Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.6, Dec. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.6.1225

임베디드 디바이스 펌웨어의 웹 인터페이스 취약점 식별을 위한 에뮬레이션 기반 퍼징 기법*

허 정 민,[1†] 김 지 민,[1] 지 청 민,[1] 홍 만 표[2‡]
[1]아주대학교 컴퓨터공학과, [2]아주대학교 사이버보안학과

Emulation-Based Fuzzing Techniques for Identifying Web Interface Vulnerabilities in Embedded Device Firmware*

Jung-Min Heo,[1†] Ji-Min Kim,[1] Cheong-Min Ji,[1] Man-Pyo Hong[2‡]
[1]Department of Computer Engineering, Ajou University,
[2]Department of Cyber Security, Ajou University

요 약

임베디드 디바이스의 대중화로 인해 펌웨어의 보안은 더욱 중요해지고 있다. 유무선 공유기와 같은 네트워크 장비는 내재된 펌웨어의 웹 인터페이스 취약점을 통해 외부의 공격자로부터 피해를 받을 수 있기 때문에 빠르게 찾아내어 제거해야 한다. 이전 연구인 Firmadyne 프레임워크는 펌웨어를 에뮬레이션 한 뒤 취약점을 찾아내기 위한 동적 분석 방법을 제안한다. 그러나 이는 도구에서 정의된 분석 방법대로만 취약점 점검을 수행하므로 찾을 수 있는 취약점의 범위가 한정되어 있다. 본 논문에서는 소프트웨어 보안 테스트 기술 중 하나인 퍼징을 통해 에뮬레이션 기반 환경에서의 퍼징 테스트를 수행한다. 또한 효율적인 에뮬레이션 기반 퍼징을 위해 Fabfuzz 도구를 제안한다. 실험을 통해 확인한 결과 기존 도구에서 식별했던 취약점뿐만 아니라 다른 유형의 취약점도 발견할 수 있다.

ABSTRACT

The security of the firmware is more important because embedded devices have become popular. Network devices such as routers can be attacked by attackers through web application vulnerabilities in embedded firmware. Therefore, they must be found and removed quickly. The Firmadyne framework proposes a dynamic analysis method to find vulnerabilities after emulating firmware. However, it only performs vulnerability checks according to the analysis methods defined in the tool, thus limiting the scope of vulnerabilities that can be found. In this paper, fuzzing is performed in emulation-based environment through fuzzing, one of the software security test techniques. We also propose a Fabfuzz tool for efficient emulation based fuzzing. Experiments have shown that in addition to the vulnerabilities identified in existing tools, other types of vulnerabilities have been found.

Keywords: Embedded Device, Dynamic Analysis, Firmware Emulation, Fuzzing

Received(08. 26. 2019), Modified(10. 10. 2019), Accepted(10. 11. 2019)

* 이 논문은 2019년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업의 지원을 받아 수행된 기초연구사업임(NRF-2017R1D1A1B03032734).

* 본 연구는 국토교통부 및 국토교통과학기술진흥원의 연구비지원(18TLRP-B117133-03)으로 수행된 연구임.

† 주저자, jungmingg@ajou.ac.kr

‡ 교신저자, mphong@ajou.ac.kr(Corresponding author)

I. 서 론

최근 IoT(Internet of Things)의 발전과 함께 임베디드 디바이스는 일상생활에 없어서는 안될 필수 소형 가전으로 자리매김 되었다. 2017년 가트너 보고서[1]에 따르면 임베디드 디바이스의 수는 시간이 지남에 따라 계속적으로 증가하고 있으며 2020년에 200억 개 이상의 장치가 네트워크에 연결되어 사용될 것으로 예상하고 있다. 또한 임베디드 디바이스의 증가와 더불어 보안 취약점을 이용한 공격 사례(예:Mirai[2])가 발생되고 있기 때문에 임베디드 디바이스의 취약점을 찾아내는 연구가 활발히 진행되고 있다[3][4][5].

유무선 공유기와 라우터 같은 네트워크 장비는 공개된 공간에서 많이 사용되고 있으며 이 장비들은 외부에서 들어오는 트래픽을 받아서 처리해야하기 때문에 안전하지 않은 펌웨어를 사용할 경우 웹 인터페이스(httpd, lighttpd)의 알려진 보안 위협 및 알려지지 않은 보안 위협에 노출될 수 있다. 또한 보안 취약점을 이용한 봇넷 공격[2]의 대상이 될 수 있으므로 이러한 공격을 예방하기 위해서는 공격자보다 빠르게 웹 인터페이스의 취약점을 찾아내고 임베디드 디바이스의 펌웨어를 수정하여 결함을 제거해야만 한다. 그러나 수천 개에 달하는 임베디드 디바이스 펌웨어의 취약점을 식별하는 것은 하드웨어에 의존적이기 때문에 물리적으로 불가능에 가깝다.

최근 연구[3][4][5]들에 따르면 이러한 물리적인 한계점을 극복하기 위해 하드웨어 및 소프트웨어 에뮬레이션 기법을 사용하였다. 특히 Chen et al. 의 Firmadyne[5] 도구는 전체 시스템 에뮬레이션 기능을 사용하여 제조사의 하드웨어 없이 펌웨어만으로도 실제 장비가 동작하는 것처럼 에뮬레이션 하였으며 실행된 프로그램을 대상으로 동적 분석을 수행하도록 구현한 프레임워크이다. 하지만 Firmadyne의 동적 분석 방법은 도구에서 정의된 취약점에 대해서만 점검이 가능하기 때문에 찾을 수 있는 취약점의 범위가 한정되어 있다.

