Received 29 March 2023, accepted 13 April 2023, date of publication 19 April 2023, date of current version 25 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3268519

RESEARCH ARTICLE

G-IDCS: Graph-Based Intrusion Detection and Classification System for CAN Protocol

SUNG BUM PARK[1], HYO JIN JO[2], AND DONG HOON LEE[1], (Member, IEEE)
[1]Graduate School of Information Security, Korea University, Seoul 02841, Republic of Korea
[2]School of Software, Soongsil University, Seoul 06978, Republic of Korea

Corresponding authors: Dong Hoon Lee (donghlee@korea.ac.kr) and Hyo Jin Jo (hyojin.jo@ssu.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Korean Government Ministry of Science and ICT (MSIT) under Grant NRF-2021R1A2C2014428 and Grant NRF-2021R1A4A1029650.

ABSTRACT The security of in-vehicle networks has become an important issue as automobiles become more connected and automated. In this paper, we propose a graph-based intrusion detection and classification system, named G-IDCS, which aims to enhance the security of the in-vehicle controller area network (CAN) protocol. Existing intrusion detection systems (IDSs) using graph theory suffer from limitations, such as requiring a large number of CAN messages for detection and being unable to classify attack types despite analyzing numerous messages. Meanwhile, machine learning or deep learning-based systems have limited sensitivity to environmental changes such as attack type change due to model overfitting, and are unable to provide explanations for classification decisions. Using various graph features, our threshold-based intrusion detection method overcomes these limitations by integrating a threshold-based IDS and a machine learning-based attack type classifier. Our threshold-based intrusion detection method of G-IDCS reduces the number of CAN messages required for detection by more than 1/30 and improves the accuracy of combined attack detection by over 9% compared to an existing intrusion detection method that uses graph theory. Furthermore, unlike existing machine learning and deep learning-based intrusion detection systems, our threshold classifier is robust to changes in attack types and can provide explanations for the features used in attack detection. In addition, our machine learning-based attack type classifier outperforms existing techniques in all performance metrics and can serve as a digital forensic tool for investigating cyber attacks on in-vehicle networks. Using the classifier to identify attack types can facilitate the design of corresponding protection methods, thereby enhancing the security of in-vehicle networks.

INDEX TERMS Attack type classification, controller area network, graph theory, intrusion detection system, in-vehicle CAN security.

I. INTRODUCTION

Modern vehicles are equipped with dozens of electronic control units (ECUs), which direct almost every operation of the vehicle from engine control to advanced driver assistance systems (ADAS), such as adaptive cruise control (ACC) and lane departure warning (LDW). The controller area network (CAN) protocol was developed for in-vehicle network applications in the mid-1980s, and has robustness, reliability, and low-cost characteristics. It has been widely used as a de facto standard for ECU communication [1]. However, the CAN protocol was designed without much consideration for security measures like encryption, access control, or message authentication because it was thought to be a closed network that did not have any external access points [2].

However, since the on-board diagnostic port (OBD-II) came to be installed in every vehicle, access to the in-vehicle CAN became possible through this port. In addition, various communication technologies such as Wi-Fi, cellular networks, and vehicle-to-everything (V2X) have been developed to allow external entities like other vehicles and infrastructure to communicate with the vehicle, which increases the attack

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

surfaces of the in-vehicle CAN [3], [4], [5]. In [6], [7], [8], [9], [10], [11], and [12], researchers showed how to perform cyber attacks on the in-vehicle CAN through remote access points including Wi-Fi, Bluetooth, and mobile (phone) networks as well as physical access points such as OBD-II ports, USB ports, and CD players. When an attacker accesses the in-vehicle CAN through various access points, a malicious packet is injected to make the vehicle operate abnormally, and in the worst case scenario, the attacker can take control of the vehicle [13].

As attacks on the in-vehicle CAN increase, researchers have studied numerous intrusion detection systems (IDSs) to detect cyber attacks on this bus standard. There are essentially two categories for in-vehicle CAN IDSs: a packet-based IDS approach and a window-based IDS approach. The number of packets required for the attack detection is the main differentiating factor between these two approaches. The packet-based IDS approach has a benefit, which is that it is relatively fast to detect intrusions since this method only requires one packet to make a decision. Systems using the clock skewness of each ECU [14], time interval of CAN packets [15], voltage characteristics of CAN packets [2], or network time protocol [16] are all classified as packet-based IDSs. However, a packet-based intrusion detection technique can also prove to be disadvantageous because it is difficult for this method to analyze the correlation of consecutive packets. In other words, packet-based IDSs can decide whether a packet is normal or abnormal (i.e., normal or derived from an attacker), but these methods may have difficulty classifying attack types, namely flooding, spoofing, fuzzy, and replay attacks.

On the other hand, the window-based detection approach is beneficial because it utilizes the correlation of consecutive packets. Thus, some of these techniques not only detect attacks but can also classify attack types [17]. The window-based detection technology is also divided into two approaches: a statistical method and a deep/machine learning (DL/ML) method. There is an IDS method that uses the entropy of the CAN ID and is also classified as a window-based detection technology using the statistical method. This method was proposed in [18], but it is difficult for this method to detect small numbers of malicious messages [19]. Han et al. proposed another window-based detection technology using the survival analysis of frequency, but the researchers did not conduct experiments regarding replay attacks [20]. Thus, more experiments should be conducted to evaluate whether or not the proposed technology can detect replay attacks.

To mitigate some of the limitations in existing window-based IDSs, one study introduced a graph theory-based detection technique using a statistical method [21]. In this research, graph theory is used to extract graph characteristics by constructing one graph per N[1] packets and a chi-square distribution is used to detect attack windows containing attack messages. However, in general, statistical testing using the chi-square distribution requires a minimum of 30 samples [22], so this work would have needed to construct at least 30 graphs—meaning it would also require more than N × 30 packets for attack detection. However, obtaining N × 30 packets for window construction leads to a non-negligible intrusion detection delay. More recently proposed window-based IDSs using DL/ML models include the deep convolutional neural network (DCNN)-based IDS [23] and the K-nearest neighbors (KNN)-based IDS [17]. However, if the in-vehicle network environment and attack types change even just slightly, it is difficult for these IDSs using DL/ML models to cope with that change because untrained data may be newly introduced. Additionally, methods using DL/ML are not suitable for direct application to intrusion detection systems for the in-vehicle CAN because they cannot explain the reasons behind the attack detection results [24].

A. MOTIVATION

Numerous IDSs have been studied for securing CAN protocol, but each technology has drawbacks. Packet-based IDSs are limited in their ability to analyze the correlation of consecutive packets, and most are unable to classify attacks. To deal with these limitations, various window-based IDSs have been studied as well. For example, Islam et al. analyzed the correlation of consecutive packets using graph theory and designed an intrusion detection system [21]. However, this work is also unable to classify attack type, and a large number of messages are required for attack detection. In addition, most IDSs using machine learning [17] and deep learning technologies [23] have high performance, but they do not provide robustness or interpretability due to the nature of the technology. To overcome the limitations of existing IDSs, there is a need for an integrated method rather than a method using only one type of technology like a statistical method or ML/DL method.

In light of this, we propose a graph-based intrusion detection and classification system called ‘‘G-IDCS,’’ which consists of a threshold-based attack classifier (TH_classifier) as well as a machine learning-based attack type classifier (ML_classifier). To construct either classifier in G-IDCS, three graph-related features—*graph elapsed time, max degree, and number of edges*—are extracted from the graphs constructed by N CAN packets. TH_classifier sets a threshold value for each feature based on the distribution of the three graph-related features under normal conditions, which allows G-IDCS to distinguish which graphs contain attack packets and which contain only normal packets. By using multiple features, the TH_classifier significantly reduces the number of messages required for detection compared to existing IDSs that use graph theory [21]. Moreover, TH_classifier can provide robustness for attack type change and interpretability about its decision (i.e., it reveals which characteristics were used to detect the attack). While ML_classifier is not as robust or interpretable as the TH_classifier, it offers a means to classify attack types, which can aid in the post-analysis (i.e., attack type classification) of

[1]N denotes window size, which is set as 200 in [21].

CAN-related attacks. In summary, TH_classifier of G-IDCS is robust against attack type changes and can provide explanations for intrusion detection decisions, while ML_classifier can classify attack types, which can be used for post-analysis of CAN-related attacks.

B. OUR CONTRIBUTIONS

Our main contributions are as follows:

1) In this study, we developed a threshold-based attack detection technology dubbed the TH_classifier of G-IDCS that utilizes various characteristics of graphs—graph elapsed time, max degree, and number of edges—which are constructed by N CAN packets. Compared to the existing window-based IDS using graph theory [21], our technique significantly reduces the number of packets needed for intrusion detection, which results in shorter detection time. Moreover, the TH_classifier's detection accuracy for combined attacks is 9% higher than that of [21].
   Lastly, the TH_classifier of G-IDCS is robust against changes in attack types and can provide interpretability about its decision, whereas DL/ML-based IDS systems are not able to do so [17], [23].
2) The proposed attack classifier named the ML_classifier of G-IDCS can identify car attack types whereas most packet-based IDSs cannot. Through our evaluation, we confirmed that ML_classifier of G-IDCS has higher accuracy than existing machine learning-based attack type classification technology [17].
3) Our proposed IDS can be used without modifying the CAN protocol, and as such, it can be applied to any intrusion detection system in any system based on the CAN protocol.

