IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 18, 2022, accepted May 2, 2022, date of publication May 11, 2022, date of current version May 19, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3174356

TTIDS: Transmission-Resuming Time-Based Intrusion Detection System for Controller Area Network (CAN)

SEYOUNG LEE[1], HYO JIN JO[2], ARAM CHO[3], DONG HOON LEE[1], (Member, IEEE), AND WONSUK CHOI[4]

[1]Graduate School of Information Security, Korea University, Seoul 02841, Republic of Korea
[2]School of Software, Soongsil University, Seoul 06978, Republic of Korea
[3]Hyundai Motors, Hwaseong-si 18280, Republic of Korea
[4]Division of IT Convergence Engineering, Hansung University, Seoul 02876, Republic of Korea

Corresponding author: Wonsuk Choi (wonsuk@hansung.ac.kr)

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant through the Korean Government [Ministry of Science and ICT (MSIT)], AI-Based Cyber Attacks and Defense for Autonomous Vehicles, under Grant 2021-0-00111.

ABSTRACT Modern vehicles are becoming complex cyber-physical systems equipped with numerous electronic control units (ECUs). Over the controller area network (CAN), these ECUs communicate with each other to share information related to vehicle status as well as commands to efficiently control the vehicle. However, the increasing complexity of modern vehicles has inadvertently expanded potential attack surfaces, making them vulnerable to cyber attacks. In light of this, researchers are currently working to demonstrate remote vehicle maneuvering by compromising ECUs, and as a countermeasure to such malicious manipulation, to study automotive intrusion detection systems (IDSs) as potential remedies. In general, CAN messages are transmitted periodically, and as such, many researchers have relied on frequency-based IDSs in their solutions proposals. However, an attacker can bypass this defense by suspending the communication of the target ECU from the network and injecting malicious messages with the same frequency as the suspended messages. As a result, an attacker is able to masquerade as the original transmission frequency. In this paper, we propose a Transmission-resuming Time-based IDS (TTIDS), which is designed to detect such attacks. TTIDS detects when an ECU periodically transmitting messages is suspended, and then it estimates when the suspended ECU resumes periodic transmission. With this projection, TTIDS detects malicious messages transmitted while the ECU is suspended. We conduct the evaluation of TTIDS on two real vehicles and present the results, which show the TTIDS is able to effectively detect an enhanced attack that bypasses existing frequency-based IDSs with a false positive rate of 0.213% and a false negative rate of 0.027%.

INDEX TERMS Automotive security, controller area network (CAN), electronic control unit (ECU), intrusion detection system (IDS).

I. INTRODUCTION

For driver safety and convenience, vehicles are now digitized via Electronic Control Units (ECUs), which are embedded computers for vehicular functions. The ECUs periodically measure the status of a vehicle along with controlling some functions based on their measurements. For example, an engine ECU has a sensor to recognize when the accelerator is being pressed. When pressure on the pedal is recognized, the ECU may actuate the engine to accelerate the vehicle. This automated control process enables vehicles to operate more efficiently and safely. It is known that a luxury vehicle has approximately 70 ECUs [1]. A set of ECUs in a vehicle constructs a network over which the ECUS communicate with each other. The controller area network (CAN), also referred to as the CAN bus, has been widely employed for data communication in this in-vehicle network. The CAN protocol, developed by Bosch in the 1980s, is a bus

The associate editor coordinating the review of this manuscript and approving it for publication was Diana Gratiela Berbecaru.

communication protocol and a standard for reliable and efficient communication in the in-vehicle network [2]. However, due to a lack of CAN bus security features, a remote attacker only needs to compromise an attack surface to be able to inject CAN messages remotely. Also, the fatal weakness of the CAN protocol is that it cannot verify if incoming CAN messages originated from valid ECUs or other sources. Ever since the car hacking demonstration by Koscher *et al.* in 2010, there has been increasing research regarding cyber attacks on a vehicle [3]. Car hacking is no longer science fiction, and myriad information about vehicle hacking is readily available on open Internet platforms like YouTube [4]. Because the CAN was designed solely for communication efficiency and stability–and not for security features–it is difficult to introduce cryptographic protocols into ECUs while maintaining standard CAN specifications [5].

The automotive intrusion detection systems (IDSs) analyze in-vehicle network traffic, and security researchers identify these systems as promising attack detection tools as they do not require any modification of existing hardware or CAN protocol. Indeed, research and development of automotive IDS products now extend beyond the academic domain. According to the University of Michigan Transportation Research Institute (UMTRI), several companies worldwide are developing automotive IDS products [6]. In response to this finding, UMTRI helmed a project on methods for evaluating the performance of these emerging products, implying that vehicle OEMs are planning to install automotive IDSs into vehicles in the very near future. In addition, United Nations Economic Commission for Europe (UNECE) regulations for the Cybersecurity Management System and Software Update Management System have been adopted by UNECE's World Forum for the Harmonization of Vehicle Regulations [7], [8].

In one variety of typical attacks on a vehicle, an attacker continuously injects a large volume of maliciously crafted messages into the CAN bus to force the receiving ECU to process the malicious messages. To detect these attacks, automotive IDSs mainly zero in on two characteristics. Since most CAN messages are periodically transmitted on the CAN bus, the periodicity of message transmissions is commonly used as a pattern for analysis purposes to detect an automotive intrusion [9]–[15]. As this periodic pattern can be measured simply and analyzed without special equipment, most commercialized versions even leverage these patterns to detect vehicular intrusions. The frequency-based IDS uses this periodicity pattern. Another commonly utilized pattern is the content of data payload (i.e., data field). Since part of the contents implies a vehicle status, there is no rapid change in normal conditions. Accordingly, machine learning-based (ML-based) IDSs learn these patterns and detect intrusions by verifying the consistency or plausibility of the data payload [16]–[19]. The consensus is that the most difficult attack to detect is a masquerade attack that injects malicious messages by mimicking the original transmission periodicity and data payload [20]–[22]. In a masquerade attack, an attacker first suspends message transmission from a target ECU and then uses a compromised ECU to inject the malicious messages with the same frequency and consistent data payload as the suspended messages. There is no difference between normal and malicious messages when it comes to transmission periodicity or content of their data payload. As a result, no frequency-based or ML-based IDSs are able to properly detect the masquerade attack. Indeed, most works for frequency-based or ML-based IDSs have not yet been able to provide evaluation results proving detection of a masquerade attack. Unfortunately, these IDSs are not designed to detect this advanced attack because it is assumed that these attacks perfectly masquerade periodicity and slowly change the content of a data payload. Even though the clock-based IDS [20] was subsequently designed to detect a masquerade attack, one recent study proves that clocks can be emulated, meaning that the clock-based IDS can be also bypassed [23], [24].

In contrast to the existing models, we present a novel method to detect a masquerade attack based on the fact that an ECU is supposed to be suspended before message injection starts in a masquerade attack. We focus on how an attacker stops message transmission from a particular ECU and when the suspended ECU will resume working. To date, two types of methods have been introduced for ECU suspension: one type is to employ diagnostic services, and the other one is to employ a bus-off attack. We designed an automotive IDS based on the Transmission-resuming Time (dubbed TTIDS) to detect a masquerade attack, which is an advanced attack. Our novel proposal is designed to simply recognize when an ECU is suspended by diagnostic services or a bus-off attack. Then, TTIDS estimates when a suspended ECU will resume its function. If CAN messages suspected to be from a suspended ECU are observed, our method considers these messages as intrusive.

The key contributions of our paper are as follows:

1) We propose a novel method that is able to detect a masquerade attack without any additional hardware.
2) According to the ways to suspend an ECU, we conduct two types of masquerade attacks on real vehicles. Through this evaluation, we show that our TTIDS method can be applied to current systems in real vehicles without any hardware modification.
3) We systematically classify ECU behaviors when ECUs are suspended and resume functioning. Based on our analysis, TTIDS is able to reliably estimate when transmission from a suspended ECU will resume.

II. RELATED WORKS

This section introduces various existing methods on how to suspend message transmission of an ECU as the suspension of ECU is the initial step in a masquerade attack. In addition, we summarize related works on automotive IDSs.

A. ECU SUSPENSION

There are two ways to suspend message transmission of a particular ECU.

The first method is to use diagnostic services, which Koscher *et al.* [3] described to illustrate how ECU communication can be deactivated this way. Miller and Valasek [25] and Nie *et al.* [26] both proposed a method to suspend an ECU transmitting CAN messages when the ECU is held in a special diagnostic mode. These works put the Electronic Parking Brake Module (EPB/EPBM) ECU into a special diagnostic session mode and make it stop transmitting messages. The open-software tool for diagnostic service scanning, such as the Caring Caribou tool [27], enables diagnostic services to be scanned and its parameter applied to the vehicles on the application layer.