소프트웨어의 취약점을 찾기 위한 방법으로는 보안 테스트 기술 중 하나인 퍼징(fuzzing)이 있으며 퍼징은 찾을 수 있는 취약점의 범위가 한정되어 있지 않기 때문에 알려진 취약점 및 알려지지 않은 취약점을 찾을 수 있다. 퍼징 테스트를 위한 기존 도구[6][7][8]들은 예기치 않은 동작(crash)을 생성하기 위해 대상 프로그램에 임의의 입력 값을 주입하여 소프트웨어를 테스트하는 기술이며 대상 프로그램에서 예기치 않은 동작이 발생될 경우 분석가는 이를 분석하여 취약점을 발견할 수 있다. 따라서 본 논문에서는 임베디드 디바이스 펌웨어의 웹 인터페이스 취약점 식별을 위한 방법으로 Firmadyne의 전체 시스템 에뮬레이션 기능과 스마트 퍼징을 수행하기 위해 오픈 소스로 공개된 Boofuzz[6]의 네트워크 프로토콜 퍼징 기능을 통합하여 에뮬레이션 기반 퍼징에 초점을 맞추어 구현된 Fabfuzz를 제안한다.

2장에서는 펌웨어 에뮬레이션 환경을 구성해 주는 Firmadyne 프레임워크에 대해 설명하고 퍼징 기법 중 하나인 네트워크 프로토콜 퍼징과 에뮬레이션 퍼징 도구인 Firm-AFL[9]에 대해 설명한다. 3장에서는 Fabfuzz 도구의 전체적인 흐름과 각 기능의 구현방법에 대해 기술하며 4장에서는 실험을 통해 제안된 도구의 테스트 및 성능 평가 및 성능 비교 그리고 추후 연구에 대해 논의한다. 마지막 5장에서는 본 논문에 대한 결론으로 마무리 한다.

II. 관련 연구

2.1 Firmadyne 프레임워크

Firmadyne은 리눅스 기반 펌웨어를 대상으로 한 최초의 자동화된 동적 분석 프레임워크이다. 기존 동적 분석 방식과 달리 여러 제조사의 펌웨어를 웹 크롤링을 통해 자동으로 수집하고 수집된 펌웨어 이미지로부터 루트 파일 시스템을 추출하여 각 아키텍처에 대한 정보를 얻는다.

Firmadyne은 QEMU[10] 전체 시스템 에뮬레이터를 사용하여 추출된 루트 파일 시스템을 사전 컴파일 된 커널(ARM, MIPS)를 사용하여 아키텍처별 커널 부팅을 실행한다. 에뮬레이션 된 펌웨어 이미지가 부팅되면 프레임워크 내에 정의된 동적 분석 방법을 사용하여 다음 세 가지 방법으로 취약점 점검을 수행한다.

첫 번째는 엑세스 가능한 웹 페이지를 탐색하는 방법이다. 이는 다양한 정보 유출, 버퍼 오버플로우 및 명령 주입 취약점을 탐지하기 위해 LAN 인터페이스에서 공개적으로 액세스할 수 있는 웹 페이지를 펌웨어 이미지로 찾는 분석 방법이다.

두 번째는 Snmpwalk 도구를 사용하여 인증되지 않은 SNMP(Simple Network Management Protocol) 정보를 덤프한다. 이를 통해 장비에 포함

Table 1. CVE(Common Vulnerabilities and Exposures) List

CVE No.	Description
CVE-2016-1555[11]	There is a web page capable of sending POST without authentication. This allows command injection into the system.
CVE-2016-1556[12]	An unauthenticated webpage exists that discloses wireless WPS PIN information
CVE-2016-1557[13] CVE-2016-1559[14]	Administrator ID and password information is included in the Management Information Base (MIB) dumped through SMNP.
CVE-2016-1558[15]	'Dlink_uid' cookie has a buffer overflow vulnerability

된 중요 정보의 존재 여부를 확인할 수 있다.

마지막 세 번째는 메타스플로잇 프레임워크[16](Metasploit Framework)에서 주로 알려진 악용 사례를 사용하여 펌웨어 이미지를 검사하는 방법이다. 메타스플로잇 프레임워크 내에서 특정 원격 셸 페이로드를 60개 정도 지정하였으며 이를 순차적으로 실행하며 점검한다.

Firmadyne 프레임워크의 동적 분석 방법으로 식별된 취약점 리스트는 Table 1.과 같다.

2.2 에뮬레이션 기반 퍼징

Firm-AFL[9]은 임베디드 디바이스의 취약점 식별을 위한 도구로 Firmadyne과 AFL(American Fuzzy Lop)[17]을 사용하여 임베디드 디바이스 펌웨어를 에뮬레이션 한 뒤 퍼징하는 도구이다. 그러나 기존 Firmadyne 에뮬레이션은 전체 시스템 에뮬레이션으로 수행되기 때문에 AFL의 퍼징 속도가 느리다. 이러한 문제를 Firm-AFL 도구에서는 퍼징 프로세스 절차를 개선한 Firmadyne을 통해 전체 시스템 에뮬레이션과 유저모드 에뮬레이션을 병행하여 속도문제를 해결한다. 위 환경에서 대상 펌웨어 프로그램(httpd, cgibin, dnsmasq dropbear 등)에 대해 개선 전과 후의 퍼징 속도를 비교한 결과 최소 3배에서 최대 13배 빠른 속도로 취약점을 찾을 수 있으며 실험 결과를 통해 임베디드 디바이스 펌웨어의 취약점을 찾기 위한 하나의 방법으로 펌웨어 에뮬레이션 기반 퍼징이 효과적임을 알 수 있다.