The rest of this paper is organized as follows. Section II gives an overview of the related works, followed by background about the CAN protocol and graph theory in Section III. The system model and detailed proposed method are described in Section IV and Section V, respectively. The experimental results and evaluation are described in Section VI. Finally, we conclude the paper in Section VII. For ease of reference, some important symbols and acronyms are summarized in Table 1.

II. RELATED WORKS

Various IDS technologies have been studied for in-vehicle CAN security, and there are several taxonomies that can be used to classify in-vehicle IDSs [25]. In the survey of [26], a CAN-based IDS is classified based on three aspects: the number of frames required for the IDS to detect an attack, the data used for detection, and how the detection model is built. In view of the first aspect, we have subdivided IDSs further into two types: packet-based IDS and window-based IDS.

A. PACKET-BASED INTRUSION DETECTION SYSTEM

In this section, we describe intrusion detection systems that detect attacks using information that can be obtained from a single packet. Kang et al. [27] extracted the high-dimensional characteristics of the CAN packet and trained the classifier with a deep neural network. This classifier was used to distinguish between normal packets and malicious packets. This study is limited in that it only uses simulation data for evaluation as opposed to real vehicle data. Lee et al. [28] subsequently proposed an active approach using the remote frame of the CAN protocol. In this method, the intrusion detection system used time interval analysis and an offset ratio between the ECU's response frame to the remote frame. However, the method increases bus traffic by injecting remote frames for analysis. Groza et al. [29] then proposed an intrusion detection mechanism based on the idea that most traffic from the in-vehicle bus is cyclic in nature, and that the format of the data field is fixed due to rigid signal allocation. The researchers designed their IDS to utilize the bloom filter and test frame periodicity based on the identifier and part of the CAN frame data field. The method is limited in that it is only suitable for periodic CAN frames, not for aperiodic ones [30]. Cho et al. next proposed Clock-based IDS (CIDS), an ECU identification and intrusion detection technology using clock skew, which is a unique characteristic of ECU hardware. In the study, the clock operation baseline of the ECU was constructed using the recursive least squares (RLS) algorithm. Then, based on the constructed model, CIDS detected the intrusion through a CUSUM analysis [14]. However, the CIDS cannot be applied to aperiodic messages and, in addition, Ying et al. also brought up the issue of a cloaking attack, in which an adversary adjusts message inter-transmission times and cloaks its clock to match the targeted ECU's clock skew to avoid being detected by CIDS [16]. Another IDS that uses ECU hardware characteristics, Choi et al. [2] leverages the inimitable characteristics of electrical CAN signals as so-called fingerprints of ECUs. In this study, various characteristics from ECU voltage signals were used to detect malicious messages, but it is limited in that the electrical characteristics may change as the vehicle ages [31]. Although these packet-based IDSs can detect attacks

TABLE 1. Description of symbols and acronyms.

Symbols/Acronyms	Description
TH_classifier	Threshold based attack classifier of G-IDCS
ML_classifier	Machine learning-based attack type classifier of G-IDCS
Upper_gtime, U_1	Upper threshold of graph elapsed time for attack detection
Lower_gtime, L_1	Lower threshold of graph elapsed time for attack detection
Upper_maxDegree, U_2	Upper threshold of max degree for attack detection
Lower_maxDegree, L_2	Lower threshold of max degree for attack detection
Upper_edgeNum, U_3	Upper threshold of number of edge for attack detection
Lower_edgeNum, L_3	Lower threshold of number of edges for attack detection

relatively fast, they are disadvantageous because the correlation of consecutive packets cannot be used.

B. WINDOW-BASED INTRUSION DETECTION SYSTEM

This section describes a system that detects intrusions using a series of CAN packet sequences called windows. A window can include a pre-defined number of packets, or it can include all packets generated for a pre-defined time.

Muter et al. designed an entropy-based IDS based on the idea that the traffic of automobile networks is much more limited compared to standard computer networks. In this study, entropy is defined as ''a measure of how many coincidences are contained in a given data set,'' and it is used to detect abnormal behavior in the network [18]. However, the experimental evaluation is limited [32], and the method had difficulty detecting a small number of attack messages injected by adversaries [19]. Olufowobi et al. proposed a real-time specification-based IDS that extracts a timing model by observing message timing and worst-case response time. The IDS can detect anomalies without predefined specifications, but it has limited capabilities in detecting aperiodic messages as well as message IDs with several message instances per period [33].

There are several frequency-based studies within window-based IDS research. Taylor et al. studied the characteristics of the CAN bus, such as the frequency of the CAN packet and the hamming distance of the data field in the packet [34]. Bozdal et al. proposed an IDS designed using continuous wavelet transform to ascertain the location of frequency components. This is a vehicle-independent IDS approach for attack detection without prior knowledge [35]. However, frequency-based IDSs are insufficient in that they cannot determine whether or not aperiodic CAN packets are being used for an attack [35], [36]. Han et al. designed an intrusion detection system that looked at the survival analysis of CAN ID frequency [20], but the researchers did not conduct experiments to test against replay attack.

Islam et al. also developed an intrusion detection system for the in-vehicle CAN using graph theory. After composing a graph with a pre-defined number of packets, the researchers extracted the graph characteristics. Then, using the chi-squared test, they developed a technique for classifying the normal window graph and the attack window graph. In addition, they introduced a median test to detect replay attacks. However, this method has a major downfall, which is that it requires too many packets[2] to detect an attack.

Recently, a window-based IDS using ML/DL has also been developed. Song et al. proposed a deep convolutional neural network (DCNN)-based IDS that reduces unnecessary complexity in the architecture of the Inception-ResNet model in order to develop a high-accuracy IDS in the in-vehicle CAN. In general, convolutional neural networks are designed to accept grid-type data as input and utilize spatial local correlation. In the study, the researchers collected CAN IDs from 29 sequential CAN packets and constructed a 29×29 2-D grid data frame to use as input for convolutional neural networks [23]. The (DCNN)-based IDS has high performance, but if the network environment or attack types change even just slightly, it is difficult for the IDS to use due to overfitting. Derhab et al. then proposed a histogram-based IDS (H-IDS). The authors constructed a histogram with each byte value of the data field in a sequence of CAN packets. Afterward, they developed a multi-class classification technique that classified attack types by classifying the histogram using the KNN algorithm [17]. However, it is easy for an attacker who can inject attack messages to change the data field, but a histogram-based IDS uses only the data field as a feature, so it is difficult for the IDS to classify such an attacker that manipulates the data field. Moreover, methods using DL/ML cannot explain the reason behind attack detection results and are difficult to use when the attack type changes.

TH_classifier of G-IDCS reduces window size by utilizing various features compared to IDS using existing graph theory, and provides interpretability and robustness. ML_classifier of G-IDCS can classify attack type by utilizing message correlation of consecutive packets, which is not utilized by a packet-based IDS. In short, the G-IDCS is a system that integrates threshold-based attack classification technology (TH_classifier) using graph theory and machine learning-based attack type classification technology (ML_classifier) to overcome the shortcomings of existing IDSs.

[2]In [21] to construct one graph—specifically, 200 packets are required. Since the chi-squared test typically requires more than 30 samples, this means that at least 6,000 packets are necessary for attack detection.

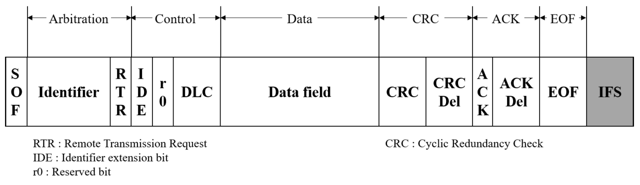


FIGURE 1. Format of CAN data frame (base frame format).

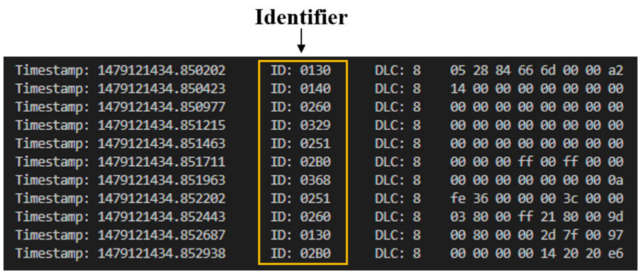
III. BACKGROUND

In this section, we introduce background information on the CAN protocol and graph theory.

A. CAN PROTOCOL

The CAN protocol is a serial communication protocol that efficiently supports distributed real-time control. It has been widely used for communication between ECUs on the vehicle bus. On the CAN bus, transmitting nodes broadcast messages and receiving nodes are able to decide via message filtering whether the data is to be acted upon or not [37]. As shown in Figure1, a CAN data frame consists of seven fields. It starts with a start-of-frame (SOF) with a dominant bit (1 bit).[3] The arbitration field includes an identifier (11 bits) and an

[3]In the CAN bus environment, bit 0 is the dominant bit, and bit 1 is the recessive bit.



```
Timestamp: 1479121434.850202    ID: 0130    DLC: 8    05 28 84 66 6d 00 00 a2
Timestamp: 1479121434.850423    ID: 0140    DLC: 8    14 00 00 00 00 00 00 00
Timestamp: 1479121434.850977    ID: 0260    DLC: 8    00 00 00 00 00 00 00 00
Timestamp: 1479121434.851215    ID: 0329    DLC: 8    00 00 00 00 00 00 00 00
Timestamp: 1479121434.851463    ID: 0251    DLC: 8    00 00 00 00 00 00 00 00
Timestamp: 1479121434.851711    ID: 02B0    DLC: 8    00 00 00 ff 00 ff 00 00
Timestamp: 1479121434.851963    ID: 0368    DLC: 8    00 00 00 00 00 00 00 0a
Timestamp: 1479121434.852202    ID: 0251    DLC: 8    fe 36 00 00 00 3c 00 00
Timestamp: 1479121434.852443    ID: 0260    DLC: 8    03 80 00 ff 21 80 00 9d
Timestamp: 1479121434.852687    ID: 0130    DLC: 8    00 80 00 00 2d 7f 00 97
Timestamp: 1479121434.852938    ID: 02B0    DLC: 8    00 00 00 00 14 20 20 e6
```

FIGURE 2. Sample CAN packets that contain a timestamp, an identifier, a DCL, and a data field.