The second method is to leverage a bus-off attack, in which an attacker takes advantage of the confinement mechanism rules defined by the CAN specification [28]–[31]. In bus-off mode, an ECU can neither transmit nor receive CAN messages on the CAN bus. Two ways to put an ECU into bus-off mode have been introduced. The first way requires a non-standard CAN controller [28], [29]. The attacker identifies the CAN ID of the target data frame and intentionally transmits a dominant bit at a recessive bit position to generate a bit error. Due to this error, the target ECU transmits an error flag and retransmits the failed data frame again. With each error, the target's transmission error counter (TEC) increases by 8, and the attacker can eventually force the target ECU into bus-off mode. However, as described by Longari *et al.* [32], this attack is hardly feasible without previous physical access to the in-vehicle network since physical modification of the CAN controller of the attack device is necessary to successfully implement this strategy. Also, the second way does not require the CAN controller to be modified [30], [31]. The attacker intentionally can increase the TEC of the target ECU by transmitting a message with the same CAN ID as the target message but with different data and at the exact same time, thus generating a bit error. By repeating this process, the target's TEC quickly exceeds the threshold of 255, and it enters bus-off mode. However, the preceded CAN messages are required to match transmission timing. Iehira *et al.* [33] proposed a spoofing attack method that uses a bus-off attack. The attacker transitions the target ECU into bus-off mode by using a bus-off attack and transmits malicious messages at the same frequency as the regular messages.

B. AUTOMOTIVE INTRUSION DETECTION SYSTEMS

To prevent cyber attacks on vehicles, researchers sought to improve the security of CAN communication by applying cryptographic protocols [34]–[37]. However, cryptographic methods required modification of current ECU software and negatively impacted CAN communication by causing heavy busload or low response times. Following this, automotive intrusion detection systems (IDSs) entered as a promising method to handle cyber attacks on the CAN protocol as these new frameworks did not require vehicle component modification.

The self-adapting nature of automotive IDSs allows for easy application in the CAN of actual vehicles, and hence, researchers have extensively studied methods to incorporate automotive IDS frameworks [9]–[19], [38], [39]. To date, the simplest approach is to make an automotive IDS check legitimate CAN identifiers (IDs) [38], [39]. Because the number of legitimate CAN IDs is relatively small (usually less than 100), malicious CAN messages with invalid CAN IDs are easily detected. However, this simple approach can be easily evaded by injecting messages with valid CAN IDs and forged data payloads.

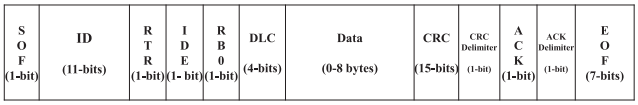


FIGURE 1. Fields defined in the CAN data frame.

Since most CAN messages are periodically transmitted on the CAN bus, the periodicity of CAN messages is also one of the characteristics used in automotive IDS technologies [9]–[15]. The injection of malicious CAN messages clearly increases transmission frequency or changes the statistical characteristics related to periodicity. The frequency-based IDS is the type that uses this periodicity pattern. Another commonly utilized pattern is the content of data payload (i.e., data field). Since a part of the contents implies a vehicle status, there would be no rapid change in normal condition. Accordingly, ML-based IDSs learn these patterns and detect intrusion by verifying the consistency or plausibility of data payload [16]–[19]. Still, any frequency-based or ML-based IDSs are unable to properly detect a masquerade attack by an advanced adversary who may be able to successfully mimic the original transmission periodicity and a data payload. In the masquerade attack, there is no difference between normal and malicious messages when it comes to their transmission periodicity and content of their data payload.

In contrast to the existing works, our approach focuses on the fact that an ECU is supposed to be suspended before message injection starts in a masquerade attack. Our method is designed to recognize a suspended ECU. If CAN messages that are supposed to be from the suspended ECU are observed, our method considers these messages as an intrusion regardless of the transmission periodicity or a data payload.

III. BACKGROUND ON CAN PROTOCOL

Next, we present background on the CAN protocol to facilitate a closer understanding of how our method operates.

A. CAN FRAME PRIORITIZATION

According to the CAN standard, four different frame types are defined: data, remote, error, and overload. In an in-vehicle network, ECUs normally communicate with each other with the data frame. As shown in FIGURE 1, a CAN data frame contains fields for the identifier (ID), the data length code (DLC), the data, and the cyclic redundancy check (CRC).

TABLE 1. Typical examples of UDS services.

Function group	Services	SID (hex)	Description
Diagnostic and communications management	*DiagnosticSessionControl*	0x10	Enable different operating sessions in the ECU
	SecurityAccess	0x27	Unlock security-critical services
	CommunicationControl	0x28	Transmitting or receiving of messages can be turned off
Data transmission	*ReadMemoryByAddress*	0x23	Read data from the physical memory at the provided address
	WriteMemoryByAddress	0x3D	Write data to the physical memory at the provided address
Stored data transmission	*ClearDiagnosticInformation*	0x14	Delete all stored Diagnostic Trouble Codes (DTC)
	ReadDTCInformation	0x19	Read the DTC from a vehicle or from a paricular ECU

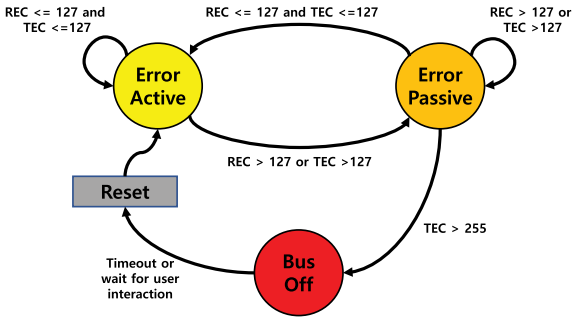


FIGURE 2. Error states of the CAN controller.

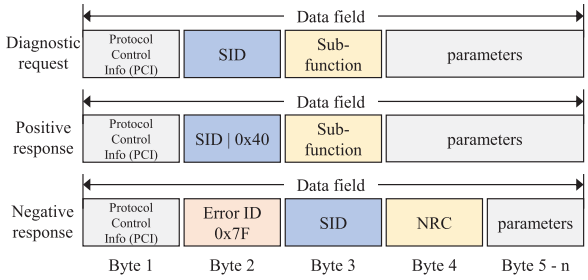


FIGURE 3. Format of request and positive/negative response messages for UDS.

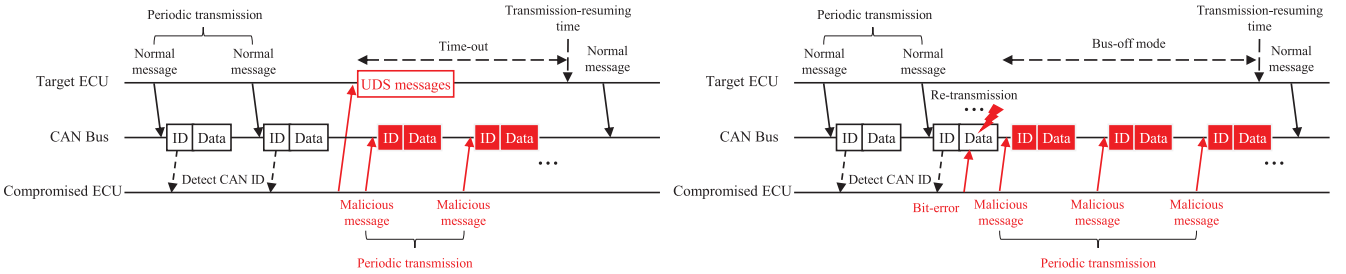
The identifier field (11 bits in the standard frame format or 29 bits in the extended frame format) refers to the CAN message identifier that determines the frame's priority and is related to a particular function (i.e., engine temperature or throttle valve angle). Note that since IDs are uniquely assigned to CAN messages, we use the terms "CAN message" and "CAN ID" interchangeably. Once the CAN bus is idle, multiple ECUs are able simultaneously attempt transmission of their CAN messages. To prioritize such collision messages, the CAN protocol supports a bit-wise arbitration decision process. The CAN specification defines dominant and recessive bits, denoted by (0) and (1), respectively. If one ECU transmits a dominant bit (0) and another ECU transmits a recessive bit (1), the dominant bit will dominate the bus. Therefore, during the arbitration decision process, the lowest ID indicates the highest priority and wins the arbitration. The ECU that loses arbitration stops transmitting its messages and re-transmits when the CAN bus is idle once more.

B. ERROR HANDLING

For fault-tolerant communication in the CAN protocol, the error-handling mechanism is implemented so that ECUs can automatically detect and resolve transmission errors and take appropriate action, such as discarding a frame, re-transmitting a frame, or broadcasting an error flag. There are five types of errors in CAN: bit, stuff, form, ACK, and CRC. Each node maintains both the transmit error counter (TEC) and the receive error counter (REC). Depending on the role of the ECU, these counters will increase when an error is detected, or alternatively, they decrease after a successful transmission or reception. When a transmitter encounters an error, it transmits an error frame and increases TEC by 8. Similarly, when a receiver encounters an error, it should transmit an error frame and increase REC by 1. When there is error-free transmission and reception, an ECU decreases both TEC and REC by 1.

To provide fault confinement, the CAN specification defines three error states as illustrated in FIGURE 2. All ECUs start in error-active mode and determine their error state according to the value of both counters. An error-active ECU can normally take part in bus communication and send an active error flag when an error is detected. When TEC or REC exceeds 127 as a result of consecutive errors, the ECU changes to error-passive mode, which prohibits the transmission of active error flags. Instead, a passive error flag is sent upon detection of an error. Finally, when TEC exceeds 255, the ECU enters bus-off mode, which wholly limits its influence on the bus. In this state, the ECU stops transmitting or receiving messages on the CAN bus. The ECU is permitted to automatically revert back to the error-active mode after 128 occurrences of 11 consecutive recessive bits (1) on the bus are monitored [2].

