Vehicular Communications 21 (2020) 100198

Contents lists available at ScienceDirect

Vehicular Communications

www.elsevier.com/locate/vehcom

In-vehicle network intrusion detection using deep convolutional neural network

Hyun Min Song [a], Jiyoung Woo [b], Huy Kang Kim [a,*]

[a] Graduate School of Information Security, Korea University, Seoul, Republic of Korea
[b] Soonchunhyang University, Asan, Republic of Korea

ARTICLE INFO

Article history:
Received 30 December 2018
Received in revised form 31 July 2019
Accepted 23 September 2019
Available online 3 October 2019

Keywords:
In-vehicle network
Controller area network (CAN)
Intrusion detection
Convolutional neural network (CNN)

ABSTRACT

The implementation of electronics in modern vehicles has resulted in an increase in attacks targeting in-vehicle networks; thus, attack detection models have caught the attention of the automotive industry and its researchers. Vehicle network security is an urgent and significant problem because the malfunctioning of vehicles can directly affect human and road safety. The controller area network (CAN), which is used as a de facto standard for in-vehicle networks, does not have sufficient security features, such as message encryption and sender authentication, to protect the network from cyber-attacks. In this paper, we propose an intrusion detection system (IDS) based on a deep convolutional neural network (DCNN) to protect the CAN bus of the vehicle. The DCNN learns the network traffic patterns and detects malicious traffic without hand-designed features. We designed the DCNN model, which was optimized for the data traffic of the CAN bus, to achieve high detection performance while reducing the unnecessary complexity in the architecture of the Inception-ResNet model. We performed an experimental study using the datasets we built with a real vehicle to evaluate our detection system. The experimental results demonstrate that the proposed IDS has significantly low false negative rates and error rates when compared to the conventional machine-learning algorithms.

1. Introduction

In the last few decades, the implementation of automotive electronics has experienced a rapid growth [1]. This trend has resulted in several changes in the vehicular ecosystem. Drive-by-wire (DBW) technology, for example, uses of electronic or electrical systems in the control systems, such as the throttle, brake, and steering, which were traditionally controlled using mechanical linkages [2]. The electronic devices that control the vehicular systems are known as electronic control units (ECUs). The controller area network (CAN) was developed as a message broadcast system to provide an efficient, reliable, and economical communication channel among ECUs. CAN is one of the most widely deployed in-vehicle networks along with local interconnected network (LIN) and FlexRay [3]. Unlike the common Ethernet network, CAN broadcasts several short messages that contain information about the vehicle status to maintain data consistency in all ECUs of the system [4].

However, CAN has no security features, such as authentication and encryption, to protect its communication from cyber-attacks. Researchers have demonstrated the security vulnerabilities of in-vehicle networks [5]. The lack of protection mechanisms in the CAN protocol permits attackers to control vehicle systems by injecting fabricated messages. For example, an attacker can change the gear of the car by injecting messages with the specific CAN identifier related to the gear function. Fabricated messages can be injected directly through the on-board diagnostic (OBD-II) port, as well as through the infotainment system or wireless communication system [6]. In recent years, various vulnerable features were discovered in vehicle systems, such as the electric window lift, warning lights, airbag [7], and tire pressure monitoring system (TPMS) [8].

There are several studies that attempted to improve the security of the CAN protocol by adding digital signatures based on pair-wise symmetric secret keys [9], [10]. However, digital signatures have a high communication overhead, and the bandwidth of CAN is limited to 500 kbps. In addition to the overhead, it is difficult to change all the devices of existing vehicles. Accordingly, research on an intrusion detection system (IDS) that does not cause communication overhead and can be added to existing systems is required.

In this paper, we propose an IDS using a deep convolutional neural network (DCNN) to secure the CAN bus from cyber-attacks,

* Corresponding author.
E-mail address: cenda@korea.ac.kr (H.K. Kim).

https://doi.org/10.1016/j.vehcom.2019.100198
2214-2096/

such as denial-of-service (DoS) and spoofing attacks, with significantly lower false negative and error rates.

We created a simple data assembly module, termed as frame builder, that allows us to use bit-stream data of a CAN bus directly into the convolutional neural network (CNN). The frame builder converts the CAN bus data into a grid-like structure, which is fitted to the DCNN. By using the grid data frame instead of the feature vector, the CNN-based classifier learned the temporal sequential patterns in the CAN traffic data efficiently without additional data preprocessing. The experimental results demonstrated that our IDS achieved high detection performance when compared to conventional machine-learning algorithms.

This paper presents the following contributions:

- First, to the best of our knowledge, this work is the first study applying the DCNN to IDS for the in-vehicle network security. We adopted a recent DCNN architecture, called Inception-ResNet, which demonstrates a state of the art performance in natural image classification tasks [11]. Moreover, we present the evaluation of the effectiveness of this network for intrusion detection in an in-vehicle network.
- Second, we constructed fully labeled in-vehicle network attack datasets using a real vehicle by injecting and logging the CAN messages. Our dataset is publicly available to foster further research [12].
- Third, from the large number of experiments using the real datasets that we constructed, we demonstrated that our detection model shows improved detection performance when compared to conventional machine-learning algorithms. In addition, the proposed data pre-processing method successfully works for in-vehicle intrusion detection.

The remainder of this paper is organized as follows. Section 2 presents the studies involving CAN-specific and sequential data-based IDSs. Sections 3 and 4 provide background material about in-vehicle network security and artificial neural networks including CNNs, respectively. In Section 5, the proposed IDS and the methodological steps are introduced. In Section 6, the evaluation metrics and experiment results are presented. Finally, we conclude this study in Section 7.

2. Related work

In this section, we introduce certain research works from both the perspective of domain and nature of data, which are the IDSs specialized in CAN and sequential data, respectively.

2.1. Intrusion detection model in controller area network (CAN)

An in-vehicle network carries critical messages that are essential for effective functioning of the vehicles; thus, the security vulnerabilities of the network closely relate to a safety problem. Several studies focused on protecting the in-vehicle network by various approaches based on the characteristics of the CAN protocol. Each CAN ID has a particular frequency at which the ECU sends messages periodically. Miller and Valasek introduced the concept of frequency distribution-based intrusion detection. Because the frequency of a specific CAN ID does not fluctuate, they asserted that attacks could easily be detected by monitoring the frequency levels [13]. Similarly, Muter and Asaj proposed an entropy based anomaly detection method [14]. They defined the entropy on the CAN bus and detected attacks by comparing the entropy to the reference set. Conversely, Larson et al. proposed a specification based attack detection method and considered the message as an attack if it did not follow protocol level or ECU behavior specifications [15]. Song et al. introduced a light-weight IDS based on timing analysis of the occurrence of CAN messages [16]. The periodicity characteristic of CAN messages was used as a significant feature. By monitoring the occurrence of an uncharacteristically shortened interval between messages, the injected messages were distinguished from normal messages. Cho and Kang [17] used the skew between the frequencies of the real clock and the true clock to profile the behavior of ECUs and proposed a clock based IDS. The aforementioned intrusion detection methods can be effective only for certain threat models or under certain premises such as the periodicity of CAN messages, massive injection of messages with a particular CAN ID, and specification violation. Recent attempts to overcome the problem resulted in the proposal of deep-learning-based techniques. Kang and Kang [18] used the deep belief network (DBN) structure to construct a classifier and tested it on a simulated dataset. Taylor et al. [19] used the long short-term memory (LSTM) network-based model and tested it on a CAN traffic log collected from a real vehicle. DBN and LSTM are generally considered to be more costly while training the model than CNNs. Seo et al. [20] proposed a novel method to train an anomaly detection model using generative adversarial network. They trained the detection model using normal CAN traffic data and generated noisy data similar to the CAN traffic data. Wang et al. [21] proposed a distributed anomaly detection system using hierarchical temporal memory (HTM). The authors designed the predictor based on the HTM algorithm and logarithmic loss function to compute the anomaly score. In other study, Levi et al. [22] attempted to solve the problem considering the sequential nature of the network traffic data. They trained a hidden Markov model (HMM) to learn the normal behavior of a vehicle.

2.2. Intrusion detection model for sequential data

Anomaly or intrusion detection in sequential data is one of the important research areas. Chandola et al. [23] introduced three kinds of approaches for anomaly or intrusion detection of sequential data: sequence-based, contiguous subsequence-based, and pattern-based approaches. The sequence-based approach identifies abnormal sequences from a given dataset of sequences whereas the contiguous subsequence-based approach determined an unusual subsequence within a given long sequence. The pattern frequency-based approach, also known as pattern mining, identifies patterns in a test sequence with a certain frequency of occurrence. From another perspective, Xing et al. [24] divided the sequence classification methods into three categories. The first was the feature-based method, which transforms a sequence into a feature vector; thus, this method requires an additional preprocessing step before classification. The second was the distance-based method, which measures the similarity between sequences. The ability of this method to process a large-scale database is limited because it needs to compute the distances between the test sequence and all the reference sequences. The last was the model-based method, which uses a classification model such as HMM and Naïve Bayes (NB). Machine learning-based methods are classified in this category; moreover they require large amounts of training data to achieve high-level performance.

The most popular application domain is the system call-based intrusion detection. Hofmeyr et al. [25] introduced an intrusion detection method by analyzing the sequences of system calls of several UNIX programs. Kosoresow et al. [26] presented a method using the trails of computer activity obtained through the analysis of system call traces. Lee and Stolfo [27] adopted a data mining technique for intrusion detection. They used the Sendmail system call data and the network TCP dump data on experiments and demonstrated that their classifier was able to detect anomalies and known intrusions. Qu [28] introduced a novel host-based

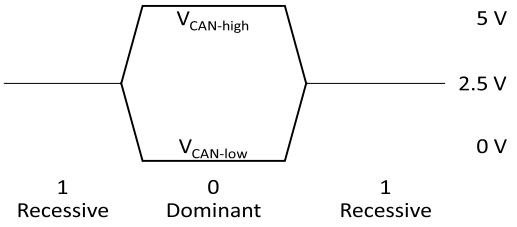

Fig. 1. Bit signaling logic of a controller area network (CAN) bus.