2.3 네트워크 프로토콜 퍼징

퍼징은 대상 프로그램의 예기치 않은 동작을 발견하기 위해 비정상적인 입력을 생성한 뒤 대상 프로그램에 전송하여 버그를 찾거나 개발 단계에서 발견하지 못한 결함을 찾아내는데 사용되는 도구이다. 퍼징은 도구에 따라 dumb 퍼저(dumb fuzzer)와 스마트 퍼저(smart fuzzer)로 구분할 수 있다.

dumb 퍼저는 대상 프로그램에 대한 정보가 없는 경우 가능한 모든 네트워크 프로토콜에 대해서 임의의 입력 값을 생성하여 전송하는 방법이며 이는 개발하기 쉽고 대상이 한정되어 있지 않다는 장점이 있다. 하지만 스마트 퍼저보다 버그를 발견할 확률이 적으며 시간이 오래 걸린다[18].

스마트 퍼저는 대상 프로그램의 프로토콜 정보를 알고 있는 경우에 사용한다. 대표적인 스마트 퍼저에는 Boofuzz[6], Peach[7], SPKIE[8] 등이 있으며 각 도구들은 프로토콜 형식 기반으로 각 필드를 변이 시키면서 테스트를 수행하여 효과적이지만 스마트 퍼저를 사용하기 위해서는 프로토콜 형식 파일을 테스터가 직접 작성해야만 하며 각 도구별로 프로토콜 형식 파일의 정의가 다르기 때문에 사용자는 이를 숙지하여 구현해야 한다[19].

Peach 퍼저의 경우 프로토콜 형식 파일을 XML(eXtensible Markup Language)로 구현해야 하고, Boofuzz 퍼저는 파이썬 스크립트를 통해 구현해야 한다. 이러한 이유로 스마트 퍼저 사용에 대한 숙련도가 부족하다면 퍼징 테스트를 위한 프로토콜 형식 파일 작성 시 오류가 발생하기 쉬우며 복잡한 작업이 될 수 있다.

하지만 이전 연구[4][18][19][20]를 통해 네트워크 프로토콜 퍼징 수행 시 호스트와 클라이언트 간 주고받은 실제 통신 패킷을 기반으로 테스트케이스를 생성하는 것이 효율적임을 알 수 있기 때문에 본 논문에서는 패킷 수집 및 테스트케이스 생성을 자동화하여 스마트 퍼징 기법을 조금 더 효율적으로 수행할

수 있다.

III. 제안 방법

본 논문에서는 에뮬레이션 환경에서의 퍼징 기법 및 도구인 Fabfuzz를 제안한다. 먼저 임베디드 디바이스의 보급이 증가하고 있기 때문에 취약점 분석을 위해 필요한 하드웨어를 모두 수집하는 것은 현실적으로 불가능하므로 리눅스 기반의 임베디드 디바이스 제조사의 펌웨어를 에뮬레이션 한다.

다음으로 기존 스마트 퍼저 도구인 Boofuzz의 경우 프로토콜 형식 파일을 사용자가 직접 작성해야 하기 때문에 복잡하고 지루한 작업을 수행해야만 퍼징 테스트가 가능하다. 그러나 제안된 도구인 Fabfuzz는 대상 펌웨어의 웹 인터페이스로부터 네트워크 통신 패킷을 수집하여 자동으로 프로토콜 형식 파일을 생성하는 방법을 사용하므로 두 가지의 이점이 존재한다. 하나는 사용자가 직접 프로토콜 형식 파일을 생성하지 않아도 된다는 점이며 다른 하나는 프로토콜에 대한 이해를 바탕으로 스마트 퍼저에 숙련된 사용자가 아니라면 사용자가 직접 작성한 임의의 프로토콜 형식 파일보다 자동으로 생성된 파일이 조금 더 정밀한 테스트가 가능하다는 점이다.

Fabfuzz는 Controller에서 모든 작업을 수행한다. 대상 펌웨어가 정해지면 Controller 내의 모듈을 통해 펌웨어의 이미지를 에뮬레이션하고 에뮬레이션 된 웹 인터페이스를 대상으로 네트워크 통신 패킷을 수집한다. 그리고 수집된 패킷 데이터를 퍼징 가능한 프로토콜 형식 파일로 만든 후 퍼징을 수행한다. 퍼징 테스트의 모든 기록은 실시간으로 모니터링 되며 예기치 않은 동작이 발생될 경우 재연을 위한 로그를 기록한다.

Fig. 1.은 제안된 도구의 전체 다이어그램으로 3장의 각 절에서는 Fabfuzz Controller의 모듈별 구현 방법에 대해 자세히 설명한다.

