Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 5650245, 13 pages
https://doi.org/10.1155/2019/5650245

Research Article

Enhanced Android App-Repackaging Attack on In-Vehicle Network

Yousik Lee,[1] Samuel Woo,[2] Jungho Lee,[3] Yunkeun Song,[1] Heeseok Moon,[4] and Dong Hoon Lee [5]

[1] *ESCRYPT, 4F, ABN Tower, 331 Pangyo-ro, Bundang-gu, Seongnam-si, Gyeonggi-do 13488, Republic of Korea*
[2] *Electronics and Telecommunications Research Institute, 218 Gajeong-ro, Yuseong-gu, Daejeon 34129, Republic of Korea*
[3] *Korea Information Certificate Authority Inc., 5th Fl C-dong, PDC, 242, Pangyo-ro, Bundang-gu, Seongnam-si, Gyeonggi-do 13487, Republic of Korea*
[4] *KATECH, 74 Yongjeong-ri, Pungse-myeon, Dongnam-gu, Cheonan, Chungcheongnam-do, Republic of Korea*
[5] *Korea University, Anam-ro, Seongbuk-gu, Seoul 02841, Republic of Korea*

Correspondence should be addressed to Dong Hoon Lee; donghlee@korea.ac.kr

Received 18 October 2018; Accepted 27 December 2018; Published 3 February 2019

Guest Editor: Karl Andersson

The convergence of automobiles and ICT (information and communication technology) has become a new paradigm for the development of next-generation vehicles. In particular, connected cars represent the most in-demand automobile-ICT convergence technology. With the development of 5G technology, communication between vehicle and external device using autonomous driving and Internet of things (IoT) technology has been remarkably developed. Control of vehicles using smart phones has become a routine feature, and over 200 Android apps are in use. However, Android apps are easy to tamper by repackaging and allowing hackers to attack vehicles with using this vulnerability, which can lead to life-critical accidents. In this study, we analyze the vulnerabilities of connected car environments when connecting with IoT technologies and demonstrate the possibility of cyberattack by performing attack experiments using real cars and repackaging for commercial apps. Furthermore, we propose a realistic security technology as a countermeasure to attain safety against cyberattacks. To evaluate the safety of the proposed method, a security module is developed and a performance evaluation is conducted on an actual vehicle.

1. Introduction

Various electronic control units (ECUs) are being mounted in the latest vehicles to enhance the safety and comfort of the driver and passengers, and this automobile-ICT (information and communication technology) convergence has become a new paradigm for the development of next-generation vehicles [1, 2]. As the development of automotive electronics is accelerating, the proportion of electric parts in automobile production has exceeded 40% and is expected to exceed 50% by 2020. Along with the increase in automotive electronic parts, communications between vehicles and outside data services are also increasing. It is expected that around 75% of all vehicles worldwide will be connected cars using wireless networks by 2020 [3]. Connected cars, one of the leading use case of IoT, are growing rapidly with the development of 5G technology. [4] 5G will be the backbone of IoT, overcoming all space-time constraints and providing a complete connection with all the user-centric “things”. [5] The existing V2X technology is evolving into C-V2X using LTE or 5G, and vehicles are connected to external devices via IoT technologies. Vehicles, which have for decades been typical machines, are evolving into large IT systems. As a result, IT experts can now participate in the development of vehicles and the general public can understand vehicles more easily.

However, with the development of automotive electronics, vehicles have become a new target for hackers [6–8]. Lee et al. published the results of a cyberattack experiment that analyzed the vulnerabilities of an ELM327-based CAN-to-Bluetooth device and apps for vehicles [9]. Furthermore, as shown in the hacking of GM OnStar by Samy Kamkar and the study on the vulnerabilities of in-vehicle smartphones by

Mikhail Kuzin and Victor Chebyshev, vulnerable smartphone apps are emerging as a new attack surface for vehicles [10, 11].

In the present study, we automate app analysis tasks that we performed manually in [9] and analyze the vulnerabilities of 213 apps registered in the Google Play store. Based on the results of this analysis, we carry out attacks to control actual vehicles after repackaging the 10 most popular apps among the commercial apps for vehicles. Finally, we propose a security technology to protect against such attacks and develop an access control device based on a whitelist for communication service security for ELM327 devices. These are universal access control devices applicable to all vehicles if CAN IDs are possible to be configured by car manufacturers.

This paper organized as follows. Background describes the overview of basic concepts and components. Next, Proposed Attack Model introduces how the attack model we proposed is constructed in both laboratory and real vehicle environment. Countermeasure describes the security measure against the attack we conducted. Finally, Conclusion presents our conclusions.

2. Background

2.1. CAN (Controller Area Network). To support efficient communication among ECUs, BOSCH developed the Controller Area Network (CAN) in the early 1980s. CAN is a sender ID-based broadcast communication technology that supports the bus network topology. CAN drastically decreased the complexity and length of in-vehicle communication lines by resolving the drawbacks of point-to-point communication. In the data transmission process between ECUs using CAN, the sending ECU includes its unique ID in the CAN data frame before sending the data. The receiving ECUs can selectively receive the data after checking the ID of the sending ECU included in the broadcast data frame. CAN bus systems are classified into the following two types depending on the length of the ID in the data frame:

(i) Standard CAN 2.0A (11-bit ID)

(ii) Extended CAN 2.0B (29-bit ID)

The data frame is composed of SOF (start of frame), Arbitration ID (arbitration), Control, Data, CRC (cyclic redundancy check), ACK (acknowledge), and EOF (end of frame) fields. The standard CAN 2.0A format uses an 11-bit ID as the arbitration bit, and the extended CAN 2.0B format uses a 29-bit ID as the arbitration bit.

Because CAN was designed only for closed network environments, it does not provide basic information protection features (e.g., confidentiality, authentication, and access control). Recently, as connected car services were commercialized, the use of FOTA (Firmware Over-The-Air) spread, in-vehicle networks, and external networks has become interconnected. As a result, CAN has been exposed to the same cyberattacks experienced in general IT environments.

2.2. Connected Car. The connected car environment implies that vehicles are always connected to an external network.

Table 1: PID codes and parameters.

PID Code	Description	Units	Formula
04	Calculated engine load value	%	A∗100/255
0A	Fuel pressure	kPa	3∗A
0C	Engine RPM	kPa	(256∗A+B)/4
0D	Vehicle speed	Km/h	A

The connected car environment consists of the following components [12, 13]:

(i) The vehicle in which ECUs have been installed and an in-vehicle network has been configured

(ii) The portal for providing various services to the vehicle

(iii) The communication link for connecting the vehicle with the portal

In such an environment, vehicles have multiple ECUs installed in them, which are connected to an in-vehicle network such as a CAN BUS system.

The communication link is constructed using a wireless communication device such as a telematics ECU or an ELM327. The portal is divided into web-based services and smartphone application-based services.

Through experiments, this study proves that vehicles are exposed to serious threats if connected car environments are constructed without solving the vulnerabilities of in-vehicle networks; a security mechanism is then proposed to solve this problem. The attack model that we propose has been designed based on the connected car environment shown in Figure 1.

2.3. OBD Protocol. OBD (on-board diagnostics) is used to diagnose the vehicle state. The first OBD was produced to control the exhaust gases of vehicles [14]. As the development of automotive electronics accelerates, more ECUs have been installed and the OBD has developed to diagnose them and check the fault list. Later, OBD was established as an international standard (ISO 15765-4), and it has become possible to confirm the vehicle state with an automotive diagnosis device through standardized terminals [15]. In 1996, the US government mandated the installation of OBD-II in all vehicles sold in the US to control the environmental problems of exhaust gases.

The checking of vehicle status using OBD-II operates via the request/response method and the OBD PID (OBD Parameter ID) which is used in checking process defined in the SAE J1989 standard [16]. The structure of the CAN ID and data frame for communication defined in this standard is shown in Figure 2. The PID codes and parameters are defined in Table 1 [17, 18].

