94 (2020) 101857

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

# CAN-ADF: The controller area network attack detection framework

Shahroz Tariq[a,1], Sangyup Lee[a,1], Huy Kang Kim[b], Simon S. Woo[c,*]

[a] Department of Computer Science & Engineering, Sungkyunkwan University, Suwon, South Korea
[b] Information Security Department, Graduate School of Information Security, Korea University, South Korea
[c] Department of Applied Data Science, SKKU Institute for Convergence, Sungkyunkwan University, Suwon, South Korea

ARTICLE INFO

*Article history:*
Received 30 September 2019
Revised 20 April 2020
Accepted 22 April 2020
Available online 28 April 2020

*Keywords:*
Intrusion detection
Security and privacy
Recurrent neural network
Controller area network
In-Vehicle network

ABSTRACT

In recent years, there has been significant interest in developing autonomous vehicles such as self-driving cars. In-vehicle communications, due to simplicity and reliability, a Controller Area Network (CAN) bus is widely used as the de facto standard to provide serial communications between Electronic Control Units (ECUs). However, prior research reveals that several network-level attacks can be performed due to the lack of defense mechanisms in the CAN bus. In this work, we propose CAN Bus Message Attack Detection Framework (*CAN-ADF*) - a comprehensive anomaly generation, detection, and evaluation system for a CAN bus. In CAN-ADF, not only various anomalies and attack characteristics can be configured but also different detection methods, and visualization frameworks are provided to effectively detect those attacks and anomalies. For the detector, we employ both a rule-based approach crafted from dynamic network traffic characteristics and Recurrent Neural Networks (RNN). For evaluation, we use 7,875,791 in-vehicle CAN packets collected from real cars, KIA Soul and Hyundai Sonata. Our detection algorithm achieves accurate intrusion detection performance, with an average accuracy of 99.45% on CAN datasets, outperforming prior approach. Furthermore, we developed a visualization tool to validate the detection of anomalies by CAN-ADF and to find new patterns in the dataset.



## 1. Introduction

As manifested in this year's Consumer Electronics Show (CES, 2018), self-driving cars and related vehicle technologies are one of the most rapidly emerging areas that many companies are interested for investment. At CES 2018, a total of 556 businesses exhibited in automotive and vehicular technology domain, which include not only major automotive companies such as Ford, Mercedes-Benz, Toyota, and Hyundai but also, other Internet corporations such as Amazon, Intel, NVIDIA, and Cisco (Levy and Kolodny, 2018). This area has become a clear rendezvous and partnership point between traditional car manufacturers and the largest Internet companies (Levy and Kolodny, 2018). Furthermore, with substantial interests and advancements in Internet of Things (IoT), we can easily envision that this automotive technology will seamlessly be connected to other IoT devices, clouds, and other cyber-physical systems (CPS) via wireless and cellular connections. However, as vehicles connect to other networks, typical network attacks such as Denial-of-Service (DoS), packet injection, and spoofing attacks can be performed to connected vehicles. Security researchers have found that cyberattacks can be exploited by manipulating data within the car's internal control bus (Lee et al., 2017; Song et al., 2016). This can enable an attacker to control over physical subsystems such as the steering wheel, engine, and brakes, which can potentially endanger passengers or pedestrians. Furthermore, new attack vectors will arise, which exploit the specific protocol vulnerabilities in automotive technology. Therefore, extenuating and mitigating cyberattacks require planting and strengthening various parametric defense mechanisms throughout the vehicle.

For in-vehicle communication, the Controller Area Network (CAN) bus (Wikipedia contributors, 2018a) has been extensively used, which is a de facto standard for serial communication, to provide an efficient, reliable, and economical link between Electronic Control Units (ECUs). The CAN bus is a broadcast-based communication protocol that supports the maximum baud rate up to 1 Mbps on a single bus, developed by Bosch in 1985. Owing to its small wiring cost, weight, and simplicity, many car manufactures have adopted the CAN bus. However, there are several weaknesses in the CAN bus, due to its simplicity and efficiency. Because the receiving node does not validate the origin of a CAN message, many network traffic injection attacks are possible, as shown by Lee et al. (2017) and Song et al. (2016). As a result, various net-

* Corresponding author.
*E-mail addresses:* shahroz@g.skku.edu (S. Tariq), sangyup.lee@g.skku.edu (S. Lee), cenda@korea.ac.kr (H.K. Kim), swoo@g.skku.edu (S.S. Woo).
[1] Both authors contributed equally to this work.

https://doi.org/10.1016/j.cose.2020.101857
0167-4048/

work attacks are easily performed and practically deployable on the CAN bus. Examples include flooding the bus with messages intended to override the legitimate ones, or using spoofed identifiers from the bus with arbitrary data. Furthermore, much more sophisticated and stealthy attacks can be generated, which appear to be legitimate-looking traffic sequences and they are difficult to be distinguished from normal traffic.

To address these challenges, effective detectors and defense mechanisms have to be developed, and evaluated against various types of attacks. In this work, we propose the Controller Area Network (CAN) Bus Message Attack Detection Framework (*CAN-ADF*), a comprehensive attack and anomaly generation, detection, visualization, and evaluation system for a CAN bus. In CAN-ADF, various anomaly and attacks can be generated with different levels of detection difficulty based on real CAN packet characteristics. This allows emulating various types of high fidelity CAN bus attacks. For designing the detector, we employ both a rule-based parametric approach crafted from analyzing dynamic network traffic characteristics and Recurrent Neural Networks (RNN) to detect anomalies and attacks based on the CAN packet sequences by formulating detection as a multiclass classification problem. The final detection algorithm is the ensemble of two different detection methods, complementing one another to yield the best detection performance. We evaluate our approach with CAN datasets collected from two different cars, KIA Soul and Hyundai Sonata, after driving more than 24 h with 10 different drivers. Also, we developed a visualization and monitoring tool to display normal and attack traffic. These are useful in providing situational awareness information to security experts for their informed decision support in detecting new abnormal events or attacks. Our contributions are summarized below:

- **Novel ensemble-based Attack Detection Framework**: We develop an ensemble-based CAN Attack Detection Framework (CAN-ADF) to successfully detect hazardous DoS, fuzzing, and replay attacks during the operation of a vehicle.
- **Evaluation with real datasets**: We evaluate the accuracy of our method with the CAN dataset collected from two real vehicles, KIA Soul (Wikipedia contributors, 2018c) and Hyundai Sonata (Wikipedia contributors, 2018b). Our experimental result shows that our algorithm achieves an average accuracy of 99.45% compared to 83.3% of the state-of-the-art OTIDS method (Lee et al., 2017) in detecting DoS, fuzzing, and replay attacks.
- **Monitoring and visualization tool**: We build a CAN traffic monitoring and visualization tool, which displays the CAN bus traffic status and shows detected attacks. Our visualization tool is useful in validating and further detecting unseen anomalies and new attacks by providing information to operators or drivers.

Our research shows highly promising results in detecting diverse types of serious attacks and it can integrate into a vehicle as an effective defense mechanism. We find that our approach utilizes the best of both worlds, where the first packet signature method can better detect the DoS and fuzzing attack, while RNN model performs better in detecting replay attack more effectively.

The remainder of this paper is organized as follows: the brief overview of CAN protocol and traffic analysis are provided in Section 2. The attack generation process, details of our detection approach and algorithm are explained in Section 3. Details of our visualization tool are provided in Section 3.4. Then, we show our experiments and results in Section 4. We offer discussion and describe limitations in Section 5. In Section 6, we discuss recent automotive security research directly relevant to our work and conclude in Section 7.

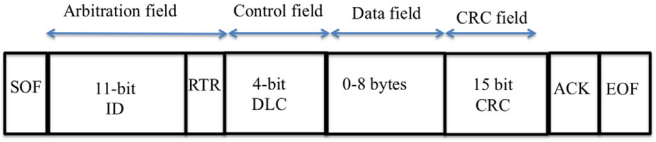



**Fig. 1.** CAN bus frame format.

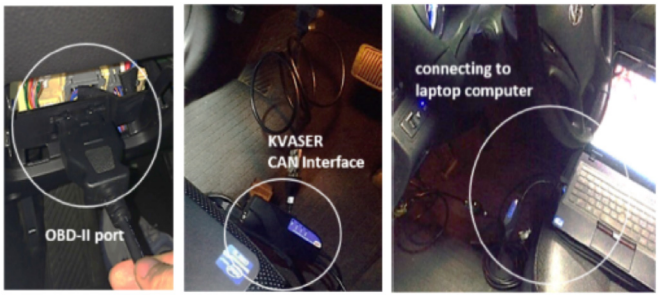



**Fig. 2.** CAN data collection using a laptop connected to OBD-II Port via KVASER CAN interface in KIA Soul.

## 2. Background on CAN BUS

CAN specification version 2.0 by Bosch et al. (1991) is a broadcast-based communication protocol. It is the primary mean of communication for the embedded Electronic Control Units (ECUs) in modern cars. ECUs manage all facets of car's behaviors, where typically cars have two CAN buses: the first type is a high-speed one (about 500Kbps) and the second type is a slower one (about 125kbps) which is dedicated to the passenger's compartment features. Data is transmitted on CAN bus in frames, where these frames contain an arbitration ID, data payload ($0-64$ bits), and additional low-level bus control fields as shown in Fig. 1. In this protocol, there are four types of standard CAN frames: (1) **data frame** for data transmission, (2) **remote frame** for requesting a data frame to a destination node, (3) **error frame** to notify an error in the currently transmitted frame, and (4) **overload frame** to delay the next message until the current message is being processed. The ID and data payload in a frame can be used as the main features for creating attacks in most of the scenarios. The ID is used for identification as well as for bus arbitration, where lower IDs have higher priority when two ECUs attempts to transmit simultaneously. For example, '0000' will have the highest priority among all. The data frames contain control instructions and status information about the vehicle state, usually in a proprietary format.

