Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.6, Dec. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.6.1115

Non-Random CAN Fuzzing을 통한 효율적인 ECU 분석 기술*

김 형 훈,[1†] 정 연 선,[2] 최 원 석,[3] 조 효 진[4‡]
[1,4]숭실대학교 (대학원생, 교수), [2]한림대학교 (학생), [3]한성대학교 (교수)

An Efficient ECU Analysis Technology through Non-Random CAN Fuzzing*

Hyunghoon Kim,[1†] Yeonseon Jeong,[2] Wonsuk Choi,[3] Hyo Jin Jo[4‡]
[1,4]Soongsil University (Graduate student, Professor),
[2]Hallym University (Undergraduate student), [3]Hansung University (Professor)

요 약

최근 출시된 차량에는 다수의 ECU(Electronic Control Unit)가 탑재되어 있고, 각 ECU들은 CAN(Controller Area Network)을 통해 통신함으로써 차량을 효율적으로 제어할 수 있다. 하지만 CAN 통신에는 암호화 및 인증 기술이 적용되어 있지 않고, 접근 제어가 없는 Broadcast 방식으로 통신이 이루어지므로 보안에 취약하다는 문제점이 존재한다. 이러한 취약점을 이용하여 차량 제어 등의 수많은 차량 해킹 공격이 이루어지고 있으며 그에 대응하기 위한 연구 또한 진행되고 있다. 차량 해킹 대응 기술들 중에는 완성차에 탑재된 ECU의 취약점을 분석할 수 있는 CAN Fuzzing 기술이 존재한다. 하지만 기존의 CAN Fuzzing 기술들은 ECU들이 전송하는 CAN 메시지 구조를 고려하지 않고 Random한 방식으로 Fuzzing을 진행하기 때문에 많은 시간이 소요된다. 또한, 기존 CAN Fuzzing 기술은 Fuzzing 결과를 모니터링하는 방법에도 한계점이 존재한다. 이러한 CAN Fuzzing 기술의 한계를 해결하고자 본 논문에서는 CAN 메시지의 구조를 분석하고, 이를 바탕으로 ECU의 이상 작동 현상을 유발시킬 수 있는 Fuzzing 입력값을 생성하는 Non-Random CAN Fuzzing 기법을 제안한다. Non-Random CAN Fuzzing은 기존 Random CAN Fuzzing에 비해 소요되는 시간을 절약할 수 있고, 이를 통해 SW 구현 오류 혹은 CAN DBC(Database CAN) 설계 오류 등으로 인해 존재할 수 있는 ECU의 이상 작동 현상과 연관된 CAN 메시지들을 빠르게 발견할 수 있다. 제안하는 Non-Random CAN Fuzzing의 성능을 평가하기 위해 제안 기법을 실제 차량에 적용하였으며 ECU에 이상 작동 현상을 일으킬 수 있는 CAN 메시지를 확인하였다.

ABSTRACT

Modern vehicles are equipped with a number of ECUs(Electronic Control Units), and ECUs can control vehicles efficiently by communicating each other through CAN(Controller Area Network). However, CAN bus is known to be vulnerable to cyber attacks because of the lack of message authentication and message encryption, and access control. To find these security issues related to vehicle hacking, CAN Fuzzing methods, that analyze the vulnerabilities of ECUs, have been studied. In the existing CAN Fuzzing methods, fuzzing inputs are randomly generated without considering the structure of CAN messages transmitted by ECUs, which results in the non-negligible fuzzing time. In addition, the existing fuzzing

Received(09. 23. 2020), Modified(11. 30. 2020), Accepted(12. 01. 2020)
* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2018R1C1B5086261)
† 주저자, axolotl0210@gmail.com
‡ 교신저자, hyojin.jo@ssu.ac.kr(Corresponding author)

solutions have limitations in how to monitor fuzzing results. To deal with the limitations of CAN Fuzzing, in this paper, we propose a Non-Random CAN Fuzzing, which consider the structure of CAN messages and systematically generates fuzzing input values that can cause malfunctions to ECUs. The proposed Non-Random CAN Fuzzing takes less time than the existing CAN Fuzzing solutions, so it can quickly find CAN messages related to malfunctions of ECUs that could be originated from SW implementation errors or CAN DBC(Database CAN) design errors. We evaluated the performance of Non-Random CAN Fuzzing by conducting an experiment in a real vehicle, and proved that the proposed method can find CAN messages related to malfunctions faster than the existing fuzzing solutions.

Keywords: ECU, CAN, Vehicle Hacking, CAN Fuzzing, Non-Random CAN Fuzzing

I. 서 론

1970년 이후 모든 차량에는 ECU(Electronic Control Unit)가 탑재되어 출시된다. ECU란 엔진, 에어백, 브레이크 등 차량의 전자 시스템을 제어하는 역할로써, 최근 출시된 고급 차량에는 약 150여 개의 ECU가 탑재되어 있다. 차량 내 ECU는 CAN(Controller Area Network), FlexRay, LIN(Local Interconnect Network), CAN-FD(CAN with Flexible Data rate), Automotive Ethernet 등 다양한 내부 네트워크를 통해 통신한다[1]. 이 중, 차량의 특성을 잘 반영한 CAN을 1993년 국제 표준화 기구(ISO)에서 ISO 11898 표준 규격으로 채택하였으며 현재까지 차량 내부 네트워크로 사용되고 있다[2].

CAN을 통해 차량을 효율적으로 제어할 수 있게 되었지만 다음과 같은 특징으로 인해 보안에 취약하다는 문제점이 있다. 첫째, CAN 데이터는 암호화가 되어 있지 않은 상태에서 Broadcast 방식으로 통신하여 ECU의 송·수신되는 모든 데이터를 도청할 수 있다. 둘째, 네트워크에 대한 접근 제어가 없는 특징으로 인하여 악의적인 데이터가 주입되면 인증 절차 없이 데이터를 수신하기 때문에 차량에 오작동을 일으킬 수 있다[3]. 대표적인 차량 해킹으로는 2015년 C. Miller 등이 원격으로 Jeep Cherokee 차량의 CAN을 공격하여 엔진과 AVN(Audio Video Navigation) 등의 전자 시스템을 제어한 사례가 존재한다[4]. 위의 공격 사례에서 소개된 Bluetooth 혹은 Wi-Fi와 같은 무선 인터페이스를 이용하는 공격 이외에도 OBD(On-Board Diagnostics)-II 포트와 같은 유선 인터페이스를 이용한 공격 및 취약점 분석도 이루어지고 있다[5].

이러한 차량 공격에 대비하기 위해서는 완성차에서 제공되는 외부 네트워크 채널에 대한 취약점 점검뿐만 아니라 완성차에 탑재된 ECU에 대한 취약점 점검이 진행되어야 한다. 외부 네트워크 채널에 대한 취약점을 점검할 수 있는 기술로는 NMAP[6], ubertooth[7] 등의 기술을 통한 분석 방법이 존재하고, ECU에 대한 취약점 점검 방법으로는 ECU의 펌웨어를 분석하는 리버스 엔지니어링[8],[9]과 CAN 메시지를 통해 ECU의 오류를 분석하는 CAN Fuzzing 기술[10],[11],[12],[13],[14]이 존재한다.

하지만 활발하게 연구되고 있는 차량 외부 네트워크 취약점 점검 기술들에 비해 완성차에 탑재된 ECU에 대한 취약점 점검 기술들에 대한 기존 연구는 다음과 같은 한계점들을 가지고 있다. 첫째, ECU의 펌웨어를 리버스 엔지니어링하는 방법은 완성차에 탑재된 모든 ECU에 물리적으로 접근해서 디버깅 포트를 찾고 펌웨어를 획득하는 어려움 등의 문제가 존재한다. 둘째, CAN 통신을 통해 Random한 입력값을 보내는 CAN Fuzzing 연구는 Random한 입력값 전송으로 인해 방대한 시간이 소요된다는 문제점과 다수의 ECU가 통신 채널을 공유하는 CAN 환경으로 인해 특정 ECU의 반응을 모니터링하기 어려운 한계점이 존재한다.