IDS that adopted an agent-based artificial immune system. Agents installed at hosts analyzed the system calls, application logs, file system modifications, and other host activities and states. Shabtai et al. [29] presented a framework for detecting malware on Android devices, named Andromaly, which monitors various events from the mobile device and then applies machine learning-based anomaly detectors to classify the anomalies. Similarly, network intrusion detection was also actively studied [30]. Karthick et al. [31] presented an adaptive network IDS that collects network traffic and detects anomalies. Goldenberg and Wool [32] presented a model-based IDS for SCADA systems using Modbus traffic. Yu et al. [33] and Kwon et al. [34] proposed a self-similarity-based IDS using the Hurst parameter, which is calculated from the network traffic features. In addition to intrusion detection, fault detection in a cyber-physical system is also considered as an important problem because it relates to safety concerns [35].

Wang et al. proposed a malicious traffic classification method using the CNN [36]. As the network traffic data was processed as an image and inputted to their CNN, the raw network traffic was used; consequently, hand-designed features were not required.

3. Overview of in-vehicle network security

3.1. Controller area network

Modern vehicles use a bus topology network, called as the CAN bus network, which is a message-based broadcast protocol and is designed to permit ECUs to communicate with each other. CAN has been widely adopted and has become the de facto standard for in-vehicle networks because of its relatively low cost and reliability. All the nodes of a CAN are connected to the bus through a twisted pair of CAN-high and CAN-low wires. Fig. 1 shows the signaling logic of a CAN bus. When an ECU transmits a zero-bit (dominant), the CAN high wire has 5 V and the CAN low wire has 0 V. Conversely, both wires have 2.5 V when the ECUs transmit a one-bit (recessive). Unlike stream-based protocols, such as TCP networks, where data is sent in the form of an unframed stream, CAN is a message-based broadcast protocol in which a node sends data in a predefined data frame called messages, as shown in Fig. 2.

In a CAN bus, several messages containing information, such as RPM, steering angle, and current speed, are broadcast to the entire network, thereby maintaining the consistency of the entire system. These messages are identified by a CAN ID. A CAN message has a unique 11- or 29-bit identifier. The base ID field contains the 11-bit ID, and the extended ID includes the remaining 18-bit ID. CAN 2.0A devices use only the base ID, whereas CAN 2.0B devices use both ID fields. ECUs can determine whether a received message

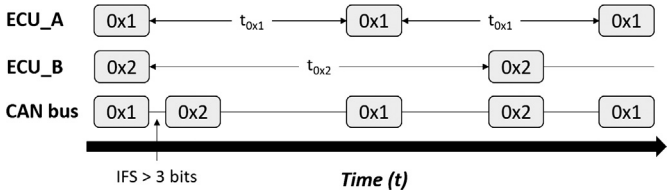

Fig. 3. Conceptual diagram of the message priority and inter-frame space (IFS).

is interesting or not based on the CAN ID; therefore, they can filter messages that are of no interest.

The meaning of each field in a CAN frame (shown in Fig. 2) are as described below:

- SOF (1 bit): The start of frame (SOF) bit denotes the start of a new message. This single bit is used for synchronizing all nodes on the bus.
- Base identifier (11 bits): This is the first part of the identifier, which is used in both the standard and extended frames.
- SRR (1 bit): The substitute remote request (SRR) bit is fixed to one and used in the extended frame.
- IDE (1 bit): The identifier extension bit (IDE) is fixed to one and used in the extended frame.
- Extended identifier (18 bits): This is the second part of the identifier, which is used only in the extended frame.
- RTR (1 bit): The remote transmission request (RTR) is used when a specific information is required from another node. The identifier specifies the node that has to respond.
- Reserved bits (2 bits): These are reserved bits for future use
- DLC (4 bits): The data length code (DLC) represents the number of bytes of data.
- Data (64 bits): The actual payload data, which can be up to 64 bits.
- CRC (16 bits): The cyclic redundancy check (CRC) contains the checksum of the previous data for error detection.
- ACK (2 bits): The transmitter sends a recessive bit (1); others change this bit to a dominant bit (0) when there is no error in the received message.
- EOF (7 bits): The end of frame (EOF) denotes the end of a current CAN message.

The CAN ID is also used to determine the ECU that has priority in the arbitration phase. As the CAN bus is a broadcast system, there may exist a situation where multiple nodes attempt to send messages at the same time resulting in a collision. In the arbitration phase, ECUs send their CAN IDs bit-by-bit. The ECU that has more dominant bits, i.e., a CAN ID with more leading zeros, wins the bus and obtains a chance to transmit a message. Alternatively, the ECU stops transmitting a message when it detects a zero bit on the CAN bus, when the current bit of its ID is one. In addition, there is a mandatory gap between consecutive messages, known as an inter-frame space (IFS). Each message is separated from the preceding frame by an IFS that consists of at least three recessive bits. If a dominant bit is detected following consecutive recessive bits, it is regarded as the SOF of the next message. In addition, for system consistency, the ECU broadcasts messages at regular intervals even if the data values have not changed. This causes the ECU to have its own message transmission cycle as shown in Fig. 3.

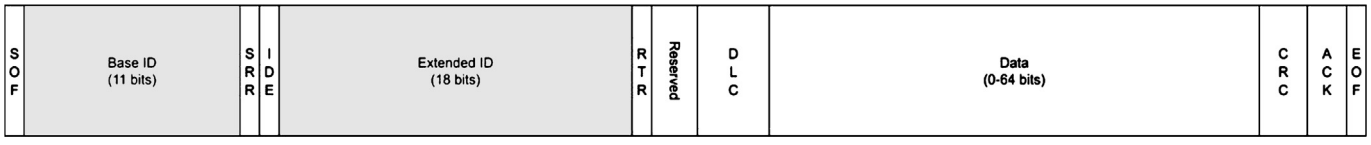

Fig. 2. Structure of a CAN 2.0B message frame.

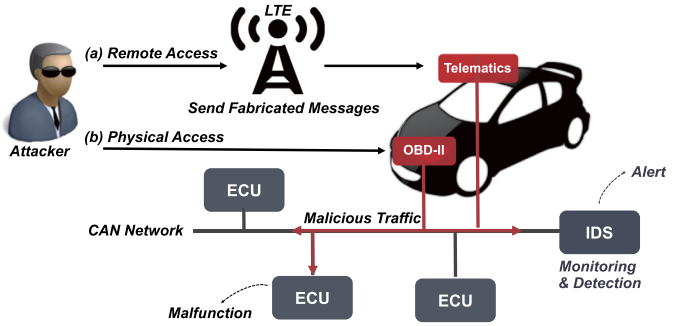

Fig. 4. Typical message injection attack scenarios. An attacker can inject fabricated messages into the CAN bus of the target vehicle via the on-board diagnostic port of the vehicle or a telematics device that provides a wireless communication channel.

ECU_A sends messages of CAN ID "0x1" after every time interval t_1 and ECU_B sends "0x2" messages after every time interval t_2. As the priority of ECU_A (0x1) is higher than ECU_B (0x2) for the first set of messages, the message of ECU_A is transmitted first, and the message of ECU_B is transmitted immediately after IFS. In practice, an ECU with information regarding the RPM of the engine sends a message containing RPM data every 10 ms. Then, the ECU managing the instrument cluster receives information about the RPM and displays it on the RPM gauge. Because of this periodicity, certain sequential patterns appear on the CAN bus.

Although CAN is an open specification, the actual identifiers and the data used in a particular vehicle depend on the car manufacturer. However, manufacturers do not provide this information for security reasons. Therefore, it is difficult to use semantic features for intrusion detection. Thus, we exploited the sequential patterns of CAN IDs that appear on the CAN bus to detect external intrusions.

3.2. Message injection attack

The most prominent security problem of the CAN protocol is the lack of security features, such as authentication and encryption. Because CAN is a broadcasting-based bus network and there is no authentication, any node can be connected to the bus and can receive all messages. Thus, an attacker can also sniff CAN bus data easily. This exposure of internal data allows an attacker to analyze the CAN data of a target vehicle and devise subsequent sophisticated message injection attacks to control the target vehicle.

An earlier research had discovered various attack surfaces that attackers can to exploit to inject messages into the CAN bus [6]. The author categorized the various attack surfaces as physical access and wireless access. An attacker can access the CAN bus physically via the OBD-II port or remotely via Bluetooth, WiFi, or telematics services, such as 3G and long-term evolution (LTE). These attack surfaces can be exploited as an entry point to the CAN bus of the target vehicle.

