Journal of The Korea Society of Computer and Information
Vol. 18, No. 9, September 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.9.021

차량 ECU의 외부 위협성 가능성을 검증하기 위한 테스트 케이스 생성 연구

이 혜 련*, 김 경 진**, 정 기 현**, 최 경 희*

Research of generate a test case to verify the possibility of external threat of the automotive ECU

Hye-ryun Lee*, Kyoung-jin Kim**, Gi-hyun Jung**, Kyung-hee Choi*

요 약

차량에서 중요 기능들을 탑재하고 있는 ECU(Electric Control Unit)들 간에 메시지를 주고받는데 하나의 내부 통신망(CAN BUS)으로 연결되어 있으나, 이 네트워크는 외부에서 쉽게 접근이 가능하여 의도하지 않게 공격자로부터 공격을 받을 수 있게 되면서 이와 관련하여 외부로부터 공격에 대한 가능성을 검증하기 위하여 공격에 사용될 수 있는 도구들을 개발하였다. 그러나 이를 개발하기 위한 시간적 비용이 발생하고, 공격에 사용될 CAN 메시지를 찾기 위하여 실제 자동차에서 분석하는 시간도 발생한다. 본 논문에서는 이러한 문제를 해결하기 위해 공개되어 있는 Sulley라는 도구를 이용하여 공격에 필요한 테스트 케이스 생성하는 방법을 제안하면서 공격에 사용될 CAN 메시지 찾는 방법을 설명한다. Sulley에서 제공하는 라이브러리 파일에 CAN 메시지의 데이터 생성 라이브러리 추가시킨 다음 Sulley을 실행시키는 파일과 정의하는 파일을 CAN 통신 환경 설정 및 메시지 규칙에 맞게 작성하여 Sulley을 실행시킨다. 제안한 방법론을 실제 자동차에 적용시켜 실험을 수행한 결과, Sulley을 통한 CAN 메시지 퍼징하여 생성된 테스트 케이스들을 자동차에 보낸 결과 별도의 개발한 도구 필요 없이 자동차를 동작시켰다.

▸ Keywords : ECU, 보안, 공격, 퍼징 알고리듬, sulley

Abstract

ECU(Electric Control Unit) on the important features of the vehicle is equipped, ECU between sending and receiving messages is connected to one of the internal network(CAN BUS), but this network easily accessible from the outside and not intended to be able to receive attacks from an attacker, In this regard, the development of tools that can be used in order to verify the possibility

•제1저자 : 이혜련 •교신저자 : 정기현
•투고일 : 2013. 3. 1, 심사일 : 2013. 3. 20, 게재확정일 : 2013. 8. 22.
* 아주대학교 컴퓨터공학과(Dept. of Computer Science, Ajou University)
** 아주대학교 전자공학과((Dept. of Electronic Science, Ajou University)

of attacks on attacks from outside, However, the time costs incurred for developing tools and time to analyze from actual car for CAN messages to be used in the attack to find. In this paper, we want to solve it, propose a method to generate test cases required for the attack is publicly available tool called Sulley and it explains how to find the CAN messages to be used in the attack. Sulley add the CAN messages data generated library files in provided library file and than Sulley execute that make define and execute file conform to the CAN communication preferences and create message rules. Experiments performed by the proposed methodology is applied to the actual car and result, test cases generated by the CAN messages fuzzing through Sulley send in the car and as a result without a separate tool developed was operating the car.

▸ Keywords : ECU, security, attack, fuzzing Algorithm, sulley

I. 서 론

기존 자동차는 단순히 기기들의 조합으로 구성되어 있었다면, 최근의 자동차들은 엔진 또는 자동변속기 등 각종 차량 내부 장치들의 동작을 컴퓨터가 제어하도록 전자 제어 장치인 ECU들로 구성되어 있으며, 이 ECU들을 이용하여 자동차의 운행을 감지하거나 오디오/에어컨 등 편의시설까지 제어할 수 있어 운전자가 자동차를 제어하는데 편리성을 제공받을 수 있도록 한다. 따라서 자동차에 다양한 기능들을 추가시키기 위해 ECU들의 사용을 증가 시키고 있으며, 이 ECU들은 하나의 내부 통신망(CAN bus)으로 연결되어 있다[1].

그러나 현재 자동차용 내부 네트워크의 컨트롤러 영역은 외부 커뮤니케이션에 노출되어 오용 및 의도하지 않게 시스템에 접근이 가능하면서 위험성을 가지고 있다. 이러한 위험성으로 ECU들은 악의적인 공격에 매우 취약하다. 해커들은 ECU들 간의 데이터 읽기, 주입 그리고 편집과 같은 기능 수행을 관찰할 수 있어 자동차의 중요한 인프라(드라이버 손상 등)에 영향을 줄 수 있다[2,3]. 특히 자동차 경우 일반적인 시스템과 다르게 악성 코드를 삽입 및 ECU 간에 사용되는 데이터 변경 등을 하여 공격이 성공할 시 전체 시스템 내에 확산시킬 수 있다. 그 결과 안전과 관련하여 심각한 결과를 가져 올 가능성이 높다.

따라서 사이버 공격으로부터 위협이 발생하게 되면서 국내외적으로 자동차에 대한 악의적인 공격 방법 및 이를 방어하기 위한 방법과 관련한 관심이 증가하고 있다. 그러나 이러한 연구들에 대한 자세한 정보들을 공개하지 않거나 공격이나 방어 방법과 관련하여 실제 실험 위주보다는 시나리오 위주의 설계 방식을 택하고 있다[4,5].

그 중에서 자동차 ECU의 외부 위협이 가능한지를 보이기 위하여 국내외적으로 연구가 진행되었으며 이 방식은 실제 자동차에 장비들을 연결하여 CAN 메시지들에 대한 패킷 스니핑(packet sniffing)을 한 다음 이 메시지들에 대해 퍼징(fuzzing) 할 수 있는 도구를 개발하여 실차에 메시지 퍼징을 시켜 공격을 성공시켜 자동차 ECU에 대한 보안 테스트의 중요성을 강조하였다[6,7]. 그러나 이 접근 방식은 테스트 케이스로 사용될 CAN 메시지를 퍼징하기 위한 도구를 개발하여야 하기 때문에 각각의 사양에 맞는 도구를 개발하여야 함으로 시간적인 비용이 발생한다.