기존 기술들의 한계점을 극복하기 위해 본 논문에서는 CAN 메시지 구조의 특성이 반영된 Non-Random CAN Fuzzing 기법을 제안한다. 제안하는 기술은 ECU의 펌웨어 획득 및 디버깅 포트를 통한 리버스 엔지니어링이 필요하지 않으며 네트워크 기반 방식의 Fuzzing 기술을 이용하고, 1) CAN 메시지 분석 단계, 2) CAN Fuzzing 입력값 생성 단계, 3) CAN Fuzzing 입력값 주입 및 결과 모니터링 단계의 총 3단계로 이루어져 있다.

제안하는 기술의 CAN 메시지 분석 단계에서는 차량에서 발생하는 모든 CAN 메시지들의 특성을 분석한다. CAN Fuzzing 입력값 생성 단계에서는 분석된 CAN 메시지들의 특성에 기반한 Rule을 정의하고, 해당 Rule에 기반한 Non-Random CAN

메시지들을 생성한다. 마지막으로 CAN Fuzzing 입력값 주입 및 결과 모니터링 단계에서는 CAN Fuzzing 입력값을 주입하고 이에 대한 차량 반응을 모니터링하여 ECU의 이상 작동 유무를 확인한다.

본 논문의 기여도는 아래와 같다.

1) CAN 메시지의 특성이 반영된 Rule 기반으로 CAN Fuzzing 입력값을 생성하고, Fuzzing 결과를 모니터링할 수 있는 방법론을 최초로 제시하였다.
2) CAN 메시지의 특성이 반영된 Non-Random CAN Fuzzing 기술을 제안함으로써, 기존 Random CAN Fuzzing 기술들에 비해 CAN Fuzzing에 소요되는 방대한 시간을 절약할 수 있다. 특히, 실험 차량의 Engine과 연관된 4개의 CAN ID를 기반으로 CAN Fuzzing 실험을 진행하였고, 기존 연구들의 Random CAN Fuzzing은 2^{66}의 Fuzzing 입력값이 테스트되어야 하지만, 제안하는 기술은 2^{7}개 수준의 Fuzzing 입력값 테스트가 필요함을 확인하였다.
3) 제안하는 Non-Random CAN Fuzzing 기술의 검증을 위해 해당 기술을 실제 차량에 적용하였고, 해당 실험 차량의 ECU에 이상 작동을 일으킬 수 있는 CAN 메시지(e.g., 차량 계기판, 조향 변화 등)를 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 연구에 필요한 배경 지식 및 관련 연구에 대해 소개하고, 3장에서는 Non-Random CAN Fuzzing에 대한 전체적인 시스템 구성도와 흐름에 대해 설명한다. 4장에서는 제안하는 기법을 실제 차량에 적용한 결과를 분석한다. 마지막으로 5장과 6장에서는 제안 기법에 대한 논의 및 결론을 다룬다.

II. 배경지식 및 관련 연구

2.1 배경지식

2.1.1 CAN

CAN(Controller Area Network)은 Robert Bosch에서 개발되었으며 1986년에 제정되었다. CAN은 구리선 두 줄이 한 쌍으로 꼬여져 있는 물리적인 구조로 전기적 잡음에 강한 특성을 지니고 있고, 정보 전송률은 최대 1Mbps를 지원하지만, 실제 차량에서는 주로 500Kbps를 사용한다. 정보를 한

SOF	ID	RTR	IDE	R	DLC	Data Field	CRC Field	ACK Field	EOF
1 bit	11 bit	1 bit	1 bit	1 bit	4 bit	0~8 byte	16 bit	2 bit	7 bit

Fig. 1. CAN Data Frame

프레임에 전송할 수 있는 최대 크기는 8byte이고, Bus형 네트워크 토폴로지에 기반하고 있다[15].

CAN Data Frame 구조는 Fig. 1.과 같다. SOF(Start Of Frame)는 데이터의 시작을 알린다. ID(IDentifier)는 데이터를 식별하고 전송되는 데이터의 우선순위를 결정하고, RTR(Remote Transmission Request)은 프레임의 종류를 판별한다. IDE(IDentifier Extension bit)는 프레임의 형식(표준, 확장)을 구분하고, R(Reserved)은 확장 프레임에 사용되기 위해 미리 지정된 부분이다. DLC(Data Length Code)는 Data Field의 크기를 나타내고, Data Field는 실질적인 데이터를 포함하고 있다. CRC(Cyclic Redundancy Check) Field는 프레임의 오류 여부를 확인하고, ACK(Acknowledgement) Field는 정상적인 데이터의 수신 여부를 판단한다. EOF(End Of Frame)는 데이터의 끝을 알린다. 본 논문에서는 CAN Data Frame의 ID와 Data Field를 중점적으로 다룬다.

2.1.2 OBD-II PID

OBD(On-Board Diagnostics)-II는 차량 시스템의 상태 정보를 파악하기 위해 사용되는 차량 진단 장치이다. PID(Parameter IDentifier)는 차량 시스템의 상태를 요청하는 코드이며 최근 출시된 모든 차량은 SAE J1962로 정의된 OBD-II PID 표준을 따른다[16]. 하지만 차량에서 표준에 따른 모든 PID를 지원하는 것은 아니며 차량 제조업체에 따라 추가적인 PID를 정의할 수 있다. 따라서 각 차량에서 지원되는 PID(Supported PIDs)를 파악하는 것이 중요하며 이는 PID 진단 쿼리와 응답을 통하여 확인이 가능하다. OBD-II 포트를 통해 연결된 차량의 CAN Bus에 CAN ID 0x7DF와 진단하고자 하는 PID Code를 이용하여 진단 쿼리를 요청하면 차량으로부터 0x7E8~0x7EF 범위의 CAN ID인 응답 메시지를 받는다. 응답 메시지에는 요청에 의한 반환값을 포함하고 있으며 이 값을 통해 차량의 상태와 Supported PIDs를 확인할 수 있다.

2.2 관련 연구

2.2.1 CAN 메시지 분석 기술

CAN DBC(Database CAN)는 공격자에게 노출될 경우, 차량의 오작동을 일으키는데 악용될 수 있기 때문에 차량 제조업체는 CAN DBC를 기밀 데이터로 유지한다. 하지만 CAN DBC는 CAN 메시지 분석(i.e., CAN 메시지 리버스 엔지니어링) 기술을 통해 유추될 수 있으며 이와 관련된 많은 연구들이 진행되고 있다.

ACA(Automated CAN Analyzer)[17],[18] 연구에서는 CAN 메시지 리버스 엔지니어링 기술에 많은 시간이 소요되는 문제를 해결하기 위해 CAN Data Frame 분석에 효율적인 도구를 개발하였다. ACA는 OBD-II PID[16] 표준의 PID 진단 쿼리를 이용하여 Supported PIDs를 파악하고, 그 중, 분석하고자 하는 PID를 입력하여 차량에 PID 진단 쿼리를 요청한다. 요청된 PID 진단 쿼리에 응답한 데이터와 CAN Bus에 흐르는 CAN 데이터 간의 분석을 통해 입력한 PID와 관련된 CAN ID와 Data Field를 도출한다.

READ(Reverse Engineering of Automotive Data Frames)[19] 연구에서는 CAN Data Frame에 포함된 신호값들을 자동적으로 분석하는 알고리즘을 제안하였다. 차량으로부터 수신한 모든 CAN 메시지를 각 CAN ID 별로 분류하여 CAN 메시지 리스트를 작성한다. CAN ID를 통해 분류된 CAN 메시지 리스트를 이용하여 특정 CAN ID의 Data Field(64bit)에서 각 bit 위치마다 bit 값이 변하는 정도를 나타내는 Bit Flip Rate를 계산한다. 계산된 Bit Flip Rate를 통해 모든 CAN ID의 Data Field에 인코딩 된 다양한 유형의 신호(Counter, CRC 등)를 식별하고 신호값의 경계를 추출한다. READ 알고리즘의 신호 분석 결과는 실험 차량의 실제 CAN DBC와 비교하였고 높은 수준의 정확성이 있음이 확인되었다.