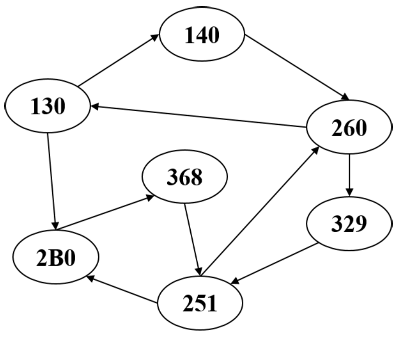


FIGURE 3. A simple example of a graph constructed from CAN packets.

RTR (1 bit) for base frame format, and the RTR must be dominant for data frames. The control field consists of an IDE (1 bit), a reserved bit (1 bit), and a DLC (4 bits) indicating the length of the data field. The data field has a different length depending on the DLC and includes data to be transmitted. The CRC field consists of a cyclic redundancy check (15 bits) and a CRC delimiter (1 bit) that has recessive bit. The ACK field (1 bit) must be dominant during normal communication, and the ACK delimiter (1 bit) must be recessive. EOF (7 bits) indicates the end of the frame and has seven recessive bits.

B. GRAPH THEORY

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. In the theory, the graph G consists of a set V of a finite number of vertices and a set E of edges having two vertices as elements. This is denoted as `G = (V, E)`. The line connecting the vertex u and the vertex v is called an edge and is denoted as `{u, v}`. When there is an edge connecting u and v, u and v are said to be adjacent. The number of vertices adjacent to a vertex x is called the degree of x and is denoted as d(x). A directed graph is a graph in which edges have orientations.

In this paper, a directed graph is used to design an intrusion detection system for CAN protocol. Each vertex of the graph represents the identifier of the CAN frame. For example, if a frame with an identifier of 0×260 appears after a frame with an identifier of 0×140, 0×140 and 0×260 are adjacent vertices and can be expressed as edge $\{0 \times 140,\ 0 \times 260\}$. Figure2 shows the message sequence of CAN data. This sequence can be represented as a graph, as shown in Figure3. As such, graph theory provides the advantage of being able to visualize complex CAN data frames within a given window and derive a relationship between frame identifiers.

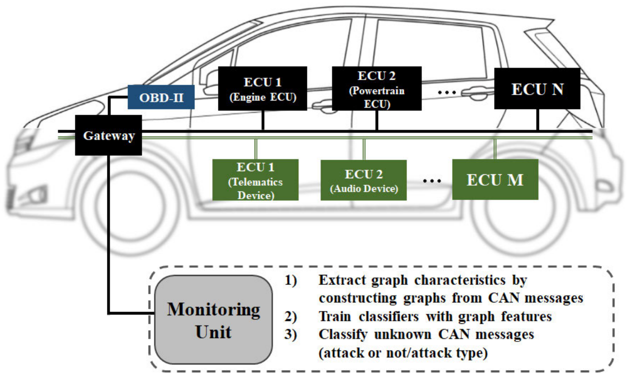

OBD-II
ECU 1 (Engine ECU)
ECU 2 (Powertrain ECU)
...
ECU N
Gateway
ECU 1 (Telematics Device)
ECU 2 (Audio Device)
...
ECU M
Monitoring Unit
1) Extract graph characteristics by constructing graphs from CAN messages
2) Train classifiers with graph features
3) Classify unknown CAN messages (attack or not/attack type)

FIGURE 4. System model.

IV. SYSTEM MODEL

In this section, we discuss the system model, the adversary model, and corresponding attack scenarios.

A. SYSTEM MODEL

In this section, we describe the system model for our method. As shown in Figure 4, the in-vehicle CAN consists of several sub-networks, and there is a gateway. Sender ECUs existing in the in-vehicle CAN broadcast the message ID and data, and the receiver ECUs receive the data after checking the message ID included in the transmitted data frame. In this system, a monitoring unit is assumed to be installed on the gateway (or directly connected to the bus).

The monitoring unit watches the CAN bus and constructs a graph whenever the number of CAN messages reaches the pre-defined value, which is also known as the window size. Then, the unit extracts several graph features from the constructed graph. When an attack on the CAN occurs, such as message injection, the message patterns within the window change, resulting in a change of the graph patterns derived from the window. In other words, characteristics extracted from the graph vary depending on the presence or absence of an attack within the window. Thus, the monitoring unit can detect and classify attacks by constructing a graph for every window and by analyzing graph-related features.

B. ADVERSARY MODEL

An attacker maliciously sends a CAN message to affect the normal operation of a vehicle or to manipulate the vehicle as desired. The attacker can physically or remotely access the in-vehicle network to perform these malicious actions. Note that physical access to the in-vehicle network is possible through the USB port, the OBD-II port, and so on. In addition, the attacker can perform wireless access-based attacks with

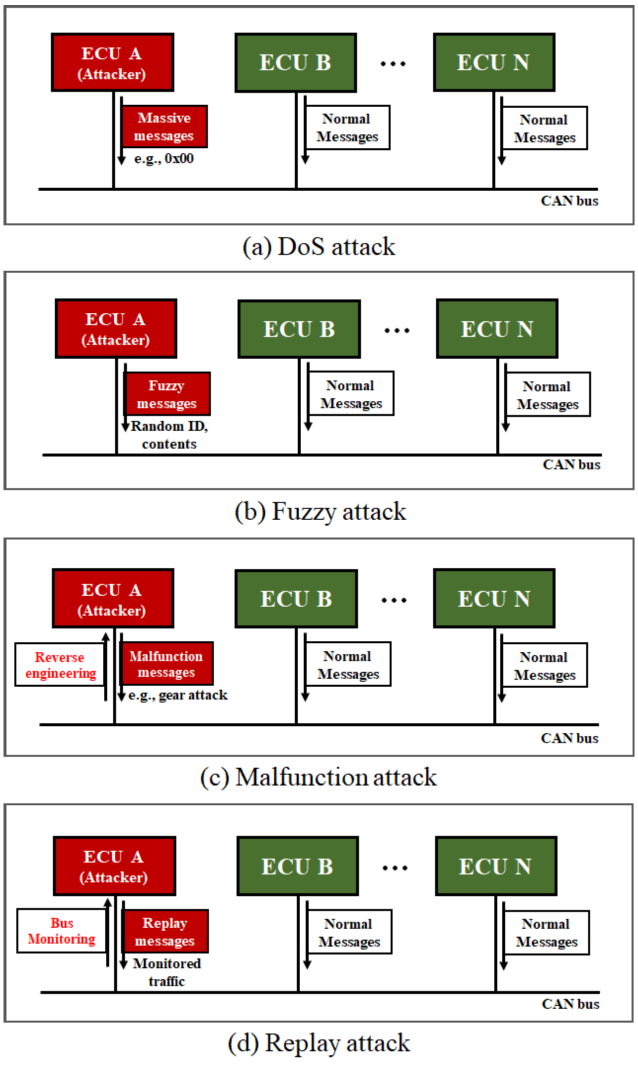


FIGURE 5. **Four attack scenarios on the in-vehicle network.**

initial physical access [7], [8], [36] or without initial physical access [3], [4], [5]. We assume that an attacker who could gain access to the in-vehicle network would perform the following attack scenarios.

C. ATTACK SCENARIOS

In this section, we present four attack scenarios.

1) DoS ATTACK

As shown in Figure 5 (a), a denial of service (DoS) attack in the CAN protocol blocks the transmission of CAN messages by injecting a massive number of packets with high priority (e.g., 0×00). A DoS attack is known as the easiest attack to mount because it only requires broadcasting a number of messages on the bus without the attacker needing to analyze the targeted in-vehicle network. If the DoS attack continues, it can cause the target vehicle to perform unexpectedly due to communication delays between legitimate ECUs.

2) FUZZY ATTACK

As shown in Figure 5 (b), a fuzzy attack in the CAN protocol injects a message with a random spoofed CAN ID and data value. Fuzzy attack messages can be sent on the bus to provoke unexpected failures [36], [38]. In the work [39], the researchers confirmed that several vehicle operations—such as changing vehicle speed, turning on the rear camera, changing gears, flashing the dashboard, moving the steering wheel, flashing lights, and so on—can be triggered by a fuzzy attack in a real vehicle.

3) MALFUNCTION ATTACK

As shown in Figure 5 (c), a malfunction attack injects a packet with a CAN ID related to a specific function in order to take over control of a vehicle (e.g., engine shutdown, steering, and emergency brake, etc.) or cause an error by using the injected packet. In order to perform a malfunction attack, it is necessary to analyze the in-vehicle network and find the specific CAN ID controlling the vehicle's functions [38], [39]. This attack is much more complex than a DoS attack or a fuzzy attack because it requires the attacker to perform a rigorous analysis of the in-vehicle network before launching the attack.