The diagnostic information of vehicles obtained through OBD-II can be easily acquired not only through the dedicated diagnostic devices used in car repair shops, but also through smartphone applications and ELM327 modules purchased from aftermarket suppliers. Therefore, malicious users can control vehicles by manipulating the diagnostic information

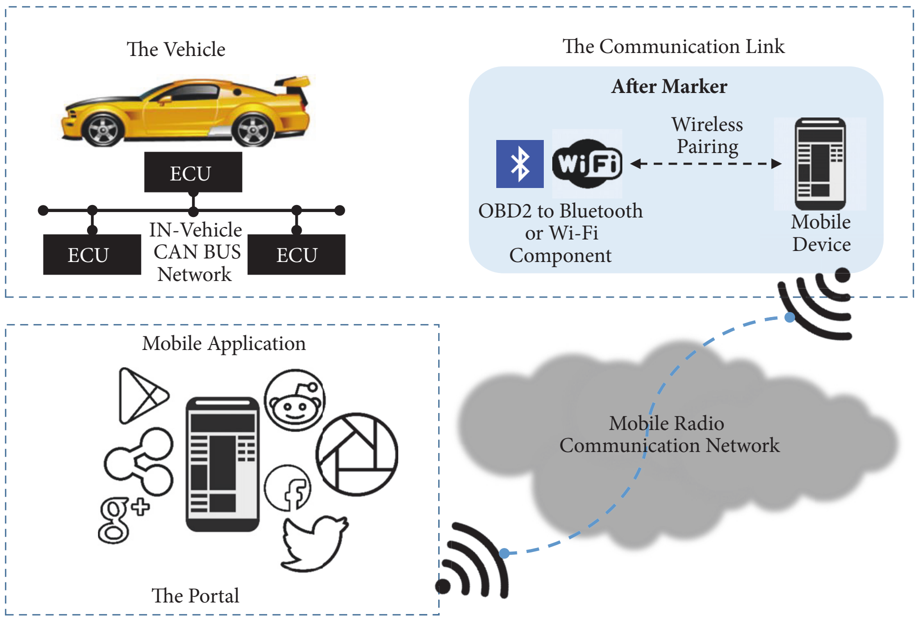


Figure 1: Connected car environment composed of ELM327 devices and mobile applications.

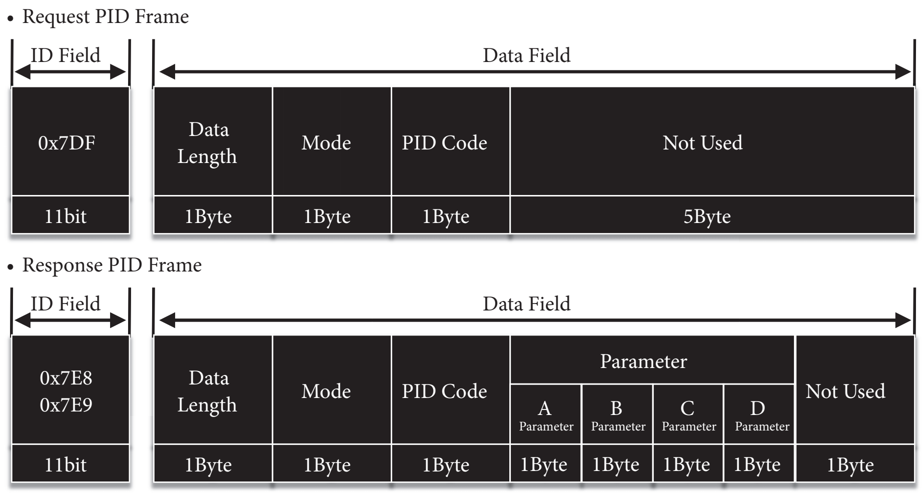


Figure 2: Structure of CAN ID and data frame for communication.

of vehicles, and security measures are required for the diagnostic devices and related protocols.

2.4. ELM327. ELM327 is a microcontroller for car repair used to diagnose the vehicle state. If a wireless communication module such as Bluetooth is added to ELM327, the driver can check the vehicle state in real time by connecting his/her smartphone with ELM327. The process of checking the vehicle state using a smartphone and ELM327 is illustrated in Figure 3

2.5. Cyber Kill Chain. Lockheed Martin proposed a cyberattack process that is generally applicable to all cyberattacks [19]. They also proposed the Cyber Kill Chain method to identify the threats in each attack phase by analyzing the cyberattack process, and to increase the resilience of organizations by defeating and neutralizing the purpose, intention, and activities of attackers. The Cyber Kill Chain consists of seven steps: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objectives. The characteristics of each phase are described in Table 2 [20].

3. Proposed Attack Model

The attack model we propose is more realistic than those of existing studies and has been designed on the basis of two threats. The first threat is that smartphone apps for

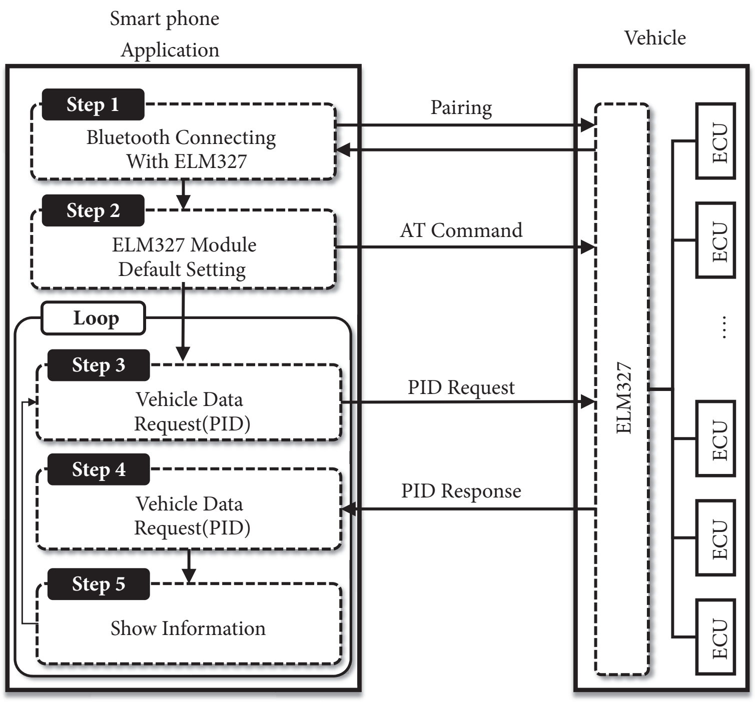


Figure 3: Process of checking vehicle status using smartphone and ELM327.

Table 2: Cyber Kill Chain phases and characteristics.

Phase	Action
Reconnaissance	Identify the target
Weaponization	Prepare the operation
Delivery	Launch the operation
Exploitation	Gain access to victim
Installation	Establish beachhead at the victim
Command and Control	Remotely control the implants
Actions on Objectives	Achieve the mission's goal

vehicles can be easily forged/repackaged and redistributed. The second threat is that attackers can attack wirelessly using repackaged malicious smartphone apps at any time. The proposed attack model is composed of an attacker, a target vehicle, and a victim. The characteristics of every entity are gathered to form one attack model. We analyzed the proposed attack model using the cyber intrusion kill chain.

3.1. Attacker Ability. Using a vehicle diagnostic device, the attacker can acquire a CAN data frame that can drive a specific ECU mounted in the target vehicle. Furthermore, the attacker can download a vehicle application that is distributed/sold in the app market and repackage it in a desired form. The attacker can then inject a desired CAN data frame into the in-vehicle CAN using a malicious app. The malicious app repackaged by the attacker can be distributed through a third-party market.

3.2. Target Vehicle. The target vehicle is assumed to utilize the connected car environment shown in Figure 1. The target vehicle uses CAN to facilitate communication among the ECUs. However, because the CAN bus system does not guarantee authentication for communicated data frames, a CAN data frame may be used in a retransmission attack [21, 22].

3.3. Victim Behavior. The driver of the target vehicle (victim) downloads a vehicle diagnosis app from the app market onto his/her smartphone and receives various services while driving his/her vehicle. The vehicle diagnosis app that the driver (victim) downloaded is assumed to be a malicious app distributed through the app market by the attacker. The driver (victim) using the malicious app cannot recognize the attack behaviors of the malicious app (injecting a data frame that can control a specific ECU).