LibreCAN[20] 연구에서는 OBD-II 프로토콜과 스마트폰에 내장된 IMU(Inertial Measurement Unit) 센서를 이용하여 차량의 Powertrain과 Body 부분의 CAN 메시지 신호를 해석하는 프레임워크를 제안하였다. LibreCAN은 READ[19] 알고리즘을 기반으로 한 메커니즘으로 CAN 데이터의 신호를 식별하고 분석한 후, 임의의 임계값을 기준으로 신호를 식별하고, 식별된 신호를 OBD-II PID[16]와 IMU 센서에 일치시켜 차량의 Powertrain 및 Body와 관련된 CAN 메시지에 대한 정보를 파악한다. LibreCAN에 의해 해석된 CAN 메시지 구조는 해당 연구팀이 소유한 CAN DBC 파일과의 비교를 통해 정확성이 증명하였다.

2.2.2 CAN Fuzzing 기술

ECU의 펌웨어 획득 및 디버깅 포트를 통한 리버스 엔지니어링이 요구되지 않는 CAN Fuzzing 기반 ECU 취약점 분석 기술들에 대한 연구가 진행되고 있다. 기존에 사용되고 있는 CAN Fuzzing 기술들은 주로 CAN ID와 Data Field를 고려하지 않고 Random한 값을 이용하여 Fuzzing을 진행한다[10],[11],[12],[13],[14].

따라서 CAN 메시지 분석 기술이 적용되지 않은 기존의 CAN Fuzzing 기술들은 64bit 크기의 CAN Data Field에 넣을 수 있는 모든 가능한 값(2^{64}개의 경우의 수)을 주입해야 하므로 많은 시간이 소요된다. 또한, CAN Fuzzing 결과 모니터링에 기술에 대한 연구가 부족한 상황이다.

III. 시스템 구성도

3.1 Overview

CAN 메시지 분석을 통한 Non-Random CAN Fuzzing에 대한 전체적인 시스템 구성은 Fig. 2.와 같으며 아래와 같이 총 3단계로 이루어져 있다.

1) CAN 메시지 분석 단계
- CAN ID-PID 관계 분석: OBD-II PID[16] 표준의 PID 진단 쿼리를 이용하여 Supported PIDs를 파악한 후, 차량의 특정 기능(e.g., Engine 등)과 연관 있는 CAN ID를 추론한다.
- CAN Data Field 분석: CAN Data Field의 Bit Flip Rate를 이용하여 Data Field에 존재하는 신호값의 특성 및 신호값 경계 범위를 추론한다.
- 상관관계 분석: CAN Data Field에 포함된 신호들의 상관관계 분석을 통해 서로 연관성이 있는 CAN ID를 식별한다.

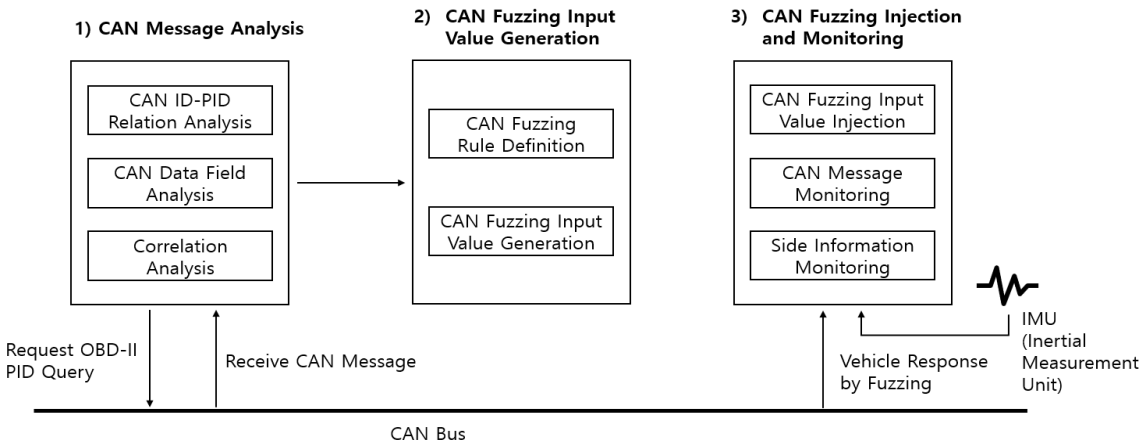

Fig. 2. Non-Random CAN Fuzzing System

2) CAN Fuzzing 입력값 생성 단계

- CAN Fuzzing Rule 정의: 각 CAN ID의 CAN Data Field 신호 특성에 기반한 Rule을 정의한다.
- CAN Fuzzing 입력값 생성: CAN Fuzzing Rule에 기반한 입력값을 생성하고, Checksum 값이 필요할 경우, Deep Learning 기술을 통해 해당 값을 유추한다.

3) CAN Fuzzing 입력값 주입 및 결과 모니터링 단계

- CAN Fuzzing 입력값 주입: CAN에서 발생하는 오류(e.g., CAN 메시지 주입으로 인한 Bus-off[21] 등)를 고려한 CAN Fuzzing 입력값 주입을 수행한다.
- CAN 메시지 모니터링: CAN Fuzzing 입력값 주입 후, CAN Data Field의 신호 특성을 위반한 상황에 대해 모니터링한다.
- Side Information 모니터링: CAN Fuzzing 입력값 주입 후, Side Information(e.g., IMU(Inertial Measurement Unit))을 이용하여 차량의 이상 작동 현상을 모니터링한다.

각 단계에 대한 자세한 설명은 다음과 같다.

3.2 CAN 메시지 분석 단계

3.2.1 CAN ID-PID 관계 분석

1) Supported PIDs 파악

Fuzzing Target 차량에서 제공되는 PID를 파악하기 위해서는 ACA[17],[18]에서 사용한 방법인 Supported PIDs 기능을 활용해야 한다. OBD-II 포트를 통해 연결된 CAN에 Supported PIDs 파악을 위한 OBD-II PID[16] 진단 쿼리를 요청하고, 반환값을 응답받는 방식을 통하여 Fuzzing Target 차량에서 제공되는 Supported PIDs를 파악할 수 있다.

즉, Fig. 3.과 같이 CAN ID를 0x7DF로 설정하고 CAN Data Field의 3번째 byte 위치에 입력되는 PID Code 부분에 Supported PIDs를 요청하는 PID(0x00, 0x20, 0x40, 0x60, 0x80, 0xA0, 0xC0)를 입력하여 OBD-II Request를 전송한 후, 해당 값에 대한 OBD-II Response를 통해 각 PID는 32개 범위의 PIDs를 확인할 수 있다.

예를 들어 OBD-II Request에 PID Code 0x20을 입력하여 진단 쿼리를 요청할 경우, OBD-II Response의 Data Field에는 PID Code가 포함된 위치(Data Field의 3번째 위치) 바로 뒤에 4byte 크기의 반환값을 포함하고 있다. 반환값을 Binary로 변환한 후, 각 PID Code가 지원하는 32개의 PIDs와 비교하였을 때, 각 bit 위치의 1에 해당하는 부분이 해당 차량에서 지원되는 Supported PID이고, Fig. 3.의 예시에서는 0x21, 0x23, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0x34, 0x3C가 지원됨을 알 수 있다. 이러한 방법으로 모든 PID Code를 이용하여 차량의 전체 Supported PIDs를 파악한다.

2) CAN ID-PID 관계 추론

Fuzzing Target 차량과 관련된 CAN ID와 PID의 관계를 분석하기 위해서 기존에 연구된 방법[17],[18],[19]을 이용한다. Fuzzing Target 차량에서 제공하는 PID를 이용하여 OBD-II PID[16] 진단 쿼리를 주기적으로 요청한 후, 응답받은 CAN 메시지를 기준으로 $\pm n$초 시간 범위의 Time Window를 지정한다.