In addition to the CAN controller, the microcontroller also stops requesting transmissions for a period of time when it recognizes bus-off mode. Normally, the microcontroller is interrupted by its CAN controller when it enters bus-off mode, but the implementation of the microcontroller for bus-off recovery is not standardized [30]. Unlike the CAN controller, when bus-off mode is detected by the microcontroller, how long it is suspended for is determined by manufacturers. It is also possible for the microcontroller to remain suspended until it is manually re-initialized. However, microcontrollers for the ECUs that we checked for this paper initialize automatically from bus-off mode. Note that other studies have also stated that the vehicles they analyzed had ECUs with an automatic initialization mechanism from bus-off mode [40]–[42].



(a) Masquerade attack using UDS services

(b) Masquerade attack using bus-off attack

FIGURE 4. Masquerade attack types.

C. UNIFIED DIAGNOSTIC SERVICES (UDS)

The Unified Diagnostic Services (UDS), codified in ISO-14229, is a diagnostic communication protocol used over the CAN protocol for vehicle diagnostics [43]. UDS has become an essential tool for manufacturers and technicians to deliver service and update to vehicles. TABLE 1 shows typical examples of the services provided by UDS along with descriptions. A message for UDS services is constructed in the data field of the CAN data frame as shown in FIGURE 3. In the data field, the Service ID (SID), sub-function, and parameters associated with diagnostic services are contained. UDS messages are transmitted via a CAN bus with a specific request CAN ID, and ECUs choose whether or not to filter these message based on their request CAN IDs. It is known that the common range of diagnostic IDs is from 0×700 to $0 \times 7FF$ [44]. Since a UDS message is always directed at a particular ECU, the ECU should respond to the request with a diagnostic response ID. If the requested SID, sub-function, and parameters are correct on the ECU, a positive response should be sent; otherwise, a negative response should be sent. A positive response is characterized by the addition of value 0×40 to the SID. A negative response is characterized by an error ID of $0 \times 7F$ and negative response code (NRC) [45]. Therefore, the request and response messages can be identified by the diagnostic request ID and the diagnostic response ID, respectively.

An attacker widely exploits to launch various attacks on vehicles via UDS services [46]. According to experiments by Miller and Valasek [47], an attacker would be able to transmit some diagnostic messages to kill engine or control fuel gauge of the Ford and Toyota in their models.

IV. ATTACK MODEL

A. ATTACKER CAPABILITIES

According to car hacking demonstrations [25], [48], an attacker may access the in-vehicle network either physically or remotely. The on-board diagnostics (OBD) port is a typical interface that enables physical access, and a telematics device can be also used as an interface to enable remote access to the in-vehicle network. In our attack model, we simply assume that the attacker is able to access the in-vehicle network regardless of physical or remote access. Moreover, we presume that the attacker may have knowledge about which CAN messages are assigned to critical functions of a vehicle, which he could glean from CAN reversing [49]–[52] or a publicly available database (e.g., OpenDBC) [53]. Once an attacker accesses the in-vehicle network, he is able to intentionally control the vehicle by injecting malicious CAN messages. As a result, an attacker is then able to conduct a masquerade attack so that the frequency-based automotive IDS is easily bypassed.

B. ATTACK TYPES

As mentioned earlier, our method is designed to detect a masquerade attack, which is the most advanced attack that can bypass existing automotive IDSs. We divide masquerade attacks by ECU suspension methods because ECU suspension attack is the initial step of any masquerade attack. A masquerade attack normally bypasses frequency-based IDSs since it does not necessarily alter the original transmission frequency of a CAN ID [20]–[22]. The attacker only needs to suspend one target ECU so that it cannot transmit CAN messages, and he then transmits the malicious messages with the same CAN ID of the suspended ECU at the original frequency. Next, we describe two ways the attacker suspends message transmission: i) suspension using the UDS services and ii) suspension using bus-off attack.

1) SUSPENSION USING UDS SERVICES

In this attack strategy, the attacker uses diagnostic services to suspend transmission of normal messages from the target ECU. After the target ECU is suspended, the compromised ECU transmits malicious messages at the same frequency as the original messages normally transmitted by the target ECU. FIGURE 4 (a) illustrates how a masquerade attack using UDS is performed on the CAN bus. An attacker can use three diagnostic services (SIDs)—*DiagnosticSessionControl* (0×10), *CommunicationControl* (0×28), and *TesterPresent* ($0 \times 3E$)—to suspend ECU message transmission [25], [26]. When the vehicle is powered on, the ECU enters the default session where it transmits messages and listens for UDS requests. At this point, the *DiagnosticSessionControl* (0×10) service is used to switch to an extended diagnostic session in the ECUs that start in the

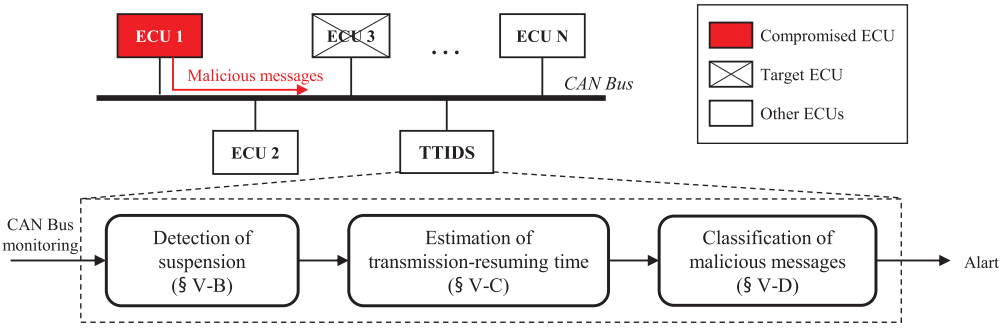


FIGURE 5. System overview.

default session. Note that ECUs with a non-default session return to the default session after a time-out. If an additional diagnostic request message is received before the time-out, the duration of a non-default session is extended. Accordingly, an attacker is able to disable message transmission of an ECU by periodically transmitting the *CommunicationControl* (0×28) service so that the ECU with a non-default session does not return to the default session. In the *CommunicationControl* (0×28) diagnostic service, sub-function "Disable Tx and Rx for ECU" is defined. In addition, an attacker can maintain the suspension by continuously requesting the *CommunicationControl* (0×28) diagnostic service or the *TesterPresent* $(0 \times 3E)$ service.

2) SUSPENSION USING BUS-OFF ATTACK

In a bus-off attack, an attacker takes advantage of the error-handling mechanism rules in the CAN specification [28]–[31]. Each time an error is detected while an ECU transmits a CAN message, the TEC increases by 8 until it exceeds 255 and forces the ECU to enter bus-off mode. As mentioned in Section II-A, there are two ways to intentionally generate an error on the CAN bus so that a target ECU is forced to enter bus-off mode. An attacker uses a non-standard CAN controller [28], [29] or a standardized CAN controller [30], [31] to generate errors when a target ECU transmits a normal message. Due to an intentional error, the target ECU transmits an error flag and then attempts to retransmit the failed transmission. As a result, the attacker can force the target ECU to increase its TEC by generating an error every time it transmits a normal message. FIGURE 4 (b) illustrates how the masquerade attack using bus-off attack is performed.

C. ASSUMPTIONS

We assume that all ECUs installed in a vehicle either revert back to an error-active state or resume transmission after the ECU enters bus-off mode or the diagnostic communication session terminates, respectively. Even though ECUs can be manually initialized from bus-off mode, all ECUs we checked performed automatic initialization processes. Moreover, we found that the ECUs investigated by research presented in [40]–[42] also contained automatically initializing ECUs from bus-off mode. In addition to this fact, AUTOSAR specification [54] states that ECUs performing a non-default diagnostic session revert to the default session if they have not received diagnostic request messages for 5 seconds (i.e., a time-out).

V. TTIDS

In this section, we present TTIDS that detects a masquerade attack. TTIDS is built on the fact that an ECU must be suspended for a masquerade attack to proceed. We first offer an overview of TTIDS and then divide important details into three steps.

A. OVERVIEW

In an in-vehicle network, TTIDS can be implemented in an existing ECU, such as a gateway ECU, so that it monitors CAN traffic generated from other ECUs. Regardless of physical or remote access, we assume that an attacker intends to compromise an ECU. For a masquerade attack, the attacker must first suspend a target ECU before malicious CAN message injection. As mentioned in Section IV-B, the attacker is able achieve this suspension via UDS service or bus-off attack. FIGURE 5 shows a system overview of TTIDS, where it can be seen that a target ECU is suspended and TTIDS analyzes CAN traffic to detect an abnormal pattern due to the suspension. TTIDS is divided into three steps: i) suspension detection, ii) transmission-resuming time estimation, and iii) malicious message classification. In the first step, suspension detection, TTIDS monitors the in-vehicle network to detect when an ECU is suspended. In the transmission-resuming time estimation step, TTIDS estimates when transmission from the suspended ECU is to resume. According to the CAN standard, suspended ECUs are designed to automatically recover from bus-off mode. Finally, in the malicious message classification step, TTIDS distinguishes between normal and malicious CAN messages.