3.4. Attack Model. The driver of the target vehicle (victim) who has downloaded the vehicle diagnosis app can receive various services while driving his/her vehicle. The attacker can inflict cyberattacks to many unspecified vehicles by abusing this service situation. The connected car environment in Figure 1 represents an environment in which a wireless communication module (ELM327) is installed in the OBD-II terminal. Through this terminal, the vehicle can connect to the in-vehicle CAN, and the smartphone app is always interconnected with the in-vehicle CAN during vehicle operation via the paring of the user smartphone and the ELM327 module. The attack model process is divided into

Table 3: Analysis of proposed attack model using Cyber Kill Chain.

Reconnaissance	(i) Analysis of vehicle control packet (using a diagnostic device) (ii) Analysis of an Android app for vehicles (iii) Analysis of ELM327 protocol
Weaponization	(i) Repacking the Android app for vehicles
Delivery	(i) Distribution to third-party app market
Exploitation Installation	(i) User downloads and installs the app on his/her smartphone
Command & Control	(i) Analysis of vehicle operation state through the in-vehicle network packets
Actions on Objectives	(i) Forced control of the automotive E/E system, causing a traffic accident

an attack preparation phase and an attack performance phase. The entire process including the attack preparation phase and attack performance phase is as follows:

(1) Attacker analyzes the communication vulnerabilities of a wireless communication module for vehicles (based on ELM327).

(2) Attacker downloads an app for vehicles from the app market, using a wireless communication module for vehicles based on ELM327.

(3) Attacker generates malicious code using the analysis results from step (1).

(4) Attacker inserts malicious code in the downloaded app for vehicles and repackages it.

(5) Attacker distributes the repacked malicious app through the app markets (including third-party markets).

(6) Victim installs the ELM327-based wireless communication module in the OBD-II terminal of his/her vehicle.

(7) Victim downloads the app for vehicles to his/her smartphone from the app market (or a third-party market) and installs it. It is assumed that the downloaded app is the malicious app distributed by the attacker.

(8) Victim runs the malicious app and drives his/her vehicle.

(9) As soon as the vehicle starts operating, the malicious app carries out a malicious act.

The attack model consists of nine steps in total as outlined above. Table 3 shows an analysis of the proposed attack model using the Cyber Kill Chain.

4. Attack Experiment

In this chapter, we prove the possibility of forced control of vehicles using an app repackaged by an attacker. Our attack experiment consists of two phases: preparation phases and actual attack phases. The preparation phase and the actual attack step are each divided into two detailed steps:

Table 4: AT commands for controlling vehicles.

AT Command	Description	Group
Z	Reset All	General
SH xyz	Set Header to xyz	OBD
AL	Allow Long(>7byte) message	ODB
AR	Automatic Receive	OBD
R0/R1∗	Responses Off/On∗	OBD
SP x	Set Protocol to h	OBD
CAF0/CAF1∗	CAN Automatic Formatting Off/On∗	CAN

(1) Preparation phase 1: communication protocol analysis of the CAN-to-Bluetooth module (ELM327 module).

(2) Preparation phase 2: vulnerability analysis of the vehicular application and production of the malicious vehicular application.

(3) Actual attack phase 1: LABCAR-based attack experiment.

(4) Actual attack phase 2: real car-based attack experiment.

Finally, we performed a risk assessment for all apps related to ELM327 sold on the Google Store. In order to risk assessment, we manufactured automated analysis tools.

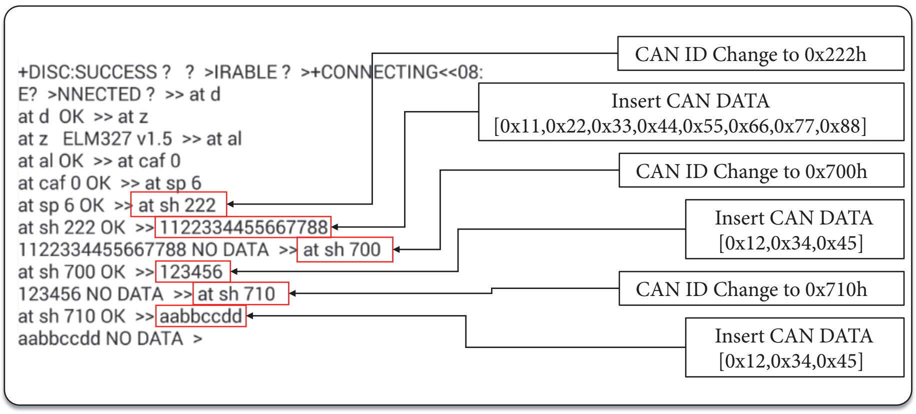
4.1. Preparation Phase (Analysis).

In this section, the vulnerabilities of the ELM327 module and the vehicle diagnosis app are analyzed. In addition, all the apps sold/distributed in the Android app market are examined and the risk of app-repackaging attacks is analyzed.

(A) Analysis of Communication Protocol between Vehicle Diagnosis App and ELM327. As described in Section 2.4, the ELM327 module is mounted in the OBD-II port of the vehicle, and delivers the vehicle state information to the vehicle diagnosis app according to the request-response method. The ELM327 module generally uses a fixed CAN ID when sending a request message. However, in a special case, the CAN ID and data of the request message can be changed using the AT command provided by ELM Electronics. The AT commands that can be used to control vehicles are listed in Table 4.

However, even if the CAN ID and data are changed, the vehicle recognizes any data as normal data if it follows the communication protocols of AT commands and ELM327. In other words, if the CAN data desired by the user is sent to ELM327 as shown in Figure 4, the same information is received in the vehicle.

Every vehicle diagnosis app based on ELM327 acquires the vehicle state information using the OBD PID. Therefore, the OBD PID code can be easily found by decompiling and analyzing the vehicle diagnosis app. In the next section, we discuss the method of finding vulnerabilities using the AT command and OBD PID information in a commercial



```
+DISC:SUCCESS ?  ? >IRABLE ? >+CONNECTING<<08:
E? >NNECTED ? >> at d
at d  OK >> at z
at z  ELM327 v1.5 >> at al
at al OK >> at caf 0
at caf 0 OK >> at sp 6
at sp 6 OK >> at sh 222
at sh 222 OK >> 1122334455667788
1122334455667788 NO DATA >> at sh 700
at sh 700 OK >> 123456
123456 NO DATA >> at sh 710
at sh 710 OK >> aabbccdd
aabbccdd NO DATA >
```

CAN ID Change to 0x222h
Insert CAN DATA [0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88]
CAN ID Change to 0x700h
Insert CAN DATA [0x12,0x34,0x45]
CAN ID Change to 0x710h
Insert CAN DATA [0x12,0x34,0x45]

Figure 4: Message transmission from outside the vehicle to ELM327.

diagnosis app, and the method of repackaging it into a malicious app.

(B) Analyzing and Repackaging Vehicle Diagnosis App. For vulnerable vehicle diagnosis apps, the strings of the ELM327 AT command and OBD PID are exposed in plain text. If the attacker has decompiled the app, he/she can easily find the AT command and OBD PID by string search only. In this chapter, the method of searching attack points and inserting attack codes using the ELM327 AT command and OBD PID is described from the standpoint of an attacker, and the process of producing a malicious app using this method is explained.

Figure 3 shows the process in which the vehicle diagnosis app operates. The vehicle diagnosis app uses AT command and OBD PID after being connected with ELM327. This allows to analyze the vehicle diagnosis app easily.

When we decompile a target app for attack and search the AT command and OBD PID, they are searched. This searched string is then replaced with the vehicle control string desired by the attacker, and a malicious app is produced by repacking the app. Figure 5 shows how a desired vehicle control command can be inserted by simply modifying the searched AT command or the OBD PID.