Fig. 4 shows two different message injection attack scenarios: (a) remote access and (b) physical access. In the case of remote access, the attacker can exploit the telematics services provided by vehicle manufacturers (e.g., GM's OnStar, Ford's Sync, and Hyundai's BlueLink) to compromise a telematics device of the target vehicle. Remote access using telematics services is the most critical part of the long-range wireless attack because these services provide a broad range of connectivity via cellular networks such as 3G and LTE communications. This permits an attacker to obtain an access remotely from long distances without being limited to physical locations. In the case of physical access, the simplest method used by an attacker is to access the CAN bus through the direct connection via the OBD-II port. The attacker can attach

Normal

Timestamp: 70.348611		ID: 0370	DLC: 8	00 20 00 00 00 00 00 00
Timestamp: 70.348851		ID: 043f	DLC: 8	10 40 60 ff 7b 07 0a 00
Timestamp: 70.349091		ID: 0440	DLC: 8	ff 00 00 00 ff 07 0a 00
Timestamp: 70.349645		ID: 0130	DLC: 8	01 80 00 ff 10 80 0c 1c
Timestamp: 70.349883		ID: 0131	DLC: 8	00 80 00 00 53 7f 0c 4a
Timestamp: 70.350123		ID: 0140	DLC: 8	00 00 00 00 0c 20 2c 6e
Timestamp: 70.350362		ID: 018f	DLC: 8	fe 52 00 00 00 3c 00 00
Timestamp: 70.35059	10 ms	ID: 0260	DLC: 8	18 1a 1b 30 08 8f 4e 33
Timestamp: 70.35085		ID: 02a0	DLC: 8	60 00 96 1d 97 02 bd 00
Timestamp: 70.351074		ID: 02c0	DLC: 8	15 00 00 00 00 00 00 00
Timestamp: 70.351310		ID: 0316	DLC: 8	05 1a 54 0a 1a 1e 00 70
Timestamp: 70.351545		ID: 0329	DLC: 8	86 ba 7f 14 11 20 00 14
Timestamp: 70.351781		ID: 0350	DLC: 8	05 20 a4 68 74 00 00 9d
Timestamp: 70.352022		ID: 0545	DLC: 8	d8 00 00 89 00 00 00 00
Timestamp: 70.357363		ID: 0002	DLC: 8	00 00 00 00 00 01 0d c7
Timestamp: 70.357599		ID: 0153	DLC: 8	00 21 10 ff 00 ff 00 00
Timestamp: 70.358343		ID: 043f	DLC: 8	10 40 60 ff 7b fa 09 00
Timestamp: 70.358591		ID: 0370	DLC: 8	00 20 00 00 00 00 00 00

Injection Attack

Timestamp: 99.607794		ID: 0002	DLC: 8	00 00 00 00 00 0a 0a 1a
Timestamp: 99.608026		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.608272		ID: 0130	DLC: 8	0d 80 00 ff 19 80 0a 6a
Timestamp: 99.608507		ID: 0131	DLC: 8	e9 7f 00 00 2b 7f 0a d6
Timestamp: 99.608753		ID: 0140	DLC: 8	00 00 00 00 04 00 2a 59
Timestamp: 99.608990		ID: 0350	DLC: 8	05 20 94 68 75 00 00 ac
Timestamp: 99.609221		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.609473	7 ms	ID: 0545	DLC: 8	d8 00 00 8b 00 00 00 00
Timestamp: 99.609707		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.609965		ID: 0370	DLC: 8	00 20 00 00 00 00 00 00
Timestamp: 99.610379		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.610629		ID: 043f	DLC: 8	10 40 60 ff 7c e6 0a 00
Timestamp: 99.610893		ID: 0440	DLC: 8	ff 00 00 00 ff e6 0a 00
Timestamp: 99.611275		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.612440		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.613599		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.614764		ID: 043f	DLC: 8	01 45 60 ff 6b 00 00 00
Timestamp: 99.615019		ID: 02c0	DLC: 8	15 00 00 00 00 00 00 00

Fig. 5. Examples of CAN traffic where fabricated messages are injected. These messages have a CAN ID of "043f". The occurrence of a "043f" message is increased to nine messages in only 7 ms. The "043f" messages are originally transmitted every 10 ms.

his tiny computing devices, such as Raspberry Pi and Arduino, as an attack node to the OBD-II port of the target vehicle.

Once the attacker obtains any one of the CAN nodes, he can control the vehicle by injecting sophisticated manipulated messages. In the CAN protocol, the connected nodes believe that the last received message represents the current condition of the vehicle. Thus, an attacker can deceive ECUs by injecting a manipulated message immediately following the target CAN ID message. For example, when a vehicle is in motion and its gear shift lever is placed in "drive", the ECU that handles transmission gear status repeatedly sends messages with a value indicating "drive". At this time, if the attacker injects messages of the CAN ID corresponding to the gear status having a value representing "reverse" immediately after the messages sent by the original ECU, other ECUs believe that the current gear status is "reverse" and not "drive". To control a vehicle successfully, the attacker has to transmit messages more frequently than the original ECU, which transmits its message continuously and periodically.

Fig. 5 shows a sample network traffic log during an injection attack. Each line contains information for each message, namely the timestamp, CAN ID, DLC and payload data consisting of up to 8 bytes. The original interval between the "043f" messages was 10 milliseconds, however nine "043f" messages are recorded in only 7 ms during the injection attack. As the injected messages appear on the network, the sequential pattern of the CAN ID changes. Thus, we capitalized on this change in the CAN ID sequential pattern to detect a message injection attack.

In this work, we assumed that the attacker already has an access to the CAN bus either physically or remotely, and has an authority over one or more CAN nodes for the message injection attack. As the CAN bus is a broadcast-based network, the source of the message is not distinguished; ECUs treat the injected messages

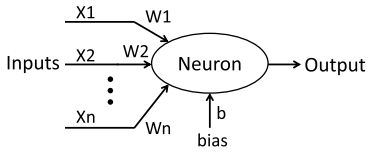

Fig. 6. Artificial neuron receives feature vectors as inputs and outputs the weighted sum of the inputs.

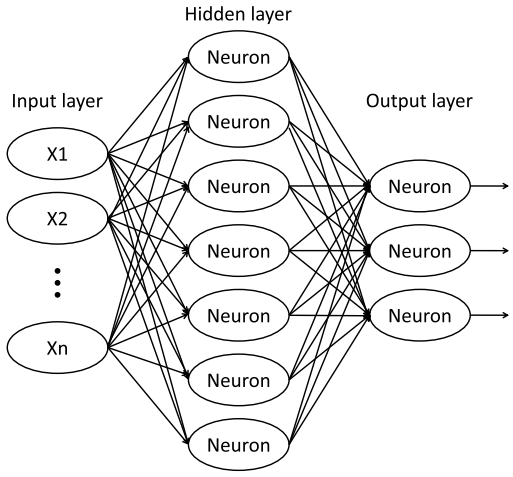

Fig. 7. Structure of a multi-layer perceptron, which consists of an input layer, multiple hidden layers, and an output layer.

through the OBD-II port or telematics service system similarly. Therefore, the IDS was also placed as one node of the CAN bus to monitor the network.

Although commercial tools, such as Kvaser and Vector, support the CAN communication via the OBD-II port, in this work, we used two customized Raspberry Pi devices, one for the message injection attack and the other for logging CAN traffic. The data on the captured CAN traffic were used as the dataset for training and evaluation of the proposed IDS.

4. Artificial neural network

4.1. Artificial neuron

Artificial neural networks (ANNs) consist of several interconnected simple artificial neurons. An artificial neuron is a mathematical function conceived as a model of biological neurons; moreover, it receives one or more inputs and produces the weighted sum of the inputs as an output. This process is illustrated in Fig. 6,

$$y(x) = W \cdot X + b, \tag{1}$$

where X is the input and y is the output of the artificial neuron. W and b are a set of weights and biases, respectively.

Traditional neural networks, such as the multi-layer perceptron (MLP), are constructed with several layers of artificial neurons, as illustrated in Fig. 7. This structured collection of artificial neurons can learn features of data and predict the results of new samples based on the learning.

4.2. Convolutional neural network

The CNN is a type of deep neural network, which is a computational approach based on a large collection of neural units. Unlike the traditional MLP models, each layer of the CNN consists of a rectangular 3D grid of neurons. The neurons of a layer are only connected to the neurons in a receptive field, which is a small region in the immediately preceding layer, rather than the entire set of neurons. By using receptive fields, CNNs exploit the spatially-local correlation of input data. The receptive fields permit CNNs to create good representations of small parts of the input; then, the CNNs use them to assemble representations of larger regions.

Image recognition is one of the application areas of CNNs. In 2016, Google announced their newest CNN models, named Inception-V4 and Inception-ResNet [11]. These models have demonstrated superior performance using the ImageNet dataset.

Generally, CNNs consist of the components shown in Fig. 8, which are the convolution layer, activation function, fully-connected layer, and pooling layer. The model in the example has two convolutional layers, one pooling layer, one fully-connected layer, and the output layer. The activation functions are applied to all neurons of the convolutional and fully-connected layers; however, they are not applied to the pooling layer. Further details of the components are described in the following subsections.

4.2.1. Fully-connected layer and convolutional layer

The fully-connected layers are identical to the hidden layers in a traditional neural network, such as the MLP. Because all neurons in the facing layers are densely connected, the fully-connected layers have an enormous number of weights. Conversely, the convolutional layers require a small number of weights but are more computationally intensive because they have dozens or hundreds of feature maps. Thus, in general, most parameters are deployed in fully-connected layers; however, most of the operations are performed in convolutional layers. The convolutional layer is a key component of the CNN and computes the dot product between the input and the weights of the filter to produce 2-D feature maps of that filter. The filter determines the receptive field of the convolutional layer described above.

Fig. 9 shows the difference between the convolutional layer and the fully connected layer. The convolutional layer computes the outputs by calculating the weighted sum of the values in the window, termed as the filter, which determines the receptive field. The convolutional layer requires considerably many fewer parameters than the fully connected layer. For example, if we process a $29 \times 29 \times 3$ volume into a $27 \times 27 \times 6$ volume, the fully connected layer requires $(29 \times 29 \times 3) \times (27 \times 27 \times 6) = 11$ M weights,

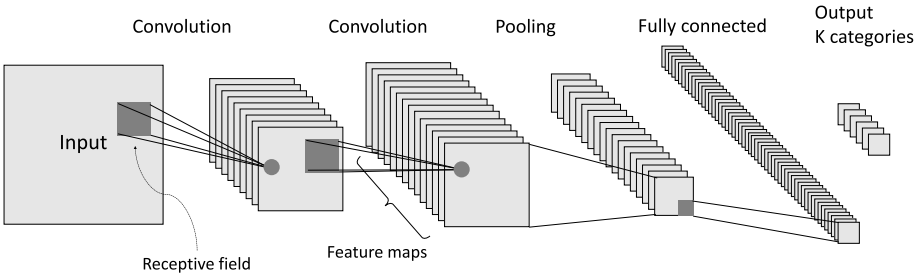

Fig. 8. Typical example of a CNN. CNN generally consists of multiple sets of convolution layers following pooling layers, fully connected layers, and the output layer.