### 2.1. CAN traffic analysis

We first analyzed high-speed CAN bus traffic to better understand the typical CAN bus traffic during the typical operation of a car. This allows us to characterize the typical and normal CAN bus traffic. We use two different cars, KIA Soul (Car 1) and Hyundai Sonata (Car 2) manufactured by different companies to better understand any varying behaviors. After driving more than 24 hours with 10 different drivers, we captured 7,875,792 packets of data. For example, Fig. 2 shows connecting a laptop to OBD-II port using KVASER CAN interface to collect packets in the CAN bus while driving.

We found that each car used different frame IDs for identical operations. This is expected as each car is manufactured by a different company. However, we found high a correlation among the occurrences and sequences of priority bits between Car 1 and 2. For example, the specific IDs used for car braking system are different in both cars, but they have higher priority compared to IDs generated from other devices. A sample of different CAN packets is

**Table 1**
Sample CAN packets, showing different ID and DLC fields.

| Timestamp | ID | DLC | Data |
|---|---|---|---|
| 1479246664.162438 | 0153 | 8 | 00 21 10 ff 00 ff 00 00 |
| 1479246664.164967 | 02b0 | 5 | bd ff 00 07 dc |
| 1479246664.165822 | 043f | 8 | 00 40 60 ff 58 28 08 00 |
| 1479246664.171065 | 05f0 | 2 | f4 00 |
| 1479246664.171695 | 0002 | 8 | 00 00 00 00 00 06 01 a2 |

shown in Table 1, where the 1st column represents the timestamp of packet arrival, 2nd column contains the ID of sender (ECU), total number of bytes occupied by the data field is given in the 3rd column, and 4th column contains the data field. As shown in Table 1, we observed the different values of ID, DLC, and data values for various CAN packets. More specifically, we found 45 and 29 unique frame IDs are employed by Car 1 and Car 2 on their CAN bus, respectively.

We observed that each ID is periodically transmitted between the range of 10ms to 1s. While the majority IDs have different data payload values over the full extent of our data collection periods, we observed that a small subgroup of data fields used in certain IDs remained nearly unchanged. For example, ID value with '018f' in Car 2 has a DLC of '8' (i.e. the data field contains 8 bytes) and 'fe 2900 00 003c00 00' is a data field. We observed that in the whole data only the **4**th,**11**th and **12**th byte in the data field of '018f' were changing and every other byte remained the same. The data field 'fe 2b00 00 004100 00' is another instance with ID value '081f', which exhibited the same behavior. Moreover, many other IDs showed the similar behavior patterns on different set of bytes. We performed this data analysis on an attack free dataset, which is collected during normal driving conditions. Therefore, finding any patterns in this dataset tell us about the normal behavior of different IDs in the car. Moreover, attacks such as DoS and Fuzzing use arbitrary data packets; therefore, if we know how the regular traffic pattern looks like, then we can distinguish between normal and abnormal traffic patterns. However, only this distinction and heuristics are not enough to detect the attacks, especially for more sophisticated attacks such as Replay attack. Therefore, we employed a Heuristics + RNN based method for detection.

To obtain a deeper insight of the structure of the CAN frames, we employed an algorithm by Markovitz and Wool (2017) to inspect the details of bits and further classify them. The algorithm by Markovitz and Wool (2017) classifies adjoining bits that behave as constant, multivalued, or sensor values, where constant fields are defined as not changing, multivalued fields are those that take a small number of values relative to their possible number, and sensor fields take many different values. Using the approach by Taylor et al. (2018), we also added a strict counter detector to the algorithm for IDs with non-recurring growing values. Also, we further categorized the sensor fields according to their variability. We observed three clusters based on the numbers of unique values in the field, which we termed low (under 65), medium (between 115 and 475), and high (over 1800). For example, the IDs 0044, 051a and 0018 are low, 0587, 04f0 and 01f1 are medium, and 0153, 0220 and 0164 are high. These results are relatively consistent with Taylor et al. (2018) observations. After analyzing the normal CAN bus traffic sequences and structures, we summarize our two main findings as follows:

- The majority of the IDs produce seemingly random data that has no clear common representations, while the relative ordering of priority bits in IDs shows some correlations. However, the continuous formation of new data fields means data sequences cannot be represented as a univariate sequence of symbols, confining the applicability of most sequence-based anomaly detection methods.
- CAN bus frames in both cars have a static rate, which allows using frequency-based algorithms to detect anomalies and attacks using its arrival rates.

Above insights obtained from the CAN frame structure and behaviors serve as a basis to define our attack generators and detectors in CAN-ADF.

## 3. CAN-ADF

The goal of CAN-ADF is to generate the anomalies and high fidelity attacks in the CAN bus and further evaluate the detectors. Hence, we first describe the attack generation process in CAN bus. Next, we provide our novel detection mechanisms in CAN-ADF.

### 3.1. Attack generation

The motivation behind this attack generation framework is to emulate and assess a wide variety of cyberattack effects in a car. To achieve this, we observe how cyberattacks manifest on the CAN bus and how we can realistically mimic these impacts. Due to the simplicity of CAN protocol, we assume that attackers can easily exploit and spoof priority bits. Additionally, they can arbitrarily inject random, carefully crafted IDs or payloads.

To hack an automobile, first an attacker needs to gain access to the vehicle's hardware. Second, he must find an hardware components that are associated with the CAN bus. Third, he must send messages on the bus intended to cause certain undesired effects or harms to a vehicle. Bear in mind that the first two steps are conceivable as demonstrated in Koscher et al. (2010). Our main focus is on the third step to attack ECU, where ECUs are directly control physical bus and frames. The ECUs have become one of the main targets for exploitation. Since the typical bus convention changes by specific car's model, attacks must be modified and customized for specific vehicle models. However, most attacks have some shared traits such as changing a data field to cause unexpected and undesired behaviors. For instance, Miller and Valasek (2013) attack on the car assistant system caused the car to move off the road during driving. Similar results can be obtained on any car, if the attacker identifies the ECU IDs for assist system of that vehicle model. The main distinction between the typical frames and the anomalous ones are the field and sequences that are abnormal for the typical operation of the vehicle. The ID field, modified, and non-modified data field values subsequently portray the attack signature on the CAN bus.

We have investigated a large number of research work (Koscher et al., 2010; Lee et al., 2017; Miller and Valasek, 2013; Palanca et al., 2017; Song et al., 2016) in this CAN bus attack. But, to the best of our knowledge, all the solutions provided so far are attack specific and their behavior in an integrated system is unknown. The motivation of this work is to provide a generic attack generation and evaluation platform, which can incorporate detection of different types of attacks, making it easier for the manufacturers to add our framework to their products for testing.

The main purpose of our attack generation framework is to allow altering data fields inside the frame sequence in a way to closely emulate real attacks. To mimic different attack behaviors, we concentrate on one or more of the three categories of data fields defined by Markovitz and Wool (2017) which are provided as sensor, constant, and counter. Hence, in this work, we consider three distinguished frame injection attack classes (DoS, replay, and fuzzing), which they behave very differently with a different level of detection difficulty. Because of the modular design, our framework is flexible enough to incorporate new attack classes and de-

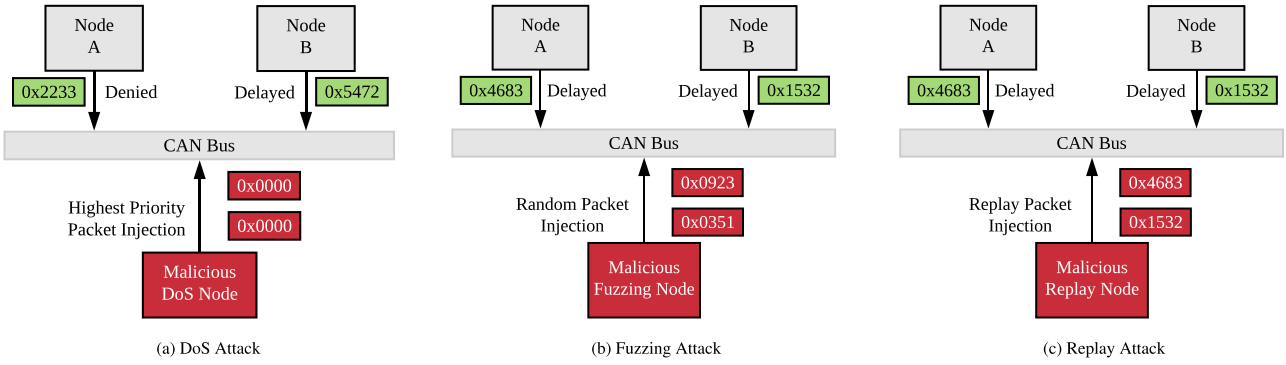



**Fig. 3.** Network-based attacks in CAN bus.

tection methodology, as they become available. Also, there are numerous deterministic attacks abusing the diagnostic packets, yet for now we overlook these, since they could be identified just by looking at a predefined list of IDs and can be included into our framework in the future. Also, for example, to cope with recently introduced Stealthy DoS Attack by Palanca et al. (2017), a separate detection module named "Stealthy DoS Attack" can be integrated into our framework, which provides the detection policy for this attack. Investigation for the detection of sophisticated attacks such as Stealthy DoS Attack (Palanca et al., 2017) is reserved for the future work. The definition of three classes of attacks is provided as follows:

*3.1.1. DoS attack class*

Attackers inject high priority packets in a short time interval on the bus as shown in Fig. 3.(a). Usually, malicious or infected ECUs (devices) can generate DoS attack frames to occupy the CAN bus, using one or multiple high priority IDs. Since all nodes share a single bus, increasing occupancy on the bus can produce delay or deny arrivals of legitimate packets. The DoS attack can cause a vehicle not to respond to the driver's commands on time. This attack is very simple and easy to detect. Therefore, we classify this attack to low-level detection difficulty class.