B. SUSPENSION DETECTION

In this step, TTIDS monitors and analyzes CAN traffic in the in-vehicle network to identify any suspended ECUs.

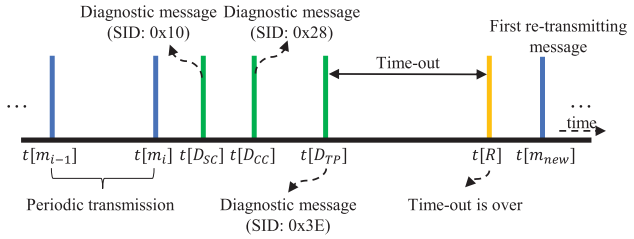


FIGURE 6. Timeline of suspended message transmission by using UDS.

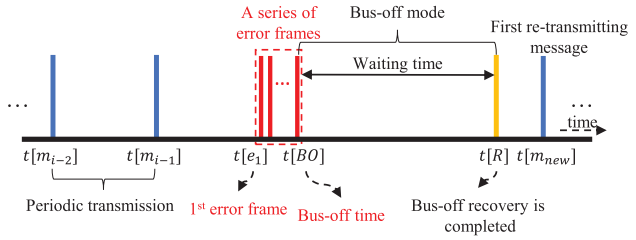


FIGURE 7. Timeline of message transmission during bus-off and recovery.

For the suspension attack using the UDS service, diagnostic request/response messages can be simply observed on the CAN bus. Accordingly, TTIDS also easily recognizes suspended ECUs even if an attacker employs the UDS service. Normally, an attacker may use a combination of the three UDS services—*DiagnosticSessionControl* (0×10), *CommunicationControl* (0×28), and *TesterPresent* ($0 \times 3E$)—to suspend an ECU. When an attacker suspends a target ECU via a bus-off attack, TTIDS looks for error frames, which can be easily detected as bit errors can be seen during a bus-off attack. TTIDS easily detects and counts the number of error frames in order to anticipate the TEC of an ECU. If the TEC exceeds the threshold, the corresponding ECU is considered to have entered bus-off mode and therefore suspended.

C. TRANSMISSION-RESUMING TIME ESTIMATION

Transmission from the suspended ECU should automatically resume if the attacker does not keep transmitting malicious messages and prolong the suspension. TTIDS is able to estimate when an ECU resumes transmission by pinpointing when the suspension attack ends. Depending on the suspension ways, TTIDS estimates the transmission-resuming time as follows.

1) TRANSMISSION RESUMPTION FROM UDS SERVICES

An ECU can be suspended by a combination of the UDS services—*DiagnosticSessionControl* (0×10), *CommunicationControl* (0×28), and *TesterPresent* ($0 \times 3E$). In particular, for *DiagnosticSessionContol*, a sub-function of 0×03 should be coupled, implying that the "Extended Diagnostic Session" is a sub-function. After the diagnostic session is enabled, an ECU is suspended for a certain time-out d if it receives the diagnostic message for *CommunicationControl* (0×28) with sub-function "Disable Tx and Rx for ECU." To maintain the suspension, a diagnostic message for *CommunicationControl* (0×28) or *TesterPresent* ($0 \times 3E$) should be transmitted before time-out. If transmissions for *CommunicationControl* (0×28) or *TesterPresent* ($0 \times 3E$) repeat, suspension is also extended. Accordingly, an attacker can use these UDS services to suspend the message transmission of the target ECU. Moreover, TTIDS can estimate when a suspended ECU will revert to the default session mode and message transmission will resume by checking if it takes time-out d after transmission of *CommunicationControl* (0×28) or *TesterPresent* ($0 \times 3E$). FIGURE 6 shows a timeline for suspended message transmission by using the UDS services. The blue bars indicate when messages with a normal CAN ID are periodically transmitted. Accordingly, the first message after the end of the diagnostic session, m_{new}, is seen the time as follows.

$$t[m_{new}] = t[m_i] + n \times T, \tag{1}$$

where T is the periodicity of the transmission suspended during a diagnostic session and n is the first positive integer that satisfies the condition $t[m_i] + n \times T > t[R]$. The $t[R]$ is determined by time-out d after $t[D_{CC}]$ or $t[D_{TP}]$, the diagnostic message for *CommunicationControl* (0×28) or *TesterPresent* ($0 \times 3E$), respectively.

2) TRANSMISSION RESUMPTION FROM BUS-OFF ATTACK

When a CAN controller recognizes that its TEC exceeds 255, it normally reports this to its microcontroller via an interruption. Even though the response operation to this interruption is not standardized, we discovered that ECUs in real vehicles share commonalities in terms of how they recover from bus-off mode. Thanks to this commonality, TTIDS is able to estimate when bus-off recovery is complete and when transmission will resume. According to the CAN standard, a CAN controller in bus-off mode should wait until there are 128 occurrences of 11 consecutive recessive bits to resume transmission. However, we found that all ECUs (i.e., microcontrollers) are programmed to conduct additional operations during bus-off recovery. Namely, the ECUs we analyzed additionally wait for a particular time duration, which is elsewhere recommended for safe CAN communication [55]. The three parameters—wait time, controller recovery type, and timer behavior—are used so that TTIDS can estimate most reliably. We elaborate on these three parameters next. FIGURE 7 shows a timeline for message transmission affected by bus-off mode, by which it can be seen that how an ECU is suspended by bus-off mode and how it recovers.

a: WAITING TIME

The first concern is how long a microcontroller will wait after bus-off mode is detected. According to Copperhill Technologies Corporation [55], recovery should not be immediate as to give other ECUs the chance to catch up or recover. In this context, immediate recovery means that a microcontroller would not conduct any operation for bus-off recovery, and the CAN controller would simply reset. Accordingly, the wait time would be minimal before bus-off mode is complete,

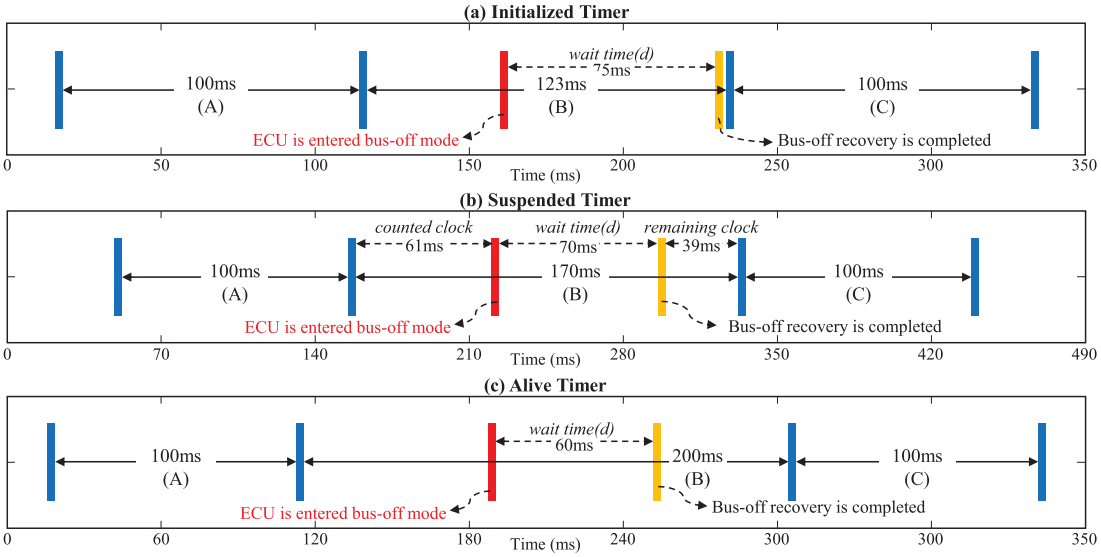


FIGURE 8. Timeline of message transmission according to timer behavior (*A* represents the time interval of the last two messages before bus-off mode, *B* represents the time interval of the last message before bus-off mode plus the first message after bus-off recovery, and *C* represents the time interval of the first two messages after bus-off recovery.)

typically taking only hundreds of milliseconds. We define this wait time as denoted by d. All ECUs we analyzed have a static duration for wait time d. In particular, some ECUs wait for the static number of timer interruptions rather than a designated amount of actual time. If an ECU waits for the static number of timer interruptions, the amount of wait time differs depending on when the ECU entered bus-off mode.