4) REPLAY ATTACK

As shown in Figure 5 (d), a replay attack extracts normal traffic at a specific time and replays (injects) it into the CAN bus [40]. When a replay attack occurs, it can interfere with normal operations of the vehicle. Since this attack retransmits normal traffic without any other changes, it is difficult for the vehicle to distinguish between normal packets and malicious packets when under attack.

V. THE PROPOSED METHOD

A. OVERVIEW

As shown in Figure 6, our proposed method, G-IDCS, consists of a classifier configuration module and a classification module. The classifier configuration module monitors the messages transmitted on the CAN bus. When the number of messages reaches the pre-defined value (i.e., window size), the module extracts the identifier of each message and constructs a graph that can discern the relationships between CAN messages. After that, graph-based features are extracted from the constructed graph, and these features are used to calculate the threshold for the threshold-based binary classifier called TH_classifier and to train the machine learning-based multi-class classifier called ML_classifier. The threshold of the TH_classifier is obtained using the maximum and minimum values of the features extracted from graphs composed of attack-free windows. When training the ML_classifier, the features extracted from graphs with windows containing each type of attack messages with normal messages and attack type labels are used as input values.

A classification module is used to distinguish between windows containing only normal messages and windows

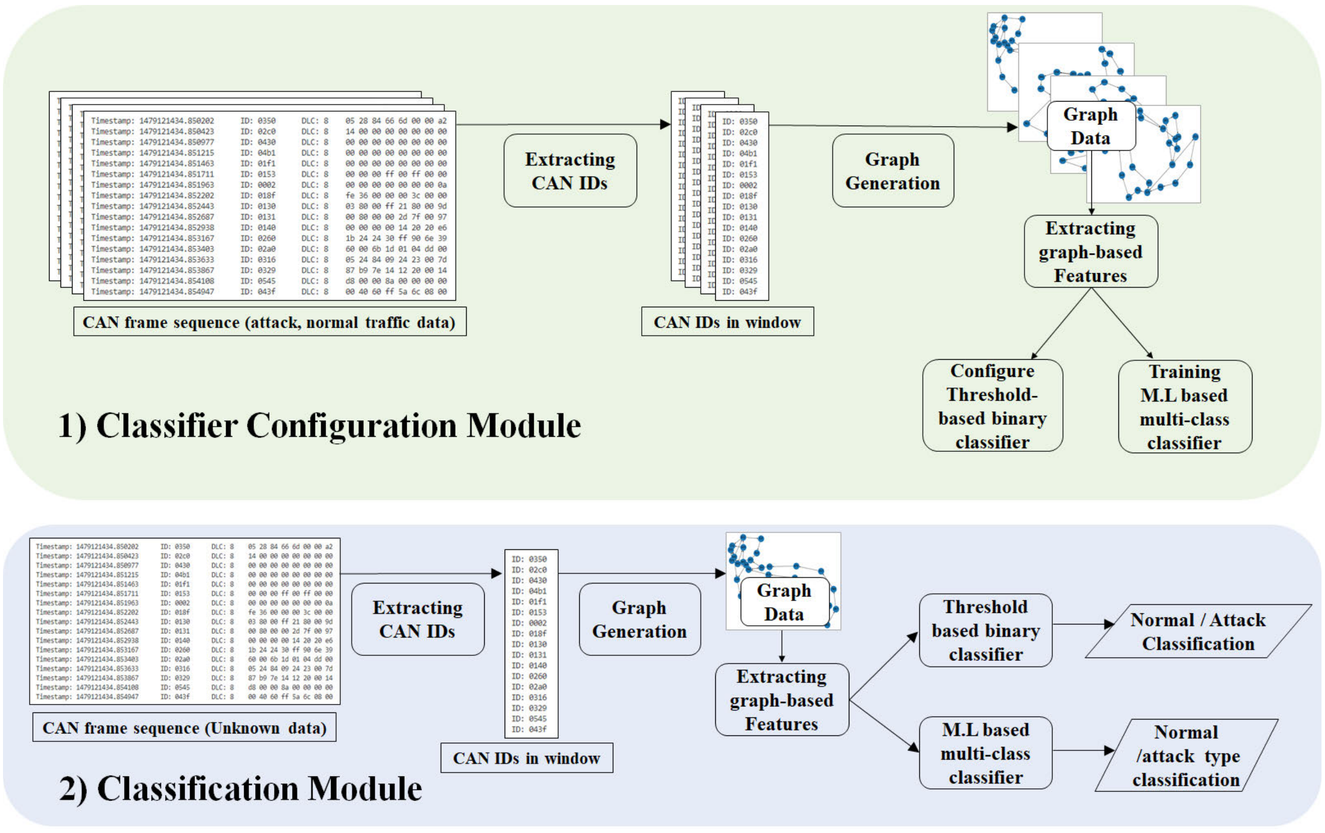

CAN frame sequence (attack, normal traffic data)
Extracting CAN IDs
CAN IDs in window
Graph Generation
Graph Data
Extracting graph-based Features
Configure Threshold-based binary classifier
Training M.L based multi-class classifier
1) Classifier Configuration Module
CAN frame sequence (Unknown data)
Extracting CAN IDs
CAN IDs in window
Graph Generation
Graph Data
Extracting graph-based Features
Threshold based binary classifier
M.L based multi-class classifier
Normal / Attack Classification
Normal /attack type classification
2) Classification Module

FIGURE 6. Overview of G-IDCS.

containing attack messages via the pre-configured TH_classifier and ML_classifier. In this classification module, the module extracts the CAN ID from each message to both construct a graph and extract graph-based features when the number of CAN messages reaches a pre-defined number (i.e., window size). After that, each graph can be classified using the TH_classifier or the ML_classifier.

B. G-IDCS

In this section, we describe the details of G-IDCS, our proposed intrusion detection system, and attack type classification system.

1) GRAPH FEATURES

In G-IDCS, we introduce three graph features—namely `graph elapsed time`, `max degree`, and `number of edges`—for intrusion detection and attack type classification. Then, we evaluate the features of G-IDCS through the latest attack dataset: the Car Hacking-Attack & Defense Challenge [41]. In order to evaluate G-IDCS's features, windows should be constructed whenever the number of CAN packets reaches the pre-defined window size.[4] The distributions of each feature are shown in Figure 7, Figure 8, and

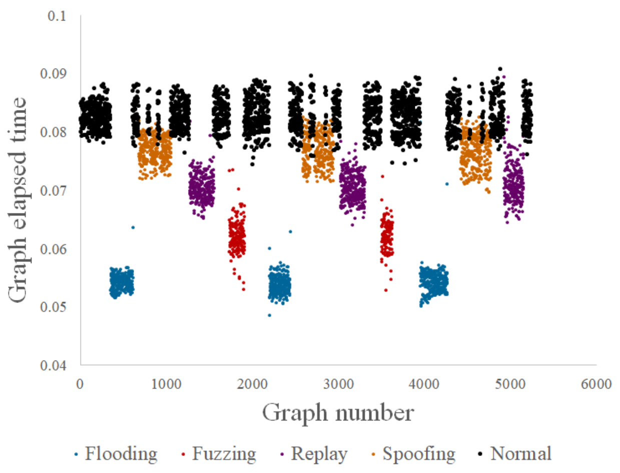


FIGURE 7. The distribution of graph elapsed time.

Figure 9 by attack type (i.e., flooding, fuzzy, replay, and spoofing) when the window size is set to 200.

- `graph elapsed time`: *graph elapsed time* indicates the time it takes to construct one graph. The *graph elapsed time* is calculated as the difference in the timestamps between the last CAN frame and the first CAN frame of the window used for graph construction.

[4]This window size is a system parameter. In this paper, the window size is set to 200 to compare G-IDCS with an existing graph-based IDS method [21].

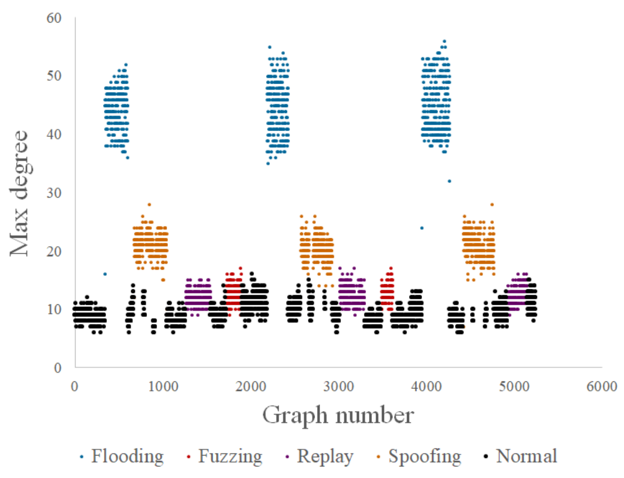


FIGURE 8. The distribution of the max degree.

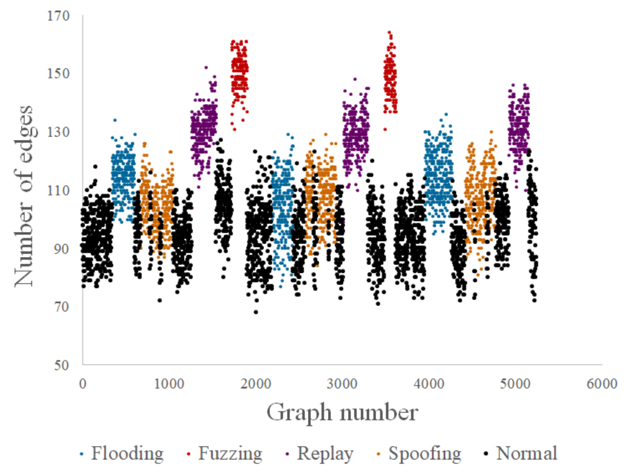


FIGURE 9. The distribution of number of edges.