*3.1.2. Fuzzing attack class*

In this attack class, attackers create messages with randomly spoofed IDs with arbitrary data, as shown in Fig. 3(b). In this case, all nodes would receive spoofed aberrant functional messages. To initiate a fuzzing attack, an attacker would first observe and monitor in-vehicle messages and select target IDs to produce unexpected behaviors. The fuzzing attack is typically generated at a slower rate than DoS attack. However, it is possible to perform fuzzing attack at higher rate. Fuzzing attack generated to the rate of normal traffic is the most challenging one to detect. Again, this attack is also simple but requires slightly more sophisticated detection method. Hence, we categorize this attack into medium-level detection difficulty class.

*3.1.3. Replay attack class*

Attackers can carefully observe data sequences during a specific time interval, and injects and repeats the whole or part of these legitimate message sequences. Each payload contains a valid CAN control message. Hence, it can cause possible damages or unexpected behavior to vehicles. Replay attack is one of the most challenging attack to detect. So we classify it to high-level detection difficulty class. The example of replay attack is shown in Fig. 3(c). Generation of DoS and fuzzing attack traffic is quite straight forward. On the other hand, replay attack generation is more challenging. First, using the OBD-II port, we listen to the traffic on the network and filter out the data coming from one or more arbitrarily selected target node IDs. We store this data with their precise packet arrival time, which is used to mimic or replay this accurately at a later timestamp by injecting these packets into the network. We have used the parametric replay attack packet generation method, which is similar to Taylor et al. (2018), which we discuss more in the related work section.

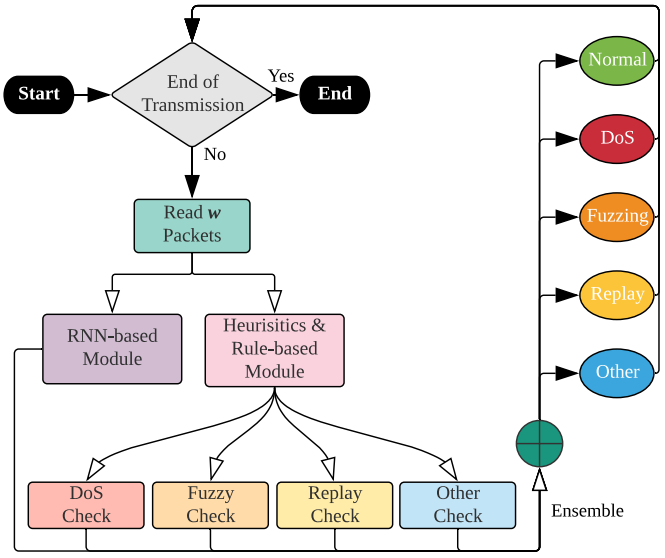



**Fig. 4.** Overall detection algorithm flowchart.

*3.2. CAN Bus detection framework*

In CAN-ADF, we use the ensemble of following two complementary detection methods to improve detection performance: 1) Heuristics/Rule based (H/R) methods derived from analyzing dynamic CAN network traffic signatures and 2) RNN-based method using training large amount of collected CAN data traffic. Specifically, we design three different intrusion detection modules for detecting DoS, fuzzing, and replay attack classes as shown in Fig. 4. For simplicity, we present the high-level functional modules in the flowchart, and explain details in the next section. From the work of Lee et al. (2017) and Song et al. (2016), we designed H/R detection and RNN-based methodologies for detecting each of the three attacks.

*3.2.1. Heuristics / rule-based detection*

In this section, we describe the details of the H/R methods, which uses different CAN packet characteristics for anomaly detection.

**DoS Attack:** The main purpose of DoS attack is to deny or delay the arrival of the normal packets. To achieve this, an attacker typically injects high priority packets in a very short time interval. Therefore, to detect DoS attack, we focus on capturing packet inter-arrival time features. Whenever IDs sent at higher rate than normal, we store it into a list of suspicious IDs and we keep monitoring them. If this behavior persists for a certain time duration (threshold), we classify it as a DoS Attack. Eq. (1) formulates the necessary conditions of heuristic ($\mathbb{H}_{\text{DoS}}$) to classify a packet as DoS or Normal.

$$\mathbb{H}_{\text{DoS}} = \begin{cases} \text{DoS} & \text{T}_{diff} \geq \mathbb{D}, \text{ and } p_{id} \leq 1000 \\ \text{Normal} & \text{T}_{diff} < \mathbb{D}, \text{ and } p_{id} > 1000 \end{cases}, \tag{1}$$

where $\text{T}_{diff}$ is the time difference between two consecutive packets, $\mathbb{D}$ is an empirically calculated upper limit on the time difference threshold, and $p_{id}$ is the packet ID. The detection difficulty for DoS attack is relatively easier to detect which can have a huge impact on network traffic pattern. However, there are some sophisticated DoS attacks such as Stealthy DoS Attack (Palanca et al., 2017) which are very difficult to detect thus have a higher detection difficulty. For this work, our attention is on typical high-rate DoS attack to emulate a low-detection difficulty intrusion.

**Fuzzing Attack:** In a fuzzing attack, data part of packet is arbitrarily generated and scrambled to muddle the normal functions of the car. That is, IDs used for fuzzing attack can be randomly generated or spoofed from the CAN bus. Hence, we first determine whether the message payload is randomized or distinguishable from normal traffic. To detect fuzzing attack, we maintain a list of last few messages in a queue for each node's ID and calculate similarity and distance (e.g. Hamming distance (Hamming, 1950)) between the new data and data in a queue to identify drastically different incoming packets. If the message payload distance is larger than a certain distance, we store this ID into a suspicious ID list. If this behavior persists, we classify it as a fuzzing attack. Eq. (2) formulates the necessary conditions of heuristic ($\mathbb{H}_{\text{Fuzz}}$) to classify a packet as Fuzzing or Normal.

$$\mathbb{H}_{\text{Fuzz}} = \begin{cases} \text{Fuzzing} & \frac{1}{n}\sum_{j=1}^{n} \text{D}_h(d_i^k, d_{i-j}^k) \geq \mathbb{F} \\ \text{Normal} & \frac{1}{n}\sum_{j=1}^{n} \text{D}_h(d_i^k, d_{i-j}^k) < \mathbb{F} \end{cases}, \tag{2}$$

where $d_i^k$ is the data of current packet $i$ with ID$= k$, $d_{i-j}^k$ is the data of a previous packet $i-j$ with ID $= K$, $\text{D}_h$ is the hamming distance between the data of two packets, and $\mathbb{F}$ is an empirically calculated upper limit on the hamming distance threshold for Fuzzing attack. Generally, detection difficulty of the fuzzing attack is higher, requiring more effort and background information than DoS attack.

**Replay Attack:** For the replay attack, the attacker captures in-vehicle messages and rebroadcasts those in the future to cause unexpected behaviors. Since the message structure for replay attack is similar to normal packets, it is very challenging to detect them using only data payload information. Nevertheless, there are some indirect ways of detecting replay attack. If the attacker tries to block some node and replace that node with its own replay data, then it can be easily detected by simply observing the faulty/blocked nodes. Therefore, in replay attack, the attacker injects replay attack packets in the presence of normal traffic. Because of this reason, the CAN bus is typically congested during the replay attack period and thus, packet inter-arrival time between CAN frames will be smaller than a normal traffic threshold.

However, a more sophisticated attack would be to rebroadcast only a small number of IDs from the captured data. In this way, it will not decrease the packet inter-arrival time that much and it is much difficult to detect. However, since both replay attack packet and normal traffic packet will have the same IDs, the arbitration will cause some of packet's IDs to be adjacent to each other, which is a strong indication of repeated (replay) sequence of packets. Also, by leveraging Markovitz and Wool (2017) work which categories the data payload fields into sensor, constant, and counter types, we can observe and measure the dissimilarity of these adjacent packets with Hamming distance. If the message payload distance is greater than a certain threshold (not as drastically different as it was in the case of fuzzing attack), it is the second strong indication of replay attack. If both conditions are satisfied, we store this ID into a suspicious ID list. If the behavior persists or happens across multiple IDs, then we classify it as a replay attack. Eq. (3) formulates the necessary conditions of heuristic ($\mathbb{H}_{\text{Re}}$) to classify a packet as Replay or Normal.

$$\mathbb{H}_{\text{Re}} = \begin{cases} \text{Replay} & \text{ID}_i = \text{ID}_{i-1}, \text{ and} \\ & \text{D}_h(d_i, d_{i-1}) \geq \mathbb{R} \\ \text{Normal} & \text{ID}_i \neq \text{ID}_{i-1} \end{cases}, \tag{3}$$

where $\text{ID}_i$ is the ID of current packet, $\text{ID}_{i-1}$ is the ID of previous packet, $\text{D}_h$ is the hamming distance between the data of packet $i$ and $i-1$, and $\mathbb{R}$ is an empirically calculated upper limit on the hamming distance threshold for Replay attack.

### 3.2.2. RNN-based detection

To overcome the limitations of rule-based detection procedure, we use neural networks to better generalize the detection of anomalies and attacks in CAN Bus traffic. Specifically, we developed the Recurrent Neural Network (RNN)-based approach, where RNN is a type of artificial neural network. In RNNs, the connections between nodes form a directed graph along a sequence. It enables the RNNs to represent temporal dynamic behavior for a time sequence. RNNs can use its memory to process sequence of inputs, making RNNs to effectively model time series data (Connor et al., 1994) as well as tasks with unsegmented handwriting recognition (Graves et al., 2009) or speech recognition (Li and Wu, 2015; Sak et al., 2014).