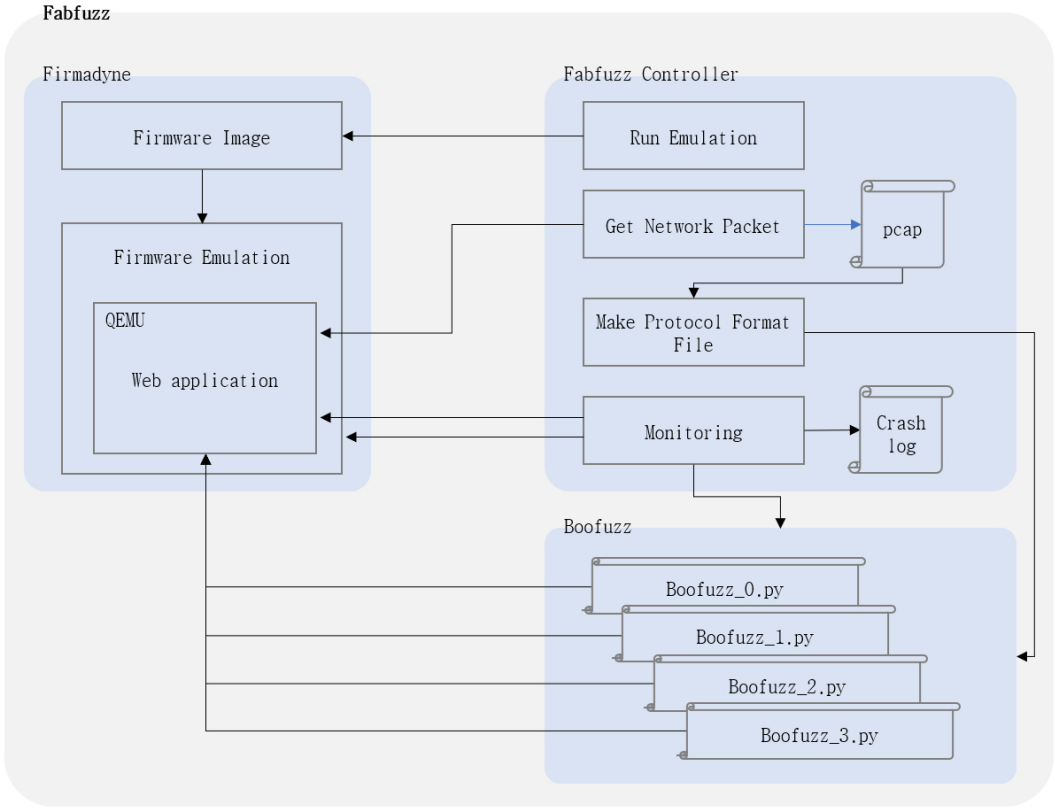

Fig. 1. Diagram of Fabfuzz showing the emulation based fuzzing

3.1 펌웨어 에뮬레이션

공유기와 라우터와 같은 임베디드 디바이스에 내재된 펌웨어들은 대부분 ARM, MIPS와 같은 리눅스 기반 운영체제를 사용한다.

이러한 펌웨어들은 Firmadyne의 전체 시스템 에뮬레이션 기능을 통해 물리적인 장비 없이 독립적인 프로세스로 실행 가능한 이미지로 형태로 만들어진다. 분석 대상 펌웨어가 지정되면 Fabfuzz에서는 Run Emulation 모듈을 호출하여 펌웨어 이미지를 에뮬레이션 한다.

3.2 네트워크 통신 패킷 수집

네트워크 프로토콜 퍼징 시 효율적인 퍼징 테스트 케이스를 생성하기 위해서는 수동으로 작성하는 것보다 실제 네트워크 통신 패킷을 기반으로 생성되는 것이 효과적임을 이전 연구[4][18][19][20]를 통해 알 수 있다. Fabfuzz에서는 대상 웹 응용프로그램의 네트워크 통신 패킷을 수집하기 위해 다음과 같은 두 가지 도구를 사용한다.

첫 번째는 tshark[21] 도구이다. tshark는 wireshark[22] 도구의 커맨드라인 버전으로 네트워크 인터페이스 이름과 프로토콜 그리고 대상 프로그램의 IP 주소만 주어진다면 명령어를 사용하여 네트워크 통신 패킷을 수집할 수 있다. 수집된 데이터는 pcap 포맷으로 분류된다.

두 번째는 파이썬에서 제공하는 selenium 패키지를 사용한다[23][24]. selenium은 웹 테스트 자동화를 위한 도구로 파이썬 스크립트를 작성하여 selenium 웹 드라이버를 통해 대상 웹 인터페이스에 요청을 보내는 등 스크립트 기반으로 웹 페이지를 동작 시킬 수 있다. 또한 자동으로 웹 페이지를 로드할 수 있기 때문에 에뮬레이션된 웹 인터페이스를 대상으로 사용자는 웹 페이지를 변경, 조작할 수 있게 되므로 다양한 네트워크 통신 패킷이 생성되기 때문에 앞서 소개한 tshark 도구와 함께 사용하면 자동으로 패킷 데이터를 얻을 수 있다.

3.3 프로토콜 형식 파일 생성

Boofuzz는 파이썬 라이브러리 형태로 구성되며 프로토콜 형식 파일을 정의하기 위해 사용자는 퍼징 테스트를 위한 파이썬 스크립트를 작성해야만 한다. 하지만 제안된 도구를 사용하게 되면 3.2절을 통해 수집된 네트워크 패킷 파일의 데이터를 기반으로 프로토콜 형식 파일을 자동으로 생성할 수가 있다. 다음은 프로토콜 형식 파일을 만들기 위한 방법이다.