The *graph elapsed time* of each window without an attack generally does not change significantly because the messages in the window are transmitted periodically. However, when a window includes attack messages, the messages in that one window gather faster than usual since attack messages are being injected in addition to the normal messages. Therefore, the *graph elapsed time* with attack messages is shorter than the *graph elapsed time* with the attack. Figure 7 shows the differences between the *graph elapsed time* without and with each of the four attack types (i.e., flooding, fuzzing, replay, and spoofing) over time. As shown in Figure 7, the *graph elapsed time* tends to decrease when a flooding attack occurs because this type of attack transmits a large number of packets over a short period of time. Likewise, the *graph elapsed time* tends to decrease when other attacks occur; a slight decrease in the *graph elapsed time* means that relatively few attack messages are being injected.

TABLE 2. Detection and misdetection rate of each feature.

Feature	Attack type	Detection rate	Misdetection rate
Graph elapsed time	Flooding	**99.75%**	**0.25%**
	Fuzzing	**100%**	**0%**
	Replay	**92.41%**	**7.59%**
	Spoofing	18.38%	81.61%
Max degree	Flooding	**99.88%**	**0.12%**
	Fuzzing	0.65%	99.34%
	Replay	0.25%	99.75%
	Spoofing	**97.56%**	**2.44%**
Number of edges	Flooding	3.33%	96.67%
	Fuzzing	**100%**	**0%**
	Replay	**66.88%**	**33.12%**
	Spoofing	0.21%	99.79%

- `max degree`: In graph theory, the degree of a vertex[5] is defined as the number of edges that are incident to the vertex. A *max degree* in this paper is defined as the degree value of the node that has the largest number of edges among all the nodes in the graph. As shown in Figure 8, when there is a flooding attack transmitting a large number of high-priority IDs (e.g., 0×00), the *max degree* increases because a high volume of messages that has ID uses for the attack exists within the window. Likewise, when there is a spoofing attack, the *max degree* increases. However, during fuzzy attacks and replay attacks, the *max degree* does not increase as much because these attacks usually exploit multiple message identifiers rather than just one.
- `number of edges`: This indicates the total number of edges in the graph. If messages with new IDs that are not transmitted as normal are injected, the *number of edges* will increase. Figure 9 shows the *number of edges* difference both with and without attacks over time. As shown in Figure 9, when a fuzzy attack injects a message with a spoofed random CAN ID, nodes with new IDs increase more than when other attacks occur, and the *number of edges* increases. Likewise, during other attacks, the *number of edges* increases because new packets are added in between normal message sequences, which creates more graph nodes than a normal window without an attack.
- `performance of graph features`: We performed an analysis of how well the three features detected each attack (i.e., flooding, fuzzing, replay, and spoofing). In the analysis, we find the maximum and minimum ranges of each feature values extracted from normal windows, and determined that the window with feature values outside of the range is under attack. As shown in Table 2, each feature has a high detection rate for a specific attack. The *graph elapsed time* feature

[5]In the directed graph, the degree of a vertex consists of indegree and outdegree, and we calculated the sum of the indegree and the outdegree as the degree of the node [42].

detects flooding, fuzzing, and replay attacks well, while the *max degree* feature detects flooding and spoofing attacks effectively. The *number of edges* feature is particularly effective at detecting fuzzing attacks, and it can also detect some replay attacks. A detailed description of the attack detection and classification method using graph features is explained in the next subsection.

2) CLASSIFIER CONFIGURATION MODULE

In order to configure the G-IDCS, the threshold of the TH_classifier is set, and the classification model for the ML_classifier is trained as follows.

- The threshold values of the TH_classifier: When calculating the threshold values of the TH_classifier, only normal CAN data (i.e., attack-free data) should be used. Thus, the data used for the TH_classifier can be obtained under normal driving conditions when there is no presence of an attack.
- The threshold value can be calculated using the maximum and minimum values of each graph feature obtained with normal CAN data. However, when extracting a message sequence to the window size during CAN communication, an error may occur in the maximum or minimum value depending on the sampling time. To reduce an error here, the error correction value[6] (ϵ_u, ϵ_l in Algorithm 1) is added to the maximum value and subtracted from the minimum value when calculating the threshold. The upper and lower threshold value can be obtained with the following equations.

$$Upper\_threshold_F = max(F) + \epsilon_u \quad (1)$$

$$Lower\_threshold_F = min(F) - \epsilon_l \quad (2)$$

  In the equations, F means a list of each characteristic values obtained from normal CAN data (F ∈ {list of *graph elapsed time*, list of *max degree*, list of *number of edges*}).
- The classification model of the ML_classifier: To train the ML-based classifier, we must have a dataset containing both normal data and attack data with attack type labels. The three features—*graph elapsed time*, *max degree*, and *number of edges*—are trained via a machine-learning algorithm. In G-IDCS, we use a random forest algorithm, which is a supervised machine learning-based algorithm.

Algorithm 1 shows the process of calculating the threshold values of the TH_classifier and training the ML_classifier. In the algorithm, *gtime* denotes *graph elapsed time*, *maxDegree* denotes *max degree*, and *edgeNum* denotes *number of edges*. The algorithm calculates the threshold of TH_classifier using the feature values obtained from normal CAN data (i.e., attack-free data), and trains ML_classifier using feature values and attack type labels. In other words, it receives three graph feature values extracted from CAN messages as input and outputs TH_classifier and ML_classifier.

[6]The error correction value can be obtained heuristically. In our experiments with a window size of 200 messages, 3% of the maximum value was suitable for obtaining the upper threshold, and 3% of the minimum value was suitable for obtaining the lower threshold.

Algorithm 1 The Graph-Based Classifier Configuration Algorithm

```
Input: $Features = [Feature_1, \ldots, Feature_n]$
Output: $ML\_classifier, TH\_classifier$
procedure G-IDCS_Configuration ($Features$)
    /* Add features to the training set */
    $Trainingset = Features \cup Trainingset$
    if $NormalDataOnly = True$ then
        /* Get the threshold of TH_classifier */
        $TH\_classifier \leftarrow getThreshold(Trainingset)$
    end
    else if $isThereAttackTypeLebel = True$ then
        /* Build random forest classifier */
        $ML\_classifier \leftarrow Rf\_Learning(Trainingset, Label)$
    end
    return $ML\_classifier, TH\_classifier$
end

procedure getThreshold ($Features$)
    /* $[U_1, L_1, \ldots, U_3, L_3]$: $[Upper\_gtime, Lower\_gtime, .., Lower\_edgeNum]$ */
    Initialize $[U_1, L_1, .., U_3, L_3]$
    $gTimes \leftarrow [gTime_1, .., gTime_n]$
    $maxDegrees \leftarrow [maxDegree_1, .., maxDegree_n]$
    $edgeNums \leftarrow [edgeNum_1, .., edgeNum_n]$
    $[f1, f2, f3] \leftarrow [gTimes, maxDegrees, edgeNums]$
    for n in range (0, 3) do
        $Max_n \leftarrow getMax(f_n)$
        /* Extract the upper threshold values of $f_n$ */
        $U_n \leftarrow Max_n + \epsilon_u$
        $Min_n \leftarrow getMin(f_n)$
        /* Extract the lower threshold values of $f_n$ */
        $L_n \leftarrow Min_n - \epsilon_l$
    end
    $Threshold \leftarrow [U_1, L_1, \ldots, U_3, L_3]$
    return $Threshold$
end
```

3) CLASSIFICATION MODULE

The classification module is composed of TH_classifier and ML_classifier. Whenever new CAN messages transmitted on the CAN bus reach the window size, one window is constructed and three graph features are extracted. The extracted features are used in the two classifiers as follows.

Algorithm 2 The Attack Detection and Type Classification Algorithm

```
/* Input: Graph features obtained
   from messages to be classified,
   which classifier to use                  */
Input: Features, Classifier type (ML, TH)
Output: Classification result
procedure Classifier ($Features$, $Classifier\_type$)
    if $Classifier\_type = ML$ then
        /* Attack type classification
           with ML_classifier               */
        $result \leftarrow ML\_classifier(Features)$
    end
    else if $Classifier\_type = TH$ then
        /* Attack classification with
           TH_classifier                    */
        $result \leftarrow TH\_classifier(Features)$
    end
    return $Classification\_result$
end
```

- TH_classifier: If any of the three calculated graph feature values are above its upper threshold or below its lower threshold, TH_classifier decides that attack messages are included in the window from which the features are extracted.
- ML_classifier: ML_classifier, which is trained with a random forest algorithm, performs attack type classification using the extracted graph features.

Algorithm 2 represents the attack detection and classification algorithm. This algorithm uses the graph characteristics and the classifier type to return the classification result as output.

VI. EXPERIMENTAL RESULTS AND EVALUATION

This section describes the dataset used to evaluate G-IDCS, presents performance metrics, and explains the experimental results.

A. DATASET

In this section, we describe two datasets: Car-Hacking Dataset for intrusion detection [23] and Car Hacking-Attack & Defense Challenge [41].