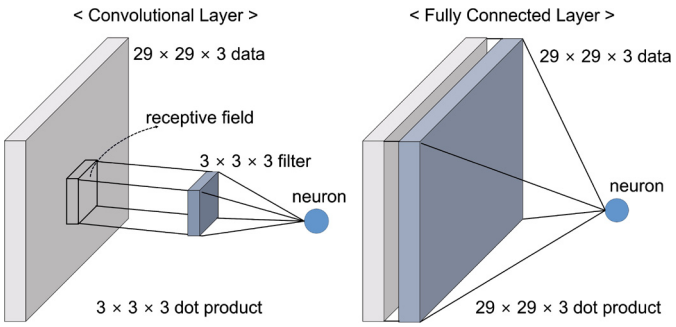

Fig. 9. Difference between a convolutional layer and a fully connected layer. The former connects neurons in a receptive field as an input whereas the latter connects all neurons in a lower layer as input.

while a convolutional layer with a 3×3 size of filter requires only $(3 \times 3 \times 3) \times 6 = 162$ weights.

4.2.2. Activation function

The purpose of the activation function is to provide the model with nonlinearity. Convolution is a linear operation; thus, when the activation function is not used, it functions similar to a single perceptron, even though several layers are adopted.

The activation functions are applied to the outputs of neurons as is presented below:

$$h(x) = g(y(x)), \tag{2}$$

where the function g is the activation function, h is the output of the activation function, and $y(x)$ is an output of a neuron, which is the input of the activation function. In CNN architectures, a rectified linear unit (ReLU) is widely used as an activation function:

$$g(a) = ReLU(a) = max(0, a), \tag{3}$$

ReLU outputs the maximum value between zero and the input, a. ReLU has the advantage of the ease of differentiation in the back-propagation to find the optimal values of weights. Although it is not differentiable at $x = 0$, differentiation is very simple in the other ranges. Its derivate value is one when $x > 0$, zero when $x < 0$.

The activation function breaks the linearity of the neural network by projecting output values of a certain range, which are positive values while using ReLU; consequently, permitting a neural network to learn the complex and nonlinear nature of the data. In addition, it decides how much information should be delivered, similar to the operation of the neurons in the human brain from which they are inspired.

The softmax function is a special activation function that is typically used on the last output layer. The softmax function normalizes a K-dimensional vector z of arbitrary real values to a K-dimensional vector $\sigma(z)$ of real values in the range $(0, 1)$ of which the sum is 1, which is calculated as:

$$\sigma(z)_i = \exp^{z_i} / \sum_{k=1}^{K} \exp^{z_k} \ (i = 0, \ \ldots, \ K). \tag{4}$$

The output of the softmax function can be considered as a probability distribution over K classes in a multiclass classification that represents the classification confidences in each class.

4.2.3. Pooling layer

The pooling layer is also referred to as the sampling layer because this layer generates a single value from each segment separated by the kernel to reduce the size of data. The pooling layers are used to adjust the amount of computation in each convolutional layer to be similar. If only the convolution layers are used without the pooling layer, the amount of computation in the subsequent convolutional layers increases significantly as the number of filter maps increases.

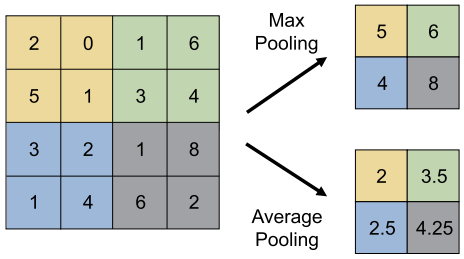

Fig. 10. Difference between max-pooling and average-pooling. Max-pooling outputs the maximum among input values whereas average pooling outputs the average of input values.

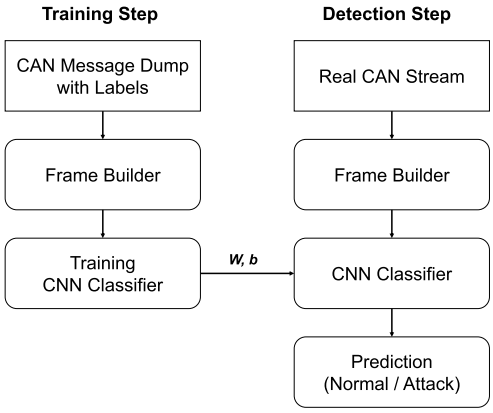

Fig. 11. Process of the proposed intrusion detection system. The process is divided into two steps: training and detection. A CAN message dump with label is used for supervised learning in the training step, whereas the real CAN stream is used in the detection step.

The max-pooling layer outputs the maximum value from each segment separated by the pooling kernel. Fig. 10 shows the difference between max-pooling and average-pooling for a 2×2 kernel with stride two. The kernel decides the size of the pooling region and the stride indicates the size of the sliding step of the kernel on data.

5. Intrusion detection system (IDS) using deep convolutional neural network

5.1. Process of the proposed IDS

The proposed IDS consists of two steps, the training and detection steps, similar to a general machine-learning-based IDS, as shown in Fig. 11. Although a labeled CAN traffic dump is used in the training step, the real CAN traffic without labels is directly used in the detection step in the proposed IDS scheme.

In the training step, the frame builder retrieves CAN IDs from the logged CAN traffic dump and assembles data frames consisting of 29 sequential CAN IDs. The data frame built by the frame builder is properly computed and classified as attack or non-attack while traversing from the first input layer to the last output layer of the CNN structure.

By building a data frame that consists of 29 sequential CAN IDs, the proposed CNN-based IDS can exploit the sequential patterns of CAN IDs that appear on the CAN bus. For supervised learning, data frames that contain one or more injected messages are labeled as attacks, and data frames that do not contain injected messages are labeled as non-attacks.

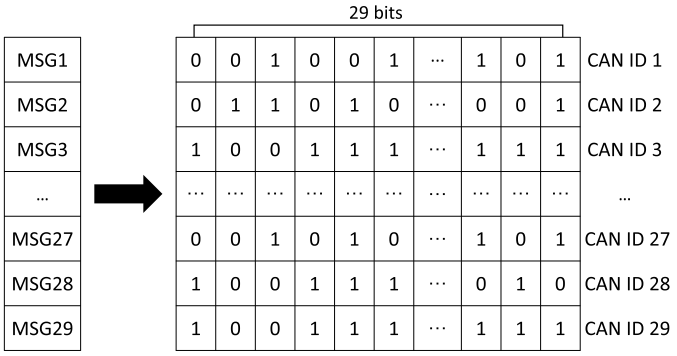

Fig. 12. Frame builder transforms a CAN ID sequence into a 2-D grid data frame. Each CAN message has a 29-bit ID and the frame builder collects IDs from 29 CAN messages to build a 2-D grid data frame.

As training the CNN classifier is a time-consuming task, it has to be performed offline. Once trained, the CNN classifier is relatively quick in calculating the results for new samples. In this paper, although the proposed IDS is tested offline, as the computing power increases, it is not unreasonable to expect that the proposed IDS will be available for real-time detection.

5.2. Frame builder

In general, CNNs are designed to accept data in the grid format as the input and exploit the spatially-local correlation. We processed the temporal sequential patterns of CAN traffic as a spatially-local correlation by using the frame builder for reshaping the sequential CAN message data to create 2-D data frames, as illustrated in Fig. 12. The frame builder extracts the 29-bit identifiers from the most recent 29 CAN messages and builds a 29×29 bitwise frame by stacking them. There are two reasons for using pure bits as input data without preprocessing. First, for efficiency, no additional data preprocessing, such as decoding, is required. As there are thousands of messages per second on a CAN bus, the IDS should be designed considering the computational efficiency to prevent a bottleneck. Second, bit representation can explicitly show the fluctuation of identifier patterns. The bit representation of the identifier is given as:

$$ID = b_i \ (for\ i = 0, \ldots, 28), \tag{5}$$

where b_i is the i-th bit value. The frame builder assembles a data frame with 29 IDs to generate the square structure that can be used as an input to a convolutional network, which is presented as:

$$FRAME = ID_i \ (for\ i = 0, \ldots, 28) \tag{6}$$

$$= b_{ij} \ (for\ i, j = 0, \ldots, 28), \tag{7}$$

where b_{ij} is the j-th bit value of the i-th ID in a frame. Then, each frame is labeled as one for attack or zero for normal, for supervised learning.

5.3. Reduced Inception-ResNet

Inception-ResNet is one of the proven DCNN models. It is designed to classify several images into 1,000 classes and shows innovative performance. To apply Inception-ResNet to our environment, we redesigned the original model by reducing its components and adjusting the parameters because of the differences in the dimensions of the input and output, and the complexity of the data structure. The original Inception-ResNet was designed to classify $299 \times 299 \times 3$ input image data into 1,000 classes whereas

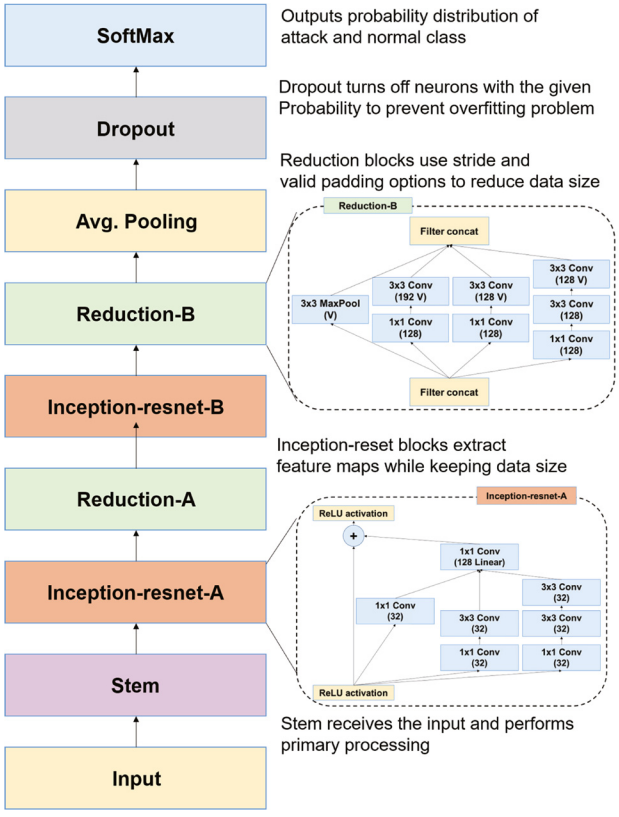

Fig. 13. Overall architecture and blocks of the reduced Inception-ResNet used in this work. The number of blocks of Inception-resnet-A and -B differ. The original Inception-ResNet has five, ten, and five blocks of Inception-resnet-A, -B, and -C, respectively. The reduced Inception-ResNet has only one of each block and no Inception-resnet-C block.

our model is designed to classify $29 \times 29 \times 1$ input data into only two classes. The input and output of the original model are so large that it is too complex to apply directly to our dataset. If a model becomes too complex, it requires excessive computing resources and has a possibility of the overfitting problem. In machine-learning-based techniques, overfitting is a common problem, where the trained model is too highly optimized to the training data set and thus degrades the performance of the test data or actual production data.