Considering that RNN can successfully model time-dependent data sequences for various classification and prediction applications, we trained our RNN model with CAN dataset. And we construct the anomaly and attack detection as the following multiclass classification problem as shown in Fig. 5: Normal, DoS, fuzzing, replay, and other, where the input to RNN is CAN packet sequences. For selecting features in input packets, we used the following attributes: time interval between packets, ID, DLC and data payload as described in Fig. 1. Data payload is further divided into 8 sub features.

Fig. 5 presents the detailed structure of the RNN, where we use fully connected dense layers with Long Short-Term Memory (LSTM). The input layer takes 40 consecutive CAN packets as a sequence with 11 features each ($40 \times 11$). Next three layers are fully connected dense layers with 256, 512 and 1024 hidden parameters and a linear rectifier unit (ReLU) is used as an activation function. After each dense layer, we added the batch normalization and applied different dropout rates. Fifth and sixth layers are LSTM layers with 512 hidden cells each. Between the two LSTM layers, we added the layer normalization. We also applied a *L2* regularizer on each layer. The final output layer has 5 classes which can classify from Normal, DoS, fuzzing, replay attacks, and other class. We formulate this classification task as a maximization problem in Eq. (4).

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y), \tag{4}$$

where $x_i$ is the input sample, $y$ is the true label, $P(y)$ is the prior, and $P(x_i|y)$ is the likelihood.

### 3.2.3. Final ensemble method

The motivation of applying the ensemble method is to combine and choose the best results from two different detection methods.

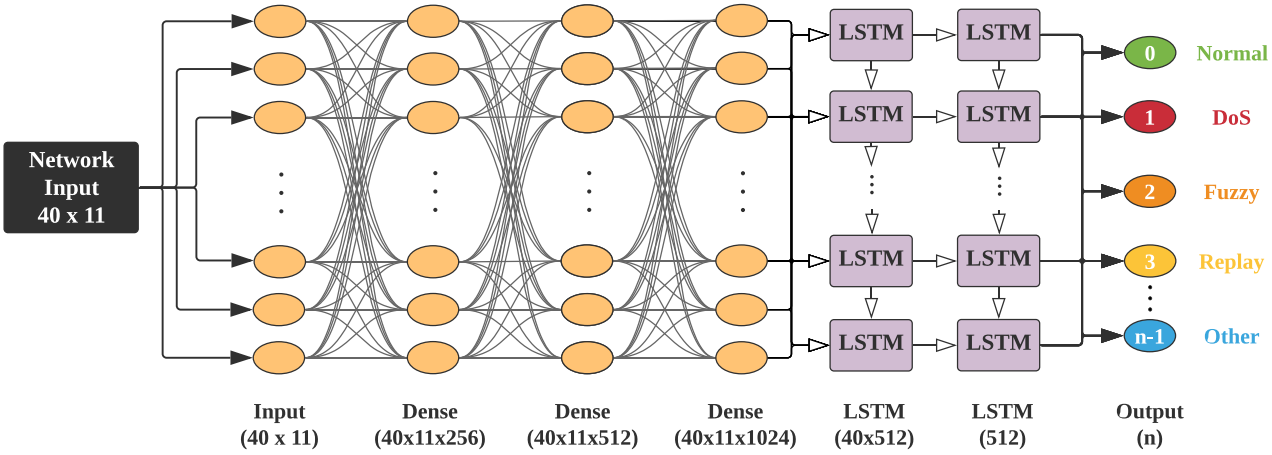



**Fig. 5.** RNN-based detection method. Here $n = 5$, where $n$ is the number of total attack types.

Our assumption is that each H/R and RNN-based approach may be good at detecting different types of attack classes. Hence, the ensemble method can provide the best final label from H/R-based and RNN-based approach as shown in Fig. 4. Using the ensemble method, we can produce the highest likelihood label and improve the detection. In the result section, we will evaluate how each detector perform in detecting different types of attack classes. In Eq. (5), we define the decision conditions for H/R based method.

$$\mathbb{H}/\mathbb{R}_{\mathrm{D}} = \begin{cases} 1, & \mathbb{H}_{\mathrm{DoS}} = \text{DoS} \\ 2, & \mathbb{H}_{\mathrm{Fuzz}} = \text{Fuzzing} \\ 3, & \mathbb{H}_{\mathrm{Re}} = \text{Replay} \\ 0, & \mathbb{H}_{\mathrm{DoS}} = \mathbb{H}_{\mathrm{Fuzz}} = \mathbb{H}_{\mathrm{Re}} = \text{Normal} \end{cases} \tag{5}$$

We formulate the decision conditions for our RNN-based method in Eq. (6).

$$\mathbb{RNN}_{\mathrm{D}} = \begin{cases} 0, & \hat{y} = 0 \text{ (Normal)} \\ 1, & \hat{y} = 1 \text{ (DoS)} \\ 2, & \hat{y} = 2 \text{ (Fuzzing)} \\ 3, & \hat{y} = 3 \text{ (Replay)} \end{cases} \tag{6}$$

Similarly, Eq. (7) defines the decision conditions for our ensemble-based method.

$$\mathbb{ENS}_{\mathrm{D}} = \begin{cases} 0, & \mathbb{RNN}_{\mathrm{D}} = \mathbb{H}/\mathbb{R}_{\mathrm{D}} = 0 \\ 1, & \mathbb{RNN}_{\mathrm{D}} = \mathbb{H}/\mathbb{R}_{\mathrm{D}} = 1 \\ 2, & \mathbb{RNN}_{\mathrm{D}} = \mathbb{H}/\mathbb{R}_{\mathrm{D}} = 2, \\ 3, & \mathbb{RNN}_{\mathrm{D}} = \mathbb{H}/\mathbb{R}_{\mathrm{D}} = 3 \\ f(\mathbb{RNN}_{\mathrm{D}}, \mathbb{H}/\mathbb{R}_{\mathrm{D}}), & \mathbb{RNN}_{\mathrm{D}} \neq \mathbb{H}/\mathbb{R}_{\mathrm{D}} \end{cases} \tag{7}$$

In Eq. (7), if $\mathbb{RNN}_{\mathrm{D}}$ is not equal to $\mathbb{H}/\mathbb{R}_{\mathrm{D}}$, then we use Eq. (8) as a tiebreaker. In Eq. (8), we take the result from the RNN-based method only if the decision probability of RNN is very high ( $\geq$ 80%); otherwise, we take the result from the H/R based method.

$$f(\mathbb{RNN}_{\mathrm{D}}, \mathbb{H}/\mathbb{R}_{\mathrm{D}}) = \begin{cases} \mathbb{RNN}_{\mathrm{D}}, & p(\mathbb{RNN}_{\mathrm{D}}(\hat{y})) \geq 0.80 \\ \mathbb{H}/\mathbb{R}_{\mathrm{D}}, & p(\mathbb{RNN}_{\mathrm{D}}(\hat{y})) < 0.80 \end{cases}, \tag{8}$$

where $p(\mathbb{RNN}_{\mathrm{D}}(\hat{y}))$ is the probability of the predicted class label for input $x_i$ as given in Eq. (4).

### 3.3. Evaluation description

To evaluate our method and the baseline method OTIDS on a set of different attack scenarios using the Car1 and Car2 datasets. These scenarios include: (1) Single instance of attack (DoS, Fuzzing or Replay), (2) multiple instances for the same type of attack, (3) a non-overlapping mix of different attacks. In the result section, we present an overall detection performance of each attack based on these three scenarios. We used F1-score to evaluate the performance of our method on Car1 and Car2 datasets and also against the OTIDS method. Furthermore, for each car, we calculate the attack detection time-delay of our model. Finally, we used our visualization tool to validate the attack detection of our CAN-ADF method.

### 3.4. Visualization tool

The visualization and monitoring tool can illustrate the inter-arrival packets within a specific time window. We built the tool using Python 3.6 with the Matplotlib library. After acquiring CAN data packets using KVASER CAN interface (Fig. 2), we can use the packets to visualize. Since the data that is used as an input consists of row-based data packets, the tool waits for the packets to arrive into the CAN bus for 40ms, and plot the number of packets arrived. In order to color the different attack types and normal state, we used the attack labels generated from our CAN-ADF Detector, as shown in Fig. 7. We developed the CAN traffic visualization tool to achieve the following two main objectives: (1) Visualize, monitor and better understand behaviors of each attack type and (2) Identify and detect anomalous events or new attack patterns.

In developing the details of the visualization tool, we focus on capturing the details of the incoming packets in the CAN bus near real-time. For this reason, we first measure the time difference $\Delta_t$ between each packets' arrival time. If an injection attack occurs in the CAN bus Line, the $\Delta_t$ will shrink as shown in Fig. 6. This behavior indicates that there are more traffic on the bus during the injection attack. In order to utilize this behavior to the visualization process, we then calculate the number of packet arrivals $w_v$ in the bus within a certain time window $w$ to differentiate an attack from a normal state. If the number of packets $w_v$ exceeds the normal behavior, we can visualize them as an anomaly. The example outputs of our visualization tool are shown in Fig 7(a) and (b). Based on empirical experiments, we set $w_v$ to 40ms, which is fast enough to capture and plot the difference between the traffic states clearly.