따라서 본 논문에서는 이러한 문제를 해결하기 위해 자동차 ECU의 외부 위협을 검증하기 위한 테스트를 진행하고자 필요한 테스트 케이스를 생성하는데, Sulley를 이용하여 테스트 케이스를 생성하는 방법을 제안한다. Sulley에서 제공하는 라이브러리 파일(Primitives.py)에서 stirng 클래스의 라이브러리 리스트를 지우고 CAN 메시지의 데이터 생성 라이브러리 추가시킨다. 그 다음 Sulley을 실행시키는 파일과 정의하는 파일을 CAN 통신 환경 설정 및 메시지 규칙에 맞게 작성하여 Sulley을 실행시킨다.

Sulley을 통해 CAN 메시지를 퍼징하여 테스트 케이스들을 생성한 결과 별도의 개발한 도구 필요 없이 테스트를 진행하여 시간적 비용을 축소시킨다. 또한 실제 자동차에서 사용되는 CAN 메시지들 중에서 입력 메시지와 출력 메시지를 구분하여 입력 메시지에 대한 퍼징하여 폭발적으로 발생할 수 있는 테스트 케이스 양을 제한한다.

마지막으로 본 논문에서 제안하는 방법의 가용성을 검증하기 위하여 실제 자동차를 대상으로 실험을 수행하고 실험 결과 ECU의 CAN 메시지 데이터 일부 영역이 변하는 것을 확

인하여 제안된 방법을 통해 자동차 ECU의 외부 위협을 검증하기 위한 테스트를 할 수 있는 기반 기술을 획득하는데 있어서 도움이 될 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 ECU 공격에 사용될 수 있는 도구들을 분석하여 서술하며, 3장에서는 자동차용 ECU의 외부 위협을 검증하기 위한 테스트를 진행하는데 필요한 테스트 케이스 생성 방법을 기술하며, 4장에서는 실험을 통하여 테스트 결과를 보인다. 마지막으로 5장에서는 결론을 맺는다.

II. 관련 연구

현재 자동차 내부 시스템을 살펴보면 약 70% 정도 ECU들로 구성되어 있으며, 이들은 하나의 네트워크 CAN bus로 연결되어 있다. 자동차 버스 시스템은 무단 액세스가 비교적 쉽게 가능하며, ECU들 간의 주고받는 CAN 메시지는 완전히 암호화되지 않는 일반 텍스트로 이루어져 있다[8]. 단, 자동차 업체별로 CAN 메시지에 대한 의미를 다르게 부여하고 있기 때문에 의미는 알지 못한다. 그러나 자동차의 시스템에 접근하여 암호화 되지 않은 CAN 메시지를 수집 및 분석하여 공격이 가능함으로 차량 ECU에 대한 외부 위협이 심각하다. 따라서 공격에 사용될 수 있는 도구들을 분석하여 어떠한 방법을 사용하였는지를 검토할 필요가 있다.

1. ECU 메시지 퍼징하는 공격자 도구 개발

이 연구에서는[6] 현대자동차의 소나타 Y20과 제니시스 BH330 자동차를 이용하여 실험 환경을 구성하였다. 좀 더 자세히 살펴보면 자동차를 고정시킨 상태에서 차 내부에 있는 ECU 케이블에서 CAN High/Low 케이블을 찾아낸 다음, 차량에 OBDII 커넥터를 연결하였다. 그 다음 자동차를 동작시키면서 이를 통해 출력되는 CAN 시그널을 PCAN-USB 장비를 사용하여 노트북에서 사용 가능한 시그널로 변환한 다음 노트북에서 CAN 메시지를 분석하였다.

1.1 실험 내용 정리

실험 환경을 구축한 다음 자동차에 4가지 형태의 실험을 하였다.

1. CAN 메시지 도청(Sniffing) : OBDII 커넥터를 사용하여 평문 상태의 CAN 메시지를 그대로 도청하였다.
2. CAN 메시지 재생(Playback) : 도청하여 얻어낸 CAN 메시지의 의미를 알 수 없기 때문에 운전자가 자동차에 탑승하여 여러 가지 조작을 하면서 CAN 메시지들을 수집하고, 수집된 CAN 메시지들을 재현 시키면서 차량의 반응을 관찰하였다.
3. CAN 메시지 인젝션(Injection) : 공격용 CAN ID와 데이터들을 찾아 차량의 메시지 전송시간 간격(7~10ms)보다 빠르게 CAN 메시지를 인젝션 시킨다. 그 결과 정상 메시지 양보다 공격자가 주입시킨 메시지 양이 더 많아 공격자가 원하는 대로 ECU가 반응하였다.
4. CAN 메시지 퍼징(Fuzzing) : 도청하여 수집한 CAN 메시지를 이용하여 퍼징을 수행하였다.