The original Inception-ResNet consists of nine different kinds of blocks: stem, Inception-resnet-A, -B, and -C, reduction-A and -B, average pooling, dropout, and softmax. In the reduced Inception-ResNet, nearly the entire original architecture was used, except the Inception-resnet-C block; moreover, the layer structures and parameters inside the other blocks were modified, as shown in Fig. 13. In our model, the size of data being processed became sufficiently small after the reduction-B block; consequently, it was directly used as an input to average pooling. Furthermore, the original model has five Inception-resnet-A blocks, ten Inception-resnet-B blocks, and one each of the other blocks; however, we used only one each of the Inception-resnet-A and -B blocks. As our data size is much smaller and simpler than the input image data of the original Inception-ResNet, we did not require the usage of such a deep architecture. Table 1 lists the output dimensions of the blocks of the original Inception-ResNet and the reduced Inception-ResNet. By creating a lighter model, we effectively reduced the memory requirement and the number of parameters. The original model requires approximately 111 MB (forward + backward) per input; hence, approximately 2.2 GB is required when the batch size is 20. However, our reduced Inception-ResNet requires only approximately 2.2 MB per input, which is approximately 2% of the

Table 1
The output dimensions of blocks.

Block	Inception-ResNet	Reduced Inception-ResNet
Input	**299 × 299 × 3**	**29 × 29 × 1**
Stem	**35 × 35 × 256**	**13 × 13 × 128**
Inception-resnet-A	**35 × 35 × 256**	**13 × 13 × 128**
Reduction-A	**17 × 17 × 896**	**6 × 6 × 448**
Inception-resnet-B	**17 × 17 × 896**	**6 × 6 × 448**
Reduction-B	**8 × 8 × 1,792**	**2 × 2 × 896**
Inception-resnet-C	**8 × 8 × 1,792**	not used
Avg. Pooling	**1,792**	**896**
Dropout	**1,792**	**896**
Softmax	**1,000**	**2**

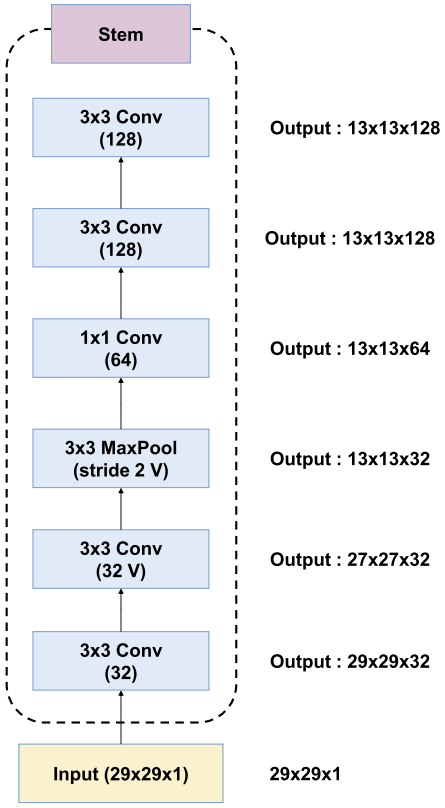

Fig. 14. Stem of the reduced Inception-ResNet. This is the input part of the network and produces feature maps from the input.

memory required by the original model. Further, the number of parameters is approximately 18% when compared to the original model, 1.76 M parameters in the reduced Inception-ResNet and 9.8 M parameters in the original Inception-ResNet.

The stem block, described in Fig. 14, generates 128 feature maps of size 13 × 13 from an input of size 29 × 29. The Inception-resnet blocks, described in Figs. 15 and 16, process the data while maintaining its dimensions. Instead, the reduction blocks, described in Figs. 17 and 18, are inserted between the inception blocks to reduce the data size. The average-pooling layer performs the role of flattening the 3-D grid data to a 1-D array data, which is used instead of a fully-connected layer to maintain the information of all elements in each feature map.

Although the general CNN structure has fully connected layers before the final softmax layer, the Inception-ResNet architecture does not have a fully connected layer. Instead, a single average-pooling layer is used before the softmax layer because the size of the output from the last convolutional layer is sufficiently small to be reduced to a single value by average-pooling. At the average-pooling layer, the dimensions of data are reduced from 3 × 3 × 896

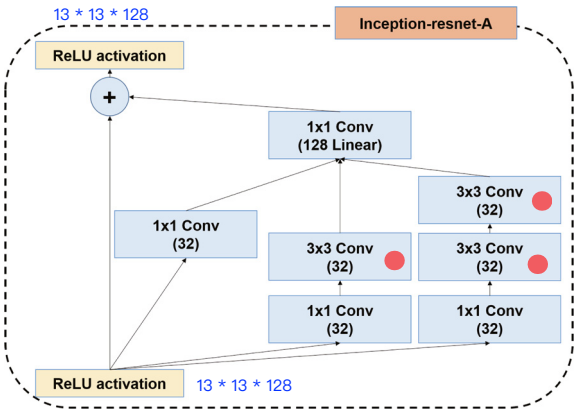

Fig. 15. Schema for a 13 × 13 grid (Inception-resnet-A) module of the reduced Inception-ResNet. In the original Inception-ResNet, it uses a 35 × 35 grid data.

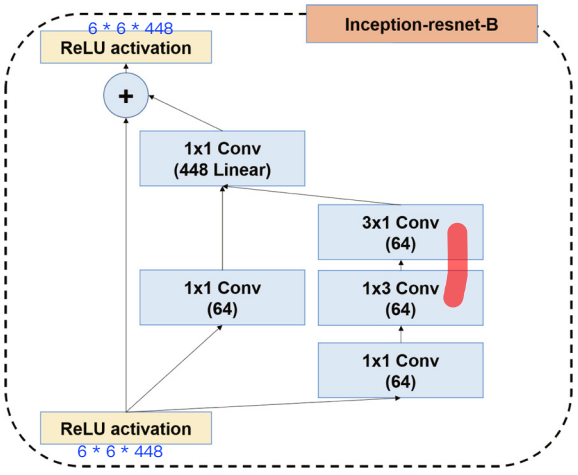

Fig. 16. Schema for a 6 × 6 grid (Inception-resnet-B) module of the reduced Inception-ResNet. In the original Inception-ResNet, it uses a 17 × 17 grid data.

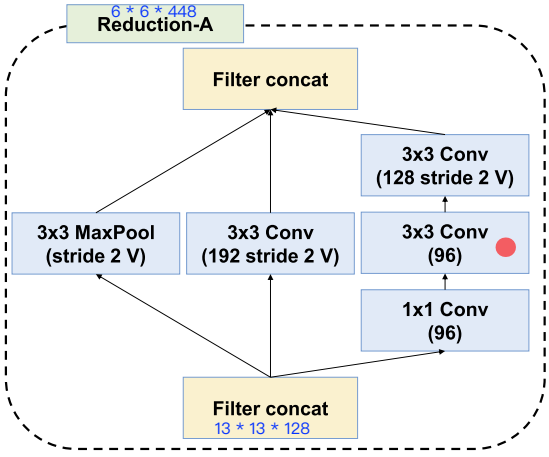

Fig. 17. Schema for a 13 × 13 to 6 × 6 grid-reduction module. In the original Inception-ResNet, it reduces a 35 × 35 grid data to a 17 × 17 grid data.

to 1 × 1 × 896, linearly arranged, and then finally delivered to the softmax layer.

As the neural network becomes increasingly deeper, the deep neural network requires regularization to prevent the overfitting problem. The most common techniques used in practice are dataset augmentation, weight penalty L1 or L2, and dropout [37]. Similar to the Inception-ResNet model, we used the dropout between the average pooling layer and the softmax layer. The

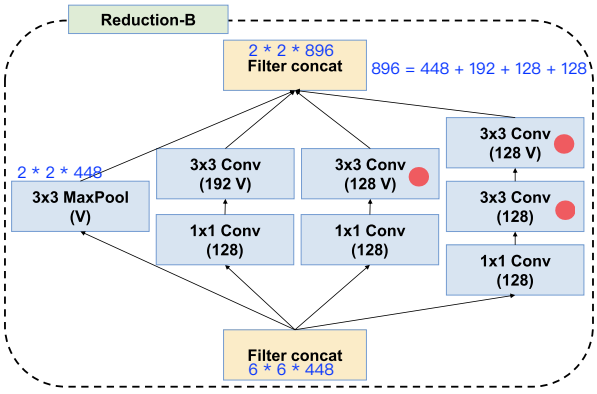

Fig. 18. Schema for a 6×6 to 2×2 grid-reduction module. In the original Inception-ResNet, it reduces a 17×17 grid data to a 8×8 grid data.

dropout reduces the extent to which the model is learned by randomly deactivating a portion of the neurons with the given probability.