1) CAR-HACKING DATASET FOR INTRUSION DETECTION [23]

The dataset was constructed by logging CAN traffic through the OBD-II port in a real vehicle during a message injection attack, which injected fabricated messages to deceive original ECUs and cause the vehicle to malfunction. The attack types include a DoS attack, a fuzzy attack, a spoofing attack targeting the drive gear, and a spoofing targeting the RPM gauge. A separate dataset is configured for each attack type, and a description of each is as follows:

- DoS attack: Injecting messages with the '0×00' CAN ID every 0.3 milliseconds. Note that '0×00' is the most dominant. The DoS attack in this dataset is identical to the flooding attack in the Car Hacking-Attack & Defense Challenge dataset [41].
- Fuzzy attack: Injecting messages with random CAN ID and data field values every 0.5 milliseconds.
- Spoofing attack (RPM/gear): Injecting messages with certain CAN IDs related to either RPM or gear information every 1 millisecond.

The dataset has CAN messages containing Timestamp (showing logging time), Arbitration_ID (the CAN identifier), DLC (data length code), and Data (the CAN data field). There is also a flag for each message, and the flag indicates whether it is a normal message or an attack message.

2) CAR HACKING-ATTACK & DEFENSE CHALLENGE [41]

This attack dataset includes normal messages and injection attack messages. Attack types include flooding attacks (same as DoS in VI-A1), fuzzy attacks, spoofing attacks, and replay attacks. All attack types are represented in the dataset, and descriptions of these attack types are as follows:

- Flooding: A flooding attack aims to consume CAN bus bandwidth by sending a massive number of messages.
- Spoofing: CAN messages are injected to control a certain desired function.
- Replay: A replay attack attempts to extract normal traffic at a specific time and replay (inject) it into the CAN bus.
- Fuzzy: Random messages are injected to cause unexpected vehicular behavior.

The dataset has CAN messages containing Timestamp (showing logging time), Arbitration_ID (the CAN identifier), DLC (data length code), and Data (the CAN data field). There is also a class for each message, indicating whether it is a normal message or an attack message, and subclass (attack type) of each CAN message.

B. PERFORMANCE METRICS

We used four performance metrics to evaluate our system, namely Accuracy, Precision, Recall, and F1-score. The meaning of each metric is as follows:

- Accuracy: Accuracy represents the average of the ratio of windows that the system correctly classifies among the total number of windows.
- Precision: Precision represents the average of the ratio of windows that are classified correctly among the number of windows classified into each class.
- Recall: Recall represents the average of the ratio of windows included in each class that are classified correctly into that class by the classifier.
- F1-score: The F1-score represents the weighted average of Precision and Recall.

Each metric is defined by Equations (3), (4), (5), and (6).

$$Accuracy = \frac{\sum_{i \in C} \frac{TP_i + TN_i}{TP_i + FP_i + FN_i + TN_i}}{Number\ of\ classes} \tag{3}$$

TABLE 3. The performance of the threshold obtained by applying each outlier detection method.

Method	Accuracy	Precision	Recall	F1
Median absolute deviation ($2MAD_e$)	94.04%	91.69%	99.62%	95.49%
Median absolute deviation ($3MAD_e$)	75.68%	72.28%	99.87%	83.87%
Maximum and minimum threshold	**99.36%**	**99.87%**	**99.11%**	**99.49%**
Z-score(|z|=3)	99.19%	99.87%	98.85%	99.36%
Z-score(|z|=2)	97.42%	96.54%	99.49%	97.99%
Tukey's method (3IQR distance)	99.03%	99.74%	98.72%	99.23%
Tukey's method (1.5IQR distance)	82.42%	100%	70.65%	82.80%
Median rule	98.23%	99.87%	97.33%	98.58%

$$Precision = \frac{\sum_{i \in C} \frac{TP_i}{TP_i + FP_i}}{Number\ of\ classes} \tag{4}$$

$$Recall = \frac{\sum_{i \in C} \frac{TP_i}{TP_i + FN_i}}{Number\ of\ classes} \tag{5}$$

$$F1 - score = 2 \times \frac{Recall \times Precision}{Recall + Precision} \tag{6}$$

In the Equations, C means class and represents normal, attack in the case of TH_classifier. In the case of ML_classifier, C includes normal and all of attack types included in the dataset. TP_i is the number of correctly assigned positive samples, TN_i is the number of correctly assigned negative samples, FP_i is the number of incorrectly assigned positive samples, and FN_i is the number of incorrectly assigned negative samples [43], [44].

C. THRESHOLD SELECTION OF TH_CLASSIFIER

To select the threshold, we applied various outlier detection methods such as median absolute deviation ($2MAD_e$ or $3MAD_e$), Z-score ($|z| = 2$ or 3), Tukey's method (1.5IQR or 3IQR distance), and Median Rule to normal (i.e., attack-free) messages using Car Hacking-Attack & Defense Challenge dataset [41]. After that, if any of the three graph feature values are above its own upper threshold or below the lower threshold, the classifier decided that attack messages are included in the window from which the features are extracted. Table 3 shows the performance of the threshold calculated using various methods. As shown in Table 3, the three methods—maximum and minimum threshold, Z-score (|z|=3), and Tukey's method (3IQR distance)—all showed high-performance metrics. Based on the experimental results, we used the maximum and minimum threshold method to obtain the thresholds of TH_classifier, which showed over 99% performance for each of the four metrics.

D. ATTACK DETECTION WITH TH_CLASSIFIER

This section describes an attack classification experiment performed to evaluate the G-IDCS TH_classifier, which is a one-class classifier. Table 4 shows the classification performance of TH_classifier and other existing IDSs using real vehicle attack datasets.

Even though the window size of G-IDCS TH_classifier and the graph-based IDS [21] is the same as 200, one graph characteristic (e.g., *number of edges*) is extracted from each window and then a statistical test technique is used in the case of the graph-based IDS [21]. In general, a minimum number of 30 samples is recommended for statistical testing [22]. Thus, to use the graph-based IDS [21], at least 30 graphs (that is, more than 6,000 packets) are required for classification, which may lead to a delay in detection time. However, the G-IDCS TH_classifier extracts various graph characteristics (i.e., *graph elapsed time*, *max dgree*, and *number of edges*) from each graph, so it can classify attack using one graph (i.e., 200 packets). Comparing the performance metrics of attack classification with the graph-based IDS [21], we confirmed that the classification accuracy rate of fuzzy and spoofing attacks was slightly lower (within 1%), but DoS attack and replay attack classification accuracy yielded a higher TH_classifier accuracy rate than the graph-based IDS [21] by over 4%. In particular, comparing the performance of the combined attack classification, we confirmed that our TH_classifier is over 9% more accurate than the graph-based IDS [21]. In summary, when comparing G-IDCS TH_classifier to the graph-based IDS [21], the number of packets required for detection was significantly reduced, and the performance for most attack classification functions improved.

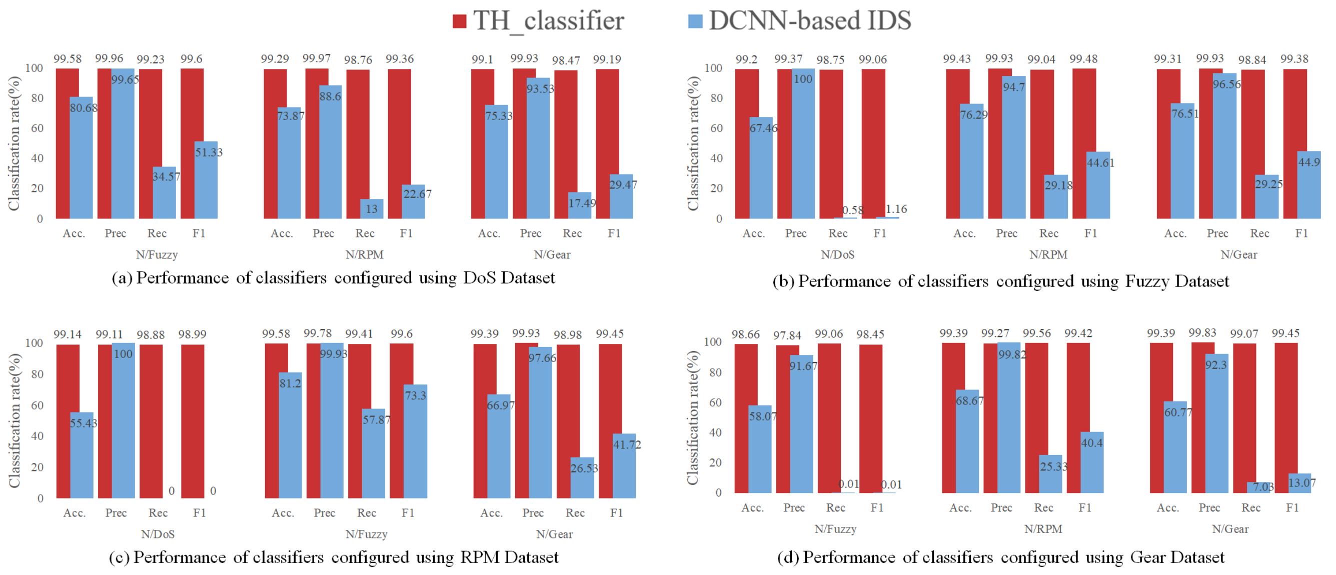
WINDS [35] is a method that uses wavelet transformation of frequency components to detect attacks. It has lower detection rates for attacks that do not target specific IDs on the bus, such as DoS or fuzzy attacks. This is because DoS and fuzzy attacks do not specifically target certain IDs and do not disrupt the period of normal messages, unlike direct attacks like Gear and RPM spoofing attacks [35]. SAIDuCANT [33] cannot detect aperiodic messages or message IDs with several message instances per period because it uses frequency as a feature. Additionally, this method has lower performance as it uses real-time specifications to detect attacks. Table 4 shows that G-IDCS TH_classifier outperforms WINDS [35] and SAIDuCANT [33] in terms of performance metrics, except for the N/RPM spoofing attack classification result of WINDS [35].