b: CONTROLLER RECOVERY TYPE

According to the CAN specification, CAN controllers should be initialized from bus-off mode after there are 128 occurrences of 11 consecutive recessive bits. We found there are two instances of recovery when CAN controllers start initialization from bus-off mode. One instance immediately starts initialization by counting the 128 occurrences of 11 consecutive recessive bits when the TEC exceeds 255. Regardless of the microcontroller waiting out wait time d, the CAN controller can recover. Accordingly, CAN controllers in bus-off mode may already recover before the microcontroller completes the wait time. The other instance of recovery starts initialization after the microcontroller completes the wait time. We refer to these two processes as immediate recovery and wait-then-recovery, respectively. Even though both types of CAN controllers may seem identical, there is a difference in terms of when the microcontroller and its CAN controller recover from bus-off mode and are ready to resume transmission. In addition, some CAN controllers flush out messages backed up in buffers during bus-off mode. If a buffer is flushed out after a CAN controller recovers, these messages are thought of as first transmissions. In other words, the suspended transmission by bus-off mode immediately resumes when bus-off recovery is complete. This type of transmission can be estimated only with the two parameters wait time and CAN controller recovery.

c: TIMER BEHAVIOR

The final concern is timer behavior during bus-off recovery. For periodic transmissions on the CAN bus, microcontrollers are periodically interrupted by timers, such that message transmissions are requested. All the timers we analyzed do not operate identically during bus-off mode. Since there is no standard for timer implementation, it seems that this depends largely on developers. However, we typically categorize timer behavior into three types: initialized, suspended and alive. Before we describe the behaviors of each type, we present the following equation for estimating when the first message will be newly transmitted after a successful bus-off recovery.

$$t[m_{new}] = t[BO] + d + r + A, \tag{2}$$

where m_{new} is the first transmitted message after bus-off recovery. $t[m_{new}]$ is the time when m_{new} is transmitted on the CAN bus. $t[BO]$ is the time when a CAN controller enters bus-off mode. Normally, 32 error frames are needed for an ECU's TEC to exceed 255. r is the amount of time needed for 128 occurrences of 11 recessive bits for the wait-then-recovery type. A is the amount of time determined by timer behavior.

Initialized Timer. This timer initializes as soon as bus-off recovery is complete. Upon initialization, the timer interrupts its microcontroller so that message transmission can be newly requested. As a result, $A = 0$ becomes the initialized timer, and the first message after the bus-off recovery m_{new} is seen the time as follows.

$$t[m_{new}] = t[BO] + d + r \tag{3}$$

This time can also be represented by $t[m_{new}] = t[R]$.

Suspended Timer. Unlike the initialized timer, the suspended timer does not interrupt after bus-off recovery.

(a) In-vehicle CAN network connection

(b) Vehicle A

(c) Vehicle B

FIGURE 9. Experimental setup.

TABLE 2. Criteria for determining timer behavior.

Timer Behavior	Time interval of two consecutive messages		
	Last two messages before bus-off mode	Last message before bus-off mode and first message after bus-off recovery	First two messages after bus-off recovery
Initialized	T	greater than $d+r$ and less than $T+d+r$	T
Suspended	T	$T+d+r$	T
Alive	T	nT	T

T is the original transmission periodicity.
d is the wait time.
r is the amount of time determined by 128 occurrences of 11 consecutive recessive bits. (If immediate recovery controller is given, $r = 0$)
n is a positive integer.

This timer is merely suspended and does not count down during bus-off mode. Accordingly, it waits until the remaining clocks run out before it enters bus-off mode. For example, if a timer that periodically interrupts every 100ms counts 15ms before bus-off mode, it would additionally count 85ms (100ms - 15ms) after bus-off recovery. The first message after bus-off recovery m_{new} is seen the time as follows.

$$\begin{aligned} t[m_{new}] &= t[BO] + d + r + T - (t[BO] - t[m_i]) \\ &= t[m_i] + T + d + r, \end{aligned} \tag{4}$$

where T is a time period for timer interruption.

Alive Timer. The alive timer continuously counts clocks even during bus-off recovery, and the microcontroller is still interrupted by this type even during bus-off mode. In other words, message transmission interruptions occur just the same as when bus-off mode does not occur. Of course, transmission requests during bus-off mode are ignored, and transmission requests after bus-off recovery are processed normally. Accordingly, we obtained the following equation to estimate transmission of the first message after bus-off recovery, in which m_{new} indicates the alive timer.

$$t[m_{new}] = t[m_i] + n \times T, \tag{5}$$

where T is the periodicity of the transmission suspended during bus-off mode, and n is the first positive integer that satisfies the condition $t[m_i] + n \times T > t[R]$. The $t[R]$ is determined by d and r. If the microcontroller enters bus-off mode and completes bus-off recovery within a single period, denoted as T, there would be no difference with when the first message m_{new} is seen outside of bus-off mode ($n = 1$).

Figure 8 shows an example of message transmissions according to the behavior of the three timers. Timer behavior can be determined by analyzing the time intervals of two consecutive messages. As a result, TABLE 2 outlines the criteria determining behavior.

D. CLASSIFICATION BETWEEN NORMAL AND MALICIOUS MESSAGES

In the above steps, we described how TTIDS detects a suspended ECU and estimates when message transmission will resume. With the correct interval of suspension, TTIDS is able to classify between normal and malicious messages during a masquerade attack. CAN messages that are observed before an ECU is suspended and after it is resumes should be classified as normal; on the other hand, CAN messages that are observed in the time interval during ECU suspension should be classified as malicious.

VI. EVALUATION

In this section, we perform a series of experiments to evaluate TTIDS and report our observations.

A. EXPERIMENTAL SETUP

TTIDS was evaluated on the CAN bus of two real vehicles (named Vehicle A and Vehicle B). In order to perform the a bus-off attack, we employed the hardware and software used in [40] conducted by Sekar *et al.* By attaching the hardware to the OBD-II port of the vehicle, we were able to generate errors on the CAN bus to force a particular ECU to enter bus-off mode. FIGURE 9 shows our experimental setup along with the real vehicles used in our experiments.

For performance metrics, we utilized a typical metric for error rates in a binary test. The false positive rate (FPR) and the false negative rate (FNR) are additionally defined to measure the performance of our method. In this evaluation, a ‘‘false positive’’ refers to the case in which a malicious CAN message is incorrectly identified as normal (i.e., missed detection) and a ‘‘false negative’’ refers to the case in which a normal CAN message is incorrectly identified as malicious (i.e., false alarm).

B. DISCOVERY OF DIAGNOSTIC PARAMETERS

Since no DBC format file was provided for Vehicle A and Vehicle B, we searched for diagnostic ID pairs implemented

TABLE 3. List of selected CAN IDs, diagnostic parameters, and classified recovery parameters of actual ECUs.

	ECU	CAN ID	Diagnostic parameters					Recovery parameters		
			Request CAN ID	Response CAN ID	SID	Sub-function ID	Time-out	Waiting time	Controller recovery type	Timer behavior
Vehicle A	A	0x018, 0x034, 0x050, 0x110, 0x120, 0x510, 0x517	0x7A0	0x7A8	0x10	0x03	5 s	70 ms	Immediate recovery	Suspended
			0x771	0x779	0x28	0x03				
					0x3E	0x00				
	B	0x042, 0x043, 0x044, 0x350	0x7B3	0x7BB	0x10	0x90	5 s	70 ms	Wait-then-recovery	Initialized
					0x28	0x01				
					0x3E	0x00				
	C	0x4F0, 0x4F1, 0x4F2, 0x51A, 0x690	0x7C6	0x7CE	0x10	0x03	5 s	60 ms	Immediate recovery	Alive
					0x28	0x01				
					0x3E	0x00				
	D	0x153, 0x164, 0x220, 0x1F1, 0x4B0, 0x4B1	0x7D1	0x7D9	0x10	0x03	5s	7-interruptions of 10 ms	Immediate recovery	Alive
					0x28	0x01				
					0x3E	0x00				
Vehicle B	A	0x07F, 0x410, 0x436, 0x50E, 0x520, 0x522, 0x541, 0x553, 0x559, 0x593, 0x5C0	0x7A0	0x7A8	0x10	0x03	5 s	70 ms	Immediate recovery	Suspended
					0x28	0x03				
					0x3E	0x00				
	B	0x042, 0x043, 0x044, 0x383	0x7B3	0x7BB	0x10	0x90	5 s	70 ms	Wait-then-recovery	Initialized
					0x28	0x01				
					0x3E	0x00				
	C	0x153, 0x164, 0x220, 0x386, 0x387, 0x507	0x7D1	0x7D9	0x10	0x03	5 s	7-interruptions of 10 ms	Immediate recovery	Alive
					0x28	0x01				
					0x3E	0x00				
	D	0x111, 0x112, 0x113, 0x502	0x7E1	0x7E9	0x10	0x03	5 s	70 ms	Immediate recovery	Alive
					0x28	0x01				
					0x3E	0x00				

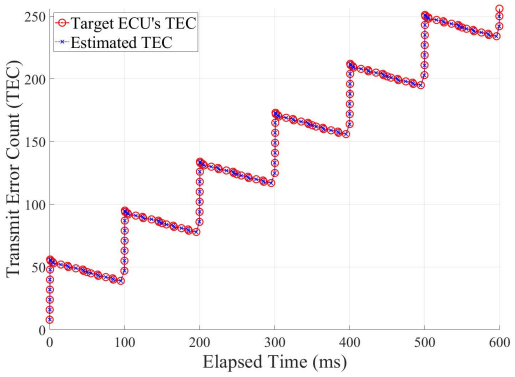


FIGURE 10. Value change of real and estimated TEC.