1.2 ECU 공격자 도구(ECU Attacker Tool)

CAN 메시지 인젝션 시키는 방법까지 수행한 결과 공격에 대한 반응하는 부분이 적어 CAN 메시지의 패턴을 분석한 결과를 가지고, CAN 메시지의 적당한 위치 데이터 영역에만 데이터 퍼징을 할 수 있도록 직접 C#을 이용한 ECU 공격자 도구를 만들었다. 컴퓨터와 PCAN-USB를 연결한 상태에서 프로그램을 활성화시킨 다음, CAN ID당 데이터 8바이트 범위 안에서 일부 데이터 영역만 선택하여 선택한 영역에 대한 퍼징을 수행하고, 자동차와 유선으로 연결된 상태에서 공격하

Destination ECU	Packet	Result	Manual Override	At Speed	Tested on Runway
IPC	00 00 ... 00 00	Falsify Speedometer Reading	No	Yes	✓
Radio	04 00 ... 00 00	Increase Radio Volume	No	Yes	
Radio	63 01 ... 39 00	Change Radio Display	No	Yes	
IPC	00 02 ... 00 00 27 01 ... 65 00	Change DIC Display	No	Yes	
BCM	04 03	Unlock Car†	Yes	Yes	
BCM	04 01	Lock Car†	Yes	Yes	
BCM	04 0B	Remote Start Car†	No	No	
BCM	04 0E	Car Alarm Honk†	No	No	
Radio	83 32 ... 00 00	Ticking Sound	No	Yes	
ECM	AE 0E ... 00 7E	Kill Engine	No	Yes	

그림 1. CarShark 도구를 이용한 실험 결과
Fig 1. Experimental results using CarShark tool

였다.

1.3 실험 분석

실험 결과 엔진 RPM이나 스피드 계량기 조작부터 차량 문 잠금 현상까지 가능하였다. 특히 소나타 같은 경우 특정 데이터를 1ms 단위로 인젝션 시킬 경우 10초 이내에 엔진을 정지시켰으며 액셀러레이터를 반응하지 않도록 하는 것까지 성공시켰다. 그러나 조종 장치 바퀴 조향 같은 경우는 피드백 메시지만 출력시키고, 브레이크는 기계적 방식인 유압식으로 출력하여 데이터를 조작하지 못하였다.

2. ECU 메시지 퍼징하는 CarShark 도구 개발

Karl Koscher 등[7]은 CarShark 이라는 도구를 개발한 다음 차량에 대한 다양한 실험 환경을 구성하였으며 그 중에서 802.11 ad hoc 네트워크를 이용한 무선 통신 환경에서 자동차에 공격을 시도하였다. 그 결과 자동차에 대한 외부 위협이 발생할 수 있음을 보였다.

2.1 CarShark 도구

실험에서 사용한 프로그램으로 사용자 정의에 따라 CAN 메시지들을 주고받을 수 있도록 CAN 버스 분석 및 패킷 주입 도구인 CarShark을 개발하였다. 이 도구는 CAN 프로토콜에서 정보를 독점적으로 처리하고 조작 할 수 있는 능력을 가지고 있으며 CAN 인터페이스에 따른 CAN 메시지 퍼징을 수행한다.

1. 패킷 도청 및 탐색(Packet Sniffing and Targeted Probing) : CAN 버스에 발생하는 트래픽을 관찰할 수 있으며 패킷 도청 및 탐색을 통해 전송된 패킷이 다양한 구성 요소들을 활성화 시킬 수 있는 것을 확인하였다. 또한, 라디오, IPC 및 BCM 함수의 수를 제어하는 방법을 발견하였다.
2. 퍼징 : 차량에서 유효한 CAN 패킷의 범위가 다소 작기 때문에 패킷에 대한 간단한 퍼징을 하여 중요한 공격을 할 수 있다. 따라서 DeviceControl 이라는 CAN 기반 서비스를 정의한 다음 퍼징을 수행한다.
3. 역공학(Reverse-Engineering) : CAN Read-Memory 서비스 및 IDA pro 디버거를 사용하여 명시적으로 특정 하드웨어 기능을 제어하였다.

2.2 실험 결과

이 연구에서는 ECU들을 정리하고 CarShark을 이용하여 각각의 ECU들에 공격을 시도하였다. 그 결과 실제 자동차에서 실험한 결과 라디오, IPC(Instrument Panel Cluster), HVAC, Generic Denial of Service 로 나누어 제어가 가능하거나 여러 가지 조작이 가능하다 라는 것을 결과로 도출하였다. 그림 1은 결과 도출한 내용 일부를 나타낸 것이다. 그 중에서 다음 3가지는 퍼징하여 제어하였다.

1. Body Controller : BCM에서는 2가지 버스를 조절하여야 하는데 저속버스에는 리버스 엔지니어링 패킷을 보내고, 고속버스에서는 패킷을 퍼징하여 그 패킷을 보냈더니 제어가 가능하였다.
2. Engine : ECM에서는 DeviceControl에 대한 퍼징을 하였다. 그 결과 센서 오류를 주기적으로 삽입하여 엔진 타이밍을 방해시키게 되면서 RPM을 높일 수 있었다.
3. Brakes : 전자 브레이크 제어 모듈에 대하여 퍼징을 함으로써 패킷을 통해 브레이크를 제어하는 부분을 발견하였으며 자동차의 전원을 끌 수 있다.

III. 테스트 케이스 생성

자동차 업체에서는 고객의 불편을 최소화시키기 위하여 ECU의 소프트웨어 부분을 점차적으로 늘리게 되면서 소프트웨어에 대한 소프트웨어의 결함들을 식별하고 이를 수정하기 위하여 원격 진단 및 펌웨어 업데이트를 수행할 수 있도록 하고 있다. 그러나 이러한 절차를 진행하기 위하여 차량에 대해 외부 통신을 허용하게 되면서 차량 네트워크에 대하여 사이버 공격의 위험성을 가지게 되면서 차량의 보안 부분이 중요한 이슈로 떠오르고 있다[9]. 특히 외부에서 공격자가 악의적인 코드를 차량에 삽입하여 엔진 정지, 브레이크 사용 등 오동작을 일으켜 운전자로 하여금 자동차를 제대로 제어할 수 없게 만들어 심각한 문제를 야기 시키게 되며, 그 중에서 가장 큰 문제점은 차량에 복잡한 공격을 할 수 있는 악성 코드를 삽입하여 차가 움직이다가 충돌하게 될 경우에 존재에 대한 증거를 지워버리는 문제까지 발생시킬 수 있다 라는 것이다[7].