The vehicle diagnosis app checks the vehicle status using the OBD PID. If the attacker can find the receiving location information in the decompiled vehicle diagnosis app, that location can be specified as the attack point. Figure 6 shows the results of finding the attack point and acquiring the vehicle information. If the attacker can insert the attack code for controlling the vehicle based on this information, he/she can arbitrarily control the vehicle. Figure 7 shows that the vehicle control command has been inserted at the desired point by the attacker.

Through the above analysis process, two attack tools can be produced. The attack tool types to be produced and the method of producing the attack tools are as follows:

(i) Delivery of attack command at the time the app is started when the vehicle diagnosis app is started; it initializes ELM327 using the AT command. Thus, the attacker replaces the AT command in the app to be attacked with the desired vehicle control command and repackages it.

(ii) Delivery of attack command in the specific condition of the vehicle.

The vehicle diagnosis app can check the vehicle status using the OBD PID. The attacker can insert a desired vehicle control command by using the repackaging method with the OBD ID in the above-mentioned analysis process.

In the next section, the forgery risk associated with the AT command and OBD PID of existing vehicle diagnosis apps distributed in the Android app market is analyzed.

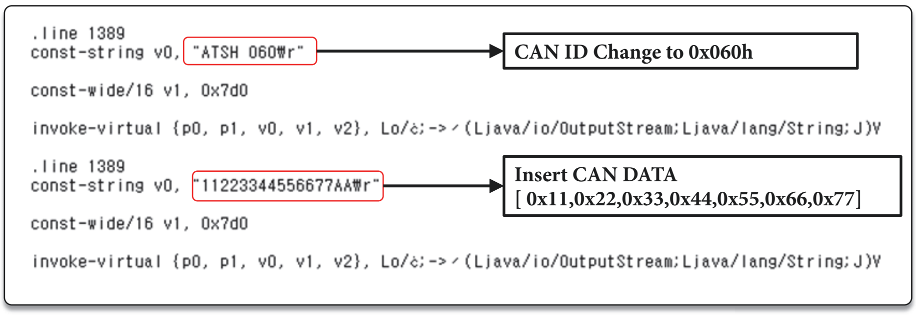
4.2. Actual Attack Experiment. This section describes the attack experiment, which utilizes an actual vehicle and the results from the preparation phase. The actual attack experiment in this section corresponds to steps (4) throw (7) of the seven steps of the Cyber Kill Chain. The attack experiment environment is outlined in Table 5.

(A) LABCAR. A LABCAR-based attack experiment was performed first to verify the effectiveness of the attack tools produced in the preparation phase. LABCAR is configured as shown in Figure 8, and each component has the functions described below.

If the attack is successful in the above-mentioned LABCAR environment, it is checked through the operation of the CANoe monitoring tool and the instrument panel. This LABCAR-based attack experiment confirmed that it is possible to inject a malicious data frame into an in-vehicle CAN through an app-repackaging attack.

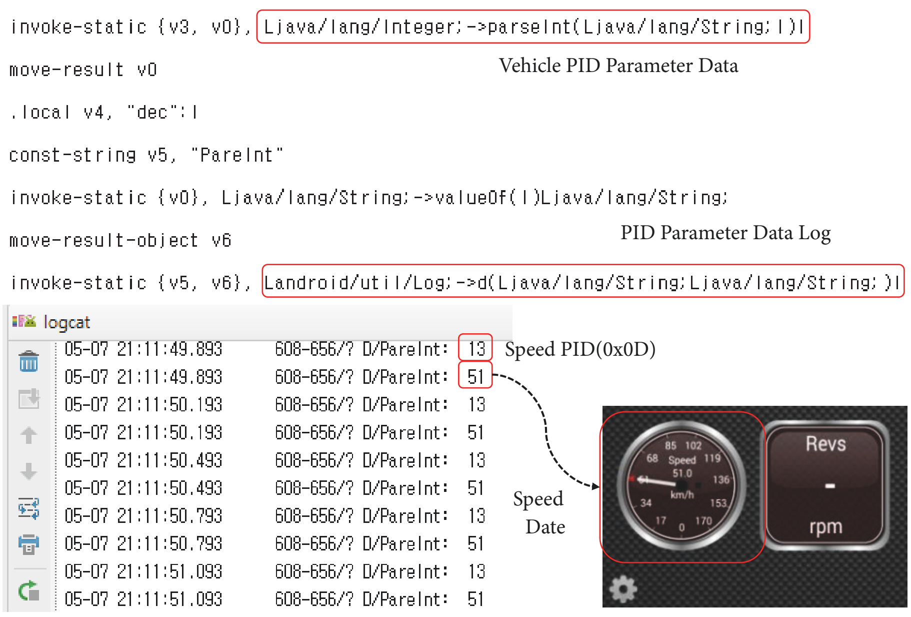
(B) Actual Vehicle. We also performed an attack experiment using an actual vehicle. The results of this experiment were recorded in a video, which was then uploaded to the web [23]. The attack experiment was carried out using the attack tools produced in the app analysis step. There were three attacks in total, which are described below.

(i) Door Unlock. After the driver leaves the vehicle in which ELM327 is installed, the attacker accesses the ELM327 through a smartphone and unlocks the door



```
.line 1389
const-string v0, "ATSH 060₩r"

const-wide/16 v1, 0x7d0

invoke-virtual {p0, p1, v0, v1, v2}, Lo/¿;->⁄(Ljava/io/OutputStream;Ljava/lang/String;J)V

.line 1389
const-string v0, "11223344556677AA₩r"

const-wide/16 v1, 0x7d0

invoke-virtual {p0, p1, v0, v1, v2}, Lo/¿;->⁄(Ljava/io/OutputStream;Ljava/lang/String;J)V
```

Figure 5: Message inserting CAN data by modifying AT command or OBD PID.



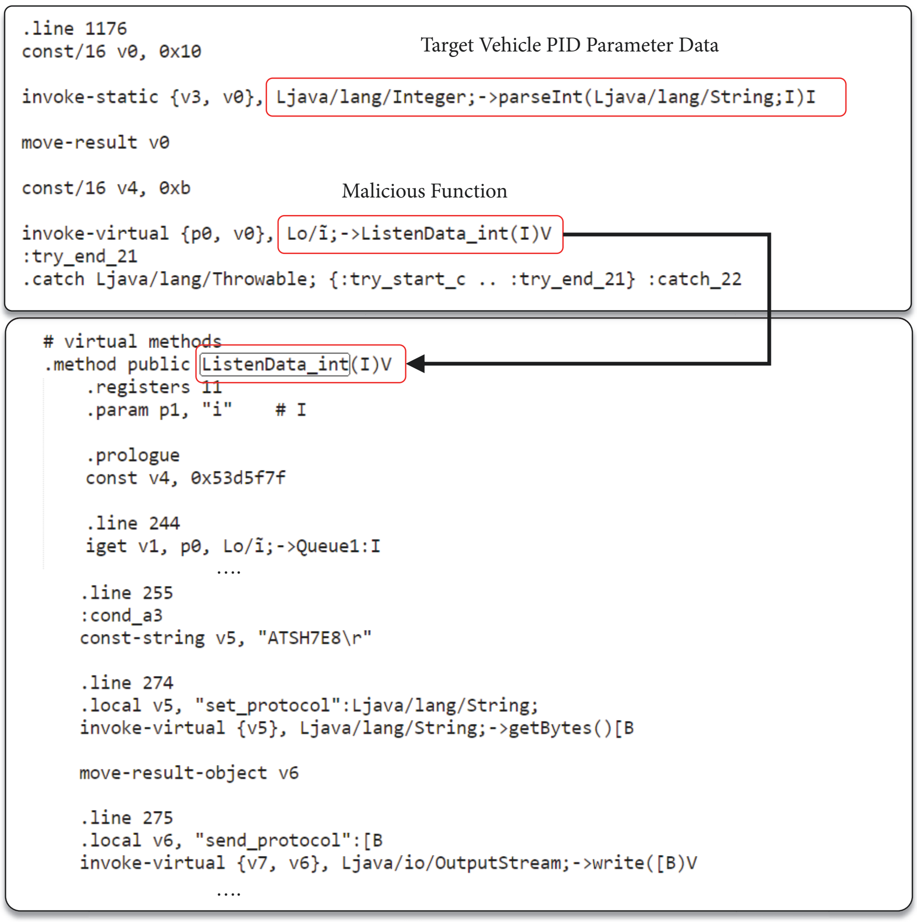
```
invoke-static {v3, v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;I)I

move-result v0

.local v4, "dec":I

const-string v5, "PareInt"

invoke-static {v0}, Ljava/lang/String;->valueOf(I)Ljava/lang/String;

move-result-object v6

invoke-static {v5, v6}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
```

```
05-07 21:11:49.893    608-656/? D/PareInt:  13
05-07 21:11:49.893    608-656/? D/PareInt:  51
05-07 21:11:50.193    608-656/? D/PareInt:  13
05-07 21:11:50.193    608-656/? D/PareInt:  51
05-07 21:11:50.493    608-656/? D/PareInt:  13
05-07 21:11:50.493    608-656/? D/PareInt:  51
05-07 21:11:50.793    608-656/? D/PareInt:  13
05-07 21:11:50.793    608-656/? D/PareInt:  51
05-07 21:11:51.093    608-656/? D/PareInt:  13
05-07 21:11:51.093    608-656/? D/PareInt:  51
```