For visualization, we use different colors to indicate different attack types, where DoS, fuzzing, and replay attacks are colored as red, orange, and yellow dots, respectively, and the green dots for normal traffic. Fig. 7(a) shows the normal and fuzzing attack traffic, where the X-axis represents the number of windows ($w_v$), the size of $w_v$ is 40$ms$, and the Y-axis is number of packets in one window. The fuzzing attack packets (shown in orange color) is increased by approximately 50% compared to normal state (shown in

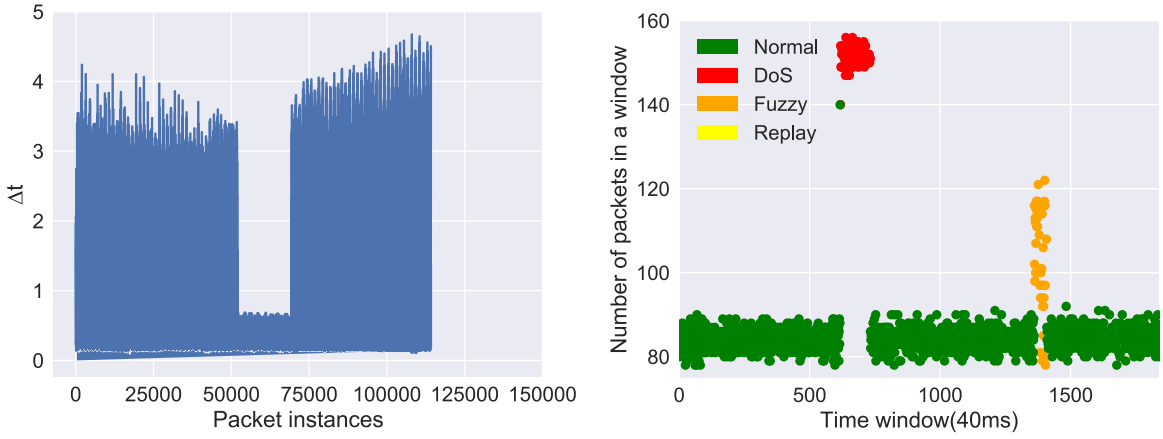
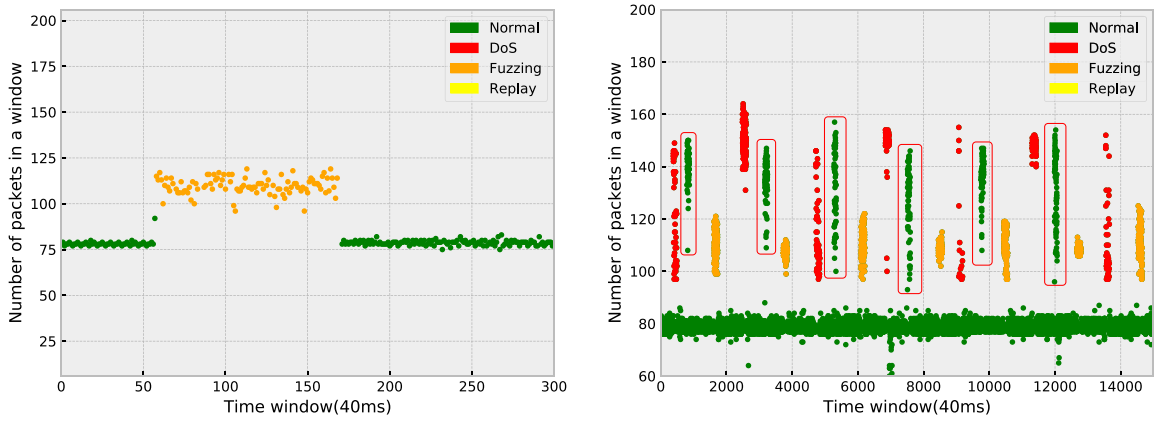



**Fig. 6.** Visualizing the incoming packets.




(a) Fuzzing traffic (shown in orange dots) vs. normal traffic (shown in green dots) with $w_v$ = 40$ms$.

(b) Possible anomaly/suspicious behavior detection, shown in red boxes.

**Fig. 7.** Visualization of different types of attacks.

green color). Also, we provide the video demo[2] to demonstrate our tool's visualization capability. The developed visualization program not only helps users understand behaviors of different attacks, but also it can uncover new types of attack patterns and anomalies, which are not detected from our detection algorithm. For example, in Fig. 7(b), we show the sections colored in green dots (normal traffic) in red boxes. But they are suspicious activities and can be potentially anomalies or new attacks. Upon detection of the possible anomaly or suspicious behaviors, we can record and store the start-time, end-time, and payloads for further analysis to end users.

## 4. Experiments

We conducted several experiments to evaluate detector performance by producing various types of attacks and anomalies in CAN-ADF. We first divide the collected dataset from each car to 70% for training (2,887,500 packets for Car 1 and 2,625,554 packets for Car 2), 25% for validation (1,031,250 packets for Car 1 and 937,698 packets for Car 2) and 5% for the testing of attack and anomaly detection model (206,250 packets for Car 1 and 187,539 packets for Car 2).

Due to the comparatively slow data rate of the CAN bus, all the detectors can be run in real-time speed; nevertheless, this might require dedicated hardware to be installed in a car. Based on the preliminary experimentation, we found that a packet sequence window $w_p$ of 30 to 40 packets is the most appropriate for this task.

The threshold values for H/R method were calculated based on statistical measurements from the training data. Both static and moving average were evaluated for calculating these thresholds and found that moving average method provides more reliable results with training set. Therefore, we chose the moving average for our approach.

### 4.1. Experiment setting

The PC specifications include Intel(R) Xeon CPU E5-1650 v4 @ 3.60GHz CPU, NVIDIA GeForce GTX 1080ti GPU and Linux 16.04 for operating system. For developing the algorithm, Python 3.6 with Keras library for deep learning methods by Chollet et al. (2017) are used.

### 4.2. Implementation details

For the implementation of the attack generation, we followed the attack descriptions mentioned in Section 3.1 and injected DoS, Fuzzing, and Replay Attack into the dataset using the OBD-II Port. However, for the detection, we developed different modules (1) Heuristics and Rule-based module and (2) RNN-based module. We used python programming language without any particular library to implement the Heuristics and Rule-based module by following

[2] Demo of visualization tool: https://youtu.be/a753Hj5C1gQ.

**Table 2**
Pre-processed and normalized dataset, where each packet has 11 features and last column is the label. The majority label in window $w_p$ is given as the label of that $w_p$.

| Time Difference | ID | DLC | Data | | | | | | | | Label |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| 0.000243 | 0.008301 | 1.0 | 0.87451 | 0.011765 | 0.996078 | 0.011765 | 0.035294 | 0.0 | 0.227451 | 0.062745 | 0 (normal) |
| 0.000175 | 0.010498 | 0.625 | 0.274509 | 0.0 | 0.0 | 0.027450 | 0.921568 | 0.0 | 0.0 | 0.0 | 2 (fuzzing) |
| 0.001288 | 0.012940 | 1.0 | 0.019608 | 0.12549 | 0.768627 | 0.356863 | 0.360784 | 0.0 | 0.0 | 0.901961 | 0 (normal) |
| 0.000368 | 0.000030 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 (DoS) |
| 0.000237 | 0.006088 | 1.0 | 0.0 | 0.109804 | 0.141176 | 0.0 | 0.0 | 0.356863 | 0.0 | 0.062745 | 0 (normal) |
| 0.000211 | 0.012055 | 1.0 | 0.019608 | 0.141176 | 0.368627 | 0.054902 | 0.141176 | 0.101961 | 0.0 | 0.498039 | 0 (normal) |
| 0.000854 | 0.012054 | 1.0 | 0.019607 | 0.129411 | 0.470588 | 0.035294 | 0.129411 | 0.117647 | 0.0 | 0.501960 | 3 (replay) |

the descriptions mentioned in Section 3.2.1. However, the implementation of the RNN-based module more complicated, and we used python with TensorFlow and Keras library. The RNN-model has three fully connected (Dense) layers with 256, 512, and 1024 connections for the feature extraction, which are followed by 2 LSTM layers with 512 connections. The last layer is the classification layer with n number of output (n=5). Both modules provide a prediction that is used in an ensemble to produce the final classification result. We train the RNN-based module with both normal and abnormal datasets. Finally, we test the modules by providing w packets at a time to perform the predictions. These predictions are then compared with the class labels to calculate the F1-score. Fig. 4 provides a flowchart of this process.

### 4.3. Pre-processing dataset

In order to better utilize Car 1 and Car2ɾs raw dataset, we performed the pre-processing and normalization on the dataset to produce 11 features. Timestamp is converted into time difference which indicates time interval between current and previous packet. The ID and DLC are normalized by dividing with the maximum of its values which are '`fxfff`' and '8' respectively. Next, data payload is divided into 8 segments. However, when data payload is smaller than 8 bytes, it can not be equally divided into 8 segments. To solve this problem, we took inspiration from the computer vision's concept of zero-paddings. We believe that, just like in the image classification task, the model can understand the zero-padded area in the image is a meaningless area. Therefore, when the data payload is less than 8 bytes, we pad the remaining part with zeros so that it can be equally divided into 8 segments. For example, if DLC=5, then the data payload will look something like this: '`aa bb cc dd ee`' (5 bytes), we pad zero at the end. Therefore, the payload will look like this: '`aa bb cc dd ee 00 00 00`'. There are different methods we can pad zero to "`aa bb cc dd ee`" to make it into 8 bytes such as "`00 00 00 aa bb cc dd ee`" or "`00 00 aa bb cc dd ee 00`'.