기존 연구들을 분석하여 보면 실제 자동차에서 CAN 메시지들을 감청하여 감청한 메시지에 대한 퍼징을 수행할 수 있는 도구들을 개발하여 자동차에 전송시켜서 공격을 성공시켰다. 그 결과 자동차가 다양한 오동작을 일으키는 것을 확인하여 공격에 굉장히 취약하다 라는 점을 보였다.

특히 기존 연구에서 개발한 퍼징 도구들은 직접 사양에 맞게 도구들을 개발하기 위하여 C++이나 C#를 이용하였다. 다양한 기능들을 추가시켰으며 초기화 및 메모리 할당 등 여러 가지 일들을 수행할 수 있다. 그러나 자동차에 공격을 하기 위하여 사용될 프로그램을 개발하는데 시간이 소요되며 프로그램을 실행시키기 위하여 성능 좋은 컴퓨터가 반드시 필수

적이다.

따라서 본 논문에서는 기존 문제점을 해결하기 위하여 퍼징을 잘 하는 Sulley을 이용하여 테스트 케이스 생성하는 방법을 제안한다. Sulley는 테스트 환경에서 공격에 사용될 테스트 케이스를 빠르게 생성한다. 따라서 기존 연구에서 가지는 프로그램 개발 시간을 단축시킬 수 있으며, UI을 실행시키지 않아도 되므로 도구를 실행시키는데 사용되는 성능 좋은 컴퓨터가 필요하지 않다. 본 절에서는 Sulley을 이용하여 어떻게 테스트 케이스 생성을 할 것인지에 대하여 설명하도록 한다.

1. Sulley

퍼징 도구 중 하나인 Sulley는 테스트 할 대상(SUT, System Under Test)을 쉽게 묘사할 수 있는 도구로 TippingPoint의 P. Amini와 A. Portnoy가 개발한 Python 기반의 퍼징(fuzzing) 프레임 워크이다[10]. Sulley는 현재 공개되어 있는 다양한 퍼징 도구들에 비해 많은 기능들을 추가하여 기능성을 크게 확장시켰으며 다양한 종류의 테스트 케이스 생성 할 수 있는 라이브러리를 제공한다.

Sulley는 블록 기반 퍼징 방법을 사용하는 방식으로 그림 1과 같이 총 5단계에 거쳐 테스트를 진행한다.

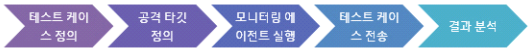

그림 2. 테스트 실행 순서
Fig 2. Test execution order

1. 테스트 케이스 정의 : 각 블록에 대해 전송될 데이터의 필드 길이와 데이터 타입 등으로 지정되며 이를 내부 정의된 다양한 형태의 라이브러리를 이용하여 테스트 케이스를 만든다.
2. 공격 타깃 정의 : 테스트에 필요한 세션 및 공격 타깃을 정의한다.
3. 모니터링 에이전트 실행 : Sulley는 테스트 환경의 감시를 위한 네트워크 모니터, 프로세스 모니터 등과 같은 모니터링 에이전트를 제공함으로 테스터는 제공되는 에이전트를 이용하여 자신의 필요에 따라 적절한 에이전트를 실행하고 이를 통해 실험 환경의 상태 변화와 예외 상황 등을 감시한다.
4. 테스트 케이스 전송 : Sulley에 내장된 웹기반 모니터링 인터페이스를 이용하여 생성된 테스트 케이스 전송 중지 및 재개와 같은 작업을 수행하여 테스트 케이스를 전송한다.
5. 결과 분석 : 네트워크에서 전송되는 패킷들에 대하여 저장하고 오류가 생긴 경우 이에 대한 레포트 기능까지 포함하고 있어 결과가 어떻게 되었는지를 분석한다.

이러한 절차에 따라 Sulley의 퍼징 방법을 이용하여 A. B. M. Omar Faruk과 Z. B. Cheah는 DNP3 프로토콜과 IEC 60870-5-105 프로토콜에 대한 취약성을 테스트한 결과를 발표하였고[11], Ganesh Devarajan은 SCADA 프로토콜들의 취약성을 분석하는데 sulley을 이용하여[12] 데이터 부분을 퍼징하기 위한 테스트 케이스를 생성한 후, 생성된 입력 데이터를 이용하여 SCADA 프로토콜에 내재된 취약성들이 존재함을 검증하였다. 따라서 많은 해커들의 관심을 끌고 있다.

2. Sulley의 테스트 케이스 생성

많은 해커들이 하나의 공격할 대상에 예측 불가능한 입력 혹은 비정상적인 입력을 보내는 방법인 퍼징 기법을 이용하고 공격을 시도한다. 퍼징은 테스트 할 대상에 대한 정상적인 입력 외에 대한 비정상적인 입력 데이터를 퍼저(fuzzer)가 생성하여 SUT에 보내고 해당 입력에 대한 SUT의 행동의 정상 동작 여부를 판단한다[13].

Sulley에서 SUT의 입력 데이터들로 사용될 테스트 케이스를 만드는데 이에 필요한 테스트 케이스 생성 알고리듬을 살펴보면 앞에서 설명한 것과 같이 테스트 케이스 생성 관련 라이브러리를 정의하여 자동으로 생성한다. Sulley의 '*Primitives.py*' 파일은 테스트 케이스 라이브러리 파일로 하나의 입력 데이터에 대한 다양한 라이브러리가 정의되어 있다. 표 1은 테스트 케이스 생성 라이브러리 일부분 중에서 *string* 관련 클래스의 일부분 이다.

표 1. Sulley의 테스트 케이스 생성 라이브러리
Table 1. Test case generation library of Sulley

```
class string (base_primitive) :
    ……
    ……
# add some long strings.
    self.add_long_strings("A")
    self.add_long_strings("1")
    self.add_long_strings("<")
    self.add_long_strings(">")
    self.add_long_strings("/")
    self.add_long_strings("\\")
    self.add_long_strings("?")
    self.add_long_strings("=")
……
```

예를 들어 설명하면 입력 데이터 'a'를 입력하였다고 가정하면 표 1의 라이브러리 규칙에 따라 정상적인 입력 데이터 'a'를 포함하여 비정상적인 입력 데이터들을 아래 표 2와 같이 생성하게 된다.