Figure 6: Message attack location of the analyzed vehicle diagnosis app and the acquisition of vehicle information.

Table 5: Cyber AT commands for controlling vehicles.

Experiment tools	Model Name	Functions & Features	Note
Vehicle	Omitted	(i) Target Vehicle (ii) Analyze CAN Data & Monitoring Target	Actual vehicle
Instrument panel	Omitted	(i) Testing Simulation ECU (ii) Target ECU	LABCAR actual vehicle
CAN BUS	CAN CASE XL	(i) Simulation CAN BUS (ii) Connecting ECU and CANoe	LABCAR
ELM327	ELM327 (Version 1.5)	(i) Diagnostic Device	LABCAR actual vehicle
Smartphone	Samsung Galaxy A3 (OS Version 5.1.1)	(i) Target Smartphone Device	LABCAR actual vehicle
Smali & BakSmali	Smali & BakSmali (Version 2.1.0)	(i) APK Assemble and Disassemble Tool	LABCAR actual vehicle
Signapk	Signapk	(i) APK Signing Tool	LABCAR actual vehicle
CANoe	CANoe (Version 7.1)	(i) CAN Monitoring Tool & Capture Tool	LABCAR actual vehicle
Diagnostic Application	Omitted	(i) Target Application (ii) Deployed Application	LABCAR actual vehicle



```
.line 1176
const/16 v0, 0x10

invoke-static {v3, v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;I)I

move-result v0

const/16 v4, 0xb

invoke-virtual {p0, v0}, Lo/ĩ;->ListenData_int(I)V
:try_end_21
.catch Ljava/lang/Throwable; {:try_start_c .. :try_end_21} :catch_22
```

```
  # virtual methods
  .method public ListenData_int(I)V
      .registers 11
      .param p1, "i"    # I

      .prologue
      const v4, 0x53d5f7f

      .line 244
      iget v1, p0, Lo/ĩ;->Queue1:I
                  ....
     .line 255
     :cond_a3
     const-string v5, "ATSH7E8\r"

     .line 274
     .local v5, "set_protocol":Ljava/lang/String;
     invoke-virtual {v5}, Ljava/lang/String;->getBytes()[B

     move-result-object v6

     .line 275
     .local v6, "send_protocol":[B
     invoke-virtual {v7, v6}, Ljava/io/OutputStream;->write([B)V
                  ....
```

FIGURE 7: Vehicle diagnosis app in which the vehicle control command has been inserted at the desired point.

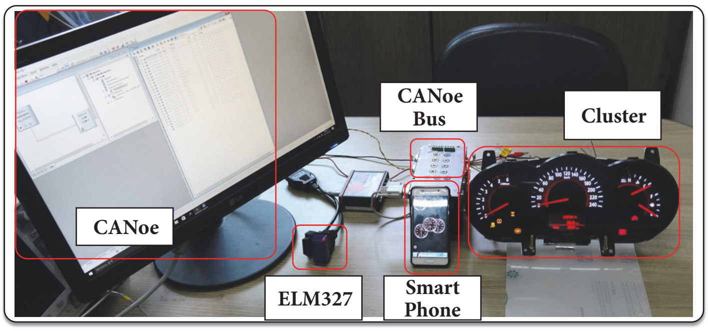


FIGURE 8: Operation LabCar simulation environment tool.

lock. Once the door lock is unlocked, the attacker can steal items from inside the vehicle and commit various other malicious acts.

(ii) Engine Off (Application Start). In this experiment, the Engine Off command set by the attacker is delivered when the repackaged app is started. The target vehicle was empty for safety and this experiment was performed with a safety device. It was confirmed that when the attacker installed a malicious app and started it, the Engine Off command was delivered and the target vehicle was stopped.

(iii) Engine Off (Vehicle's Status). In this experiment, the repackaged app delivered the Engine Off command in a specific state of the vehicle. It was confirmed that the engine stopped at the vehicle speed specified by the attacker.

Through the above three experiments, it was found that various apps sold/distributed in the Google Play store were

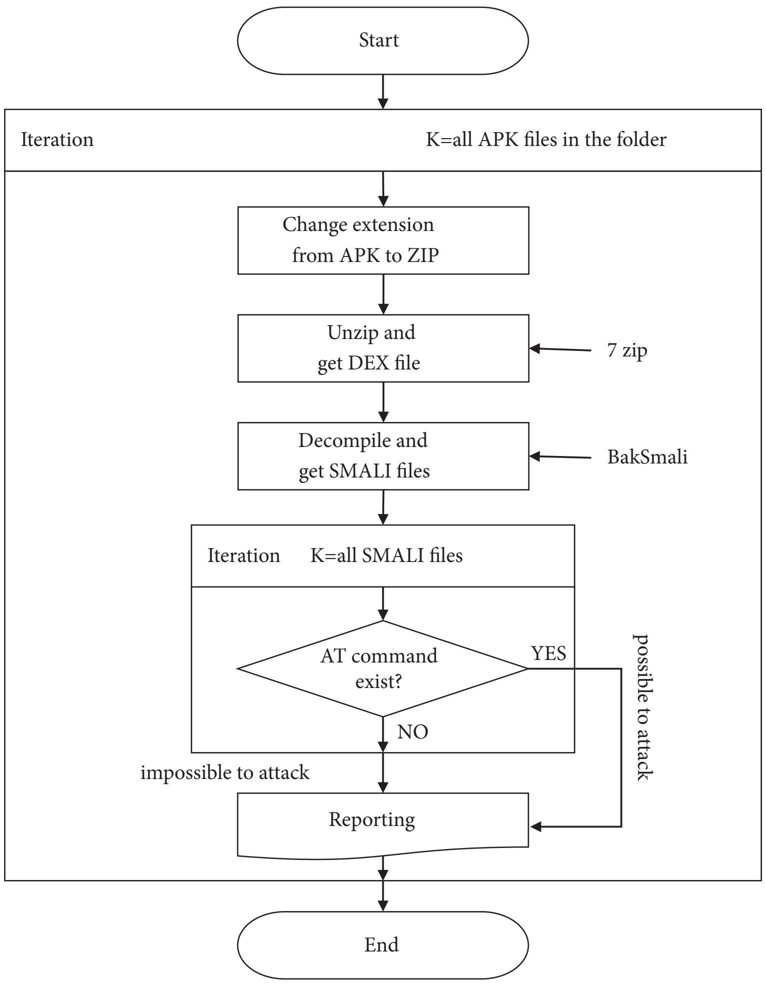


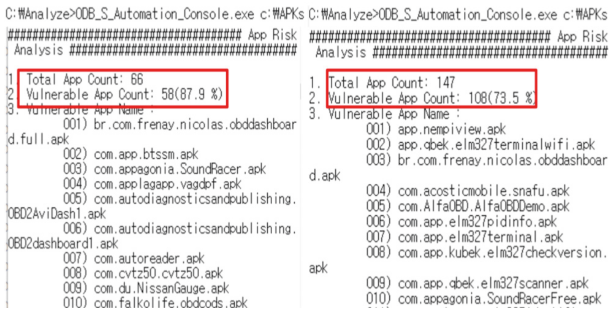
Figure 9: Operation process of the automatic vulnerability analysis.

exposed to the app-repackaging attack. There are primarily two reasons that the app-repackaging attack is possible. First, it is easy to analyze applications in terms of software. Second, the attack command can be delivered to the internal network through the OBD-II port of the vehicle in terms of hardware.