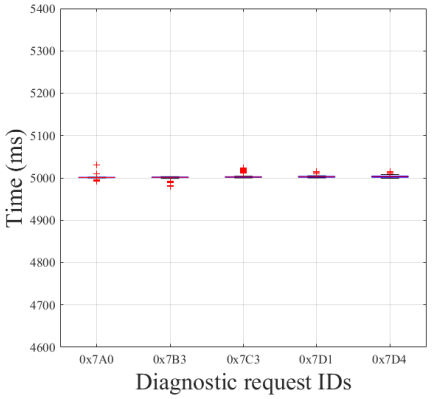


FIGURE 11. Distribution of time-out as a function of diagnostic request IDs.

on ECUs via a query for diagnostic request messages with request IDs ranging from 0×700 to $0 \times 7FF$. In doing so, we also discovered valid SIDs, sub-functions, and parameters for the diagnostic services. When an ECU receives a CAN message with a diagnostic request CAN ID, the ECU transmits a positive or negative response with a corresponding diagnostic response CAN ID. Depending on the responses, we can identify the valid diagnostic ID pairs and SIDs. After that, we are able to identify the valid sub-function and its parameters. For example, we discovered that an ECU has a SID of 0×28 for *CommunicationControl* and a sub-function of 0×03 for "Disable TX and RX". If the ECU finds some invalid sub-function or parameters, then it transmits a negative response message with a negative response code (NRC). Since we are able to observe via the NRC which parameters are invalid, such as service IDs (SID) or sub-functions, we rather understand which parameters are correct by repeating the transmission of the diagnostic request messages. As a result, we found 6 and 8 diagnostic request/response CAN ID pairs from Vehicle A and Vehicle B, respectively. TABLE 3 shows these results for diagnostic request/response CAN IDs, SIDs, and sub-functions for suspension of transmission. Due to security concerns, we only disclose partial information about those parameters.

C. NETWORK MAPPING

TTIDS knows which CAN IDs are sent from an ECU and can estimate the TEC by observing how many times the corresponding CAN messages are corrupted. Moreover, TTIDS is able to differentiate between normal and malicious CAN messages based solely on whether or not the ECU is suspended. Even amid suspension, CAN messages originally designed to be transmitted from the suspended ECU are observable during a masquerade attack. Accordingly, our method groups CAN IDs with their assigned ECUs, and we can group CAN IDs by suspending each ECU. While an ECU is suspended, the CAN messages that are originally designed to be periodically transmitted but are not observed should be associated with the suspended ECU. TABLE 3 shows the results of network mapping from Vehicle A and Vehicle B. We were able to suspend an ECU using the UDS services and

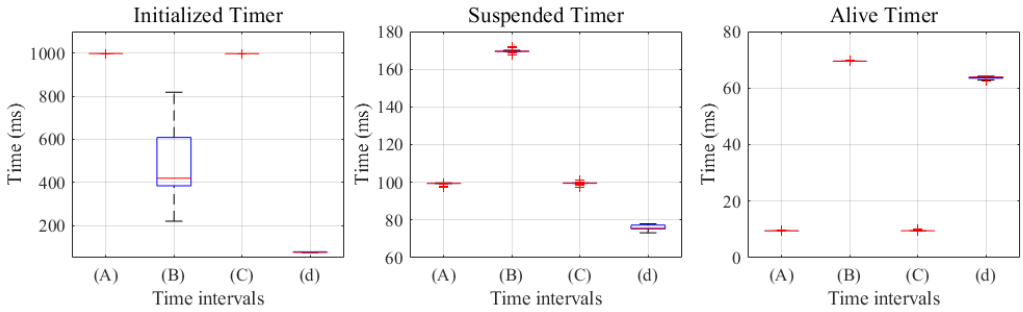


FIGURE 12. Distribution of time intervals for classification of timer behavior and waiting time (*A* represents the time interval of the last two messages before bus-off mode, *B* represents the time interval of the last message before bus-off mode plus the first message after bus-off recovery, *C* represents the time interval of the first two messages after bus-off recovery, and *d* represents the waiting time.)

a bus-off attack. Moreover, both techniques produced identical results, and we use these results to verify that the network mapping is correct.

D. DETECTION OF SUSPENDED ECUS

TTIDS is designed to detect when an ECU is suspended. In this subsection, we demonstrate how correctly TTIDS is at detecting a suspended ECU. As mentioned earlier, an ECU can be suspended via bus-off attack and UDS services. First, a suspended ECU by UDS services are simply detected by identify the corresponding diagnostic messages in the CAN bus. Because the diagnostic request/response messages are observed in the CAN bus, TTIDS can then identify the corresponding suspension of an ECU. On the other hand, there is no clear indicator of when an ECU enters bus-off mode. If an ECU is suspended via bus-off attack, TTIDS check the TEC because when the TEC exceeds a threshold of 255, the ECU enters bus-off mode. Accordingly, TTIDS is designed to estimate the TEC of an ECU by observing error frames. Each time an error frame is observed, TTIDS increases the TEC by 8 and decreases it by 1 for each successful transmission from the ECU. We construct a CAN bus prototype with Arduino Uno boards whose TEC can be read. Accordingly, we are able to compare the estimated TEC by observing CAN traffic with the real TEC and reading its register. FIGURE 10 shows the value changes for estimated and real TECs as a result of intentional bus-off attacks in the CAN bus prototype, and it can be seen that TTIDS perfectly estimates the TEC.

E. TRANSMISSION RESUMPTION FROM UDS SERVICES

When an ECU is suspended by UDS services, we can estimate when it will resume by counting how much time elapses after the last diagnostic messages is observed. As mentioned before, the diagnostic messages *CommunicationControl* (0×28) or *TesterPresent* ($0 \times 3E$) are transmitted to extend ECU suspension time. If an ECU does not receive any additional diagnostic messages, it automatically returns to the default session. TTIDS measures how much time an ECU takes to return to the default session from when we transmit the last diagnostic messages. FIGURE 11 shows the

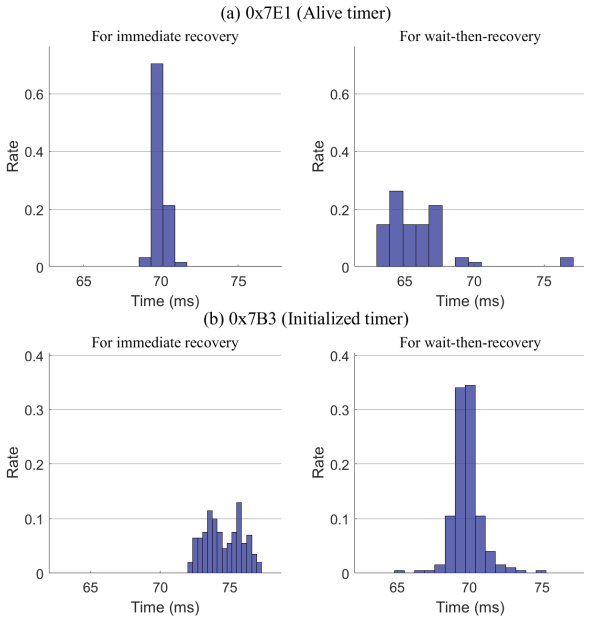


FIGURE 13. Distribution of time intervals to classify controller recovery types.

distribution of suspended time duration for one diagnostic message. All suspended ECUs we analyzed start working again approximately 5 seconds after the last diagnostic message is transmitted. From this observation, we conclude that TTIDS is able to estimate when an ECU suspended via UDS services will resume transmission.

F. TRANSMISSION RESUMPTION FROM BUS-OFF ATTACK

With the three parameters—wait time, controller recovery type, and timer behavior—TTIDS estimates when message transmission suspended by bus-off mode will resume. To demonstrate the feasibility of TTIDS, we first checked whether or not the actual ECUs are assigned to parameters classified. Even though the parameters for each ECU can be provided by automotive manufacturers, we were unfortunately unable to obtain this information. To overcome this

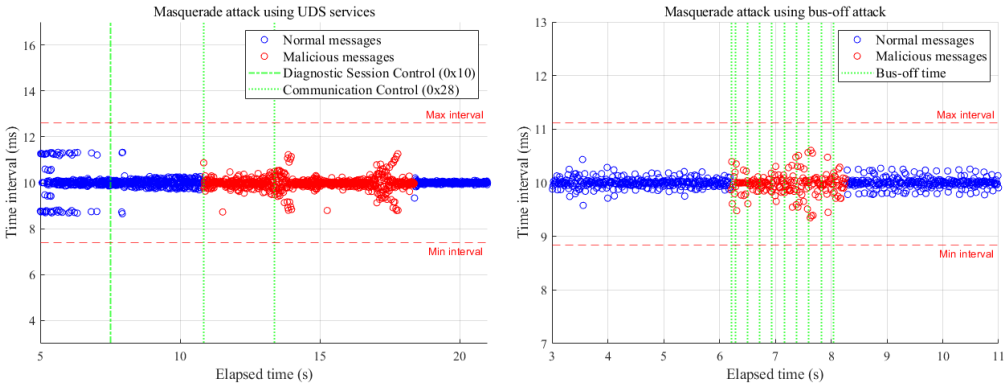


FIGURE 14. Time intervals during masquerade attacks using UDS services and bus-off attacks.

limitation, we manually analyzed patterns in transmissions directly affected by bus-off mode. We presented the criteria for determining timer behavior in TABLE 2, and we note that timer behavior can be determined by analyzing the transmission intervals of two consecutive messages during bus-off mode. FIGURE 12 shows the distribution of time intervals for classification of timer behavior, where the parameters of timer behavior can be simply determined.