표 2. 테스트 케이스 생성 예제
Table 2. Example of test case generation

```
1 'aA'
2 'a1'
3 'a('
4  'a)'
5 'a/'
6 'a\\'
7 'a?'
8 'a='
9 ……
```

3. CAN 메시지의 퍼징 알고리듬

앞에서는 설명한 내용은 소프트웨어에 대한 테스트 케이스 생성 라이브러리이므로 본 논문에서 사용할 CAN 데이터에 대한 테스트 케이스를 자동 생성하기 위해 '*Primitives.py*'의 라이브러리 일부분을 수정하여 퍼징 알고리듬으로 사용한다. 따라서 *string* 클래스에서 테스트 케이스 생성하는 라이브러리 부분을 주석처리 한 다음 아래 표 3과 같이 라이브러리를 추가한다. 추가한 소스 코드에는 CAN 메시지의 데이터 1 바이트에 대하여 값이 0부터 255까지 생성되도록 하였다. 단, CAN 메시지에서 데이터 8바이트에 대한 각각 바이트를 구별하기 위해 키워드로 ':' 을 사용함으로 키워드를 값 출력 시 함께 사용한다.

표 3. CAN 데이터 테스트 케이스 생성 라이브러리
Table 3. Test case generation library of CAN Data

```
# CAN Data fuzzing library
for item in range(256):
    self.fuzz_library.append(str(item) + ":")
```

이렇게 수정한 다음 Sulley을 실행시키면 CAN 메시지의 데이터 영역에 대한 각각 바이트 값이 0부터 255까지 바뀌므로 CAN 데이터를 8바이트 전부 다 사용한다고 가정하면 CAN ID 하나 당 총 입력 데이터로 사용될 수 있는 테스트 케이스는 2^64(18,446,744,073,709,551,616)개이다.

4. 테스트 케이스 생성 방법

ECU로부터 수집된 CAN 메시지의 패턴을 분석하여 보면 CAN 메시지의 데이터 일부분은 주기적으로 바뀌는 부분인 방면에 나머지 데이터 일부분은 정적으로 바뀌지 않는다. 따라서 CAN 메시지에 대한 패턴을 구분할 수 있어 자동차에 공격으로 사용될 CAN ID 하나 당 데이터 전체에 대한 테스트 케이스를 만들어 수행할 필요 없이 동적으로 바뀌는 데이터 영역에 대하여 테스트 케이스를 만든다.

CAN 메시지 규칙에 따라 데이터 일부분 값을 바꿔어 테스트 케이스를 생성하기 위하여 CAN 메시지 값을 정의하는 파일 '*can_message.py*' 파일에서는 다음 표 4와 같이 정의한다. 이 표를 예제로 한다고 가정하면 CAN 메시지 하나에 대하여 데이터 영역 2군데가 바뀌도록 정의하였다. 바뀌는 영역은 s_string 함수를 사용하였고, 바뀌는 않는 영역은 s_static 함수를 사용하였다. CAN ID 하나 당 바뀌는 데이터 영역에 따라 이 정의 파일을 생성하여 Sulley을 실행 시키는 파일에서 import 하면 된다.

표 4에서 사용된 예제를 설명하면 첫 번째 p0는 CAN 메시지의 데이터 영역의 값에 대해 테스트 케이스를 생성한다고 가정하면 CAN 메시지는 다음과 같이 규칙을 따르므로,

syn(1):PID(1):Leng(1):CAN ID(4):
CAN Data(8):checksum(1):End byte(1)

첫 번째 static 함수는 syn(1):PID(1):Leng(1):CAN ID(4): 를 작성하고, 두 번째 static 함수는 CAN 데이터의 앞 5 바이트를 작성한다. 세 번째 string 함수는 CAN 데이터의 6번째 바이트로 string 함수를 사용하여 string 클래스의 라이브러리를 호출한다. 앞에서 정의한 테스트 케이스 생성 라이브러리에 따라 0부터 255까지 값을 생성한다. 4번째 static 함수는 뒤 CAN 데이터 2바이트와 chacksum(1), End byte(1)를 작성하였다.

두 번째 p1 에서는 CAN 데이터의 2번째 바이트 1부터 255까지 값이 바뀔 때마다 CAN 데이터의 6번째 바이트가 0부터 255까지 바뀌도록 CAN 데이터의 2번째 바이트 위치에 s_string(temp) 함수를 추가시킨다.

표 4. CAN 메시지 정의파일
Table 4. File of CAN message define

```
s_initialize("p0")
s_static("240:132:12:243:0:0:0:")
s_static("0:0:127:0:255:")
s_string("0:")
s_static("128:0:0:224")
s_static("\r\n")
```

```
for i in range(255):
    temp = "p1_" + str(i)
    s_initialize(temp)
    cnt = i+1
    temp = str(cnt) + ":"
    s_static("240:132:12:243:0:0:0:")
    s_static("0:")
    s_static(temp)
    s_static("127:0:255:")
    s_string("0:")
    s_static("128:0:0:224")
    s_static("\r\n")
```

Sulley을 실행시키는데 사용되는 '*can_test.py*' 파일에서 CAN 통신 환경에 맞게 실행하는 부분을 추가시키고, 입력 데이터를 입력할 시 CAN 메시지의 동적으로 바뀌는 데이터 개수에 따라 for문을 이용한 connect문을 입력한다. 총 CAN ID 하나 당 데이터는 8바이트 이므로 최대 7까지 만들 수 있으며 '*can_test.py*'에서 import로 사용될 '*can_message.py*' 파일에서 2개의 데이터 영역을 변경하였으므로 아래 표 5와 같이 정의할 수 있다. 그 결과 데이터의 2바이트(2,6번)만 바뀌며 CAN ID 한 개에 대한 총 테스트 케이스는 2^16(65,536)개이다.