However, we believe that all the methods are correct as long as we choose one way and be consistent with it for the whole dataset. For normalization, we divided payloads hex values with "`ff`", which is the maximum value of the raw payload data. Labeling is performed by using packet sequence window $w_p$. For example, if $w_p$ is 30 packets and that window $w_p$ is in "Normal state duration", all those windows" packets are labeled as 0 for normal traffic. Similarly, we labeled 1 for DoS, 2 for fuzzing and 3 for replay based on the aforementioned criteria for convenience. The sample pre-processed and normalized dataset is shown in Table 2. Each packet has 11 features and last column is the label. The majority label in window $w_p$ is given as the label of that $w_p$

### 4.4. Overall performance

We tested our algorithm with both Car 1 and Car 2 dataset, where a total of 119 attacks were generated. Our CAN-ADF was able to successfully detect all DoS attacks (total 43), fuzzing attacks (total 39) and replay attack instances (total 37). We achieved an average F1-score of 99.9% on Car 1 and 98.41% on Car 2 with rule-based method, and RNN-based model yielded 98% on Car 1 and 99% on Car 2. Our ensemble method showed the best performance with 99.9% on Car 1 and 99% on Car 2 dataset. Fig. 8 summarizes our final result on each dataset comparing rule-based, RNN-based model and ensemble method. As we can observe R/H achieves slightly better performance on Car 1, while RNN-based approach achieves slightly higher detection performance of Car 2. The precision, recall and F1-score of ensemble method for detecting (1) nominal traffic are 99%, 100% and 100%, (2) DoS attack are 99%, 100% and 100%, (3) fuzzing attack are 100%, 100% and 100%, and (4) replay attack are 97%, 93% and 95% for H/R, RNN, and ensemble method, respectively.

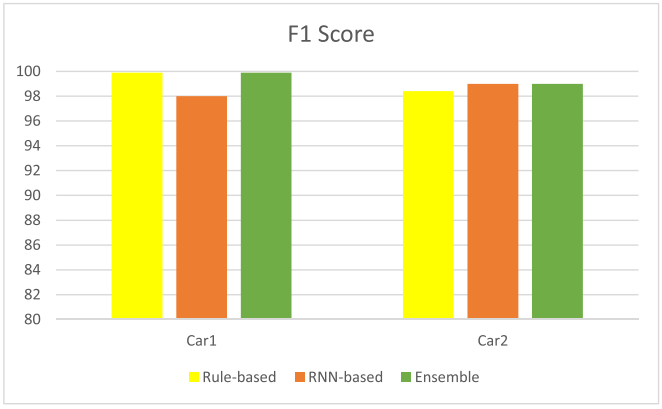



**Fig. 8.** Overall F1-score performance for Car 1 and Car 2.

The H/R method performed better compared to RNN method on Car 1 dataset. Based on our observation we found that Car 1 dataset has significantly more outliers/noise than Car 2, which might have caused this performance decline on RNN method. However, H/R method seems to be less affected by the outliers and noise thanks to the running averages on threshold values. The effect of these outliers on H/R can be also observed in Fig. 9, especially on replay (Car 1) plot. In the future, we plan to explore a better pre-processing and design an improved RNN model, which can can be more robust against outliers and noise.

### 4.5. Attack detection time

Based on the experiments, H/R method detected all **DoS, fuzzing, and replay** attacks within 0.073 *sec* on average. The boxplot representing the time-delay in detecting the attack using H/R method is shown in Fig. 9, where the X-axis represents the different attack types for each car and the Y-axis shows the absolute time differences (*sec*) of when our algorithm detects the attack and when it actually occurred in the dataset. The red line is

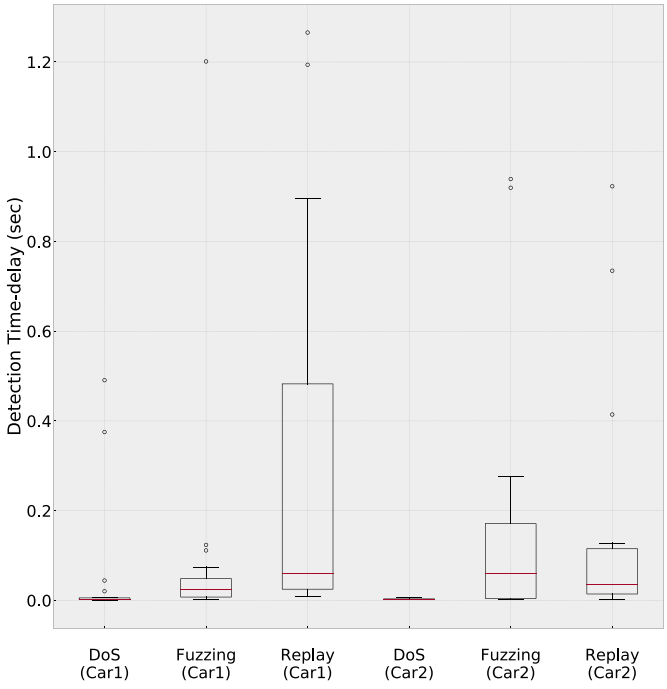



**Fig. 9.** Error box-plot for H/R method, where the X-axis represents the different attack types for each car and the Y-axis is time differences between our algorithm and actual label in *sec*, and the red line is a median and edges of the box are the 25$^{th}$ and 75$^{th}$ percentiles. The whiskers extend to the most extreme data points not considering outliers, and outliers are plotted individually as circles.

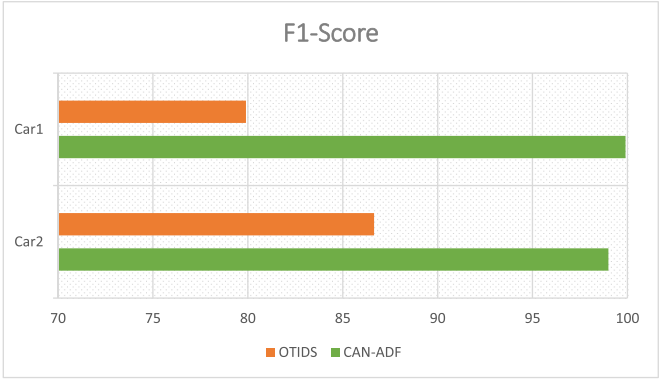



**Fig. 10.** F1-score performance comparison between CAN-ADF vs. OTIDS.

the median and edges of the box are the 25$^{th}$ and 75$^{th}$ percentiles. The whiskers extend to the most extreme data points not considering outliers, and outliers are plotted individually as hollow circles. The RNN method can detect attacks within 0.014 *sec* on average, which is approximately the time interval of single $w_p$. Using the ensemble of H/R and RNN produced an average detection time-delay of 0.020 *sec*. Detection time delay of ensemble method is slightly higher than the RNN method due to the H/R method's high attack detection time delay. However it provides better F1-score.

### 4.6. Comparison to prior research

We conducted direct comparison between CAN-ADF and the state-of-the-art OTIDS intrusion detection algorithm by Lee et al. (2017), where OTIDS uses the offset ratio and time interval between request and response messages in CAN message to detect attacks. We used the same Car 1 and Car 2 dataset for evaluation and the F-1 score result is provided in Fig. 10. As shown in Fig. 10, CAN-ADF outperforms the state-of-the-art OTIDS in both cases by 10 to 15%. We achieved the F1-score of 99.9% using our method versus the 79.9% of OTIDS on Car 1 (KIA Soul) dataset. On Car 2 (Hyundai Sonata) dataset, we obtained the F1-score of 99.0% using CAN-ADF compared to the 86.7% of OTIDS. We believe the OTIDS detection method focuses mainly on the packets' inter-arrival time. If the inter-arrival time of packets rises more than the nominal threshold, it detects those as an anomaly deterministically. We observed that OTIDS does not perform well for some of attack patterns, while our approach performed better due to more features we take into account in our CAN-ADF. Moreover, the time taken to detect any attack is 15% longer in OTIDS than those of our method on average. As a result, OTIDS misclassified many of the initial $w_p$ before it can detect the true anomaly causing performance degradation compared to ours.

### 4.7. Attack visualization

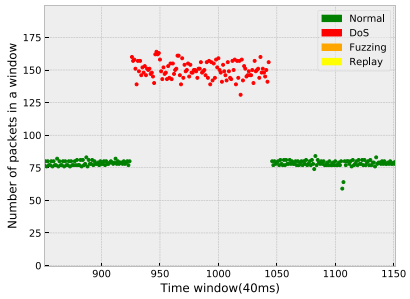
We present the sample outputs produced from our visualization tool in Fig. 11, where the X-axis is the number of time windows $w_v$, and the Y-axis is the number of packets per window ($w_v$ = 40*ms*). From this, not only we can observe how normal traffic and attacks behave, but also we can validate the performance of our algorithm. Fig. 11(a) shows one instance of the DoS attack with red circles, where green circles are nominal traffic. During the DoS attacks, the number of packets in each $w_v$ jumps to near 150, while those in the normal traffic are slightly lower than 90 packets/$w_v$. Fig. 11(b) shows one instance of the fuzzing attack in orange circles. In fuzzing attack, packets per $w_v$ shifts from the nominal range of 80–90 packets to 100–120 which is greater than that of normal traffic. Fig. 11(c) shows one instance of the replay attack in yellow circles. The number of packets sent in one $w_v$ also increases to a range of 100–150 due to the replay injections, in addition to normal traffic. Fig. 11(d) shows the entire data points using our visualization tool. We accurately captured 7 DoS attacks (in red), 7 fuzzing attacks (in orange), and 6 replay attacks (in yellow) from a subset of Car 2 dataset by using the visualization tool as shown in Fig. 11(d).