Subsequently, we also checked two distribution types to classify variations in controller recovery. One distribution type applies to time intervals between when bus-off mode starts and recovery completes, making it possible to count clocks in reverse by timer behavior to check where bus-off recovery completes. The other distribution type relates to time intervals between when bus-off mode starts and when wait-then-recovery CAN controller begins recovery. This timing can also be checked by counting in reverse the 128 occurrences of 11 consecutive recessive bits, which is required for automatic recovery. Given these two different distributions, we were able to determine parameters for controller recovery types. Since the wait time is a static number, the controller recovery type is also determined by checking for a static distribution. If the distribution of time intervals between when bus-off mode starts and when recovery completes are relatively normal distribution, the controller recovery type can be classified as the parameter of immediate recovery. FIGURE 13 shows two different distribution models, from which we obtained parameters per ECU by manually analyzing CAN traffic. TABLE 3 shows the parameters we found for each actual ECU.

G. MASQUERADE ATTACK DETECTION

We conducted masquerade attacks on the CAN bus of real vehicles by performing a suspension attack. As mentioned earlier, the suspension attack is performed via either UDS services or a bus-off attack. For UDS services, we transmit a diagnostic request message so that the target ECU is suspended; at the same time, we inject CAN messages that are originally designed to be transmitted from the suspended ECU. We also inject them at the same frequency as

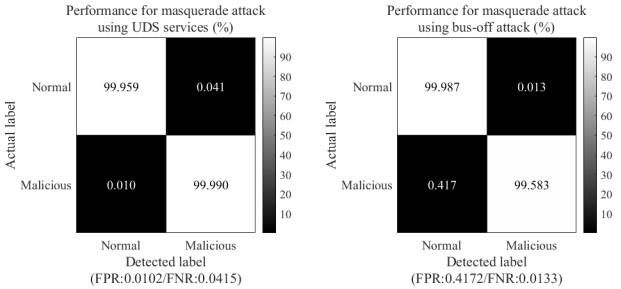


FIGURE 15. Performance of TTIDS under masquerade attacks.

the original frequency of one the suspended ECU transmits. Moreover, we are able to suspend a particular ECU by the bus-off attack. When the target ECU is suspended, we similarly conduct the injection of CAN messages that are designed to be transmitted from the suspended ECU. Our dataset for the masquerade attack and detection code are publicly available to foster further research.[1]

FIGURE 14 shows time intervals between two consecutive CAN messages, where blue indicates time intervals of normal CAN messages and red indicates time intervals of malicious CAN messages. As can be seen from the figure, there is no significant difference between normal and malicious CAN messages during a masquerade attack. Accordingly, it is expected that any frequency-based methods for automotive IDSs would be not able to detect a masquerade attack. On the other hand, TTIDS is able to detect such an attack by identifying ECU suspension statuses. FIGURE 15 illustrates the confusion matrices showing the accuracy of TTIDS in detecting masquerade attacks. We obtained an FPR of 0.213% and an FNR of 0.027%. As a result, we conclude that TTIDS reliably detects masquerade attacks that, up to this point, have gone undetected by existing methods.

H. COMPARISON WITH OTHER AUTOMOTIVE IDSS

In this subsection, we compare our method with four existing methods [19], [20], [24], [56] in terms of the

[1] https://forms.gle/FRL5Ptrzqh7DjJey9

TABLE 4. Comparison with other methods.

Automotive IDS	Category	Where to be implemented	Where to bu evaluated	How to suspend ECU
[56]	CAN ID filtering-based	All ECUs	On ptorotype	None
[20], [24]	Clock-based	Single ECU (e.g., Gateway)	On real vehicle	None
[19]	Payload-based	Single ECU (e.g., Gateway)	On real vehicle	Only UDS services
TTIDS (Ours)	Transmission -resuming time-based	Single ECU (e.g., Gateway)	On real vehicle	UDS services / Bus-off attack

four different matters shown in TABLE 4. Ansari *et al.* proposed a filtering-based automotive IDS that detects a self-identifier violation [56]. Even though this method can be easily applied, SW modification is necessary for filter configuration. This implies that every ECU should conduct additional computation to detect any self-identifier violations. Because they have constrained resources, though, the necessary functions would not be properly provided. Clock-based IDSs detect a masquerade attack by analyzing clock offsets that are inherent characteristics of ECUs [20], [24]. However, these clock offsets are evaluated using a dataset for a simple injection attack because it is difficult to create a dataset for a masquerade attack. In short, existing research has not suspended a real ECU. The injected messages were only assumed to be derived from a masquerade attack. Alternatively, Nowdehi *et al.* proposed an automotive IDS that analyzes a series of data payloads [19]. Their method detects a masquerade attack when the data payloads demonstrate a high deviation compared with normal ones. Even though these models indeed suspend a real ECU for a masquerade attack, the suspension is only done using the UDS service and there is no bus-off attack conducted to suspend a real ECU—a key improvement in our method. We thus conclude that our method is able to properly detect a masquerade attack more successfully than existing methods.

VII. DISCUSSION

A. EXTENDED INTRUSION PREVENTION SYSTEMS (IPS)

TTIDS can detect exactly when an ECU is suspended. If suspended messages are transmitted from the time when the EUC is suspended until the time when the suspended ECU resumes message transmission, the messages are deemed malicious. Therefore, since TTIDS can distinguish between normal and malicious messages in message units, it is possible to expand this framework to an extended intrusion prevention system (IPS) that blocks detected attack messages before transmission is complete. We hope to expand TTIDS into an IPS in future work.

VIII. CONCLUSION

In this paper, we proposed a Transmission-resuming Time-based IDS (TTIDS) to detect masquerade attacks on vehicles. It is widely known that masquerade attacks are the most advanced of attack techniques, and following this, it is difficult for existing automotive IDSs to detect them. The most effective automotive IDSs in existence that use the periodicity of CAN messages are unable to detect injections by advanced adversaries who can suspend the target ECU and mimic the original message frequency. To detect a masquerade attack in these circumstances, we focus on the fact that an attacker must first and foremost suspend transmission in a target ECU. Moreover, according to the CAN standard, the suspended ECU automatically reverts back to the default state. We also systemically analyzed when the suspended ECU will resume its periodic transmission. With this projection, TTIDS can detect malicious messages transmitted while the ECU is suspended. We performed masquerade attacks on two real vehicles to evaluate the performance of TTIDS. Overall experimental results demonstrate a low error rate, a false positive rate of 0.213%, and a false negative rate of 0.027%. In conclusion, TTIDS is able to effectively detect masquerade attacks.

REFERENCES

[1] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1204–1223, Jun. 2005.

[2] R. Bosch, "Can specification version 2.0," *Rober Bousch GmbH, Postfach*, vol. 300240, p. 72, Sep. 1991.

[3] 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.

[4] T. O. Career. (2016). *Serial Communication-Hack Your CanBus*. Accessed: Jan. 26, 2022. [Online]. Available: https://www.youtube.com/watch?v=1kd5nl0Akfo

[5] N. Nowdehi, A. Lautenbach, and T. Olovsson, "In-vehicle CAN message authentication: An evaluation based on industrial criteria," in *Proc. IEEE 86th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2017, pp. 1–7.

[6] S. Stachowski, R. Gaynier, and D. J. LeBlanc, "An assessment method for automotive intrusion detection system performance," Univ. Michigan, Transp. Res. Inst., Ann Arbor, MI, USA, Tech. Rep. DOT HS 812 708, 2019.

[7] (2021). *Cyber Security and Cyber Security Management System*. UNECE. Accessed: Jan. 26, 2022. [Online]. Available: https://unece.org/sites/default/files/2021-03/R155e.pdf

[8] (2021). *Software Update and Software Updates Management System*. UNECE. Accessed: Jan. 26, 2022. [Online]. Available: https://unece.org/sites/default/files/2021-03/R156e.pdf

[9] 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.

[10] 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.

[11] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell, "Modeling inter-signal arrival times for accurate detection of CAN bus signal injection attacks: A data-driven approach to in-vehicle intrusion detection," in *Proc. 12th Annu. Conf. Cyber Inf. Secur. Res.*, Apr. 2017, pp. 1–4.

[12] D. Stabili and M. Marchetti, "Detection of missing CAN messages through inter-arrival time analysis," in *Proc. IEEE 90th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2019, pp. 1–7.

[13] S. Otsuka, T. Ishigooka, Y. Oishi, and K. Sasazawa, "Can security: Cost-effective intrusion detection for real-time control systems," SAE Tech. Paper 2014-01-0340, 2014.

[14] M. Gmiden, M. H. Gmiden, and H. Trabelsi, "An intrusion detection method for securing in-vehicle CAN bus," in *Proc. 17th Int. Conf. Sci. Techn. Autom. Control Comput. Eng. (STA)*, Dec. 2016, pp. 176–180.