표 5. 테스트 케이스 생성 예제
Table 5. Example of test case generation

```
…
sess.connect(s_get("p0"))
for i in range(255):
    temp = "p1_" + str(i)
    sess.connect(s_get("p0"), s_get(temp))

sess.fuzz()
…

ex)
0:0:0:0:0:0:0:0
0:0:0:0:0:1:0:0
0:0:0:0:0:2:0:0
……
……
0:1:0:0:0:0:0:0
0:1:0:0:0:1:0:0
0:1:0:0:0:2:0:0
```

다음 그림 3은 '*can_test.py*'을 실행시키면 총 65536개의 테스트 케이스가 생성되는 모습이다.

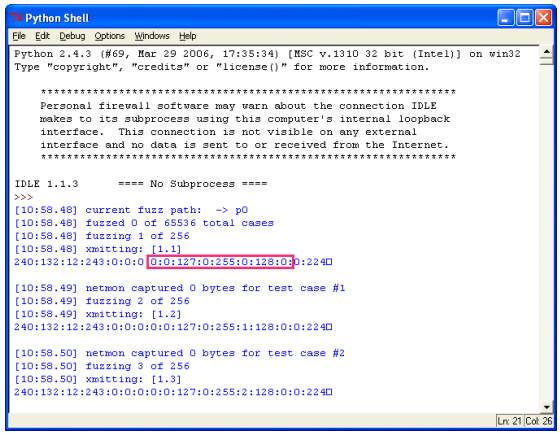

그림 3. 테스트 케이스 생성 모습
Fig 3. State of test case generation

IV. 실험

지금까지 본 논문에서 제안한 테스트 케이스 생성 알고리듬에 대한 가용성을 평가하기 위하여 실험을 수행하였다. 본 논문에서는 총 3단계의 실험 순서대로 진행한 다음 이를 분석하여 자동차 ECU의 외부 위협 가능성을 검증한다.

1. 테스트 실행 순서

1.1 CAN 메시지 도청(Sniffing)

CAN 메시지를 도청하기 위하여 2가지 방법을 수행한다.

첫 번째 그림 4와 같이 실제 자동차에서 OBDII 커넥터를 차량에 연결하고 커넥터의 CAN High와 Low 라인을 찾아서 PCAN-USB가 읽을 수 있도록 한다. 그 다음 노트북에 PCAN-USB을 구입한 회사에서 제공하는 프로그램을 설치한다. 기본적인 장비를 갖춘 상태에서 차량에 여러 동작들을 수행하여 그림 5와 같이 CAN 프로토콜에서 사용되는 CAN 메시지들을 도청한다.

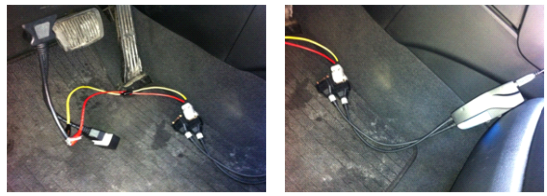

그림 4. 차량 연결 부분
Fig 4. Vehicle connections

표 6. CAN 메시지 내용 일 부분
Table 6. One part of CAN message

CAN ID	1 byte	2 byte	3 byte	4 byte	5 byte	6 byte	7 byte	8 byte
0x2	0	0	0	0	0	0	fuzzing	fuzzing
0x130	fuzzing	127	0	255	fuzzing	128	fuzzing	fuzzing
0x131	fuzzing	128	0	0	fuzzing	127	fuzzing	fuzzing
0x140	0	0	0	0	fuzzing	fuzzing	fuzzing	fuzzing
0x2B0	fuzzing	0	0	fuzzing	fuzzing			
0x350	5	32	fuzzing	81	96	0	0	fuzzing

두 번째 차량에서 엔진 ECU만 분리하여 전원과 시동 신호를 ECU에 입력하여 ECU가 동작할 수 있도록 한다. 그 다음 이 ECU로부터 출력되는 CAN 메시지들을 수집한다. 기존 연구에서는 자동차에서 직접 CAN 메시지에 대한 동작을 살피면서 입력과 출력 CAN 메시지를 구분하기 위하여 CAN 메시지를 분석하였지만 이렇게 분리하여 실험을 진행하면 입력과 출력CAN 메시지를 바로 분류 할 수 있어 실험에 유리하다.

1.2 CAN 메시지 분석(Analysis)

수집한 CAN 메시지들을 분석한다. 본 논문에서 사용한 자동차 모델에서는 첫 번째 실험을 통하여 총 31개의 CAN ID가 ECU 간에 사용되는 것을 확인하였으며 두 번째 실험에서는 총 14개의 CAN ID을 찾아냈다.

두 번째 실험에서 찾아낸 CAN ID는 엔진 ECU에서 출력되는 CAN 메시지들로 입력으로 사용되는 CAN 메시지들이 아니다. 따라서 첫 번째 실험에서 찾아낸 CAN ID 31개 중에서 14개를 제외한 17개의 CAN ID가 입력 메시지로 사용된 것으로 판단할 수 있다.

첫 번째 실험 결과 차량에서 여러 동작을 수행하였을 때 CAN ID 14개를 제외한 나머지 CAN ID에서 표 6과 같이 CAN 메시지의 데이터 영역 일부분이 바뀌는 것을 확인하였다. 표 6은 본 논문에서 실험한 자동차 모델에서 수집한 CAD ID 일부분을 나타낸 것이다.