Training the model, which implies adjusting the parameters, requires the measurement of the loss of the current model. The softmax layer is used as the output layer. Thus, our model outputs the probability distribution over target classes zero and one, as presented below:

$$\hat{y}_i = p(c = i|x) \ (for\ i = 0, 1), \tag{8}$$

where x and $\hat{y}_i$ are the input frame data and the predicted probability of the target class when c is the target class. Thus, we must maximize the probability $p(c = y|x)$ such that the target class c is equal to the true class y of a given x. In the training step, instead of maximizing the probability of y, we minimize the log-likelihood function, referred to as cross-entropy, as shown below:

$$H(y, \hat{y}) = -(1 - y) \log \hat{y}_0 - y \log \hat{y}_1, \tag{9}$$

where y is the true label, which is zero or one, and $\hat{y}$ is the predicted probability distribution over two classes. The cross-entropy of $\hat{y}$ and y is computed by summing the products of the true probability and the predicted probability. In classification problems, when the true probability is zero or one, the cross-entropy can be rewritten as the negative log of the predicted probability of the true label because the true probability of the wrong label is zero, which causes another term to be zero.

Then, the loss function is computed by taking the average of the cross-entropies of the given N samples, which is derived as:

$$L(W) = \frac{1}{N} \sum_{n=1}^{N} H(y, \hat{y}), \tag{10}$$

where W is the set of parameters of the current model. More specifically, our intrusion detection model classifies the given input data into two possible classes labeled zero and one, signifying normal and attack, respectively. Thus, the model predicts the probabilities of the target classes and outputs the predicted label $y \in \{0, 1\}$, for given input data x.

6. Experiment results

6.1. Datasets

A real vehicle was used to build the datasets for our experiments. We utilized two custom Raspberry Pi devices, one for logging the network traffic and the other for injecting fabricated messages as the attack node. They were connected to the in-vehicle network via the OBD-II port located below the steering wheel of the car. Through the OBD-II port, our custom nodes were able to send to/receive from real ECU nodes on the CAN bus. There are 26 distinct CAN IDs on the CAN bus at normal status.

Table 2
Datasets overview.

Attack type	Normal messages	Injected messages
DoS Attack	3,078,250 (84%)	587,521 (16%)
Fuzzy Attack	3,347,013 (87%)	491,847 (13%)
Gear Spoofing	2,766,522 (82%)	597,252 (18%)
RPM Spoofing	2,290,185 (78%)	654,897 (22%)

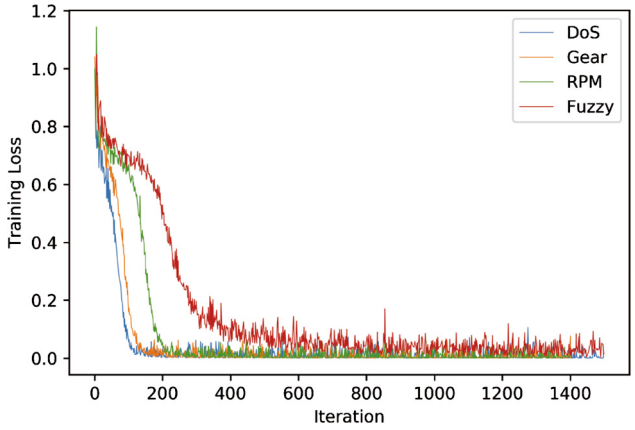

Fig. 19. Training loss on each dataset. From left to right are DoS, gear, RPM, and fuzzy attack datasets. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 2 lists the numbers of normal messages and injected messages in the experiment datasets. We constructed four datasets, which are DoS attack, fuzzy attack, spoofing the drive gear, and spoofing the RPM gauge. Each dataset was created by logging the CAN traffic while injecting fabricated messages in the controlled environment. The car was parked with the engine turned on while data was being collected. Each dataset contained 300 intrusions by injected messages. Each intrusion was performed for 3 to 5 seconds, and each dataset contains CAN traffic collected for thirty to forty minutes.

For the DoS attack, we injected CAN messages with a CAN ID of 29 zero bits, which is the most dominant CAN ID in the CAN bus, every 0.3 ms. The purpose of the DoS attack is impairment of the network availability. As the ECU that attempts to send a message with the most dominant CAN ID always wins the bus in the arbitration phase, other ECUs are prevented from transmitting their messages. The fuzzy attack is similar to the DoS attack; however, the difference is that the CAN ID and data values of messages are entirely random in the fuzzy attack. We injected random messages every 0.5 ms. The fuzzy attack is performed to cause the vehicle to malfunction. Spoofing the drive gear and RPM gauge datasets were achieved by injecting messages of a certain CAN ID every 1 ms. These messages contained information of the drive gear and RPM gauge, respectively. The spoofing attack enabled us to deceive the original ECU and change the RPM gauge and the drive gear on the instrument panel.

Fig. 19 shows the training losses on each dataset in the early training steps (∼1,500). We can observe that as the complexity of the attack data increases, the convergence speed of the training loss become slower. The training loss of the DoS attack, which has a relatively simple pattern converges the earliest and the training loss of the fuzzy attack, which injects random data, converges the last.

6.2. Evaluation metrics

We measured the false negative rate (FNR) and error rate (ER) to evaluate the classification performance. In our experiment, FNR is the fraction of undetected frames that are attack frames, and the ER is the fraction of incorrectly classified frames, which are calculated as:

$$FNR = FN/(TP + FN), \tag{11}$$

and

$$ER = (FN + FN)/(TP + TN + FP + FN), \tag{12}$$

where TP (true positive) and TN (true negative) are the number of frames that are classified correctly as attack and normal, respectively, and FP (false positive) and FN (false negative) are the number of frames that are classified incorrectly as attack and normal, respectively. The FNR should be small and is considered to be more important for attack detection in an in-vehicle network. This is because even a small number of undetected attacks could cause the vehicle to momentarily malfunction and become a safety threat.

We also calculated precision, recall, and F1-score for comparison with other algorithms. Precision is the fraction of actual attack frames among the frames detected as attack. High precision relates to a low FP rate. Frequent false alarms annoy users; therefore, they should be managed to improve the service quality. Precision is calculated using the equation:

$$Precision = TP/(TP + FP). \tag{13}$$

Recall is a fraction of the correctly detected attack frames and is called as true positive rate (TPR). We can easily calculate the recall by subtracting FNR from 1 or using the equation:

$$Recall = 1 - FNR = TP/(TP + FN). \tag{14}$$

The F1-score represents a balance between precision and recall. The F1-score is usually used to measure classification performance when a dataset has uneven class distribution, which is calculated as:

$$F1 = 2 \times (Precision \times Recall)/(Precision + Recall). \tag{15}$$

6.3. Hyperparameter

Depending on the hyperparameters, the performance of the model is changed. We considered the experiments by changing the value of the learning rate. The learning rate controls the degree to which the weights of the model are adjusted with respect to the gradient of training loss. The lower the value of learning rate, the more slowly the weights are updated along the downward slope of the loss surface, and a longer time is required for convergence. If the learning rate value is too small, training a model is not only time consuming, but it can also lead the model to a local minimum; instead of a global minimum. Conversely, if the learning rate is too large, the gradient descent can overshoot the minimum and may fail to converge. Thus, choosing the proper value of learning rate helps us to identify the optimal weights.

Initially, we measured FNR and ER while changing the learning rate from 10^{-6} to 10^{-2}. In Table 3, we can observe that FNR and ER are growing as the learning rate is decreased. However, when the learning rate was set to 0.01, the gradient descent failed to converge and could not create a model. Therefore, we set 10^{-3} as the optimal learning rate.

Table 3
Impact of learning rate.

Learning rate	10^{-3}	10^{-4}	10^{-5}	10^{-5}
# of iterations		10k		20k
FNR	0.267%	0.603%	1.04%	0.727%
ER	0.197%	0.292%	0.692%	0.574%

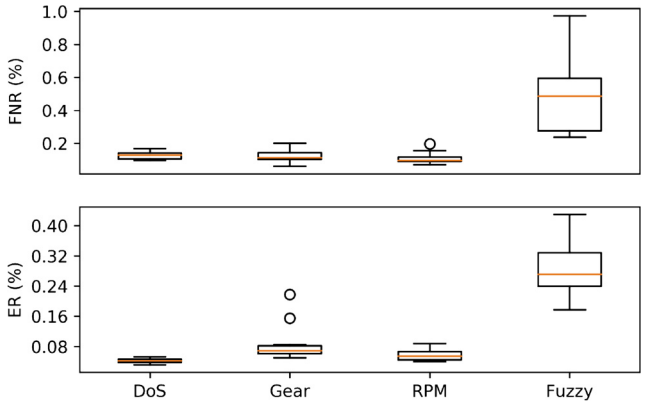

Fig. 20. Boxplot of false negative rates and error rates measured in 30 repeated experiments.

Table 4
The summarized test results on each dataset.

Dataset (%)	DoS		Gear		RPM		Fuzzy	
	FNR	ER	FNR	ER	FNR	ER	FNR	ER
mean	0.13	0.04	0.12	0.09	0.11	0.06	0.50	0.28
std	0.02	0.01	0.04	0.05	0.04	0.02	0.24	0.07
min	**0.10**	**0.03**	**0.06**	**0.05**	**0.07**	**0.04**	**0.24**	**0.18**
25%	0.11	0.04	0.10	0.06	0.09	0.04	0.28	0.24
50%	0.13	0.04	0.11	0.07	0.10	0.05	0.49	0.27
75%	0.14	0.05	0.14	0.08	0.12	0.07	0.60	0.33
max	0.17	0.05	0.20	0.22	0.20	0.09	0.97	0.43

6.4. Detection performance

According to the result of Section 6.3, we set the learning rate to 0.001 for training the IDS model. We performed 30 tests for each attack dataset and measured FNR and ER. For each test, we initially shuffled the data randomly to break the locality between sequential samples because the locality of the data can cause learning performance degradation of the model. Then, we split the shuffled dataset into the training and test data, 70% for training and 30% for test.

The test results are shown in Fig. 20. As illustrated in the figure, the proposed model shows stable detection performance in the DoS, gear, and RPM attack datasets, but relatively unstable performance in the fuzzy attack dataset. It appears that the complexity of the fuzzy attack data is much higher than the others, requiring more training iterations to create a stable model.