In the next chapter, we will discuss effective countermeasures against the above-mentioned attack.

4.3. Analysis of the Risk of Commercial Vehicle Diagnosis Apps. In this section, the forgery risk associated with the AT command and OBD PID of apps is analyzed. The risk is analyzed by checking the existence of the AT command in the target app. In this study, the risk analysis was conducted for 213 apps related to ELM327 among the vehicle-related apps in the Google Play store (as of June 2017).

To check the existence of the AT command, the DEX (Dalvik Executable) file must be found first, which is a byte code-level executable file that exists in the APK (Android Application Package) file. Then the smali file is extracted by decompiling the DEX file, and the existence of the AT command is determined. The searching and analyzing process for checking for AT commands in several hundred



```
C:₩Analyze>ODB_S_Automation_Console.exe c:₩APKs
##################################### App Risk
 Analysis ####################################

1. Total App Count: 66
2. Vulnerable App Count: 58(87.9 %)
3. Vulnerable App Name :
        001) br.com.frenay.nicolas.obddashboar
d.full.apk
        002) com.app.btssm.apk
        003) com.appagonia.SoundRacer.apk
        004) com.applagapp.vagdpf.apk
        005) com.autodiagnosticsandpublishing.
OBD2AviDash1.apk
        006) com.autodiagnosticsandpublishing.
OBD2dashboard1.apk
        007) com.autoreader.apk
        008) com.cvtz50.cvtz50.apk
        009) com.du.NissanGauge.apk
        010) com.falkolife.obdcods.apk
```

```
C:₩Analyze>ODB_S_Automation_Console.exe c:₩APKs
##################################### App Risk
 Analysis ####################################

1. Total App Count: 147
2. Vulnerable App Count: 108(73.5 %)
3. Vulnerable App Name :
        001) app.nempiview.apk
        002) app.qbek.elm327terminalwifi.apk
        003) br.com.frenay.nicolas.obddashboar
d.apk
        004) com.acosticmobile.snafu.apk
        005) com.AlfaOBD.AlfaOBDDemo.apk
        006) com.app.elm327pidinfo.apk
        007) com.app.elm327terminal.apk
        008) com.app.kubek.elm327checkversion.
apk
        009) com.app.qbek.elm327scanner.apk
        010) com.appagonia.SoundRacerFree.apk
```

Figure 10: Running screen of the automatic vulnerability analysis.

apps can be automated because it is a mechanical, repetitive task [24]. In this study, an automatic vulnerability analysis tool was developed for this task [25]. Figure 9 shows the process used by the automatic vulnerability analysis tool, and Figure 10 shows the running screen.

The results of analyzing the apps using the automatic vulnerability analysis tool are outlined in Table 6. The number of apps that have a plain text AT command that can be

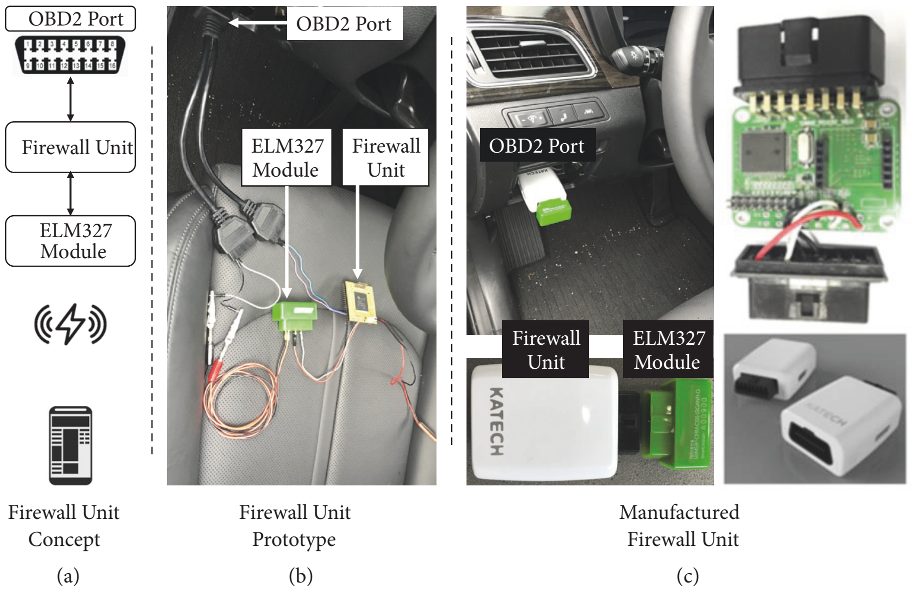


FIGURE 11: Access control system production process, (a) concept design, (b) prototype, and (c) actual manufacturing tool.

TABLE 6: App analysis results using automatic vulnerability analysis tool.

	Number of apps	Possible to attack (%)	Impossible to attack (%)
Total	213	166 (78)	47 (22)
Paid apps	66	58 (88)	8 (12)
Free apps	147	108 (73)	39 (27)

used for attack is 166, which accounts for approximately 78% of the 213 apps in total. In other words, most of the apps in the Google Play store can be used for attacks through app-repackaging. Furthermore, both free and paid apps are exposed to attacks.

The results of analyzing the apps using the automatic vulnerability analysis tool are outlined in Table 5. The number of apps that have a plain text AT command that can be used for attack is 166, which accounts for approximately 78% of the 213 apps in total. In other words, most of the apps in the Google Play store can be used for attacks through app-repackaging. Furthermore, both free and paid apps are exposed to attacks.

5. Countermeasure

In this chapter, we propose a security mechanism to protect against app-repackaging attacks. We designed a security mechanism considering the constraints of in-vehicle CANs and verified it. The proposed security mechanism is largely divided into two methods:

(i) Malicious data frame blocking method using a whitelist-based access control system

(ii) Code analysis defense method using app obfuscation

(A) Whitelist-Based Firewall. Most external devices such as ELM327 access the in-vehicle CAN through the OBD-II port. Car manufacturers must establish an access control policy for CAN data frames that are sent from external devices connected to the OBD-II port. In order to construct a safe in-vehicle CAN communication environment, a firewall is required to verify the ID of a CAN data frame (which comes in through the OBD-II port) and selectively send it to the subnetwork.

Car manufacturers can perform access control for CAN data frames that flow in through the OBD-II port during vehicle operation by generating a whitelist. The whitelist-based firewall can be designed/developed as follows:

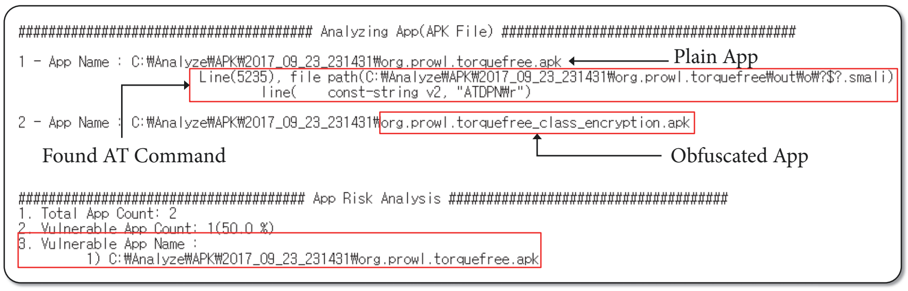
(1) Car manufacturer defines a CAN ID that can be flowed into the in-vehicle CAN through the OBD-II port during vehicle operation.

(2) Car manufacturer develops a whitelist-based firewall using the CAN ID defined in step (1) and installs it in the OBD-II port.