DCNN-based IDS [23], an attack detection technique using a deep convolutional neural network, performed slightly higher in all types of attack classification compared to G-IDCS TH_classifier. However, since a DCNN-based IDS [23] uses deep learning, it may be sensitive to environmental changes. To confirm this, we evaluated the performance of the pre-configured DCNN-based IDS [23] and TH_classifier when the attack type changes. We implemented a DCNN-based IDS [23], configured the classifier using a dataset that includes each attack type, and performed an experiment to classify other types of attacks that were not included in the training dataset. The same experiment was performed with TH_classifier.

Figure 10 (a), (b), (c), and (d) respectively show the performance of TH_classifier and a DCNN-based IDS when

TABLE 4. Attack detection performance comparison with existing methods using real vehicle attack datasets.

Model	Task	Accuracy	Precision	Recall	F1-score	Dataset
G-IDCS TH_classifier	N/DoS	99.44%	99.81%	98.86%	99.33%	Car-Hacking Dataset for intrusion detection [23]
	N/Gear Spoofing	99.52%	100%	99.08%	99.54%	
	N/RPM Spoofing	99.20%	99.85%	98.69%	99.27%	
	N/Fuzzy	99.28%	99.71%	99.00%	99.35%	
	N/Replay	99.68%	100%	96.51%	98.22%	OTIDS [28]
	N/Combined attack (3 types of attacks)	99.45%	100%	97.95%	98.97%	
Graph-based IDS [21]	N/DoS	94.74%	-	-	-	
	N/Fuzzy	100%	-	-	-	
	N/Spoofing	100%	-	-	-	
	N/Replay	95.24%	-	-	-	
	N/Combined attack (3 types of attacks)	90.16%	-	-	-	
WINDS [35]	N/DoS	94.97%	97.97%	94.15%	96.02%	Car-Hacking Dataset for intrusion detection [23]
	N/Gear Spoofing	98.83%	99.58%	98.45%	99.01%	
	N/RPM Spoofing	99.26%	99.86%	98.90%	99.38%	
	N/Fuzzy	87.78%	98.16%	83.39%	90.17%	
SAIDuCANT [33]	N/DoS	98.08%	97.71%	100%	98.84%	
	N/Gear Spoofing	82.62%	82.45%	97.02%	89.14%	
	N/RPM Spoofing	80.33%	80.10%	96.36%	87.48%	
	N/Fuzzy	87.82%	86.39%	99.58%	92.52%	
DCNN-based IDS [23]	N/DoS	99.97%	100%	99.89%	99.95%	
	N/Gear Spoofing	99.95%	99.99%	99.89%	99.94%	
	N/RPM Spoofing	99.97%	99.99%	99.94%	99.96%	
	N/Fuzzy	99.82%	99.95%	99.65%	99.80%	



(a) Performance of classifiers configured using DoS Dataset

(b) Performance of classifiers configured using Fuzzy Dataset

(c) Performance of classifiers configured using RPM Dataset

(d) Performance of classifiers configured using Gear Dataset

FIGURE 10. Robustness evaluation of TH_classifier and DCNN-based IDS.

the attack type changes. Both were configured for DoS, fuzzy, RPM spoofing, and gear spoofing attacks. As shown in Figure 10 (a), TH_classifier obtained using the DoS dataset and can classify attacks from other datasets, but a DCNN-based IDS cannot due to overfitting in the deep learning model. Figure 10 (b), (c), and (d) also show that TH_classifier returns high classification metrics even if the attack type changes, but the DCNN-based IDS does not. When the attack type changes, the accuracy of the DCNN-based IDS mostly falls under 80%, and in some cases, it was less than 60%. In addition, the Recall and F-1 scores were very low, with the lowest results coming out to less than 1%. However, TH_classifier performed very well in attack classification even when the attack type changed. Overall, due to over-fitting, a DCNN-based IDS is not effective in classifying attack types when the attack type changes, but TH_classifier is robust despite attack type change. In other words, compared to a DCNN-based IDS [23], we conclude that our G-IDCS TH_classifier can adapt well to environmental changes.

TABLE 5. Attack detection performance of TH_classifier and survival analysis-based IDS on different types of vehicles.

Model		G-IDCS TH_classifier				IDS based on survival analysis
Vehicle	Task	Accuracy	Precision	Recall	F1-score	Recall
HYUNDAI Sonata	N/Flooding	99.73%	99.72%	**99.72%**	92.72%	98.61%
	N/Fuzzy	100%	100%	**100%**	100%	97.35%
	N/Malfunction	99.85%	100%	99.64%	99.82%	**99.95%**
Kia Soul	N/Flooding	98.57%	96.92%	**99.74%**	98.31%	99.69%
	N/Fuzzy	99.92%	100%	**99.84%**	99.92%	98.87%
	N/Malfunction	98.15%	95.92%	99.03%	97.45%	**99.98%**
CHEVROLET Spark	N/Flooding	99.83%	99.65%	**100%**	99.83%	92.35%
	N/Fuzzy	99.70%	98.92%	**100%**	99.46%	87.08%
	N/Malfunction	99.75%	99.36%	**100%**	99.68%	99.83%

TABLE 6. Attack type classification performance comparison of ML_classifier and H-IDS.

Model	Task	Accuracy	Precision	Recall	F1-score	Dataset
G-IDCS ML_classifier (Our method)	multi-class	98.17%	98.27%	96.45%	98.18%	Car Hacking-Attack & Defense Challenge [41]
H-IDS [17]		97.25%	95.88%	90.97%	93.36%	

E. ATTACK DETECTION IN DIFFERENT TYPES OF VEHICLES WITH TH_CLASSIFIER

We evaluated the performance of our model on a dataset that included attack experiments on three different types of vehicles: HYUNDAI YF Sonata, KIA Soul, and CHEVROLET Spark. The dataset consisted of separate datasets for DoS, fuzzy, and malfunction attacks, including normal driving data. In the dataset [20], a DoS attack is the same as a flooding attack in Car-Hacking Dataset for intrusion detection [23]. For the malfunction attack, they used selected extractable CAN IDs and random data values. Specifically, they used CAN IDs 0×316, 0×153, and $0 \times 18E$ from the HYUNDAI YF Sonata, KIA Soul, and CHEVROLET Spark vehicles, respectively.

After obtaining the G-IDCS TH_classifier threshold from normal driving data, we experimented to detect the attack using TH_classifier. Table 5 summarizes the experimental results, indicating that our model can effectively detect and classify each attack targeting the three vehicles. Additionally, we compared our results to the recall values (the average recall value presented in the paper) of an IDS using a survival analysis of CAN IDs [20]. G-IDCS TH_classifier outperformed all other results, except for the malfunction detection results in the HYUNDAI YF Sonata and the KIA Soul.[7] Overall, experimental results demonstrate the effectiveness of our proposed method for detecting attacks on different types of vehicles, indicating its potential for use in real-world applications.

F. ATTACK TYPE CLASSIFICATION WITH ML_CLASSIFIER

This section describes the attack type classification result of our G-IDCS ML_classifier, which is a multi-class classifier trained with a random forest algorithm. To evaluate the classification performance of ML_classifier, we implemented the histogram-based IDS (H-IDS) [17], which is a window-based ML classifier using a KNN algorithm that can classify attack types. We performed attack type classification via ML_classifier and H-IDS[8] using the recently released Car Hacking-Attack & Defense Challenge dataset [41].

The results in Table 6 show that the multi-class classification performance of the ML_classifier outperformed H-IDS in all performance metrics. We believe this is because the H-IDS was designed without consideration for a replay attack that could extract normal traffic at a specific time and thus replay (inject) it into the CAN bus. In addition, attacks are classified in H-IDS by comparing the histogram of the data field. However, it is not difficult for an attacker to change the data field if they already have access to the CAN bus and can inject messages. If an attacker performs a message injection attack by composing their data field similarly to a normal packet, it would be difficult for H-IDS to classify the attack. Unlike the H-IDS, our proposed G-IDCS ML_classifier classifies attacks with a graph composed of message IDs, such that attacks that change the data field can still be detected in the same way.

G. TIME TO DETECT ATTACK AND CLASSIFY ATTACK TYPE

In this section, we evaluate the computational cost of G-IDCS by measuring detection time and attack classification time. In the experiments, we used 3.60GHz Intel Core i7-9700K CPU, 32.0GB RAM, running on Windows 10 to test TH_classifier and ML_classifier of G-IDCS. First, we performed an experiment measuring the time to extract features and classify normal/attack with TH_classifier. As a

[7] Among the performance metrics, only recall was presented in the paper.

[8] We set the window size of H-IDS to 110, which has the best performance according to the experimental results of the paper.

result of the experiment, it took about 0.7478 seconds for graph construction, feature extraction, and classification in messages of 5,242 windows (5,242 * 200 messages). That is, it took about 142.6 microseconds on average to configure one graph and detect an attack using TH_classifier. Second, we performed an experiment measuring the time to extract features and classify attack types with ML_classifier. In the case of the second experiment, it took about 0.7292 seconds for graph construction, feature extraction, and attack type classification in messages of 5,242 windows (5,242 * 200 messages). In other words, it took about 139.1 microseconds on average to construct one graph and classify the attack type using ML_classifier. The graph-based IDS [21]—utilizing graph theory with the same window size of G-IDCS—took about 258.9 microseconds to detect an attack. In summary, the detection time of TH_classifier and attack type classification time of ML_classifier of G-IDCS is faster than that of the graph-based IDS [21].