1.3 CAN 메시지 퍼징(Fuzzing)

CAN 메시지를 분석하여 얻은 결과를 가지고 첫 번째는 CAN 메시지에 대하여 패턴을 분석하고 일정한 패턴 규칙을 가지는 CAN 메시지를 제외한 나머지 CAN 메시지(6~7개)에 제안한 알고리듬을 적용하여 퍼징한다. 퍼징을 수행하여 정확한 CAN 메시지를 찾아낼 수 있다. 자동차 안에서 여러 가지 동작을 수행하여 CAN 메시지를 수집할 때 다양한 CAN 메시지를 출력시킨다. 엔진 ECU를 통한 출력 CAN 메시지들을 찾아냈지만 입력 CAN 메시지의 정확한 데이터 값을 알 수는 없다. 자동차에서 ECU가 CAN 메시지를 받는데 20ms 라는 짧은 시간을 사용하기 때문에 여러 동작을 수행할 경우 똑같은 CAN ID 을 사용한다고 하더라도 데이터 일부분은 바뀐다. 따라서 퍼징을 수행하여 CAN 메시지를 찾아낸다.

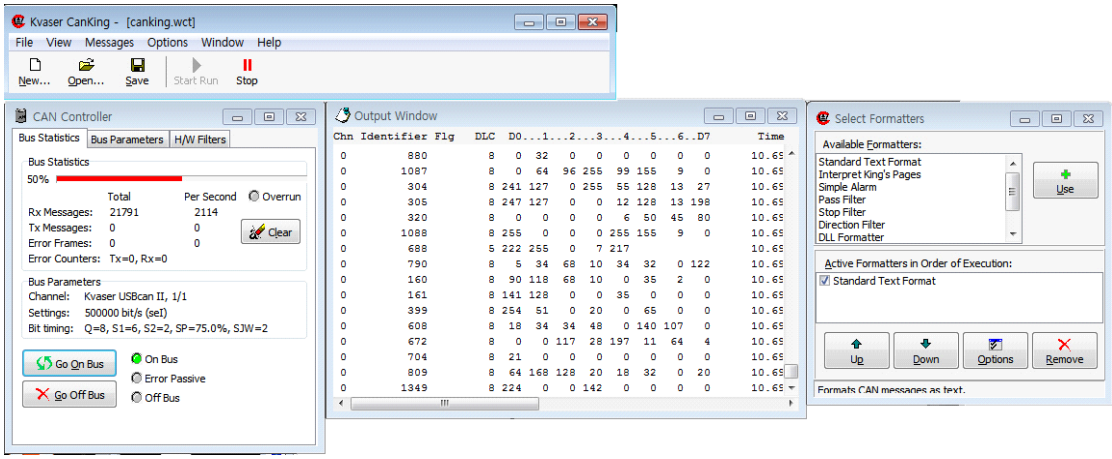

그림 5. PCAN-USB 도구 사용을 통한 CAN ID 수집
Fig 5. CAN ID collect using PCAN-USB tool

370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0

370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	10	F	0	0	0	0
370	0	20	40	0	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0
370	0	20	40	80	0	0	0	0

그림 6. 실험 결과
Fig 6. Experiment results

2. 테스트 분석

본 논문에서 제안한 테스트 케이스 생성 알고리듬을 적용하여 퍼징을 수행한 다음 실험을 수행하였다.

실험 결과 엔진 ECU에서는 CAN 메시지 16개가 존재한 상태에서 1개의 CAN 메시지를 퍼징 할 경우 엔진 ECU의 출력 CAN 메시지의 데이터 값이 바뀌는 것을 확인할 수 있었으며 엔진 ECU의 출력 CAN 메시지 0x370이 그림 6과 같이 데이터 영역 일부분이 바뀌는 것을 확인하였으며 공격으로 사용될 수 있는 입력 CAN 메시지를 찾았다.

찾아낸 CAN 메시지들을 차량에 삽입하여 실험한 결과 엔진 ECU의 기능을 기준으로 그림 7과 같이 자동차가 운전자 없이도 동작을 수행하였다.

CAN 메시지 0x2를 삽입할 경우 VDC 시그널이 발생하였고 차량에서 찾아낸 CAN 메시지 0x316을 삽입할 경우 RPM 시그널이 상승하는 것이 발생하였다.

따라서 실험을 통해 ECU의 외부 위협 가능성을 보임으로써 기존 연구에서 가지는 시간적 비용을 줄일 수 있었다. 기존에 발표한 연구에서 개발한 도구들을 비교하였을 경우 공격자가 공격하기 위하여 필요한 프로그램을 전부 다 개발할 필요가 없으며 공격용 CAN 메시지를 찾기 위하여 CAN 메시지의 반응을 살피기 위한 CAN 메시지를 재생할 필요가 없어 시간을 단축시킬 수 있었다. 그 결과 실험을 통하여 얻은 CAN 메시지들을 정리하여 하나의 데이터베이스 화 할 경우 본 논문에서 실험한 자동차 모델에 대하여 운전자가 없는 상태에서 공격을 가할 수 있어 실험한 자동차 모델이 도로에서 많이 있을 경우 무선 네트워크로 공격을 시도하여 운전자의 운전을 방해시켜 큰 교통사고를 일으킬 수 있을 것으로 예상된다.

그림 7. 운전자 없는 상태에서 RPM 상승 발생 및 VDC 발생
Fig 7. RPM rise and VDC occurs in the absence of the driver.

V. 결 론

자동차를 사용하는 운전자에게 사용의 편의성을 제공하고자 내부의 장치의 동작을 ECU 의해 제어하도록 하였으며 다양한 ECU들은 하나의 내부 네트워크 CAN로 연결되어 있다. 이러한 내부 시스템의 발달은 점점 가속화 되는 반면에 공격자에 의한 공격이 발생 가능하게 되면서 자동차 업체와 보안 업계쪽에서는 많은 관심을 가지고 있다.

따라서 본 논문에서는 여러 참고 문헌들을 통해서 공격에 사용될 수 있는 도구들을 분석하였다. 이 도구들은 자동차에 공격을 시도하여 성공시켰으며 차량에 대한 보안이 적실히 필요함을 강조하고 있다.