이렇게 지정된 Time Window 내에서 OBD-II PID[16] 진단 쿼리에 대한 반환값과 동일한 값을 소유하고 있는 CAN ID 및 해당 Data Field의 위치를 카운팅한다. 해당 실험을 반복적으로 진행하여 모든 Time Window 내에서 카운팅 된 횟수가 가장 많은 CAN ID를 진단 쿼리에 사용된 PID와 높은 연관성을 가지고 있다고 판단한다. 따라서 해당 CAN ID들이 PID 진단 코드에 명시된 차량의 기능과 연관되어 있을 것이라고 추론한다. 알고리즘에 대한 자세한 사항은 [17],[18],[19]을 참고한다.

3.2.2 CAN Data Field 분석

각 CAN ID의 Data Field의 신호를 분석하는 연구[19],[20]가 활발히 진행 중이며 주로 Bit Flip Rate 기술을 이용한다. Bit Flip Rate란 bit가 변하는 정도(0↔1)를 의미하며 구하는 과정은 식 (1)과 같다.

$$BFR_{c,k} = \frac{1}{n_c}\sum_{i=2}^{n_c} b_{c,k,i} \qquad (b_{c,k,1} = 0) \qquad (1)$$

- $BFR_{c,k}$: 임의의 CAN ID (c)의 Data Field에서 k번째 bit 위치에 대한 Bit Flip Rate
- n_c : $BFR_{c,k}$ 값 계산을 위해 수집한 CAN Log 데이터 중, CAN ID (c)를 가지는 CAN 메시지의 수
- $b_{c,k,i}$: CAN ID (c)를 가지는 n_c개의 CAN 메시지 중, i번째 CAN 메시지의 Data Field에서 k번째 bit와 $i-1$번째 CAN 메시지의 Data Field에서 k번째 bit 값이 다르면 1, 같으면 0

제안하는 CAN Data Field 분석 방법에서는 (1)번 식을 통해 얻어진 Bit Flip Rate의 특징에 따라 신호를 4bit, 8bit, 16bit 단위로 나누어 분류하였다. 분류된 신호값 Field는 Unused, Constant, Multi-Value, Sensor, Counter, Checksum 그리고 Undefined Field로 7가지이며 분류 기준은 다음과 같다.

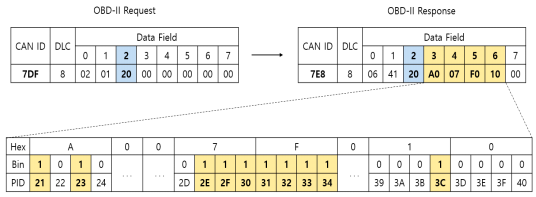

Fig. 3. OBD-II PID Diagnostic Query Request and Response

- Unused Field: 64bit의 Data Field에서 임의의 4bit 값이 0x0으로 고정되어 있고, Bit Flip Rate가 0인 필드
- Constant Field: 64bit의 Data Field에서 임의의 4bit 값이 0x0이 아닌 특정 값으로 고정되어 있고, Bit Flip Rate가 0인 필드
- Multi-Value Field: 64bit의 Data Field에서 임의의 4bit 값이 몇 개의 상수 값으로만 구성되는 특징을 지니고 있으며 Bit Flip Rate가 0에 수렴한 필드
- Sensor Field: 64bit의 Data Field에서 임의의 4bit, 8bit, 16bit 값이 LSB(Least Significant Bit)에서 MSB(Most Significant Bit)로 이동할수록 Bit Flip Rate가 감소하는 추세를 보이는 필드(일반적으로 센서값과 같이 특정 물리량(Physical Value)에 대한 수치를 나타내는 Field는 LSB 근처의 bit position 일수록 해당 위치에 저장된 bit 값이 자주 변동되므로 Bit Flip Rate가 높고, MSB 근처의 bit position 일수록 Bit Flip Rate가 작은 값을 가진다. 따라서 Bit Flip Rate가 감소하는 추세를 나타내는 Field는 차량 내의 센서에서 발생된 물리량(Physical Value)과 연관 되었을 것으로 추론할 수 있다.)
- Counter Field: 64bit의 Data Field에서 임의의 4bit, 8bit, 16bit 값이 LSB에서 MSB로 이동할수록 Bit Flip Rate가 1부터 1/2씩 감소하는 필드(Sensor Field와 달리 Counter Field는 해당 위치에 저장된 값을 1씩 순차적으로 증가하는 특징을 보이므로 LSB에서 MSB로

1bit씩 이동할 경우, Bit Flip Rate가 1/2씩 감소한다.)
- Checksum Field: 64bit의 Data Field에서 임의의 4bit, 8bit, 16bit 값의 Bit Flip Rate가 0.3~0.6 사이의 값으로 고르게 분포한 필드
- Undefined Field: 위의 6가지 필드로 분류되지 않는 필드

3.2.3 상관관계 분석

임의의 CAN ID와 연관성 있는 CAN ID를 분류하기 위해서 각 CAN ID의 Data Field를 이용하여 상관관계 분석을 진행한다. 차량 내의 ECU들은 차량의 특정 기능 및 상태 공유를 위해 서로 연관성이 있는 값들을 CAN 메시지를 통해 주고 받는다. 따라서 Data Field에서 분류된 Sensor Field 값을 이용하여 각 값들 간의 상관관계를 분석하여 CAN ID들 간의 상관관계를 유추한다.

3.3 CAN Fuzzing 입력값 생성 단계

3.3.1 Non-Random CAN Fuzzing Rule 정의

Fuzzing Target 차량에서 SW 구현 오류 혹은 CAN DBC 설계 오류 등으로 인해 존재할 수 있는 이상 작동 현상을 찾기 위해 CAN Fuzzing 입력값 생성 Rule을 정의한다. 해당 Rule은 이상 작동 현상이 유도될 수 있도록 정의하며 Value, Sensor, Field Violation으로 총 3개로 구성된다.

- Value Violation: Unused, Constant 그리고 Multi-Value Field에서 나타나지 않았던 임의의 값을 입력하여 CAN Fuzzing 입력값을 생성한다. 각 Field에서 입력되는 경우의 수를 줄이면서 차량 내에 존재할 수 있는 오류에 대한 빠른 Scanning을 위해서 입력되는 값의 크기를 순차적으로 증가시키는 것이 아닌 2^k 단위로 증가시키면서 입력값을 생성한다. 즉, 각 Field 값의 최댓값보다 큰 값이 나오기 전까지 k값을 1씩 증가시키면서 Value Violation을 일으킬 수 있는 CAN Fuzzing 입력값을 생성한다.
- Sensor Violation: Sensor Field의 최댓값보다 큰 값 또는 최솟값보다 작은 값을 입력하여 CAN Fuzzing 입력값을 생성한다.
- Field Violation: Undefined Field를 제외한 6가지 Field들의 특성이 무시된 CAN Fuzzing 입력값에 대한 ECU 반응을 보기 위해서 Data Field의 모든 byte 위치에 임의의 같은 값을 입력한 CAN Fuzzing 입력값을 생성한다. Data Field의 모든 byte 위치에 입력되는 경우의 수를 줄이면서 차량 내에 존재할 수 있는 오류에 대한 빠른 Scanning을 위해서 입력되는 값의 크기를 순차적으로 증가시키는 것이 아닌 2^k 단위로 증가시키면서 입력을 진행한다. 즉, k값을 0부터 7까지 1씩 증가시키면서 Field Violation을 일으킬 수 있는 CAN Fuzzing 입력값을 생성한다.

3.3.2 CAN Fuzzing 입력값 생성

1) Value Violation 및 Sensor Violation 입력값 생성

Value Violation 및 Sensor Violation Rule에 의해 CAN Fuzzing 입력값을 생성하는 과정에서 Fuzzing 입력값의 Data Field 내에 Checksum Field가 존재하는 경우, 생성된 입력값에 대한 Checksum을 계산해야 한다. [8]에 따르면 Checksum을 고려하지 않은 CAN 메시지가 차량에 주입되게 될 경우, Checksum 검증 오류가 발생하여 해당 CAN 메시지가 누락될 수도 있다. 또한, Checksum 값을 생성할 때, 차량 제조업체에 의해 ECU에 주입된 비밀키가 사용되는 경우도 존재한다[8].