The elaborate test results including average, standard deviation, minimum, and maximum values are summarized in Table 4. The proposed IDS shows a FNR and an ER of less than 0.1% in the DoS, gear, and RPM attack datasets, 0.24% of FNR and 0.18% of ER in the fuzzy dataset, in the best case scenario.

The confusion matrices of one of the multiple tests on each attack dataset are listed in Table 5. The numbers in the table represent the number of samples of TN, FP, FN, and TP values. Samples with data frame size of 29×29 were created by the frame builder, which contained 29 sequential bit-represented CAN IDs.

Table 6 lists the detection performance of our DCNN-based technique when compared to those of the other machine-learning techniques. The results clearly demonstrate that the DCNN model

Table 5
Classification results of the proposed IDS.

DoS	Predicted normal	Predicted attack
True normal	26,555	0
True attack	12	11,354
Fuzzy	Predicted normal	Predicted attack
True normal	17,050	3
True attack	36	13,405
Spoofing RPM	Predicted normal	Predicted attack
True normal	26,412	3
True attack	9	21,386
Spoofing Gear	Predicted normal	Predicted attack
True normal	26,096	5
True attack	27	19,835

outperforms the other machine-learning models on all datasets. It can be observed that the proposed model is well aware of the sequential nature of network traffic and captures the changes in sequential patterns satisfactorily.

Interestingly, we can see that the DCNN model is superior to the LSTM model. It is generally considered that LSTM models are more appropriate for processing sequential data. However, the proposed model was designed to capture sequential characteristics between consecutive CAN messages using a convolution operation. The CAN messages in a certain time window were regarded as Two-Dimensional matrix snapshot of CAN traffic, and the model was induced to learn the pattern of the snapshot. LSTM still demonstrates a performance advantage when compared to other traditional machine-learning algorithms; however, it does not satisfy the performance requirements of the proposed DCNN model.

When compared to other traditional algorithms, the ANN model also demonstrated decent detection performance against DoS, gear spoofing, and RPM spoofing attacks; however, it failed to detect fuzzy attacks satisfactorily with approximately 2% FNR and 1.37% ER. On the fuzzy attack dataset, the deep neural network models, such as DCNN and LSTM, demonstrated significant detection performance with low FNR and ER, whereas the other models performed poorly. These results are owing to the complexity of the attack. The DoS attack and spoofing attacks are relatively simple because they repeatedly inject specific messages. However, fuzzy attack injects random messages, which creates more complex traffic patterns. This results in difficulty for models to learn the attack pattern and degrades the detection performance.

Support Vector Machine (SVM), k-nearest neighbors (kNN) and NB demonstrate high precision, but have a low recall, which results in low F1-scores. This implies that there are no false alarms, but these classifiers tend to be biased toward the normal class.

kNN algorithm, which is a clustering-based algorithm, demonstrated adequate performance for DoS, gear spoofing, and RPM spoofing attacks, but did not detect fuzzy attacks. The kNN model classified most of the data as normal. As the fuzzy attack data is randomly generated and scattered, it appears that the kNN algorithm could not generate appropriate clusters.

6.5. Time cost of training and test

In this research, the hardware used to test the performance of the proposed model were two 2.30 GHz Intel Xeon CPUs and a Nvidia Tesla K80 GPU. The CPUs utilized are single core processors with two threads and the GPU has 2,496 CUDA cores with 12 GB GDDR5 VRAM.

Table 7 shows that the training time on the "CPU only setting" takes longer than the "CPU with GPU acceleration setting". We achieved nearly 50 times faster performance by using the GPU acceleration when compared to training using only CPUs; moreover, the time cost was proportional to the batch size.

Table 6
Comparison with other algorithms.

DoS	FNR	ER	Precision	Recall	F1
Reduced Inception-ResNet	**0.10%**	**0.03%**	**1.0**	**0.9989**	**0.9995**
LSTM (256 hidden units)	0.22%	0.07%	1.0	0.9978	0.9988
ANN (2 hidden layers)	0.18%	0.07%	0.9995	0.9982	0.9988
Support Vector Machine	0.56%	0.17%	1.0	0.9944	0.9972
k-nearest neighbors (k = 5)	0.70%	0.22%	0.9998	0.9929	0.9964
Naïve Bayes	1.18%	0.35%	1.0	0.9882	0.9941
Decision Trees	1.18%	1.34%	0.9762	0.8681	0.9776
Gear Spoofing	FNR	ER	Precision	Recall	F1
Reduced Inception-ResNet	**0.11%**	**0.05%**	**0.9999**	**0.9989**	**0.9994**
LSTM (256 hidden units)	0.32%	0.24%	0.9975	0.9968	0.9972
ANN (2 hidden layers)	0.16%	0.11%	0.9989	0.9984	0.9986
Support Vector Machine	0.35%	0.15%	1.0	0.9965	0.9982
k-nearest neighbors (k = 5)	1.58%	0.67%	1.0	0.9842	0.9920
Naïve Bayes	0.84%	0.36%	1.0	0.9916	0.9958
Decision Trees	2.19%	1.72%	0.9815	0.9781	0.9798
RPM Spoofing	FNR	ER	Precision	Recall	F1
Reduced Inception-ResNet	**0.05%**	**0.03%**	**0.9999**	**0.9994**	**0.9996**
LSTM (256 hidden units)	0.30%	0.13%	1.0	0.9971	0.9985
ANN (2 hidden layers)	0.11%	0.09%	0.9990	0.9989	0.9989
Support Vector Machine	0.23%	0.11%	1.0	0.9977	0.9988
k-nearest neighbors (k = 5)	0.80%	0.36%	0.9999	0.9920	0.9960
Naïve Bayes	0.51%	0.23%	1.0	0.9949	0.9974
Decision Trees	2.18%	1.68%	0.9842	0.9782	0.9812
Fuzzy	FNR	ER	Precision	Recall	F1
Reduced Inception-ResNet	**0.35%**	**0.18%**	**0.9995**	**0.9965**	**0.9980**
LSTM (256 hidden units)	0.84%	0.65%	0.9936	0.9916	0.9926
ANN (2 hidden layers)	1.97%	1.37%	0.9886	0.9803	0.9844
Support Vector Machine	4.45%	2.26%	0.9928	0.9555	0.9738
k-nearest neighbors (k = 5)	93.42%	41.18%	1.0	0.0658	0.1236
Naïve Bayes	9.03%	4.25%	0.9933	0.9098	0.9497
Decision Trees	10.26%	7.23%	0.9359	0.8974	0.9163

Table 7
Training and test time of the DCNN model on different hardwares.

Hardware	Batch size	Training (sec/batch)	Test (sec/batch)
CPUs only	64	1.1805	0.2783
	128	2.2950	0.5338
	256	4.5776	1.0632
CPUs w/ GPU	64	0.0221	0.0063
	128	0.0400	0.0109
	256	0.0749	0.0203

Table 8
Training and test time of the LSTM model on different hardwares.

Hardware	Batch size	Training (sec/batch)	Test (sec/batch)
CPUs only	64	0.0451	0.0158
	128	0.0737	0.0260
	256	0.1205	0.0423
CPUs w/ GPU	64	0.0167	0.0103
	128	0.0169	0.0106
	256	0.0172	0.0115

We also measured training and test time of the LSTM model as listed in Table 8 for comparison. Unlike the DCNN model, the efficiency of the GPU acceleration depends on the batch size for this model. Even if the batch size was increased, the time cost of the LSTM model was not significantly changed. For a batch size of 64, the training time per iteration using CPUs only and using GPU acceleration were approximately tripled at 0.0451 s and 0.0167 s, respectively; however, the difference increased to approximately seven times at 0.1205 s and 0.0172 s for a batch size of 256.

In the learning phase using GPU acceleration, the LSTM model showed a strength that was approximately four times faster than DCNN, for large batch sizes. Conversely, in the test phase, both models demonstrated a similar inference time of approximately 0.01 s for a batch size of 128. The DCNN model is advantageous when the batch size is smaller than 128, and the LSTM model is advantageous when the batch size is larger than 128. However, in practice, it is better to set a smaller batch size because large batches require a longer time to collect messages and can delay intrusion detection.

6.6. Detection latency

Because CAN is a time critical system, the detection latency is an important safety-related metric for the real-time detection performance. Detection latency t_l is the time from when an attack occurs to when it is detected, which is defined as:

$$t_l = t_d - t_a \ (where\ t_{cs} \le t_a \le t_{ce}), \tag{16}$$

where, t_d represents the time when an attack is detected, and t_a represents the time when the attack occurred. t_{cs} and t_{ce} are the start time and end time of data collection duration for an input batch, respectively.

t_d is the sum of t_{ce} and t_p, where t_p is the processing time of a model. Then, detection delay t_l can be rewritten as:

$$t_l = t_{ce} + t_p - t_a, \tag{17}$$

and then,

$$t_p \le t_l \le \Delta t_c + t_p \ (where\ \Delta t_c = t_{ce} - t_{cs}). \tag{18}$$

The detection delay becomes maximum when t_{cs} and t_a are equal and becomes minimum when t_{ce} and t_a are equal. Therefore, both Δt_c and t_p should be reduced to minimize the detection delays. However, the processing time t_p and data collection duration Δt_c depend on the batch size. Thus, the only factor we have to consider is the batch size.

Although a larger batch size is more efficient in terms of computing operation, we set the batch size to one and measured the inference time to minimize the detection latency.

With GPU acceleration, the proposed model required 5 ms to inference one sample, which consists of 29 CAN messages. This implies that the model can process 5,800 messages per second by performing 200 inferences. Conversely, with CPUs only, the model required 6.7 ms to inference one sample and could process approximately 4,300 messages per second by performing 150 inferences.

As the CAN bus of the test vehicle delivers approximately 2,000 messages per second, the proposed model has enough capacity to process twice using CPUs only and thrice using GPU acceleration for real-time detection.

6.7. Discussion and limitation

The dataset used in this study was collected from a real vehicle and is publicly available online [12]. This dataset is the only open dataset that contains normal and attack CAN traffic with labels. Our research group is the only group distributing in-vehicle network intrusion datasets for security research. Currently, the published datasets contain only message injection attacks; however, it will be continuously updated to include sophisticated attack types, such as message tampering.

In the experiment conducted in this study, the proposed DCNN-based model demonstrated good detection performance against message injection attacks. However, the model has a fundamental limitation in detecting unlearned types of attacks because it is based on supervised learning. To solve this problem, additional research on unknown attack detection is required using advanced learning techniques, such as adversarial training. In addition to detecting messages injected from the outside, semantic features must be utilized to detect the transmission of erroneous data owing to vehicle failure, which requires cooperation from vehicle vendors.

Obviously, the time performance depends on the computing power. Although the performance evaluation of the model was performed in a computer-based environment in this study, we believe that the proposed model will be suitable for real-time detection when a computing device with higher performance is utilized.

7. Conclusion

This research primarily focused on building an intrusion detection model that learns sequential patterns of in-vehicle network traffic and detects message injection attacks based on the changes in the traffic. We proposed a DCNN-based IDS. The proposed IDS was designed utilizing the structure of the Inception-ResNet model, which was originally designed for large-scale image classification. The Inception-ResNet architecture is too large and complicated to fit directly into the in-vehicle network data; consequently, we redesigned the DCNN by reducing the size and the number of layers throughout the architecture. To use the bit-wise CAN ID sequence of CAN messages from the in-vehicle network traffic directly as an input to the DCNN classifier without additional feature engineering, we proposed a novel module, named frame builder, which generates a 2-D data frame similar to an image with sequential bit-wise identifiers of CAN messages. This module enabled the DCNN classifier to learn temporal sequential patterns of input data.

Our experiments involved four categories of message injection attacks that exploited the CAN bus and are likely to happen in a vehicle environment connected to an external network. We constructed datasets of the four types of message injection attacks, which are publicly available. The experiment results showed that

the proposed IDS was able to identify the sequential patterns between the in-vehicle network traffic data. It demonstrated that the DCNN-based IDS is reliable and efficient in intrusion detection for identifying the attacking traffic, including DoS, spoofing of RPM, spoofing of the gear, and fuzzy attacks from normal traffic.

One of the primary focuses of this study was to evaluate the performance of DCNN when compared to other famous machine-learning algorithms, such as LSTM, ANN, SVM, kNN, NB, and decision trees. Although the LSTM and ANN showed better performance than other traditional algorithms, they still have a FNR and an ER that is double that of the proposed algorithm. In particular, for the fuzzy attack dataset, the proposed DCNN model demonstrated a significant performance improvement over other algorithms; thus, the DCNN model is much more effective in the case of complex irregular random attacks than other machine-learning techniques.

In the future, we plan to conduct further research to consider semantic features through cooperation with vendors and to improve the DCNN model to detect unknown attack types applying adversarial training techniques.

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.

Acknowledgement

This work was supported by the Institute for Information & Communications Technology Promotion (IITP) grant, funded by the Korea government (MSIT) (No. R7117-16-0161, Anomaly Detection Framework for Autonomous Vehicles).

References

[1] G. Leen, D. Heffernan, Expanding automotive electronic systems, Computer 35 (1) (2002) 88–93.
[2] N.A. Stanton, M. Young, B. McCaulder, Drive-by-wire: the case of driver workload and reclaiming control with adaptive cruise control, Saf. Sci. 27 (2) (1997) 149–159.
[3] S.-H. Kim, S.-H. Seo, J.-H. Kim, T.-M. Moon, C.-W. Son, S.-H. Hwang, J.W. Jeon, A gateway system for an automotive system: LIN, CAN, and FlexRay, in: 6th IEEE International Conference on Industrial Informatics, 2008, INDIN 2008, IEEE, 2008, pp. 967–972.
[4] M. Farsi, K. Ratcliff, M. Barbosa, An overview of controller area network, Comput. Control Eng. J. 10 (3) (1999) 113–120.
[5] C. Miller, C. Valasek, Remote exploitation of an unaltered passenger vehicle, in: Black Hat USA, 2015.
[6] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al., Comprehensive experimental analyses of automotive attack surfaces, in: USENIX Security Symposium, San Francisco, 2011, pp. 77–92.
[7] T. Hoppe, S. Kiltz, J. Dittmann, Security threats to automotive CAN networks—practical examples and selected short-term countermeasures, Reliab. Eng. Syst. Saf. 96 (1) (2011) 11–25.
[8] R.M. Ishtiaq Roufa, H. Mustafaa, S.O. Travis Taylora, W. Xua, M. Gruteserb, W. Trappeb, I. Seskarb, Security and privacy vulnerabilities of in-car wireless networks: a tire pressure monitoring system case study, in: 19th USENIX Security Symposium, Washington, DC, 2010, pp. 11–13.
[9] C. Szilagyi, P. Koopman, Low cost multicast authentication via validity voting in time-triggered embedded control networks, in: Proceedings of the 5th Workshop on Embedded Systems Security, ACM, 2010, p. 10.
[10] C.-W. Lin, A. Sangiovanni-Vincentelli, Cyber-security for the controller area network (CAN) communication protocol, in: 2012 International Conference on Cyber Security (CyberSecurity), IEEE, 2012, pp. 1–7.
[11] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[12] H.M. Song, H.K. Kim, Can network intrusion datasets, http://ocslab.hksecurity.net/Datasets/car-hacking-dataset. (Accessed 30 December 2018).
[13] C. Miller, C. Valasek, Adventures in automotive networks and control units, in: DEF CON, vol. 21, 2013, pp. 260–264.
[14] M. Müter, N. Asaj, Entropy-based anomaly detection for in-vehicle networks, in: Intelligent Vehicles Symposium (IV), 2011 IEEE, IEEE, 2011, pp. 1110–1115.
[15] U.E. Larson, D.K. Nilsson, E. Jonsson, An approach to specification-based attack detection for in-vehicle networks, in: Intelligent Vehicles Symposium, 2008 IEEE, IEEE, 2008, pp. 220–225.
[16] H.M. Song, H.R. Kim, H.K. Kim, Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network, in: 2016 International Conference on Information Networking (ICOIN), IEEE, 2016, pp. 63–68.
[17] K.-T. Cho, K.G. Shin, Fingerprinting electronic control units for vehicle intrusion detection, in: 25th USENIX Security Symposium (USENIX Security 16), USENIX Association, 2016, pp. 911–927.
[18] M.-J. Kang, J.-W. Kang, Intrusion detection system using deep neural network for in-vehicle network security, PLoS ONE 11 (6) (2016) e0155781.
[19] A. Taylor, S. Leblanc, N. Japkowicz, Anomaly detection in automobile control network data with long short-term memory networks, in: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2016, pp. 130–139.
[20] E. Seo, H.M. Song, H.K. Kim, GIDS: GAN based intrusion detection system for in-vehicle network, in: 2018 16th Annual Conference on Privacy, Security and Trust (PST), IEEE, 2018, pp. 1–6.
[21] C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu, X. Cheng, A distributed anomaly detection system for in-vehicle network using HTM, IEEE Access 6 (2018) 9091–9098.
[22] M. Levi, Y. Allouche, A. Kontorovich, Advanced analytics for connected car cybersecurity, in: 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), IEEE, 2018, pp. 1–7.
[23] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection for discrete sequences: a survey, IEEE Trans. Knowl. Data Eng. 24 (5) (2012) 823–839.
[24] Z. Xing, J. Pei, E. Keogh, A brief survey on sequence classification, ACM SIGKDD Explor. Newsl. 12 (1) (2010) 40–48.
[25] S.A. Hofmeyr, S. Forrest, A. Somayaji, Intrusion detection using sequences of system calls, J. Comput. Secur. 6 (3) (1998) 151–180.
[26] A.P. Kosoresow, S. Hofmeyer, Intrusion detection via system call traces, IEEE Softw. 14 (5) (1997) 35–42.
[27] W. Lee, S.J. Stolfo, et al., Data mining approaches for intrusion detection, in: USENIX Security Symposium, San Antonio, TX, 1998, pp. 79–93.
[28] C.-M. Ou, Host-based intrusion detection systems adapted from agent-based artificial immune systems, Neurocomputing 88 (2012) 78–86.
[29] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, Y. Weiss, "Andromaly": a behavioral malware detection framework for android devices, J. Intell. Inf. Syst. 38 (1) (2012) 161–190.
[30] M.H. Bhuyan, D.K. Bhattacharyya, J.K. Kalita, Network anomaly detection: methods, systems and tools, IEEE Commun. Surv. Tutor. 16 (1) (2014) 303–336.
[31] R.R. Karthick, V.P. Hattiwale, B. Ravindran, Adaptive network intrusion detection system using a hybrid approach, in: 2012 Fourth International Conference on Communication Systems and Networks (COMSNETS), IEEE, 2012, pp. 1–7.
[32] N. Goldenberg, A. Wool, Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems, Int. J. Crit. Infrastruct. Prot. 6 (2) (2013) 63–75.
[33] S.J. Yu, P. Koh, H. Kwon, D.S. Kim, H.K. Kim, Hurst parameter based anomaly detection for intrusion detection system, in: 2016 IEEE International Conference on Computer and Information Technology (CIT), IEEE, 2016, pp. 234–240.
[34] H. Kwon, T. Kim, S.J. Yu, H.K. Kim, Self-similarity based lightweight intrusion detection method for cloud computing, in: Asian Conference on Intelligent Information and Database Systems, Springer, 2011, pp. 353–362.
[35] R. Mitchell, I.-R. Chen, A survey of intrusion detection techniques for cyber-physical systems, ACM Comput. Surv. 46 (4) (2014) 55.
[36] W. Wang, M. Zhu, X. Zeng, X. Ye, Y. Sheng, Malware traffic classification using convolutional neural network for representation learning, in: 2017 International Conference on Information Networking (ICOIN), IEEE, 2017, pp. 712–717.
[37] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.