(3) Every external device can send data frames to the in-vehicle CAN only through the firewall installed at the OBD-II port.

(4) The firewall checks the data frames sent from external devices and drops abnormal data frames.

We designed and developed a whitelist-based firewall as shown in Figure 11. This firewall is installed between the OBD-II port and external device (ELM327 module) as shown in Figure 11(a). After producing the prototype shown in Figure 11(b), we connected it to an actual vehicle and performed an access control experiment. The prototype module was developed using an F28335-based microcontroller unit (MCU). All CAN data frames sent by ELM327 to the F28335-based MCU were checked first before being delivered to the



```
######################################## Analyzing App(APK File) ########################################

1 - App Name : C:₩Analyze₩APK₩2017_09_23_231431₩org.prowl.torquefree.apk
                       Line(5235), file path(C:₩Analyze₩APK₩2017_09_23_231431₩org.prowl.torquefree₩out₩o₩?$?.smali)
                               line(    const-string v2, "ATDPN₩r")

2 - App Name : C:₩Analyze₩APK₩2017_09_23_231431₩org.prowl.torquefree_class_encryption.apk

####################################### App Risk Analysis #######################################
1. Total App Count: 2
2. Vulnerable App Count: 1(50.0 %)
3. Vulnerable App Name :
         1) C:₩Analyze₩APK₩2017_09_23_231431₩org.prowl.torquefree.apk
```

Figure 12: Result of running the automatic vulnerability analysis tool for obfuscated app.

OBD-II port. Therefore, all data frames that do not have the predefined CAN ID could be dropped.

We developed the firewall device as a finished product with assistance from the Korea Automotive Technology Institute. The firewall device that we developed is shown in Figure 11(c). This firewall is equipped with an MCU based on an Infineon XC2265N (16-bit) microcontroller, and the whitelist can be easily updated. This access control policy for CAN data frames must be determined by the car manufacturer.

(B) App Obfuscation. The app-repackaging attack was possible because readability was provided for easy analysis of the internal app code, and the addition and repackaging of the app contents were possible. The app-repackaging attack is carried out largely in two steps.

In the first step, the AT command of the target app is found. In the second step, the app is repackaged by inserting malicious code. From the viewpoint of the Cyber Kill Chain, if only one of the attack steps is disabled, the attack cannot be performed.

Practically, the most effective method for preventing an app-repackaging attack is to prevent the AT command from being found.

The obfuscation technique that is mainly applied in Java is described in Table 7 [26].

When class encryption is not utilized, it is difficult to defend against attacks that find the AT command through string encryption, renaming, control flow, and API hiding.

Figure 12 shows the analysis result obtained using the automatic vulnerability analysis tool developed in this study, in a case where analyzing source code inside the app is made difficult by applying the class encryption method, which encrypts the DEX file and dynamically loads it when running the app.

In order to defend against the attacks performed in this study, it is recommended to apply class encryption, which makes the AT command unsearchable by encrypting the DEX file that includes the original source code.

6. Related Studies

K. Kosher et al. first reported ECU forced control attacks using actual vehicles. CAN data frames that can control vehicles by force using CAN message fuzzing, ECU firmware reverse engineering, and vehicle diagnostic device analysis were obtained. Using the acquired CAN data frames, wired and wireless attack models were proposed and actual vehicle hacking experiments were conducted. Among the proposed attack models, the short-range wireless attack and the long-range wireless attack showed very threatening results. It has been proven that a short-range wireless attack can hack vehicles in a Bluetooth pairing between a smartphone infected with malicious code and the hands-free function of vehicles. Furthermore, it has been experimentally proven that long-range wireless attacks can be performed using the vulnerabilities of the communication protocol and the telematics devices installed in vehicles [7].

M. Wolf, T. Hoppe, and others proved through an experiment based on the CANoe simulator that a message retransmission attack is possible in an in-vehicle CAN. To solve this problem, they proposed a message authentication method based on certificate-based ECU authentication and symmetric keys. However, it is impossible to apply their proposed security mechanisms to an actual vehicle environment because the computing power of ECUs and the payloads of CAN data were not considered [27, 28].

D. K. Nilsson et al. proposed the DDA (Delayed Data Authentication) method, which considered the computing capacity of ECUs and the limited CAN message structure. The DDA method uses the message authentication code (MAC) to prevent message retransmission attacks. They proposed a method of using the CRC field, while pointing out the shortage of areas available in the CAN message structure. Furthermore, they proposed a DDA-based method of authentication in which four messages are grouped. However, the CRC field in CAN cannot be used dynamically. In addition, real-time data processing must be guaranteed in vehicles. It is impossible to apply the DDA method to the vehicle environment because it generates a delay of at least 80 ms in authentication [29].

Woo et al. proposed a data frame authentication method using 32-bit truncated MAC. In consideration of the limited characteristics of the CAN data frame, they proposed a method of using the CRC and Extended ID fields for message authentication. In addition, they proposed a method of updating the session key to strengthen the safety of the 32-bit truncated MAC. This method ensures real-time data processing without generating additional data frames. However, it is impossible to apply their method to an actual

Table 7: App analysis results using automatic vulnerability analysis tool.

Obfuscation	Description
string encryption	The used string is replaced with an encrypted string, and a decryption method is added to the class file and the encrypted string is decrypted during runtime.
renaming	The classes, fields, and methods are renamed with meaningless names to make it difficult to analyze the decompiled source code.
control flow	The positions of commands in the code area of the class file are changed or trash commands are inserted to make it difficult to analyze flow during decompiling.
API hiding	Sensitive libraries are used or the method calling is hidden.
class encryption	A specific class file is encrypted and stored, and the dynamically decrypted code is run during runtime.

vehicle environment because the CAN protocol itself must be modified in order to use the CRC field for sending the message authentication code [6].

7. Conclusion

In this study, we have shown that vehicles can be easily attacked by analyzing ELM327s, which are commercial products that send vehicle information to smartphones and examining Android apps in the app market (Google Play store) that show vehicle information to owners after receiving the information. As countermeasures to such attacks, app obfuscation and a whitelist-based firewall were proposed. For the attack method used in this study, commercial products that are now sold or distributed were used. Because anyone can repackage Android apps if they have abilities to develop and analyze them, this has large ripple effects in terms of reality and risk. The more the 5G is expanded, the more the connection between the vehicle and the external device is increased; thus, the risk of cybersecurity will be increased. Therefore, multistep, multifaceted security measures are required to attain security against realistic threats. In the case of the firewall proposed in this study, a separate device that operates based on a whitelist is attached to an aftermarket device. However, car makers must install the security feature in the OBD-II interface, because we cannot realistically depend on an external device to counter threats to the vehicle itself. Furthermore, from the Cyber Kill Chain perspective, the security measures proposed in this study are countermeasures in terms of "reconnaissance" and "command and control." The security measures in the command and control step are still effective because using Wi-Fi instead of Bluetooth for the communication between the OBD-II device and smartphone apps, using another dongle instead of ELM327, or using iOS instead of Android is in the "reconnaissance," "identification," or "vulnerability" steps. In the "Actions on Objectives" step, the attacks can be detected and defended using IDS. Research on this topic is left for future studies.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by the Korea Ministry of Land, Infrastructure and Transport. It was also supported by the Korea Agency for Infrastructure Technology Advancement (Project no. 17TLRP-B117133-02).

References

[1] A. Saad and U. Weinmann, "Automotive Software Engineering and Concepts," *GI Jahrestagung*, vol. 34, pp. 318-319, 2003.

[2] E. Nickel, "IBM Automotive Software Foundry," in *Proc. Conf. Comput. Sci. in Automot. Ind.*, Frankfurt University, Frankfurt, Germany, 2003.

[3] J. Greenough, "The Connected Car Report: Forecasts, Competing Technologies, and Leading Manufacturers," Business Insider, 2016, http://www.businessinsider.com/connected-car-forecasts-top-manufacturers-leading-car-makers-2015-3.

[4] S. Pandi, F. H. P. Fitzek, C. Lehmann et al., "Joint design of communication and control for connected cars in 5G communication systems," in *Proceedings of the 2016 IEEE Globecom Workshops, GC Wkshps 2016*, December 2016.