따라서 Non-Random CAN Fuzzing 입력값 생성 모듈에서는 Deep Learning 기술을 이용하여 ECU에 대한 리버스 엔지니어링 없이 CAN Fuzzing 입력값에 대한 Checksum 값을 유추한다. 제안하는 Checksum 유추 과정은 데이터셋을 구축하는 전처리 단계와 Deep Learning 알고리즘을 이용하여 모델을 생성하는 단계로 구성된다.

- Checksum 유추를 위한 전처리 단계

Deep Learning에 사용할 데이터셋을 구축하기 위해 Data Field에서 Checksum Field로 분류된 구간을 출력 데이터로 지정하고, 그 외의 나머지 구간을 입력 데이터로 지정한다. 입력 데이터는 Binary로 변환하고, 출력 데이터는 Checksum Field가 표현할 수 있는 값의 범위에서 Checksum

값의 위치에 맞게 One-hot Encoding 한다. 예를 들어 Checksum Field가 n-bit일 경우, 입력 데이터는 크기가 $(64-n)$인 배열을 가지고, 출력 데이터는 크기가 2^n인 배열로 구성되어 있다. 전처리 과정은 Fig. 4.와 같다.

- Checksum 유추를 위한 모델 생성 단계

Deep Learning의 대표적인 알고리즘인 DNN(Deep Neural Network)[22]을 이용하여 Sequential 모델을 생성한다. 예를 들어 Checksum Field가 n-bit일 경우, 입력층(Input Layer)은 $(64-n)$개의 노드로 구성되며 Binary로 변환된 입력 데이터를 주입한다. 출력층(Output Layer)은 2^n개의 노드로 구성되며 One-hot Encoding 된 출력 데이터를 주입한다. 은닉층(Hidden Layer)은 Fully-Connected Layer로 구성되며 모델의 구조는 Fig. 5.와 같다.

Checksum Field 유추에 대한 정확성을 검증하기 위해 F1-score를 계산하여 모델의 성능을 확인한다. 모델의 정확도에 대한 실험 및 평가는 4장에서 자세히 기술한다.

2) Field Violation 입력값 생성

Field Violation Rule에 의해 CAN Fuzzing 입력값을 생성하는 과정에서는 Field들의 특성이 무시된 CAN Fuzzing 입력값을 생성하므로 Deep Learning 기반 Checksum 유추 과정이 필요하지 않다. 따라서 해당 Rule에 의거하여 CAN Fuzzing 입력값을 순차적으로 생성한다.

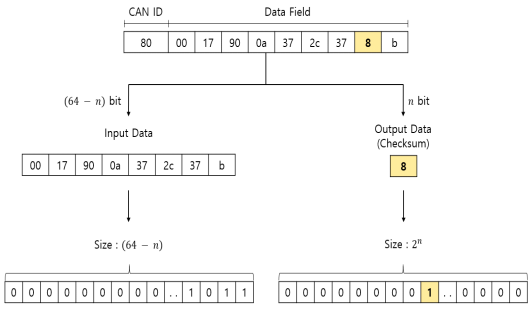

Fig. 4. Pre-processing steps for Checksum Estimation

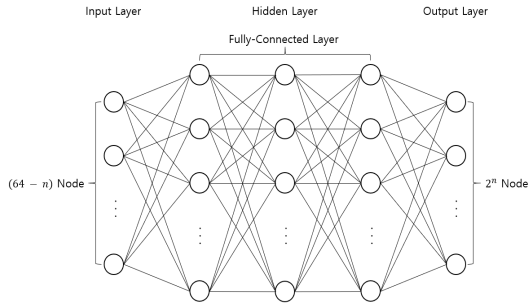

Fig. 5. DNN Classification Model for Checksum Estimation

3.4 CAN Fuzzing 입력값 주입 및 결과 모니터링 단계

3.4.1 CAN Fuzzing 입력값 주입

본 논문에서는 정의된 CAN Fuzzing 입력값 생성 Rule에 따라 생성된 CAN 메시지를 Fuzzing Target 차량의 CAN에 주입한다. 만약 CAN Fuzzing 진행 중, 실제 차량에서 생성되는 CAN 메시지와 CAN Fuzzing 입력값 충돌로 인해 CAN Fuzzing을 수행하는 장비의 TEC(Transmission Error Counter)가 임계치(e.g. 255)를 넘어 Bus-off[21]가 발생할 경우, 해당 장비의 CAN 컨트롤러를 Reset하여 Fuzzing을 다시 진행한다.

3.4.2 CAN 메시지 모니터링

CAN Fuzzing으로 인해 ECU에서 발생할 수 있는 이상 작동 현상을 CAN 메시지를 이용하여 모니터링하기 위해 다음과 같은 값의 변화를 확인한다.

- CAN ID Monitoring : 새로운 CAN ID가 발생했는지 확인한다.
- CAN DLC Monitoring : CAN ID의 DLC가 변했는지 확인한다.
- Value Violation Monitoring : Unused, Constant 그리고 Multi-Value Field에 계산된 Bit Flip Rate에서 벗어나는 다양한 값들이 나타나는지 확인한다.
- Sensor Violation Monitoring : Sensor Field 데이터가 기존 최댓값과 최솟값의 범위를 벗어났는지 확인한다.

- Correlation Monitoring : 각 CAN ID의 Data Field에 존재하는 Sensor Field 간의 상관관계가 임계치 이상 변화되었는지 확인한다.

3.4.3 Side Information 모니터링

CAN Fuzzing에 의한 차량의 이상 작동 현상을 측정하기 위해 IMU 센서를 이용한다. IMU 센서는 관성 측정 장치로 자이로와 가속도 값을 측정하는 장치이다. x, y, z축을 기준으로, 자이로는 방향을 감지하고 가속도는 속도 변화를 측정한다. 차량에 Fuzzing 입력값을 주입하지 않은 상태의 센서값과 Fuzzing을 진행하는 동안의 센서값을 비교하여 차량에서 이상 작동 현상이 발생하였는지 확인한다.

IV. 실험 및 분석

본 논문에서 제안한 Non-Random CAN Fuzzing 기술의 검증을 위한 실험 환경은 Fig. 6.과 같다. 라즈베리 파이와 PiCAN[23] CAN Bus Board를 결합한 후, 차량의 OBD-II 포트를 통해 CAN 메시지를 송·수신하였다. 추가적으로 Side Information 수집을 위해 라즈베리 파이와 IMU 센서인 MPU-6050을 결합하여 차량의 자이로와 가속도의 값을 측정하였다. 실험 차량은 All new Soul 2014를 이용하였다.

4.1 CAN 메시지 분석 실험

4.1.1 Supported PIDs 확인

실험 차량에 OBD-II PID[16] 진단 쿼리를 요청함으로써 Supported PIDs를 확인하였다. 총 37개의 PIDs를 지원하고 있으며 결과는 Table 1.과 같다.

4.1.2 CAN ID-PID 관계 확인

실험 차량의 CAN ID와 PID의 관계를 분석하기 위해 기존 연구[17],[18]에 기반한 실험을 진행하였다. 본 실험에서는 차량의 Engine과 연관된 PID를 선택한 후, 해당 PID와 관련된 CAN ID를 파악하였다.

실험 결과 Table 2.와 같이, PID Code 0x04(Calculated Engine Load)와 0x05(Engine Coolant Temperature)는 CAN ID 0xA0와 관련이 있고, PID Code 0x0C(Engine RPM)는 CAN ID 0x80, 0xA0, 0x316과 연관이 있음을 확인되었다. 또한, PID Code 0x0D(Vehicle Speed)는 0xA0, 0x316, 0x440과 연관되어 있음을 확인하였다.

Fig. 6. Experiment environment for Non-Random CAN Fuzzing System

실험을 통해 파악된 CAN ID-PID 관계에 대한 정확도 검증을 위해 CAN DBC에 대한 공개적인 리소스 파일[24]과 비교를 하였고, Table 2.를 보는 바와 같이 CAN ID 0x440의 Data Field 7번째 byte를 제외한 다른 CAN ID들은 모두 정확히 분석되었음을 확인하였다.

4.1.3 CAN Data Field 분류

각 CAN ID의 Data Field의 신호를 분석하기 위해 운전(Driving) 중인 상태에서 실험 차량의 CAN Log를 확보하였다.

확보된 CAN Log를 통해, Engine과 관련된 각 CAN ID의 Data Field에 대해 Bit Flip Rate를 계산하였고, 총 7가지의 신호가 Fig. 7.과 같이 분

Table 1. Supported PIDs of All new Soul 2014

PID (Hex)	Supported PIDs of Soul (Hex)
00	01, 03, 04, 05, 06, 07, 0B, 0C, 0D, 0E, 0F, 11, 13, 15, 1C, 1F
20	21, 23, 2E, 2F, 30, 31, 32, 33, 34, 3C
40	41, 42, 43, 44, 45, 46, 47, 49, 4A, 4C, 56

Table 2. Relation between PID code and CAN ID of All new Soul 2014
(√: CAN signals analyzed by PID Code requests, Red Tagging: CAN signals provided by [24])

Vehicle	Description (PID)	CAN ID (Frequency)	Data Field							
			0	1	2	3	4	5	6	7
Soul	Calculated Engine Load (04)	0xA0 (100ms)	√							
	Engine Coolant Temperature (05)	0xA0 (100ms)		√						
	Engine RPM (0C)	0x80 (10ms)			√	√				
		0xA0 (100ms)			√	√				
		0x316 (10ms)			√	√				
	Vehicle Speed (0D)	0xA0 (100ms)					√			
		0x316 (10ms)							√	
		0x440 (10ms)			√					

Table 3. Sensor Field Value-based CAN ID Correlation

Vehicle	Description (PID)	CAN ID (Offset)	CAN ID with High Correlation (Offset)
Soul	Calculated Engine Load (04)	0xA0(0)	0xA0(5), 0xA1(1, 4)
	Engine Coolant Temperature (05)	0xA0(1)	0xA0(4), 0x18F(1)
	Engine RPM (0C)	0x80(2)	0x316(2)
		0x80(3)	0x80(0), 0x260(2, 5), 0x316(3, 6), 0x329(6), 0x43F(6), 0x440(2, 6), 0x545(1)
		0xA0(2)	X
		0xA0(3)	X
		0x316(2)	0x80(2)
		0x316(3)	0x80(0, 3), 0x260(2, 5), 0x316(6), 0x329(6), 0x43F(6), 0x440(2, 6), 0x545(1)
	Vehicle Speed (0D)	0xA0(4)	18F(1), 0xA0(1)
		0x316(6)	0x18F(5), 0x370(2), 0x43F(6), 0x440(2, 6)
		0x440(2)	0x18F(5), 0x316(6), 0x43F(6), 0x440(6)

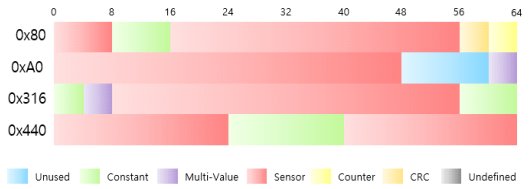

Fig. 7. Classification of CAN Data Field (CAN IDs related to Engine)

류됨을 확인하였다.

4.1.4 CAN 메시지 간의 상관관계 분석

CAN 메시지 간의 상관관계 분석을 위해서는 각 CAN ID의 데이터에서 Sensor Field로 식별된 값들의 상관관계를 이용해야 한다. 본 실험에서는 Engine과 관련된 CAN ID들의 Data Field에서 분류된 Sensor Field와 그 외 CAN ID 간의 상관관계를 계산한 후, 상관관계 수치가 0.7 이상이거나 -0.7 이하인 CAN ID와 Data Field를 추출하였다. Table 3.을 보는 바와 같이 각 CAN ID의 Data Field에 대해 높은 상관관계를 가지는 다양한 CAN ID와 Data Field를 분석하였으며 해당 CAN ID 모두 Engine과 연관이 있다고 추론하였다.

4.2 CAN Fuzzing 입력값 생성 실험

4.2.1 입력값 생성 및 Checksum 추론

본 실험에서는 Engine과 관련된 CAN ID들에 대해 CAN Fuzzing 입력값을 생성하였다. 입력값 생성에는 3.3.1절에 정의된 CAN Fuzzing Rule인 Value Violation, Sensor Violation, Field Violation을 이용하였다.

Value와 Sensor Violation에 기반한 CAN Fuzzing 입력값 생성 과정에서 CAN ID 0x80의 Data Field에는 Checksum Field가 존재하므로

생성한 입력값에 대한 Checksum을 유추하는 과정을 진행하였다.

CAN ID 0x80에 대한 Checksum Field는 4bit로 식별되었으므로, 크기가 60인 배열에 Checksum 이외의 값들이 Binary로 변환된 데이터를 포함하는 입력 데이터와 크기가 2^4인 배열에 Checksum 값의 위치에 맞게 One-hot Encoding 된 데이터를 포함하는 출력 데이터를 생성하였다. Checksum을 유추하기 위해 60개의 노드로 구성된 입력층, 120개의 노드와 3개의 층으로 구성된 은닉층, 16개의 노드로 구성된 출력층으로 이루어진 DNN 모델을 생성하였다. 또한, Optimizer는 Adam을 사용하였으며 Loss Function은 categorical_crossentropy를 사용하였다.

CAN ID 0x80에 대한 메시지 중, 90%는 훈련 데이터, 10%는 테스트 데이터로 나누어 학습 및 평가를 진행하였다. 모델 성능 평가 결과, 정확성은 98%로 Fig. 8.과 같으며 정밀도, 재현율 그리고 F1-score 모두 98%로 Table 4.와 같다.

- 정밀도(Precision)

$$= \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- 재현율(Recall)

$$= \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- F1-score

$$= 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Engine과 연관된 CAN ID들 중, CAN ID 0x80 이외에는 Data Field 내에 Checksum Field가 존재하지 않으므로 Checksum을 고려하지 않고 CAN Fuzzing 입력값을 생성하였다.

4.3 CAN Fuzzing 주입 및 결과 모니터링 실험

CAN Fuzzing Rule 기반으로 생성된 입력값들을 주입하였고, 해당 입력값으로 인해 발생할 수 있는 ECU의 이상 반응을 CAN 메시지와 Side Information 모니터링을 통해 확인하였다.

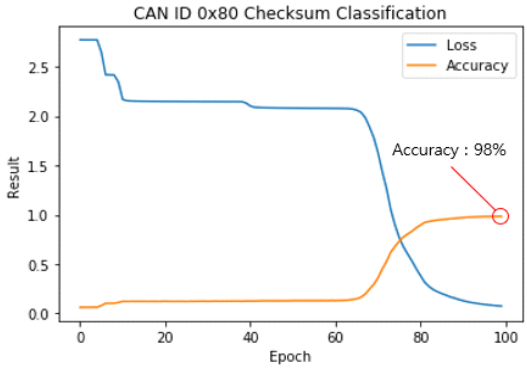

Fig. 8. Checksum Estimation for CAN ID 0x80

Table 4. Evaluation Metrics for DNN Model

CAN ID (Hex)	Precision	Recall	F1-score
80	98%	98%	98%

4.3.1 CAN 메시지 모니터링

CAN Fuzzing 입력값에 대한 ECU의 이상 반응이 CAN 메시지에 반영될 수 있으므로, 이를 분석하기 위해 아래와 같은 CAN 메시지 모니터링을 진행하였다.

- CAN ID Monitoring: 0x4, 0x40, 0x100인 새로운 CAN ID가 추가되었지만, CAN Bus Board가 Bus-off[21] 되어 컨트롤러를 Reset하는 상황에서 발생되는 것으로 확인하였다[25].
- CAN DLC Monitoring: 변화가 있는 DLC는 존재하지 않았다.
- Value Monitoring: Unused, Constant 그리고 Multi-Value Field의 값이 변하는 것은 존재하지 않았다.
- Sensor Monitoring: Sensor Field의 최댓값과 최솟값을 벗어나는 것은 존재하지 않았다.
- Correlation Monitoring: 기존에 서로 높은 상관관계를 가지는 CAN ID 간의 상관관계가 유지되거나 깨지는 것을 확인하였다.

4.3.2 Side Information 모니터링

CAN Fuzzing 입력값을 주입하지 않은 상태의 센서값(Normal)과 주입하는 동안의 센서값

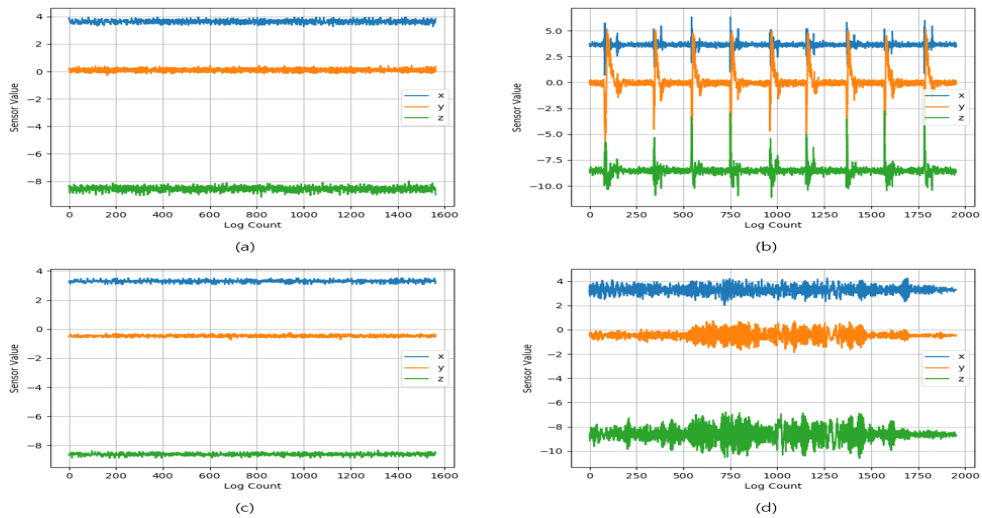

Fig. 9. Side Information Monitoring Results. (a) CAN ID 0x164 Normal. (b) CAN ID 0x164 Value Violation Fuzzing. (c) CAN ID 0x370 Normal. (d) CAN ID 0x370 Field Violation Fuzzing

Fig. 10. Vehicle Malfunction due to CAN Fuzzing. (a) RPM Instrument Panel. (b) Movement of Steering

(Violation)을 기록하여 두 값의 변화를 비교하였다. 그래프로 표현한 결과, 진폭의 변화가 있는 CAN ID들을 발견하였으며 결과는 Fig. 9.와 같다.

Fig. 9.의 그래프에서 파란색, 주황색, 초록색은 각각 IMU 센서의 x, y, z축을 의미하며 상단 그래프 (a), (b)는 CAN ID 0x164, 하단 그래프 (c), (d)는 0x370에 대해 나타낸다. (a)와 (b)를 비교하였을 때, (b)에서 주기적으로 진폭의 변화가 있는 것을 볼 수 있다. 이는 Fig. 10.의 (b)와 같이 차량 조향의 주기적인 움직임으로 인해 센서값이 변한 것을 확인하였고, 조향이 오른쪽 방향으로 5~10° 움직임에 따라 차량 바퀴도 움직이는 것을 확인하였다. (c)와 (d)를 비교하였을 때, (d)에서 특정 구간에서 진폭이 커지는 것을 볼 수 있다. 이는 차량 Engine이 가속되어 차량의 떨림으로 인해 센서값이 변하는 것을 확인하였고, Fig. 10.의 (a)와 같이 차량 계기판의 RPM이 변하는 것을 확인하였다. 추가적으로

Table 5. Number of cases for Input Value of CAN Fuzzing

	Random CAN Fuzzing [10][11][12][13][14]				Non-Random CAN Fuzzing			
	0x80	0xA0	0x316	0x440	0x80	0xA0	0x316	0x440
Value Violation	2^{64}				8	16	16	16
Sensor Violation					12			
Field Violation					8			
Total Number of cases	2^{66}				$8+(16\times3)+(12\times4)+(8\times4)\approx2^7$			

CAN ID 0x316에 대한 CAN Fuzzing 입력값을 주입하였을 때도 차량 계기판의 RPM이 변하는 것을 확인하였다.

4.4 CAN Fuzzing 입력값 경우의 수

제안하는 기법인 Non-Random CAN Fuzzing과 기존 기술인 Random CAN Fuzzing 기술 [10],[11],[12],[13],[14]의 패킷 주입 횟수를 비교하고, 결과는 Table 5.와 같다. 정확한 비교를 위해 비교 범위는 실험 차량에서 파악된 Engine 관련 CAN ID인 0x80, 0xA0, 0x316, 0x440을 비교 대상으로 한정하였고, 각 CAN ID 별로 필요한 CAN Fuzzing 입력값 경우의 수를 아래와 같은 과정을 통해 계산하였다.

- Value Violation 경우의 수: Value Violation과 관련 있는 Unused, Constant 그리고 Multi-Value Field(4bit)는 각 Field에 2^k 단위로 k값을 0부터 4까지 1씩 증가시키며 입력하므로, 각 Field 입력값에 대한 경우의 수는 4개이다. 만약 CAN Fuzzing 대상 CAN ID의 Data Field에 존재하는 Unused, Constant, Multi-Value Field의 수가 n_1일 경우, 총 $(4\times n_1)$개의 입력값 경우의 수가 존재한다. 따라서 CAN ID 0x80, 0xA0, 0x316, 0x440은 Unused, Constant, Multi-Value Field의 총 개수가 각각 2개, 4개, 4개, 4개이므로, 8, 16, 16, 16의 입력값 경우의 수가 존재한다.
- Sensor Violation 경우의 수: Sensor Violation과 관련 있는 Sensor Field(8bit)에 최댓값과 최솟값의 범위를 벗어난 값을 입력하므로 입력값의 경우의 수는 2개이다. 만약 CAN Fuzzing 대상 CAN ID의 Data Field에 존재하는 Sensor Field의 수가 n_2일 경우, 총 $(2\times n_2)$개의 입력값 경우의 수가 존재한다. 따라서 CAN ID 0x80, 0xA0, 0x316, 0x440은 Sensor Field의 총 개수가 모두 6개이므로, 각각 12개의 경우의 수가 존재한다.
- Field Violation 경우의 수: 64bit의 Data Field를 8bit씩 8개의 Field로 나누고, 각 Field마다 2^k 단위로 k값을 0부터 7까지 1씩 증가시키며 입력값을 생성한다. 즉, 8개의 Field는 k값에 따라 모두 동일한 값을 가지게 된다. 따라서 CAN ID 0x80, 0xA0, 0x316, 0x440은 각 8개의 경우의 수가 존재한다.

Table 5.의 결과에서 보듯이, Non-Random CAN Fuzzing 입력값은 약 2^7개의 경우의 수가 존재하며, 이는 4개의 CAN ID에 대해 각 Violation의 경우의 수를 총합한 값이다. Value Violation과 Sensor Violation의 입력값은 해당 Data Field에 대해 값을 생성하게 되고, Field Violation의 입력값은 Data Field의 모든 Field에 대해 값을 생성하게 된다. 즉, 각 Violation에서 생성된 입력값은 모두 다르다. 최종적으로, Non-Random CAN Fuzzing 입력값의 경우의 수는 Random CAN Fuzzing에서 요구되는 경우의 수 2^{66}에 비해 매우 적음을 확인하였다.

V. 논 의

5.1 CAN Fuzzing 기술 확장

5.1.1 조건적 CAN Fuzzing 분석 범위 확대

제안하는 Non-Random CAN Fuzzing 진행 중, 특정 CAN 메시지의 Data Field 내에 특정 위치에서 차량의 이상 작동 현상이 발견되면, 해당 위

치에 모든 경우의 수가 반영된 입력값을 주입함으로써, 테스트 범위를 늘릴 수 있다. 즉, 특정 CAN 메시지에 대한 세부적인 분석이 필요할 경우, 특정 조건에 따라 Non-Random CAN Fuzzing을 Random CAN Fuzzing으로 전환이 가능하다.

5.1.2 Side Information 확장

본 논문에서는 IMU 센서만을 이용하여 Side Information을 파악하였다. IMU 센서 이외에도 차량 내부 블랙박스 등을 통한 영상 데이터, Sound 센서 등을 활용한다면, 다양한 정보들을 통해 차량의 이상 작동 현상에 대해 더 면밀하게 분석할 수 있다.

5.2 CAN Fuzzing 기술 검증

본 논문에서 제안하는 Non-Random CAN Fuzzing 기술의 효율성이 All new Soul 2014 차량 실험에서 검증되었다. 즉, 기존에 사용되던 CAN Fuzzing 기술에 비해 Fuzzing에 소요되는 시간을 크게 줄일 수 있다.

하지만 다양한 차량에 대해 제안 기술을 검증하지 못한 한계점이 존재하므로 제안하는 CAN Fuzzing을 다양한 차량 환경에 적용하고, 해당 결과를 분석하는 향후 연구가 진행되어야 한다. 또한, Fuzzing 입력값 생성 시, Checksum을 맞춰서 주입하더라도 차량이 반응하지 않는 경우가 존재한다. 따라서 CAN 메시지를 검증하는 또 다른 요소인 Counter Field를 고려하여 주입하거나 서로의 상관관계가 높은 CAN ID들을 동시에 주입하는 등의 Fuzzing 입력값 생성 및 주입에 대한 추가 연구가 필요하다.

VI. 결 론

차량과 IT 기술의 융합으로 인해, 차량 해킹 가능성이 증대되고 있다. 차량 해킹 대응 기술로써, ECU의 취약점을 분석할 수 있는 ECU 리버스 엔지니어링 기술과 CAN Fuzzing 기술이 존재하지만, 전자는 완성차에 탑재된 ECU에 대한 물리적인 접근이 필요하고, 후자는 CAN 메시지 신호의 특징을 고려하지 않고 Random한 방식으로 Fuzzing을 진행하기 때문에 많은 시간이 소요되는 문제점 및 Fuzzing 결과 모니터링에 대한 한계점이 있다. 따라서 본 연구에서는 기존 ECU의 취약점 분석 기술의 한계를 해결하기 위해 CAN 메시지 신호 특성을 반영한 Non-Random CAN Fuzzing 기술을 최초로 제안하였다. Random CAN Fuzzing에 비해 제안하는 Non-Random CAN Fuzzing 기술은 CAN Fuzzing에 소요되는 시간을 단축시킴과 동시에 ECU에 대한 물리적인 접근 없이 CAN Fuzzing 결과를 자동적으로 모니터링할 수 있다. 또한, 제안하는 기술은 All new Soul 2014 차량에서 그 효율성이 검증되었다.

향후 연구로는 제안한 기술을 다양한 차량에 적용하여 실험 결과를 도출할 계획이다.

References

[1] Pierre Kleberger, Tomas Olovsson, and Erland Jonsson, “Security Aspects of the In-Vehicle Network in the Connected Car,” IEEE intelligent Vehicles Symposium, June 2011

[2] ISO 11898-1:2015, “Road Vehicles - Controller Area Network (CAN),” 2015

[3] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, and Tadayoshi Kohno, “Experimental Security Analysis of a Modern Automobile,” in Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp. 447-462, May 2010

[4] Charlie Miller and Chris Valasek, “Remote Exploitation of an Unaltered Passenger Vehicle,” Black Hat USA, Aug. 2015

[5] Dan Klinedinst and Christopher King, “On Board Diagnostics: Risks and Vulnerabilities of the Connected Vehicle”, Software Engineering Institute Carnegie Mellon University, Mar. 2016

[6] https://nmap.org/

[7] http://ubertooth.sourceforge.net/

[8] Charlie Miller and Chris Valasek, “Adventures in automotive networks and control units,” Defcon, 2013

[9] Karl Koscher, Alexei Czeskis,

Franziska Roesner, and Tadayoshi Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in Proceedings of the 20th USENIX conferences on Security, pp. 447-462, Aug. 2011
[10] https://github.com/CANToolz/CANToolz
[11] https://github.com/TianTianlove/ATG-python
[12] https://github.com/bhass1/pyfuzz_can
[13] https://github.com/zombieCraig/UDSim
[14] https://github.com/CaringCaribou/caringcaribou
[15] Robert Bosch GmbH, "CAN Specification Version 2.0," 1991
[16] https://en.wikipedia.org/wiki/OBD-II_PIDs
[17] Tae Un Kang, Hyun Min Song, Seonghoon Jeong, and Huy Kang Kim, "Automated Reverse Engineering and Attack for CAN using OBD-II," 2018 IEEE 88th Vehicular Technology Conference, pp.1-7, Aug. 2018
[18] Hyun Min Song and Huy Kang Kim, "Discovering CAN specification using On-Board Diagnostics", IEEE Design and Test, July 2020
[19] Mirco Marchetti and Dario Stabili, "READ: Reverse Engineering of Automotive Data Frames," IEEE Transactions on Information Forensics and Security, April 2019
[20] Mert D.Pesé, Troy Stacer, C. Andrés Campos, Eric Newberry, Dongyao Chen and Kang G. Shin, "LibreCAN: Automated CAN Message Translator," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, Nov. 2019
[21] Sekar Kulandaivel, Tushar Goyal, Arnav Kumar Agrawal and Vyas Sekar, "CANvas: Fast and Inexpensive Automotive Network Mapping," in Proceedings of the 28th USENIX Conference on Security Symposium, pp. 389-405, Aug. 2019
[22] Geoffrey Hinton, Li Deng, et al, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," IEEE Signal Processing Magazine, pp. 82-97, Nov. 2012
[23] http://skpang.co.uk/catalog/pican-canbus-board-retired-replacement-available-p-1196.html
[24] https://github.com/commaai/opendbc
[25] https://github.com/rhyttr/SocketCAN

〈저 자 소 개〉

김 형 훈 (Hyunghoon Kim) 정회원
2019년 8월: 한림대학교 컴퓨터공학과 졸업
2019년 9월~현재: 숭실대학교 융합소프트웨어학과 석사과정
〈관심분야〉 자동차 보안, IoT/CPS 보안

정 연 선 (Yeonseon Jeong) 정회원
2021년 2월: 한림대학교 융합소프트웨어학과 졸업 예정
〈관심분야〉 자동차 보안, IoT/CPS 보안

최 원 석 (Wonsuk Choi) 정회원
2008년 2월: 서울시립대학교 수학과 졸업
2013년 2월: 고려대학교 정보보호대학원 석사 졸업
2018년 2월: 고려대학교 정보보호대학원 박사 졸업
2020년 2월: 고려대학교 정보보호연구원 연구교수
2020년 3월~현재: 한성대학교 IT융합공학부 교수
〈관심분야〉 센서 보안, 자동차 보안, 암호 프로토콜

조 효 진 (Hyo Jin Jo) 정회원
2009년 2월: 고려대학교 산업공학과 졸업
2016년 2월: 고려대학교 정보보호대학원 박사 졸업
2018년 8월: University of Pennsylvania 박사 후 연구원
2019년 9월~현재: 숭실대학교 소프트웨어학과 교수
〈관심분야〉 자동차 보안, IoT/CPS 보안, 프라이버시