먼저 수집된 네트워크 패킷 파일을 pyshark[25] 도구를 사용하여 로드한다. 파이썬 모듈인 pyshark는 tshark 또는 wireshark를 통해 수집된 pcap 파일을 구문 분석해 주는 도구이며 수집된 네트워크 통신 패킷에는 동일한 구조로 된 패킷이 많기 때문에 중복된 구조의 패킷을 제거하는 작업이 필요하다. 이를 제거하지 않는 경우 모든 패킷을 테스트케이스로 만들게 되며 프로토콜 형식 파일이 각 패킷 별로 생성되기 때문에 퍼징 테스트 시간이 굉장히 오래 걸릴 수 있으므로 구문 분석된 패킷에서 동일한 구조의 패킷 제거 작업을 선행해야만 한다.

선행된 패킷 제거 작업이 끝난 후 퍼징 테스트를 위해 해당 패킷 내의 헤더 값과 파라미터 값을 기반으로 Boofuzz에서 실행 할 수 있는 프로토콜 형식 파일로 변환한다. 해당 파일에는 다양한 테스트 케이스가 포함된다.

3.4 퍼징 및 모니터링

만약 Fabfuzz를 사용하지 않고 펌웨어를 에뮬레이션 한 뒤 퍼징을 통해 취약성 테스트를 수행하는 경우 다음과 같은 문제점이 발생될 수 있다.

첫 번째는 임의로 생성된 퍼징 테스트케이스를 웹 인터페이스에 주입 시 에뮬레이션 된 프로세스는 예기치 않은 동작을 발생 시킬 수 있으며 이 경우 에뮬레이션 프로세스를 다시 실행시켜주어야 하는데 Firmadyne에서는 이러한 기능을 제공하지 않는다. 두 번째는 예기치 않은 동작으로 인해 대상 프로세스에 문제가 발생 되었다면 퍼저는 이를 감지하고 로그를 생성해야 하지만 이러한 기능도 존재하지 않는다. 세 번째는 특정 테스트케이스의 예기치 않은 동작이 발생되고 난 후 퍼징 테스트를 다시 실행하는 경우 이전에 문제가 생겼던 직후부터 테스트가 이어져서 실행되어야 하나 처음부터 다시 수행하는 문제가 있다. Fabfuzz는 에뮬레이션 기반 퍼징을 위한 도구로 펌웨어 에뮬레이션, 웹 인터페이스의 상태 점검 및 퍼징을 원활하게 수행하기 위해 Fig. 2.처럼 프로세스 모니터링이 가능하다. 대상 웹 인터페이스의 경우 퍼징 테스트를 수행하게 되면 예기치 않은 동작으로 인해 해당 프로세스가 죽거나 정상적으로 응답

```
[2019-08-03 15:01:30.732313] Process state(boofuzz)    : Test Case 9205 of 13789
[2019-08-03 15:01:31.741925] Process state(boofuzz)    : Alive
[2019-08-03 15:01:32.758525] Process state(emulator)   : Alive
[2019-08-03 15:01:33.782899] Process state(httpd)      : Alive
[2019-08-03 15:01:34.784361] Process state(boofuzz)    : Test Case 9227 of 13789
[2019-08-03 15:01:35.798443] Process state(boofuzz)    : Alive
[2019-08-03 15:01:36.807132] Process state(emulator)   : Alive
[2019-08-03 15:01:36.809280] Detected a crash          : _httpd/949:_potentially_unexpected_fatal_signal_11.
```

Fig. 2. Monitoring and Crash detection in Fabfuzz

하지 않는 경우가 많이 발생된다. 이를 해결하기 위해 Fabfuzz에서는 에뮬레이션 및 웹 응용프로그램 프로세스를 모니터링 하여 상태 및 응답을 주기적으로 체크하도록 하여 정상적인 경우가 아닌 경우 웹 인터페이스가 응답할 때까지 기다리거나 에뮬레이션을 다시 실행하여 프로세스를 정상 구동시키도록 한다. 퍼징 테스트를 수행하는 도중 웹 인터페이스에서 예기치 않은 동작이 발생되면 모니터링 도구는 이를 감지하여 해당 테스트케이스를 로그로 기록한다. 또한 사용자는 해당 로그를 통해 크래시를 재연할 수 있다. 모니터링의 최종 목표는 정해진 테스트케이스를 마지막까지 원활하게 모두 진행되도록 하는 것이며 퍼징 테스트 중 대상 웹 인터페이스의 이상 동작이 발생되는 경우 퍼징 테스트케이스를 문제 발생 지점의 테스트케이스부터 다시 재실행 하도록 하여 효율적인 테스트를 할 수 있도록 도와준다.

IV. 실험 및 평가

본 4장에서는 Firm-AFL에서 퍼징을 통해 크래시가 발생된 3개의 펌웨어와 Firmadyne에서 동적 분석 방법으로 취약점이 식별된 28개의 펌웨어를 대상으로 3장에서 제안한 도구를 사용하여 퍼징 테스트 및 평가를 수행한다. 실험에 사용된 시스템은 Intel Core i7-8550U CPU 1.8GHz와 4GB RAM이 탑재된 16.04 Ubuntu 64bit 운영체제를 사용한 가상머신이다.