## 5. Discussion and limitations

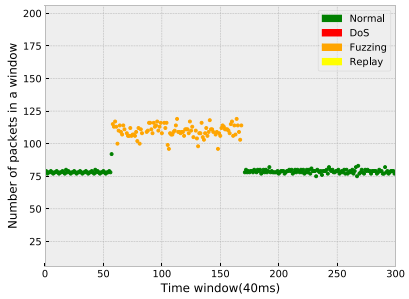
Our algorithm detects the majority of attacks accurately. However, there is much room for improvement especially in the pre-processing of data. In addition, we are currently using the Hamming distance to calculate the difference between two data payloads. However, the Hamming distance is sensitive to the size of the data payload. We plan to evaluate different distance metrics such as Levenshtein distance (Levenshtein, 1966), Damerau-Levenshtein distance (Damerau, 1964), Jaro-Winkler distance (Winkler, 1990) and the most frequent k characters (Seker et al., 2014) in the future.

After detecting attacks, the most important question is how we should respond and what actions or countermeasures we should take. We can envision an integrated safety and defense system as an extension of CAN-ADF with the followings additional countermeasures: (1) Automatically put the car to safety mode. (2) Slow down the car and alert the driver. (3) Send it the attack signature to cloud for further analysis. For example, a service such as General Motors subscription-based communications, in-vehicle security, emergency services, hands-free calling, turn-by-turn navigation, and remote diagnostics systems known as OnStar (OnStar, 2018) can be alerted on detecting an intrusion attack. These types of proactive actions can be made to protect a driver and a vehicle after detecting attacks from CAN-ADF.

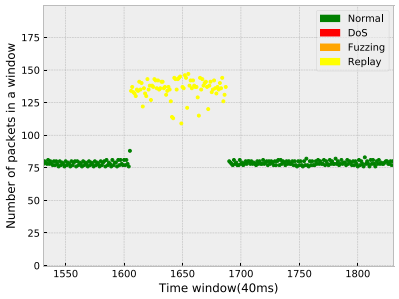
In particular, the effects of detection time delay on properties such as stability and safety of car are considered in this work. The processing time for the analysis of packets is kept to minimum,



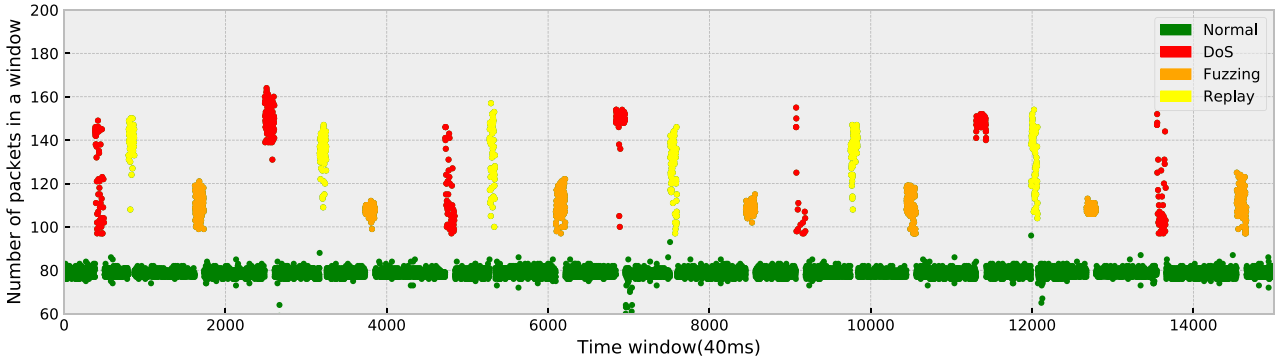

(a) The one instance of the DoS attack shown in red dots, where green dots are nominal traffic. During the DoS attacks, the number of packets per window spikes to approximately 150, while those in the nominal traffic are slightly lower than 90 packets/window.




(b) The one instance of the fuzzing attack shown in orange dots. The fuzzing attack traffic peaks over 120 which is greater than that of normal traffic.




(c) The one instance of the replay attack shown in yellow dots. The number of packets sent in one window also increases due to the replay injections, in addition to normal traffic.




(d) Displaying the entire data points using our visualization tool. We accurately capture the entire 7 DoS attacks (in red), 7 fuzzing attacks (in orange), and 6 replay attacks (in yellow).

**Fig. 11.** Visualization tool validating the performance of our algorithm.

which enables our proposed approach to detect attacks in real-time. With the average detection delay of 20 milliseconds, we can maintain that the stability and safety of the vehicle. In this work, we only considered DoS, fuzzing, and replay attacks, and slightly variations of those. For future work, we aim to generate more diverse and new attack traffic using Generative Adversarial Networks (GAN) to strengthen our detection performance. We also plan to incorporate detection mechanism for Stealthy DoS attack (Palanca et al., 2017). In addition, we need to explore other regularization in CAN-ADF to better generalize defense method. Another limitation is that we only evaluated CAN-ADF with two different cars. However, it might not achieve the expected performance with a car from different manufacturers. An effective strategy to cope with different car models is to explore transfer learning, and we plan to research more in this area. We will also consider reinforcement learning, which can work very well without large data to harness our detection performance.

## 6. Related work

Cho and Shin (2016) proposed a recursive least square algorithm to detect irregularities inside the in-vehicle network. Boudguiga et al. (2016) discussed the lack of security mechanisms in the CAN protocol which makes it vulnerable to DoS, impersonation and isolation attacks. They also proposed a simple intrusion detection method for CAN by making each micro-controller monitor the CAN bus to detect malicious frames. Since the origin of a CAN message is never validated, it is susceptible to spoofing attacks. Wolf et al. (2004) identified such vulnerabilities inside in-vehicle networks. Müter and Asaj (2011) defined the concept of entropy on CAN bus and used it to detect anomalies by associating entropy to a reference set. They also used a more systematized method, which involves sensors to monitor the state of traffic for anomaly detection. A specification based attack detection approach, which was created from CANopen draft 3.01 and CAN 2.0, was used by Larson et al. (2008). Our work is different from the above research, as we provide the attack detection algorithm utilizing more fine-grained CAN frame features to detect DoS, replay and fuzzing attacks.

Choi et al. (2018) proposed VoltageIDS which can secure in-vehicle CAN networks. In their work, they mainly use the inimitable characteristics of an electrical CAN signal as a fingerprint of ECUs. VoltageIDS can distinguish errors from the bus-off attack. Also, it uses hardware related characteristics that cannot be easily avoidable. Nilsson and Larson (2008) tested the in-vehicle networks by simulating credible attacks on the CAN bus. They concluded that, due to lack of security mechanisms, the in-vehicle networks are vulnerable to attacks. The in-vehicle intrusion detection using the number of messages sent during an interval on CAN bus is discussed by Hoppe et al. (2008, 2011) and Miller and Valasek (2014). Hoppe et al. (2009) presented basic attacks from the black box attack on a gateway ECU which enables an attacker to sniff arbitrary internal communications. Hoppe et al. (2008) proposes a intrusion detection method based on IT-Forensic measures.

Also, Miller and Valasek (2015) showed that they were able to take complete control over the automobile from a remote site by using wireless network, provided the IP address of the vehicle is known. Miller and Valasek (2013) demonstrated the degree of control needed for injecting messages onto the CAN bus, and also suggested countermeasures. A tool named as EcomCat, which can receive and transmit messages on CAN bus, was also released by Miller and Valasek (2013). Koscher et al. (2010) developed an analysis and message injecting tool, CARSHARK, which is used for experimental analysis. They investigated the security issues in vehicle systems and were able to gain control of critical components such as brakes, engine and so on. They also identi-

fied the internal and external attack surfaces were identified by which an attacker might compromise a component and gain access to in-vehicle networks. Anomaly detection is another vastly researched area in cyber-physical systems such as the work of Cho et al., 2019 and Tariq et al., 2019; however, the focus of this work is primarily on detecting and identifying intrusions.

Song et al. (2016) also used the time intervals of CAN messages. They identified that the time interval between the messages gets changed during the attack state. Using this information, they were able to detect anomalies by analyzing the time intervals. In our work, we extend the prior research by providing new capabilities in monitoring, and visualizing real-time attack traffic with higher accuracy in detecting more complex attack patterns. Recently, Lee et al. (2017) demonstrated that they can detect intrusions with high fidelity by using the response offset ratio and time interval of remote frames. We compare our approach with OTIDS (Lee et al., 2017) detection method, and show that our approach achieves higher accuracy. Tariq et al., 2019 proposed a heuristic-based approach for detecting DoS, fuzzing and replay attacks in the controller area network. Furthermore, Tariq et al., 2020 proposed a transfer learning-based method to detect new intrusions with a small amount of data. Recently, Taylor et al. (2018) introduced an attack framework for automobiles that describes automotive cyber attack characteristics. They performed simulation of attacks such as replay attack. They concluded that longer events are more easily detectable than shorter ones. In this paper, we incorporated the work by Taylor et al. (2018) for the attack generation process.

## 7. Conclusion

During the vehicle operation, DoS, fuzzing, and replay attacks can be dangerous threats and have the real serious ramifications and harms to a driver, passengers, pedestrians, and a vehicle. Our proposed CAN-ADF demonstrates the effectiveness in detecting these attacks by combining the benefits of using rule-based attack traffic signatures as well as neural networks to cope with new attack patterns. We can also monitor and visualize attacks and other anomalies. Our approach is modular and can be easily integrated in a vehicle system as an effective defense and safety mechanism. For the future work, we plan to work on developing and integrating the actionable mitigation strategy upon detection of different attacks, and more effectively face with unseen and new attacks.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## CRediT authorship contribution statement

**Shahroz Tariq:** Conceptualization, Methodology, Software, Writing - original draft, Validation, Data curation, Investigation. **Sangyup Lee:** Conceptualization, Visualization, Methodology, Software, Writing - original draft, Validation, Formal analysis. **Huy Kang Kim:** Data curation, Resources. **Simon S. Woo:** Writing - review & editing, Supervision, Project administration, Funding acquisition.