VII. CONCLUSION

In this paper, we presented a graph-based intrusion detection and classification system called G-IDCS to enhance the security of the controller area network (CAN) protocol. The proposed system consists of a threshold-based intrusion detection system (TH_classifier) and a machine learning-based attack classification system (ML_classifier), which can accurately detect and classify various types of attacks on the CAN network.

In our experiments, we found that G-IDCS TH_classifier significantly reduces the number of packets required for detection and improves the accuracy of attack detection by more than 9% compared to existing intrusion detection methods that use graph theory. Moreover, our experimental results show that TH_classifier is more robust to changes in attack types than DCNN-based IDS and can provide an explanation of the features used in classification, which existing machine learning and deep learning-based systems cannot do. Additionally, G-IDCS ML_classifier outperforms the existing attack classification techniques, making it a useful tool for designing countermeasures based on attack types in digital forensic investigations. In future research, we plan to extend our proposed system to the V2X environment by designing a methodology for identifying anomalous vehicles using graph theory.

REFERENCES

[1] K. Pazul, "Controller area network (CAN) basics," Microchip Technol. Inc, Preliminary DS00713A, AN713, 1999, p. 1.
[2] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "VoltageIDS: Low-level communication characteristics for automotive intrusion detection system," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 2114–2129, Aug. 2018.
[3] S. Checkoway et al., "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. USENIX Secur. Symp.*, San Francisco, CA, USA, 2011, pp. 77–92.
[4] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *Black Hat USA*, vol. 2014, p. 94, Aug. 2014.
[5] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, Aug. 2015.
[6] E. Aliwa, O. Rana, C. Perera, and P. Burnap, "Cyberattacks and countermeasures for in-vehicle networks," *ACM Comput. Surv.*, vol. 54, no. 1, pp. 1–37, Jan. 2022.
[7] S. Woo, H. J. Jo, and D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle CAN," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 993–1006, Sep. 2015.
[8] Y. Lee, S. Woo, J. Lee, Y. Song, H. Moon, and D. H. Lee, "Enhanced Android app-repackaging attack on in-vehicle network," *Wireless Commun. Mobile Comput.*, vol. 2019, pp. 1–13, Feb. 2019.
[9] A. K. Mandal, F. Panarotto, A. Cortesi, P. Ferrara, and F. Spoto, "Static analysis of Android auto infotainment and on-board diagnostics II apps," *Softw., Pract. Exper.*, vol. 49, no. 7, pp. 1131–1161, May 2019.
[10] H. Wen, Q. A. Chen, and Z. Lin, "Plug-N-Pwned: Comprehensive vulnerability analysis of OBD-II dongles as a new over-the-air attack surface in automotive IoT," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 949–965.
[11] H. J. Jo, W. Choi, S. Y. Na, S. Woo, and D. H. Lee, "Vulnerabilities of Android OS-based telematics system," *Wireless Pers. Commun.*, vol. 92, no. 4, pp. 1511–1530, 2017.
[12] S. Mazloom, M. Rezaeirad, A. Hunter, and D. McCoy, "A security analysis of an in-vehicle infotainment and app platform," in *Proc. 10th USENIX Workshop Offensive Technol.*, 2016, pp. 1–12.
[13] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 447–462.
[14] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 911–927.
[15] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2016, pp. 63–68.
[16] X. Ying, S. U. Sagong, A. Clark, L. Bushnell, and R. Poovendran, "Shape of the cloak: Formal analysis of clock skew-based intrusion detection system in controller area networks," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 9, pp. 2300–2314, Sep. 2019.
[17] A. Derhab, M. Belaoued, I. Mohiuddin, F. Kurniawan, and M. K. Khan, "Histogram-based intrusion detection and filtering framework for secure and safe in-vehicle networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 3, pp. 2366–2379, Mar. 2022.
[18] M. Muter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2011, pp. 1110–1115.
[19] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, and K. Li, "A survey of intrusion detection for in-vehicle networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 3, pp. 919–933, Mar. 2020.
[20] M. L. Han, B. I. Kwak, and H. K. Kim, "Anomaly intrusion detection method for vehicular networks based on survival analysis," *Veh. Commun.*, vol. 14, pp. 52–63, Oct. 2018.
[21] R. Islam, R. U. D. Refat, S. M. Yerram, and H. Malik, "Graph-based intrusion detection system for controller area networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 3, pp. 1727–1736, Mar. 2022.
[22] U. Sekaran and R. Bougie, *Research Methods for Business, a Skill Building Approach*. Hoboken, NJ, USA: Wiley, 2003.
[23] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Veh. Commun.*, vol. 21, Jan. 2020, Art. no. 100198.
[24] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018.
[25] G. Karopoulos, G. Kambourakis, E. Chatzoglou, J. L. Hernández-Ramos, and V. Kouliaridis, "Demystifying in-vehicle intrusion detection systems: A survey of surveys and a meta-taxonomy," *Electronics*, vol. 11, no. 7, p. 1072, Mar. 2022.
[26] G. Dupont, J. Den Hartog, S. Etalle, and A. Lekidis, "A survey of network intrusion detection systems for controller area network," in *Proc. IEEE Int. Conf. Veh. Electron. Saf. (ICVES)*, Sep. 2019, pp. 1–6.
[27] M.-J. Kang and J.-W. Kang, "A novel intrusion detection method using deep neural network for in-vehicle network security," in *Proc. IEEE 83rd Veh. Technol. Conf. (VTC Spring)*, May 2016, pp. 1–5.
[28] H. Lee, S. H. Jeong, and H. K. Kim, "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame," in *Proc. 15th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2017, pp. 57–5709.

[29] B. Groza and P.-S. Murvay, "Efficient intrusion detection with Bloom filtering in controller area networks," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 4, pp. 1037–1051, Apr. 2019.
[30] Y. Wei, C. Cheng, and G. Xie, "OFIDS: Online learning-enabled and fingerprint-based intrusion detection system in controller area networks," *IEEE Trans. Dependable Secure Comput.*, early access, Dec. 19, 2022, doi: 10.1109/TDSC.2022.3230501.
[31] H. M. Song and H. K. Kim, "Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1098–1108, Feb. 2021.
[32] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *Proc. IEEE 2nd Int. Forum Res. Technol. Soc. Ind. Leveraging Better Tomorrow (RTSI)*, Sep. 2016, pp. 1–6.
[33] H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, "SAIDuCANT: Specification-based automotive intrusion detection using controller area network (CAN) timing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1484–1494, Feb. 2020.
[34] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *Proc. World Congr. Ind. Control Syst. Secur. (WCICSS)*, Dec. 2015, pp. 45–49.
[35] M. Bozdal, M. Samie, and I. K. Jennions, "WINDS: A wavelet-based intrusion detection system for controller area network (CAN)," *IEEE Access*, vol. 9, pp. 58621–58633, 2021.
[36] H. J. Jo and W. Choi, "A survey of attacks on controller area networks and corresponding countermeasures," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 6123–6141, Jul. 2022.
[37] R. Bosch, "Can specification version 2.0," *Rober Bousch GmbH, Postfach*, vol. 300240, p. 72, Sep. 1991.
[38] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based intrusion detection system for in-vehicle network," in *Proc. 16th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2018, pp. 1–6.
[39] B. I. Kwak, M. L. Han, and H. K. Kim, "Cosine similarity based anomaly detection methodology for the CAN bus," *Exp. Syst. Appl.*, vol. 166, Mar. 2021, Art. no. 114066.
[40] H. Kim, "Car hacking: Attack & defense challenge 2020 dataset," *IEEE Dataport*, Feb. 2021.
[41] H. Kang, B. I. Kwak, Y. H. Lee, H. Lee, H. Lee, and H. K. Kim, "Car hacking and defense competition on in-vehicle network," in *Proc. 3rd Int. Workshop Automot. Auto. Vehicle Secur.*, 2021, p. 25.
[42] J. L. Gross and J. Yellen, *Graph Theory and Its Applications*. Boca Raton, FL, USA: CRC Press, 2005.
[43] A. Amira, A. Derhab, E. B. Karbab, O. Nouali, and F. A. Khan, "TriDroid: A triage and classification framework for fast detection of mobile threats in Android markets," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 2, pp. 1731–1755, 2021.
[44] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009.

SUNG BUM PARK received the B.S. degree in industrial engineering from Korea University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree. His research interests include applied cryptography, smart car security, and the IoT/CPS security.

HYO JIN JO received the B.S. degree in industrial engineering and the Ph.D. degree in information security from Korea University, Seoul, South Korea, in 2009 and 2016, respectively. From 2016 to 2018, he was a Postdoctoral Researcher with the Department of Computer and Information Systems, University of Pennsylvania, Philadelphia, USA. He is currently an Assistant Professor with the School of Software, Soongsil University, Seoul. His research interests include applied cryptography, security and privacy for ad-hoc networks, and IoT/CPS security.

DONG HOON LEE (Member, IEEE) received the B.S. degree from Korea University, Seoul, South Korea, in 1985, and the M.S. and Ph.D. degrees in computer science from The University of Oklahoma, Norman, OK, USA, in 1988 and 1992, respectively. Since 1993, he has been with the Faculty of Computer Science and Information Security, Korea University, where he is currently a Professor and the Director of the Graduate School of Information Security. His research interests include cryptographic protocol, applied cryptography, functional encryption, software protection, mobile security, vehicle security, and ubiquitous sensor network security.