이와 같은 연구들을 바탕으로 본 논문에서는 차량 ECU의 외부 위협 가능성을 검증하기 위하여 테스트를 진행하는데 필요한 테스트 케이스 생성 알고리듬에 대한 아이디어를 제시하였다. 이 알고리듬에 대한 가용성을 검증하기 위하여 실제 자

동차와 엔진 ECU에서 CAN 메시지를 수집한 다음 CAN 메시지를 분석하여 알고리듬을 적용한 CAN 메시지 퍼징 작업을 수행하였다. CAN 메시지 퍼징 한 결과 엔진 ECU의 출력 CAN 메시지의 일부 데이터 영역이 바뀌는 것을 확인하였고, 실제 자동차에서도 동일하게 실험한 결과 운전자 없이도 자동차가 동작하는 것을 확인하였다. 그 결과 기존 연구에서 자동차에 공격을 하기 위한 도구를 개발하는데 필요한 시간을 단축시킬 수 있었다. 따라서 ECU에 대한 보안 테스트를 할 수 있는 중요 기반 기술을 획득하는데 도움이 될 수 있다.

자동차 같은 경우 IT 시스템과 다르게 고려해야 할 상황들이 많다. 가장 큰 문제점은 하나의 공격 대상에 공격하여 차량 네트워크에 연결된 다른 시스템에 영향을 줄 수 있다 라는 것이다. 사람의 생명과 직결되기 때문에 자동차용 ECU 대한 보안과 관련 분석을 통하여 해당 영역에 대한 정확한 연구가 적실히 필요하다.

향후에 이러한 연구들을 바탕으로 공격에 대한 방어를 하기 위해서는 차량에 대한 취약성 분석이 반드시 필요함을 알 수 있었다. 이를 위해서는 현재 제안한 테스트 케이스 알고리듬을 이용하여 다양한 공격 시나리오 기반의 실제 실험을 통한 방어책 관련 연구가 진행되어야 할 것이다.

참고문헌

〔1〕 Phung, Phu H. "A model for safe and secure execution of downloaded vehicle applications," Road Transport Information and Control Conference and the ITS United Kingdom Members' Conference (RTIC 2010), pp.1-6, May. 2010.

〔2〕 Nilsson, D.K., "Efficient In-Vehicle Delayed Data Authentication Based on Compound Message Authentication Codes," in Proceedings of IEEE 68th Vehicular Technology Conference, 2008. VTC 2008-Fall. pp.1-5, Sep. 2008.

〔3〕 Muter, M., "A structured approach to anomaly detection for in-vehicle networks," Information Assurance and Security (IAS), 2010 Sixth International Conference on, pp.92-98, Aug. 2010

〔4〕 Marko Wolf, Andr´e Weimerskirch, Christof Paar, "security in automotive bus systems," In Proceedings of the Workshop on Embedded Security in Cars 2004, pp.1-13, 2004.

〔5〕 Xiao Ni, "AES Security Protocol Implementation for Automobile Remote Keyless System," in Proceedings of IEEE 65th Vehicular Technology Conference, 2007. VTC2007-Spring. pp. 2526-2529, April 2007

〔6〕 Gang-seok Kim, "Vehicle ECU through CAN communication from eavesdropping and manipulation of the analysis of the possibility of external threats", Korea University, 2011

〔7〕 Karl Koscher, Alexei Czeskis, Franziska Roesner, "Experimental Security Analysis of a Modern Automobile," IEEE Symposium on Security and Privacy, pp. 447 - 462, 2010. May

〔8〕 T. Hoppe, S. Kiltz, A. Lang, and J. Dittmann. "Exemplary Automotive Attack Scenarios: Trojan horses for Electronic Throttle Control System (ETC) and replay attacks on the power window system," in Proceedings of 23th VDI/VW Gemeinschaftstagung Automotive Security, pp. 165-183, 2007.

〔9〕 Nilsson, Dennis K., "Vehicle ECU classification based on safety-security characteristics," Road Transport Information and Control - RTIC 2008 and ITS United Kingdom Members' Conference, IET, pp.1-7, May. 2008.

〔10〕 Aditya P. Mathur, "Foundations of Software Testing", Pearson Education, pp.193-278, 2008.

〔11〕 M. Sutton, A. Greene, P. Amini, "Fuzzing: Brute Force Vulnerability Discovery," Addison-Wesley, pp. 386-417, 2007

〔12〕 G. Devarajan, "Unraveling SCADA Protocols: Using Sulley Fuzzer", DEFCON 15, Aug 2007.

〔13〕 Hye-ryun Lee, Seung-hun Shin, Kyung-hee Choi, Ki-hyun Chung, Seung-kyu Park, Jun-yong Choi, "Detecting the vulnerability of software with cyclic behavior using Sulley," in Proceedings of the Advanced Information Management and Service (ICIPM), 2011 7th International Conference on, pp. 83-88, Dec. 2011.

저 자 소 개

이 혜 련

2006: 조선대학교
인터넷소프트웨어공학과 공학사.
2008: 아주대학교
정보통신전문대학원 공학석사.
현 재: 아주대학교
컴퓨터공학과 박사수료
관심분야: 보안테스팅, 소프트웨어공학
Email : cocom12@ajou.ac.kr

김 경 진

2012: 아주대학교 전자공학과 공학사
현 재: 아주대학교 전자공학과 석사과정
관심분야: 임베디드 시스템, 테스팅,
실시간 시스템 등
Email : klm0012@ajou.ac.kr

최 경 희

1976: 서울대학교 수학교육과 학사.
1979: 프랑스
그랑데꼴 Enseeiht대학 석사.
1982: 프랑스 Paul Sabatier대학
정보공학부 박사
현 재: 아주대학교 컴퓨터공학과 교수
관심분야: 컴퓨터공학, 운영체제,
실시간시스템, 테스팅 등
Email : khchoi@ajou.ac.kr

정 기 현

1984: 서강대학교 전자공학과 학사.
1988: 미국 Ilinois주립대 EECS(석사)
1990: 미국 Purdue대학
전기전자공학부 박사
현 재: 아주대학교 전자공학과 교수
관심분야: 컴퓨터구조, VLSI 설계,
실시간시스템, 테스팅 등
Email : khchung@ajou.ac.kr