4.1 에뮬레이션 기반 퍼징 테스트

에뮬레이션 기반 퍼징 테스트를 수행하기 위해서는 에뮬레이션 된 웹 인터페이스의 프로세스가 실행되어 있어야 하며 퍼징을 위한 프로토콜 형식 파일이 필요하다. 펌웨어 별로 프로토콜 형식 파일은 복수개로 존재할 수 있으며 파일 개수에 비례하여 테스트케이스의 개수가 정해지게 된다. 만약 3.2절의 네트워크 통신 패킷 수집을 통해 소량의 패킷을 수집한 경우 프로토콜 형식 파일의 개수는 적게 만들어지고 그로인해 테스트케이스의 개수도 줄어들게 된다. 대상 펌웨어의 테스트케이스가 모두 생성되면 퍼징 테스트를 수행하기 위해 Boofuzz는 프로토콜 형식 파일을 하나씩 실행하며 대상 웹 인터페이스에 테스트케이스를 순차적으로 주입한다. 퍼징 테스트 중 예기치 않은 동작 발생 시 Controller의 모니터링 모듈은 에뮬레이션 된 커널의 이벤트 로그를 감지하고 크래시가 발생된 로그를 기록한다.

4.2 동일한 웹 인터페이스 프로그램 성능 비교

Table 2.는 동일한 펌웨어에서 동일한 프로그램을 대상으로 Fabfuzz와 Firm-AFL의 크래시가 발생된 시간을 비교한 결과이다. 본 연구에서는 실제 패킷을 기반으로 퍼징 테스트 데이터를 생성할 경우 조금 더 효과적인 퍼징을 수행할 수 있을 것으로 생각하여 스마트 퍼저인 Boofuzz를 통해 퍼징을 수행한다. 그 결과 Table 2.의 결과와 같이 DLink 제

Table 2. Fabfuzz vs Firm-AFL Performance Comparison

Manufacturer	Model	Ver.	Device	Program	Firm-AFL Time to Crash	Fabfuzz Time to Crash
TPLink	WR840N	V4	Router	httpd	〉24h	〉24h
DLink	DIR-825	2.02	Router	httpd	22h3min	2h10min
DLink	DAP-2695	1.11.RC04	Router	httpd	2h32min	2h18min

조사의 펌웨어에서 보다 높은 성능 향상을 보임을 확인할 수 있다.

4.3 알려진 취약점에 대한 퍼징 성능 테스트

Table 3.은 Firmadyne에서 동적 분석 방법으로 취약점이 식별된 펌웨어와 Fabfuzz의 퍼징 테스트로 발생된 크래시 결과를 보여준다. Table 1.에서 소개한 CVE-2016-1555, CVE-2016-1556, CVE-2016-1557, CVE-2016-1559의 경우 특정 웹 페이지의 인증 관련 취약점 및 SNMP dump 점검을 통한 취약점으로 퍼징 테스트를 통해 취약점을 발견할 수 없었다.

CVE-2016-1558은 버퍼 오버 플로우 취약점으로 퍼징 테스트 시 Dlink 제조사의 펌웨어 7개 중 6개에서 다수의 크래시가 발생됨을 확인하였으며 Fabfuzz에서 기록한 크래시 로그를 통해 재연한 결과 CVE-2016-1558 취약점과 동일한 것으로 확인되었다.

다음은 Netgear 제조사의 펌웨어 21개에 대해 퍼징 테스트를 수행하였다. Table 3.의 결과에 따라 일부 크래시가 발생된 펌웨어를 분석한 결과 Out of Memory 취약점을 11개 펌웨어에서 발견할 수 있었으며, 기존 도구인 Firmadyne에서 식별되지 않았던 취약점을 제안된 도구를 사용하여 새로 발견했다고 볼 수 있다.