## Acknowledgement

We thank KISA and KIISC for the release of CAN dataset. We also thank anonymous reviewers for their helpful feedback on drafts of this paper. This Research was supported by Energy Cloud R&D Program through the National Research Foundation (NRF) of Korea Funded by the Ministry of Science, ICT (No. 2019M3F2A1072217) and was supported by the National Research Foundation of Korea (NRF) grant funded bythe Korea government (MSIT) (No. 2017R1C1B5076474 and No. 2020R1C1C1006004).

## References

Bosch, R., et al., 1991. Can Specification Version 2.0, 300240. Rober Bousch GmbH, Postfach, p. 72.
Boudguiga, A., Klaudel, W., Boulanger, A., Chiron, P., 2016. A simple intrusion detection method for controller area network. In: Proceedings of the IEEE International Conference on Communications (ICC). IEEE, pp. 1–7.
CES, 2018. Consumer electronics show. https://ces.tech/.
Cho, K.-T., Shin, K.G., 2016. Fingerprinting electronic control units for vehicle intrusion detection.. In: Proceedings of the USENIX Security Symposium, pp. 911–927.
Cho, Jinwoo, Tariq, Shahroz, Lee, Sangyup, et al., 2019. Robust Anomaly Detection in Cyber Physical System using Kullback-Leibler Divergence in Error Distributions. 5th Workshop on Mining and Learning from Time Series (MileTS '19), Anchorage, Alaska, USA.
Choi, W., Joo, K., Jo, H.J., Park, M.C., Lee, D.H., 2018. Voltageids: low-level communication characteristics for automotive intrusion detection system. IEEE Trans. Inf. Forensics Secur. 13 (8), 2114–2129. doi:10.1109/TIFS.2018.2812149.
Chollet, F., et al., 2017. Keras. https://keras.io.
Connor, J.T., Martin, R.D., Atlas, L.E., 1994. Recurrent neural networks and robust time series prediction. IEEE Trans. Neural Netw. 5 (2), 240–254.
Damerau, F.J., 1964. A technique for computer detection and correction of spelling errors. Commun ACM 7 (3), 171–176.
Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J., 2009. A novel connectionist system for unconstrained handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. 31 (5), 855–868.
Hamming, R.W., 1950. Error detecting and error correcting codes. Bell Labs Tech. J. 29 (2), 147–160.
Hoppe, T., Kiltz, S., Dittmann, J., 2008. Security threats to automotive can networks–practical examples and selected short-term countermeasures. Comput. Saf. Reliab. Secur. 235–248.
Hoppe, T., Kiltz, S., Dittmann, J., 2009. Automotive it-security as a challenge: basic attacks from the black box perspective on the example of privacy threats. In: Proceedings of the International Conference on Computer Safety, Reliability, and Security. Springer, pp. 145–158.
Hoppe, T., Kiltz, S., Dittmann, J., 2011. Security threats to automotive can networks?practical examples and selected short-term countermeasures. Reliab. Eng. Syst. Saf. 96 (1), 11–25.
Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al., 2010. Experimental security analysis of a modern automobile. In: Proceedings of the IEEE Symposium on Security and Privacy (SP). IEEE, pp. 447–462.
Larson, U.E., Nilsson, D.K., Jonsson, E., 2008. An approach to specification-based attack detection for in-vehicle networks. In: Proceedings of the Intelligent Vehicles Symposium, 2008 IEEE. IEEE, pp. 220–225.
Lee, H., Jeong, S.H., Kim, H.K., 2017. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In: Proceedings of the Privacy, Security and Trust (PST).
Levenshtein, V.I., 1966. Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet Physics Doklady, 10, pp. 707–710.
Levy, A., Kolodny, L., 2018. Self-driving cars take over ces: Here's how big tech is playing the market. https://www.cnbc.com/2018/01/12/intel-cisco-and-amazon-introduce-self-driving-car-technology-at-ces.html, [Online; accessed 4-September-2018].
Li, X., Wu, X., 2015. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 4520–4524.
Markovitz, M., Wool, A., 2017. Field classification, modeling and anomaly detection in unknown can bus networks. Veh. Commun. 9, 43–52.
Miller, C., Valasek, C., 2013. Adventures in automotive networks and control units. DEF CON 21, 260–264.
Miller, C., Valasek, C., 2014. A survey of remote automotive attack surfaces. Black Hat USA 2014.
Miller, C., Valasek, C., 2015. Remote exploitation of an unaltered passenger vehicle. Black Hat USA 2015.
Müter, M., Asaj, N., 2011. Entropy-based anomaly detection for in-vehicle networks. In: Proceedings of the Intelligent Vehicles Symposium (IV), 2011 IEEE. IEEE, pp. 1110–1115.
Nilsson, D.K., Larson, U.E., 2008. Simulated attacks on can buses: vehicle virus. In: Proceedings of the IASTED International conference on communication systems and networks (AsiaCSN), pp. 66–72.
OnStar, 2018. Onstar in-vehicle safety and security. https://www.onstar.com/us/en/home/, [Online; accessed 26-December-2018].
Palanca, A., Evenchick, E., Maggi, F., Zanero, S., 2017. A stealth, selective, link-layer denial-of-service attack against automotive networks. In: Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, pp. 185–206.
Sak, H., Senior, A., Beaufays, F., 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In: Proceedings of the Fif-

teenth Annual Conference of the International Speech Communication Association.
Seker, S. E., Altun, O., Ayan, U., Mert, C., 2014. A novel string distance function based on most frequent k characters. arXiv:1401.6596.
Song, H.M., Kim, H.R., Kim, H.K., 2016. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In: Proceedings of the International Conference on Information Networking (ICOIN). IEEE, pp. 63–68.
Tariq, Shahroz, Lee, Sangyup, Kim, Huy Kang, Woo, Simon S., 2019. Detecting In-vehicle CAN message attacks using heuristics and RNNs. International Workshop on Information and Operational Technology Security Systems 39–45. doi:10.1007/978-3-030-12085-6_4.
Tariq, Shahroz, Lee, Sangyup, Shin, Youjin, et al., 2019. Detecting Anomalies in Space Using Multivariate Convolutional LSTM with Mixtures of Probabilistic PCA. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining 2123–2133. doi:10.1145/3292500.3330776.
Tariq, Shahroz, Lee, Sangyup, Woo, Simon S., 2020. CANTransfer: transfer learning based intrusion detection on a controller area network using convolutional LSTM network. Proceedings of the 35th Annual ACM Symposium on Applied Computing 1048–1055. doi:10.1145/3341105.3373868.
Taylor, A., Leblanc, S., Japkowicz, N., 2018. Probing the limits of anomaly detectors for automobiles with a cyberattack framework. IEEE Intell Syst (2) 54–62.
Wikipedia contributors, 2018a. Can bus — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=CAN_bus&oldid=857951504, [Online; accessed 4-September-2018].
Wikipedia contributors, 2018b. Hyundai sonata — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Hyundai_Sonata&oldid=857139549, [Online; accessed 4-September-2018].
Wikipedia contributors, 2018c. Kia soul — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kia_Soul&oldid=857653915, [Online; accessed 4-September-2018].
Winkler, W. E., 1990. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage..
Wolf, M., Weimerskirch, A., Paar, C., 2004. Security in Automotive Bus Systems. In: Proceedings of the Workshop on Embedded Security in Cars.

**Shahroz Tariq** received his B.S. in Computer Science from National University of Computer & Emerging Sciences, (FAST-NUCES), Islamabad, Pakistan, M.S. in Computer Science from Sangmyung University, Cheonan, South Korea.He worked as a Software Engineer in Bentely Systems (2014-2015). He was a Ph.D. research assitant at Stony Brook University and SUNY Korea (2017-2019).He is currently a Ph.D. candidate at Sungkyunkwan University, Suwon, South Korea.

**Sangyup Lee** received his B.S. in Computer Science from Kwangwon University, Seoul, South Korea. He was a member of IT Development department for 1 and a half year in Ssangyong Information and Communication Corp. Seoul, Korea. He was a Ph.D. research assitant at Stony Brook University and SUNY Korea (2017-2019).He is currently a Ph.D. student at Sungkyunkwan University.

**Huy Kang Kim** received his Ph.D. in industrial and systems engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2009. He received an MS degree in industrial engineering from KAIST in 2000. He received a BS degree in industrial management from KAIST in 1998. He founded A3 Security Consulting, the first information security consulting company in South Korea in 1999. Also, he was a member and the last leader of KUS (KAIST UNIX Society), the legendary hacking group in South Korea. Currently, he is an associate professor in the Graduate School of Information Security, Korea University. Before joining Korea University, he was a technical director (TD) and a head of the information security department of NCSOFT (2004–2010), one of the most famous MMORPG companies in the world. His recent research is focused on solving many security problems using AI and Machine Learning in various online services such as online games and internet banking. Based on this experience, he recently founded AI Spera, data-driven fraud detection service company.

**Simon S. Woo** received his M.S. and Ph.D. in Computer Science from Univ. of Southern California (USC), Los Angeles, M.S. in Electrical and Computer Engineering from University of California, San Diego (UCSD), and B.S. in Electrical Engineering from University of Washington (UW), Seattle. He was a member of technical staff (technologist) for 9 years at the NASA's Jet Propulsion Lab (JPL), Pasadena, CA, conducting research in satellite communications, networking and cybersecurity areas. Also, He worked at Intel Corp. and Verisign Research Lab. Since 2017, he was a tenure-track Assistant Professor at SUNY, South Korea and a Research Assistant Professor at Stony Brook University. Now, he is a tenure-track Assistant Professor at the SKKU Institute for Convergence and Department of Software in Sungkyunkwan University, Suwon, Korea.