[15] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 1577–1583.

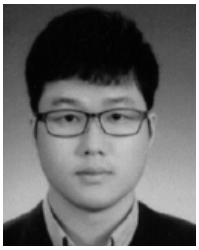
[16] A. Derhab, M. Belaoued, I. Mohiuddin, F. Kurniawan, and M. K. Khan, "Histogram-based intrusion detection and filtering framework for secure and safe in-vehicle networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 3, pp. 2366–2379, Mar. 2022.
[17] D. Stabili, M. Marchetti, and M. Colajanni, "Detecting attacks to internal vehicle networks through Hamming distance," in *Proc. Int. Annu. Conf. (AEIT)*, Sep. 2017, pp. 1–6.
[18] 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(RTSI)*, Sep. 2016, pp. 1–6.
[19] N. Nowdehi, W. Aoudi, M. Almgren, and T. Olovsson, "CASAD: CAN-aware stealthy-attack detection for in-vehicle networks," 2019, *arXiv:1909.08407*.
[20] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *Proc. 25th Secur. Symp. (USENIX)*, 2016, pp. 911–927.
[21] A. Tomlinson, J. Bryans, and S. A. Shaikh, "Towards viable intrusion detection methods for the automotive controller area network," in *Proc. 2nd ACM Comput. Sci. Cars Symp.*, 2018, pp. 1–9.
[22] M. E. Verma, M. D. Iannacone, R. A. Bridges, S. C. Hollifield, P. Moriano, B. Kay, and F. L. Combs, "Addressing the lack of comparability & testing in CAN intrusion detection research: A comprehensive guide to CAN IDS data & introduction of the ROAD dataset," 2020, *arXiv:2012.14600*.
[23] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran, "Cloaking the clock: Emulating clock skew in controller area networks," in *Proc. ACM/IEEE 9th Int. Conf. Cyber-Phys. Syst. (ICCPS)*, Apr. 2018, pp. 32–42.
[24] 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.
[25] C. Miller and C. Valasek, "Advanced CAN injection techniques for vehicle networks," Black Hat USA, Las Vegas, NV, USA, Tech. Rep., 2016.
[26] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking Tesla from wireless to CAN bus," *Briefing, Black Hat USA*, vol. 25, pp. 1–16, Jul. 2017.
[27] S. Christian, K. Kasper, L. Tobias, J. Mattias, W. Johannes, and H. Filip. (2018). *Caring Caribou*. Accessed: Jan. 26, 2022. [Online]. Available: https://github.com/CaringCaribou/caringcaribou
[28] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, "A stealth, selective, link-layer denial-of-service attack against automotive networks," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2017, pp. 185–206.
[29] P.-S. Murvay and B. Groza, "DoS attacks on controller area networks by fault injections from the software layer," in *Proc. 12th Int. Conf. Availability, Rel. Secur.*, Aug. 2017, pp. 1–10.
[30] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1044–1055.
[31] G. Bloom, "WeepingCAN: A stealthy CAN bus-off attack," in *Proc. 3rd Int. Workshop Automot. Auto. Vehicle Secur.*, 2021, p. 25.
[32] S. Longari, M. Penco, M. Carminati, and S. Zanero, "CopyCAN: An error-handling protocol based intrusion detection system for controller area network," in *Proc. ACM Workshop Cyber-Phys. Syst. Secur. Privacy (CPS-SPC)*, 2019, pp. 39–50.
[33] K. Iehira, H. Inoue, and K. Ishida, "Spoofing attack using bus-off attacks against a specific ECU of the CAN bus," in *Proc. 15th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2018, pp. 1–4.
[34] A.-I. Radu and F. D. Garcia, "LeiA: A lightweight authentication protocol for CAN," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2016, pp. 283–300.
[35] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede, "LiBrA-CAN: A lightweight broadcast authentication protocol for controller area networks," in *Proc. Int. Conf. Cryptol. Netw. Secur.* Cham, Switzerland: Springer, 2012, pp. 185–200.
[36] S. Nürnberger and C. Rossow, "–vatican–vetted, authenticated can bus," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2016, pp. 106–124.
[37] J. Van Bulck, J. T. Mühlberg, and F. Piessens, "VulCAN: Efficient component authentication and software isolation for automotive control networks," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, Dec. 2017, pp. 225–237.
[38] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks–practical examples and selected short-term countermeasures," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.* Berlin Springer, 2008, pp. 235–248.
[39] C. Ling and D. Feng, "An algorithm for detection of malicious messages on CAN buses," in *Proc. Nat. Conf. Inf. Technol. Comput. Sci.*, 2012, pp. 1–4.
[40] S. Kulandaivel, T. Goyal, A. K. Agrawal, and V. Sekar, "CANvas: Fast and inexpensive automotive network mapping," in *Proc. 28th Secur. Symp. (USENIX)*, 2019, pp. 389–405.
[41] H. Olufowobi, S. Hounsinou, and G. Bloom, "Controller area network intrusion prevention system leveraging fault recovery," in *Proc. ACM Workshop Cyber-Phys. Syst. Secur. Privacy (CPS-SPC)*, 2019, pp. 63–73.
[42] K. Serag, R. Bhatia, V. Kumar, Z. B. Celik, and D. Xu, "Exposing new vulnerabilities of error handling mechanism in CAN," in *Proc. 30th Secur. Symp. (USENIX)*, 2021, pp. 4241–4258.
[43] *Road Vehicles–Unified Diagnostic Services (UDS)—Part 1: Specification and Requirements*, Standard ISO 14229-1: 2013, 2013.
[44] 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.
[45] F. Sommer, J. Durrwang, M. Wolf, H. Juraschek, R. Ranert, and R. Kriesten, "Automotive network protocol detection for supporting penetration testing," in *Proc. SECURWARE*, 2019, pp. 114–119.
[46] Y. L. Le Yu, P. Jing, X. Luo, L. Xue, K. Zhao, Y. Zhou, T. Wang, G. Gu, S. Nie, and S. Wu, "Towards automatically reverse engineering vehicle diagnostic protocols," in *Proc. USENIX Secur.*, 2022.
[47] C. Valasek and C. Miller, "Adventures in automotive networks and control units," *Def Con*, vol. 21, nos. 260–264, pp. 15–31, 2013.
[48] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, Aug. 2015.
[49] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin, "LibreCAN: Automated CAN message translator," in *Proc. Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 2283–2300.
[50] M. Marchetti and D. Stabili, "READ: Reverse engineering of automotive data frames," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 4, pp. 1083–1097, Sep. 2018.
[51] D. Frassinelli, S. Park, and S. Nürnberger, "I know where you parked last summer: Automated reverse engineering and privacy analysis of modern cars," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1401–1415.
[52] W. Choi, S. Lee, K. Joo, H. J. Jo, and D. H. Lee, "An enhanced method for reverse engineering CAN data payload," *IEEE Trans. Veh. Technol.*, vol. 70, no. 4, pp. 3371–3381, Apr. 2021.
[53] *OpenDBC From Comma.AI*. Accessed: Jan. 26, 2022. [Online]. Available: https://github.com/commaai/opendbc
[54] AUTOSAR. (2017). *Specification of Diagnostic Communication Manager Release 4.3.1*. Accessed: Jan. 26, 2022. [Online]. Available: https://www.autosar.org/standards/classic-platform/classic-platform-431
[55] O. Pfeiffer, A. Ayre, and C. Keydel, *Embedded Networking With CAN and CANopen*. MA, USA: Copperhill Media, 2008.
[56] M. R. Ansari, W. T. Miller, C. She, and Q. Yu, "A low-cost masquerade and replay attack detection method for CAN in automobiles," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

SEYOUNG LEE received the B.S. degree in mathematics and computer science from the University of Seoul, Seoul, South Korea, in 2014, and the M.S. degree in information security from the Graduate School of Information Security, Korea University, Seoul, in 2016, where he is currently pursuing the Ph.D. degree in information security. His research interests include applied cryptography, embedded system security, and automotive 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. He was a Postdoctoral Researcher at the Department of Computer and Information Systems, University of Pennsylvania, Philadelphia, PA, USA, from 2016 to 2018. 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 the IoT/CPS security.

ARAM CHO received the B.S. degree in industrial engineering (major) and electronics engineering (bimajor) from Korea University, Seoul, South Korea, in 2011, and the M.S. degree in information security from the Graduate School of Information Security, Korea University, in 2013. He was a Research Engineer at Samsung Electronics. In 2015, he joined Hyundai Motor Company, Hwaseong, South Korea, as a Senior Engineer. His research interests include security for in-vehicle networks and privacy for vehicle.

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



WONSUK CHOI received the B.S. degree in mathematics from the University of Seoul, Seoul, South Korea, in 2008, and the M.S. and Ph.D. degrees in information security from Korea University, Seoul, in 2013 and 2018, respectively. He was a Postdoctoral Researcher at the Graduate School of Information Security, Korea University. In 2020, he joined the Division of IT Convergence Engineering, Hansung University, Seoul, as an Assistant Professor. His research interests include security for body area networks, usable security, applied cryptography, and smart car security.