4.4 평가 결과 및 추후 연구

Table 2.의 TPLink 제조사 펌웨어의 경우 모두

Table 3. Target Firmware and Fabfuzz execution result

Manufacturer	Device	Ver.	Firmadyne	Fabfuzz		
			CVE No.	No. of Test case	No. of Crash	Crash type
DLink	DAP-1353	3.15	CVE-2016-1559	33,293	0	
	DAP-2230	1.02	CVE-2016-1558	64,633	1,103	BOF
	DAP-2330	1.06	CVE-2016-1558	71,616	1,078	BOF
	DAP-2660	1.11	CVE-2016-1558	64,633	911	BOF
	DAP-2695	1.16	CVE-2016-1558	64,633	772	BOF
	DAP-3320	1.00	CVE-2016-1558	64,633	824	BOF
	DAP-3662	1.01	CVE-2016-1558	69,229	652	BOF
Netgear	WN604	2.1	CVE-2016-1555,1556	40,838	0	
	WN604	2.3.1	CVE-2016-1555,1556	24,608	0	
	WN604	2.3.2	CVE-2016-1555,1556	37,962	0	
	WN604	3.0.0	CVE-2016-1555,1556	40,838	0	
	WN604	3.0.2	CVE-2016-1555,1556	40,838	0	
	WNAP210	2.1.1	CVE-2016-1555,1556	59,125	0	
	WNAP210	2.1.4	CVE-2016-1555,1556	40,838	0	
	WNAP320	2.0	CVE-2016-1555,1557	44,961	2	OOM
	WNAP320	2.0.3	CVE-2016-1555,1557	44,901	1	OOM
	WND930	2.0.0	CVE-2016-1556	48,782	2	OOM
	WNDAP350	2.0.27	CVE-2016-1555	44,901	1	OOM
	WNDAP350	2.0.9	CVE-2016-1555	23,170	0	
	WNDAP350	2.1.2	CVE-2016-1555,1556	44,901	4	OOM
	WNDAP350	2.1.6	CVE-2016-1555,1556	40,838	1	OOM
	WNDAP350	2.1.7	CVE-2016-1555,1556	24,608	0	
	WNDAP360	2.0.4	CVE-2016-1555,1557	40,838	1	OOM
	WNDAP360	2.0.7	CVE-2016-1555,1557	40,838	0	
	WNDAP360	2.1.1	CVE-2016-1555,1556	40,838	1	OOM
	WNDAP360	2.1.5	CVE-2016-1555,1556	40,838	4	OOM
	WNDAP360	2.1.6	CVE-2016-1555,1556	40,838	6	OOM
	WNDAP360	2.1.7	CVE-2016-1555,1556	44,901	2	OOM

BOF : Buffer over flow
OOM : Out of memory

크래시를 발견하지 못하였는데 분석 결과 해당 웹 인터페이스의 동작을 유발하기 위해서는 실행을 위한 세션 값이 필요하다. 그러나 웹 인터페이스 프로그램에서 세션 값을 수시로 변경하기 때문에 네트워크 프로토콜 퍼징 시 웹 인터페이스에서 패킷을 처리하지 않은 것으로 보인다. 이 문제를 해결하기 위해서는 퍼징 패킷을 보낼 때 현재 필요한 웹 인터페이스의 세션 값을 취득하고 패킷에 반영하여 퍼징 테스트를 수행한다면 보다 향상된 네트워크 퍼징을 수행하는 것이 가능하다.

다음은 Table 3.의 Netgear 제조사 펌웨어와 Dlink 제조사 펌웨어에서 발생된 크래시 수의 결과를 비교해보면 Dlink 펌웨어에서 발생된 크래시의 수가 Netgear에서 발생된 크래시 수 보다 훨씬 높다는 것을 확인할 수 있다. 이는 Dlink 제품이 보안에 취약해서가 아니라 발생된 크래시의 종류가 다르기 때문이다. 3.3절에서 선행한 동일 구조의 패킷 제거 작업으로 인해 프로토콜 형식 파일을 생성할 때에는 서로 다른 구조의 헤더끼리 파일로 만들어지므로 각 파일의 헤더 구조는 다르지만 같은 헤더 값을 가진 파일이 여러 개가 존재할 수 있다. 이러한 문제로 특정 헤더 값에서 버퍼 오버 플로우와 같은 크래시가 발생 되었을 경우 동일한 헤더 값을 가진 모든 파일에서 퍼징 테스트하는 현상이 발생되어 크래시의 개수가 증가한다. Netgear 제조사 펌웨어의 경우는 특정 헤더 값에 의한 크래시가 아니기 때문에 이와 관련이 없다. Table 3.의 더 나은 퍼징 성능을 위해서는 동일 구조의 패킷 제거 작업뿐만 아니라 각 프로토콜 형식 파일의 헤더 값 중복을 고려하여 가능한 하나의 파일에서 많은 테스트케이스가 수행될 수 있도록 중복된 헤더 값 제거를 위한 작업이 추가적으로 필요하다.

마지막으로 Firmadyne을 통해 알려진 취약점인 CVE-2016-1555,1556,1557,1559에 대해서는 제안된 도구의 퍼징을 통해 확인할 수가 없었다. 이는 웹 페이지의 인증과 관련된 취약점 문제로 퍼징의 경우 무작위 입력 값을 통해 특정 프로그램의 이상 현상을 감지하는 것을 목적으로 수행되기에 이와 같은 취약점을 점검하기 위해서는 Firmadyne에서 제공하는 동적 분석 기능을 사용하여 추가적인 취약점 점검 절차를 갖는 것이 필요할 것으로 판단된다.

V. 결 론

본 연구는 임베디드 디바이스의 취약점을 식별하고자 기존 하드웨어 종속적인 부분을 제거하기 위해 에뮬레이션 기반 환경에서 진행하였으며 기존 도구[5]의 취약점 식별 방법과 달리 새로운 취약점을 찾기 위한 방법으로 퍼징 기법을 사용하여 단 하나의 대상 임베디드 디바이스의 장비 없이 제조사의 펌웨어만으로 취약점을 식별할 수 있었다. 논문에서 제안된 도구인 Fabfuzz는 펌웨어 에뮬레이션 실행, 테스트케이스 생성 및 모니터링을 수행함으로써 에뮬레이션 환경에서 퍼징하기 위한 도구로 활용된다. 4장의 에뮬레이션 환경에서의 퍼징 테스트 결과에 따르면 제안된 도구는 CVE-2016-1558 취약점과 동일한 버퍼 오버 플로우 크래시를 식별할 수 있었으며 추가적으로 새로운 유형의 Out of Memory 취약점을 발견할 수 있었다. 이는 Firmadyne의 동적 분석 방법과 함께 사용하게 될 시 보안 인증과 관련된 취약점뿐만 아니라 제안된 방법을 통해 더 많은 취약점을 점검할 수 있다는 의미이며 보다 더 향상된 취약점 식별 성능을 보일 것으로 기대된다.

References

[1] Gartner, “Internet of Things (IoT) Market,” https://www.gartner.com/newsroom/id/3598917, Feb. 2017.

[2] B. Herzberg, D. Bekerman, and I. Zeifman, “Breaking Down Mirai: An IoT DDoS Botnet Analysis,” https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html, Nov. 2017.

[3] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, “AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems’ Firmwares,” in Proceedings of the 21st Annual Network and Distributed System Security Symposium ,NDSS ,Feb. 2014.

[4] Jiongyi Chen,Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. “IoTFuzzer: Discovering memo

ry corruptions in iot through app-based fuzzing," In Networked and Distributed System Security Symposium, NDSS, Feb. 2018.
〔5〕 Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. "Towards automated dynamic analysis for Linux-based embedded firmware," In Network and Distributed System Security Symposium, NDSS, Feb. 2016.
〔6〕 Joshua Pereyda, "boofuzz Documentation," https://buildmedia.readthedocs.org/media/pdf/boofuzz/latest/boofuzz.pdf, Aug. 2019.
〔7〕 M. Eddington, "Peach fuzzing platform," http://community.peachfuzzer.com/WhatIsPeach.html, Feb 2014.
〔8〕 D. Aitel, "An introduction to SPIKE, the fuzzer creation kit," in Proceedings of the Black Hat USA, 2001.
〔9〕 Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu and Limin Sun, "Firm-AFL: High-Throughput Greybox Fuzzing of IoT Firmware via Augmented Process Emulation," in Proceedings of the USENIX 2019 Annual Technical Conference, Aug, 2019
〔10〕 F. Bellard, "QEMU, a fast and portable dynamic translator," in Proceedings of the USENIX 2005 Annual Technical Conference, pp. 41-41, Apr. 2005,
〔11〕 CVE, "https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1555", Apr. 2017.
〔12〕 CVE, "https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1556", Apr. 2017.
〔13〕 CVE, "https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1557", Apr. 2017.
〔14〕 CVE, "https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1558", Apr. 2017.
〔15〕 CVE, "https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1559", Apr. 2017.
〔16〕 Metasploit, "http://www.metasploit.com/", Aug. 2019
〔17〕 M.Zalewski. American fuzzy lop. http://lcamtuf.coredump.cx/afl/.
〔18〕 Serge Gorbunov and Arnold Rosenbloom, "AutoFuzz: Automated Network Protocol Fuzzing Framework," International Journal of Computer Science and Network Security, VOL.10 No.8, pp. 239-245 Aug 2010.
〔19〕 Xing Han, Qiaoyan Wen and Zhao Zhang, "A Mutation-based Fuzz Testing Approach for Network Protocol Vulnerability Detection," Proceedings of 2012 2nd International Conference on Computer Science and Network Technology, pp. 1018-1022, Dec. 2012.
〔20〕 Rune Hammersland, Einar Snekkenes, "Fuzz testing of web applications," https://www.semanticscholar.org/paper/Fuzz-testing-of-web-applications-Hammersland-Snekkenes/40882fb2cc230e1e8c7859ae3b5e96999e7ec45d#citing-papers, 2008.
〔21〕 tshark version 3.1.1, https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html
〔22〕 wireshark version 3.1.1, https://www.wireshark.org/download.html
〔23〕 selenium(python) version 3.14.0, https://www.seleniumhq.org, Aug. 2018.
〔24〕 Ivan Andrianto, M.M. Inggriani Liem and Yudistira Dwi Wardhana Asnar, "Web Application Fuzz Testing," 2017 International Conference on Data and Software Engineering, ICoDSE, Nov. 2017.
〔25〕 pyshark version 0.3.8, https://github.com/KimiNewt/pyshark, Aug. 2019.

〈저자소개〉

허 정 민 (Jung-Min Heo) 학생회원
2018년 3월~현재: 아주대학교 컴퓨터공학과 석사과정
〈관심분야〉 임베디드/IoT 보안

김 지 민 (Ji-Min Kim) 학생회원
2015년 2월: 아주대학교 정보컴퓨터공학과 학사
2015년 3월~현재: 아주대학교 컴퓨터공학과 석·박사 통합과정
〈관심분야〉 임베디드/IoT 보안

지 청 민 (Cheong-Min Ji) 학생회원
2012년 2월: 아주대학교 정보컴퓨터공학과 학사
2012년 3월~현재: 아주대학교 컴퓨터공학과 석·박사 통합과정
〈관심분야〉 차량보안, 임베디드/IoT 보안, 블록체인 보안, 영상데이터 보안

홍 만 표 (Man-Pyo Hong) 종신회원
1981년 2월: 서울대학교 계산통계학과 학사
1983년 8월: 서울대학교 계산통계학과 석사
1991년 2월: 서울대학교 전산과학과 박사
1985년 3월~2016년 2월: 아주대학교 정보컴퓨터공학부(과) 교수
2016년 3월~현재: 아주대학교 사이버보안학과 교수
〈관심분야〉 기반시설보안, 차량보안, 임베디드/IoT 보안, 금융보안, 병렬처리