[5] J. Ni, X. Lin, and X. S. Shen, "Efficient and Secure Service-oriented Authentication Supporting Network Slicing for 5G-enabled IoT," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 644–657, 2018.

[6] 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 Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 993–1006, 2015.

[7] K. Koscher, A. Czeskis, F. Roesner et al., "Experimental Security Analysis of a Modern Automobile," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pp. 447–462, Oakland, CA, USA, May 2010.

[8] C. Miller and C. Valasek, "Demo: Adventures in Automotive Networks and Control Units," in *Proceedings of the DEFCON 2013*, 2013.

[9] J. H. Lee, S. Woo, S. Y. Lee, and D. H. Lee, "A Practical Attack on In-Vehicle Network Using Repackaging Android Applications," *Journal of the Korea Institute of Information Security and Cryptology*, vol. 26, no. 3, pp. 679–691, 2016.

[10] S. Karmar, "Drive It Like You Hacked It: New Attacks and Tools to Wirelessly Steal Cars," in *Proceedings of the DEFCON 23*, 2015.

[11] M. Kuzin and V. Chebyshev, "Hey Android, Where is My Car?" in *Proceedings of the RSA Conference 2017*, 2017.

[12] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *Proceedings of the 2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 528–533, Baden-Baden, Germany, June 2011.

[13] D. K. Nilsson, U. E. Larson, and E. Jonsson, "Creating a Secure Infrastructure for Wireless Diagnostics and Software Updates in Vehicles," in *Proc. Conf. Comput. Saf., Reliab., and Secur.*, pp. 207–220, Newcastle upon Tyne, UK, 2008.

[14] A. Tahat, A. Said, F. Jaouni, and W. Qadamani, "Android-based universal vehicle diagnostic and tracking system," in *Proceedings of the 2012 IEEE 16th International Symposium on Consumer Electronics (ISCE 2012)*, pp. 137–143, Harrisburg, Pa, USA, June 2012.

[15] The International Organization for Standardization (ISO), "The International Organization for Standardization (ISO)," The International Organization for Standardization standard 15765-4:2016, 2016.

[16] Society of Automotive Engineers (SAE) International, "Recommended Service Procedure for the Containment of Cfc-12," Society of Automotive Engineers standard J1989, 2011.

[17] S. Chen, J. Pan, and K. Lu, "Driving Behavior Analysis Based on Vehicle OBD Information and AdaBoost Algorithms," in *Proceedings of the in International MultiConference of Engineers and Computer Scientists (IMECS) 2015 Vol I*, pp. 102–106, 2015.

[18] C. Furmanczyk, D. Nufer, B. Sandona et al., *Integrating ODB-II, Android, and Google App Engine to Decrease Emissions and Improve Driving Habits*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.469.359&rep=rep1&type=pdf.

[19] Lockheed Martin Corporation, "Seven Ways to Apply the Cyber Kill Chain® with a Threat Intelligence Platform," white paper, 2015.

[20] Lockheed Martin Corporation, "Gaining the Advantage: Applying Cyber Kill Chain® Methodology to Network Defense," Guide, 2015.

[21] T. Hoppe and J. Dittman, "Sniffing/Replay Attacks on CAN Buses: A Simulated Attack on the Electric Window Lift Classified Using an Adapted CERT Taxonomy," in *Proc. Conf. on Embed. Syst*, 2007.

[22] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, 2011.

[23] 2017, https://sites.google.com/team-aegis.com/webpage/publish_data.

[24] F. Palmieri, U. Fiore, and A. Castiglione, "Automatic security assessment for next generation wireless mobile networks," *Mobile Information Systems – Emerging Wireless and Mobile Technologies*, vol. 7, no. 3, Article ID 404328, pp. 217–239, 2011.

[25] GitHub repository, 2017, https://github.com/song4jang/ODB-S-app_repackaging-.

[26] Y. Piao, J. Jung, and J. H. Yi, "Structural and functional analyses of proguard obfuscation tool," *The Journal of Korean Institute of Communications and Information Sciences*, vol. 38, no. 8, pp. 654–662, 2013.

[27] M. Wolf, A. Weimerskirch, and T. Wollinger, "State of the art: Embedding security in vehicles," *EURASIP Journal on Embedded Systems*, vol. 2017, 2007.

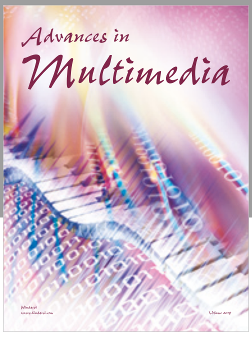
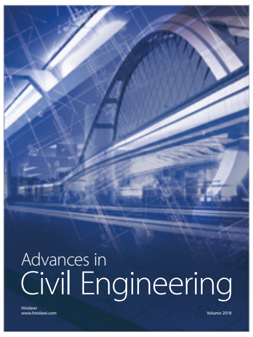
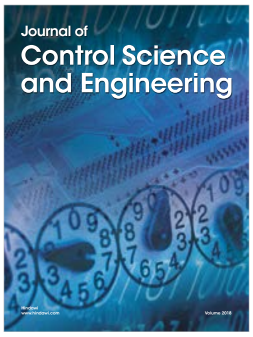
[28] T. Hoppe and J. Dittman, "Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy," in *Conf. Embedded Syst. Security*, pp. 1–6, 2007.

[29] D. K. Nilsson, U. E. Larson, and E. Jonsson, "Efficient In-Vehicle Delayed Data Authentication Based on Compound Message Authentication Codes," in *Proceedings of the 2008 IEEE 68th Vehicular Technology Conference (VTC 2008-Fall)*, pp. 1–5, Calgary, Canada, September 2008.

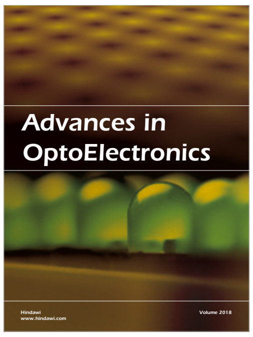
Journal of
Engineering

The Scientific World Journal

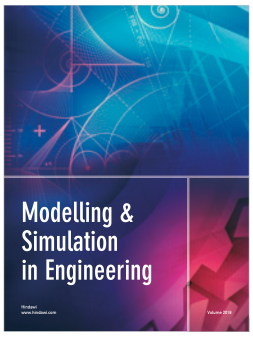
International Journal of
Rotating Machinery

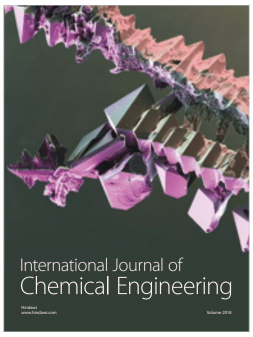
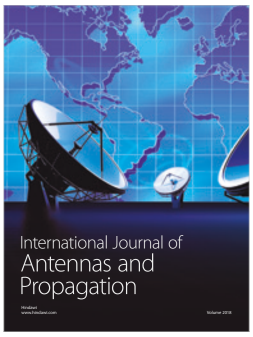
Journal of
Sensors

Advances in
Multimedia

Advances in
Civil Engineering

Journal of
Control Science and Engineering

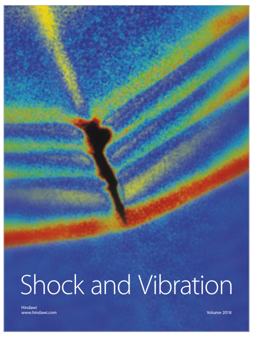
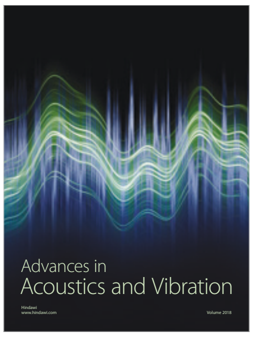
Journal of
Robotics

Journal of
Electrical and Computer Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and Observation

Modelling & Simulation in Engineering

International Journal of
Aerospace Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and Propagation

Active and Passive Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration