Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.3, Jun. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.3.541

스마트폰 미디어 서버 데몬에 대한 파일 포맷 인식 기반의 퍼징 연구*

신 민 식,† 유 정 빈, 권 태 경‡
연세대학교 정보보호연구실

A Study of File Format-Aware Fuzzing against Smartphone Media Server Daemons*

MinSik Shin,† JungBeen Yu, Taekyoung Kwon‡
Information Security Lab, Graduate School of information, Yonsei University

요 약

스마트폰은 오디오 서비스를 처리하기 위해 미디어 서버 데몬을 운용한다. 백그라운드에서 높은 권한으로 실행되고 있는 미디어 서버 데몬은 스마트폰을 포함한 스마트 기기에서 가장 많이 사용하는 기능인만큼 관련 취약점이 많이 발생하고 있다. 소프트웨어 취약점을 찾기 위해 널리 사용되는 기존의 퍼징 기법은 미디어 서버 데몬과 같이 입력 파일 포맷 요구사항이 엄격한 환경에서 효과적이지 않다. 본 연구에서는 미디어 서버 데몬의 취약점을 효율적으로 찾기 위해 파일 포맷 인식 기반의 퍼징 기법을 제안한다. 실험을 통해 iOS/tvOS/MacOS/watchOS에서 원격 임의코드 실행 취약점을 발견하였으며 상용 퍼징 도구 FileFuzz, ZZUF와 비교하여 본 논문의 효율성을 검증하였다.

ABSTRACT

The smartphone operates the media server daemon to handle audio service requests. Media server daemons, running with a high privilege in the background, caused many vulnerabilities to applications most frequently used in smart devices including smartphones. Fuzzing is a popularly used methodology to find software vulnerabilities. Unfortunately, fuzzing itself is not much effective in such format-strict environments as media services. In this paper, we propose a file format-aware fuzzing in order to efficiently detect vulnerabilities of media server daemon. We acquired a remote arbitrary code execution vulnerability on iOS/tvOS/MacOS/watchOS, and we verified the effectiveness by comparing our methodology with the fuzzers FileFuzz and ZZUF.

Keywords: Mutation-based Fuzzing, Audio/Video file, Format Awareness, Media Server Daemon, Smartphone

I. 서 론

오늘날 세계 주요 50개국의 스마트폰 보급률이 70%에 육박하면서 스마트폰과 관련된 보안 이슈의 중요성도 커지고 있다. 특히 대부분의 스마트폰이 채택하고 있는 모바일 운영체제인 iOS와 Android에 대한 보안 취약점 수가 빠르게 증가하고 있어 이를 예방하기 위한 연구가 필요하다.

스마트폰에서 가장 많이 사용하는 기능 중 하나는 알람, 카메라 셔터음, 게임, 벨소리, 영상 스트리밍, 음악 등과 같은 오디오 서비스이다. 오디오 서비스 요청을 처리하기 위해서는 일반적으로 미디어 서버 데몬을 운용하는데 백그라운드에서 대기하다가 요청이 들어오면 처리하고 에러가 발생할 경우 커널에 의

Received(02. 23. 2017), Modified(04. 04. 2017), Accepted(04. 05. 2017)
* 본 연구는 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2015R1A2A2A01004792)
† 주저자, msshinaktl@yonsei.ac.kr
‡ 교신저자, taekyoung@yonsei.ac.kr(Corresponding author)

해 죽었다가 다시 되살아난다. 또한 오디오/비디오 처리의 모든 부분을 담당하고 있어 높은 권한을 유지하고 있기 때문에 악성 멀티미디어 파일을 이용한 공격이 많이 이루어지고 있다. 예를 들어 2015년 CVE 취약점을 살펴보면 Android 6.0.x 버전에서 오디오/비디오 재생에 관여하는 Stagefright 엔진 관련 취약점이 급격히 증가한 것을 볼 수 있다[1]. 특히 멀티미디어 파일은 스마트폰 뿐만 아니라 스마트워치 등 거의 모든 스마트 기기에서 실행할 수 있으며 운영체제가 서로 호환성을 가질 시 동일한 취약점이 적용되어 공격 대상 플랫폼을 확장할 수 있다.

소프트웨어 보안 취약점 발굴에는 주로 퍼징(Fuzzing)이라 불리는 소프트웨어 보안 테스트 기법을 이용한다. 퍼징은 유효하지 않은 데이터를 어플리케이션에 입력해 취약점이 없는지 확인하는 고도로 자동화된 테스트 기법이다[2].

기존의 퍼징 도구로는 찾을 수 없는 취약점을 찾고, 보다 효율적인 퍼징을 위해 2007, 2008년 PC 기반에서 미디어 파일 포맷 인식 기반의 퍼징을 연구한 사례[3, 4]가 있다. 하지만 PC 기반이 아닌 스마트폰 환경에서의 연구와 파일 포맷 인식 관점에서의 시드파일 수집 연구는 부족한 편이다.

이를 극복하기 위해 본 연구에서는 멀티미디어 파일 포맷을 면밀히 분석하고 이를 바탕으로 iOS와 Android 환경에서 변이 기반의 퍼징을 수행한다. 미디어 서버 데몬의 보안 취약점을 찾기 위해 멀티미디어 파일을 시드파일로, 미디어 서버 데몬을 퍼징 대상으로 퍼징을 수행한다. 특히 본 연구는 파일 포맷 인식을 통해 최소한의 포맷 요구사항을 지켜 어플리케이션이 단순히 파일 재생을 거부하는 경우가 없도록 한다. 또한 포맷 구조 파악에 공부가 다소 많이 필요한 단점을 AtomicParsley[5]와 같은 GNU-GPL parser 도구를 활용하여 극복하고, 파악한 파일 포맷 구조를 토대로 크래시 발생 시 문제를 일으킨 부분을 쉽고 빠르게 파악한다. 멀티미디어 관련 취약점은 해커의 주요 공격 수단이 되므로 이에 대한 대책 기술을 제안하고 개선하는데 활용될 것으로 기대된다.

II. 퍼징 대상 및 시드파일

본 연구의 퍼징 대상인 스마트폰 미디어 서버 데몬과 시드파일인 멀티미디어 파일은 파일 포맷 인식 기반의 퍼징에 있어 다양한 장점을 지닌다.

2.1 미디어 서버 데몬

스마트폰, 스마트워치, 스마트 TV 등 대부분의 스마트 디바이스는 동영상 처리, 음악 파일 재생, 코덱 활용 등의 미디어 서비스를 제공하기 위해 공통적으로 미디어 서버 데몬을 운용한다. 이는 하나의 미디어 서버 대상 취약점이 크로스 플랫폼으로 여러 스마트 디바이스에도 동일한 취약점을 기대할 수 있게 한다.

예를 들어 Apple의 경우 iOS, tvOS, Mac OS, watchOS가 미디어 서버 데몬(mediaserverd)를 공통적으로 운용하기 때문에 하나의 취약점이 모든 플랫폼에 적용될 수 있다. 이와 유사하게 Android 운영체제 기반의 Android Auto, Android TV, Android Wear, Android Things도 동일하게 오디오 서비스를 제공하므로 미디어 서버 데몬을 퍼징 대상으로 삼는 것에 의의를 찾을 수 있다.

2.2 멀티미디어 파일

시드파일이 특정 프로그램 혹은 플랫폼에서만 받아들여지고 사용자가 적다면 퍼징 결과의 파급력이 제한적일 수밖에 없다. 반면 멀티미디어 파일, 특히 오디오/비디오 파일은 시드파일로서 다음 네 가지 장점을 가진다. 1) 인터넷과 스마트폰 보급률이 증가하면서 YouTube, Facebook, uTorrent 등 언제 어디서든 오디오/비디오 컨텐츠를 공유할 수 있다. 2) 오디오/비디오 파일 재생이 사용자 자신에게 보안 위협이 될 거라는 인식이 부족하다. 3) 코덱과 같이 멀티미디어 파일 재생 및 생성에 필요한 구성요소는 구조가 매우 복잡하여 보안성을 고려하기 힘들 뿐더러 개발자들의 보안의식 또한 높지 않다. 4) 스마트폰에서 오디오/비디오 파일 재생에 특별한 권한이나 조건이 필요하지 않다. 이러한 다양한 장점을 지닌 시드파일을 파일 포맷 인식 관점에서 활용하기 위해서는 오디오/비디오 파일의 구조를 면밀히 파악할 필요가 있다.

오디오/비디오 파일 포맷 중 실험에 사용할 MPEG-4 Part 14 파일은 Fig. 1.와 같은 아톰 구조로 나타낼 수 있다[6]. MPEG-4 파일을 구성하는 각 데이터 단위를 `아톰(atom)′이라 부르는데 아톰은 크기를 나타내는 크기 필드와 기능을 나타내는 타입 필드, 타입 필드의 데이터 혹은 하위 아톰으로 이루어져 있는 데이터 필드로 구성되어 있다. 하

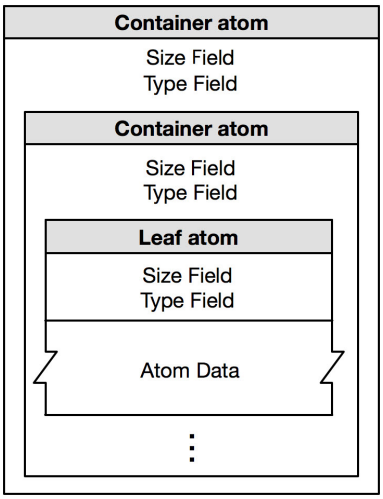


Fig. 1. MPEG-4 Part 14 Atom Structure

위 아톰을 포함하는 경우 상위 아톰은 container atom, 하위 아톰은 leaf atom으로 정의한다.

MPEG-4 파일은 코덱, 인코더, 프로그램, 장치 등의 생성 환경에 따라 아톰 구조가 상이하기 때문에 어떤 파일이 더 좋은지 판단할 수 있는 기준이 필요하다.

III. 파일 포맷 인식 기반의 퍼징 기법 설계

기존의 시드파일 변이 전략은 파일의 오프셋 처음부터 끝까지 순차적으로 변이하거나 bit-flipping 방식을 사용한다. 때문에 얼마나 많은 테스트 케이스를 만들어야 하는지, 크래시가 발생했을 경우 파일의 어느 위치가 문제를 일으키는지 파악하는데 한계가 있다. 이를 해결하고자 MPEG-4 Part 14 파일을 수집하고 Size 아톰만을 변이하기 위해 다음 두 단계로 퍼징 기법을 설계한다.

3.1 시드파일 수집 단계

시드파일 수집은 다운로드와 검증 단계로 이루어져 있다. 파일 포맷 인식 관점에서 질 좋은 시드파일을 얻기 위한 전략을 세우고 이를 자동화시켜 적은 수의 시드파일로도 다양한 아톰의 크래시 여부를 확인할 수 있도록 한다.

자동화된 시드파일 수집을 위해서는 다음 네 가지 1) 크롤링 대상 웹사이트가 로봇 검사 기능이 있는지, 2) 모든 비디오/오디오 파일에 접근하기 위한 사이트내 검색 기능이 있는지, 3) 파일 변환을 거치지 않고 원본 형태의 파일을 얻을 수 있는지, 4) 충분히 많은 파일이 업로드 되어 있는지를 고려한다. 특히 파일 변환 여부는 다양한 아톰을 수집하기 위해 꼭 필요한 사항이다. YouTube와 같이 webm 파일 포맷으로 변환을 거치는 웹사이트는 다시 m4a/mp4 파일로 변환하더라도 코덱, 인코더 등의 설정에 의해 고정된 개수의 아톰만을 얻게 된다.

마지막으로 시드파일 검증 전략을 세우기 위해 스마트폰이 비디오/오디오 파일을 어떻게 처리하는지 파악한다. Android 6.0.1 MPEG4Extractor.cpp 소스코드 분석을 통해 Table 3.과 같이 94개의 유효 아톰 목록을 얻을 수 있었으며 해당 아톰을 포함한 시드파일만을 유지한다.

3.2 변이기반 퍼징 단계

시드파일 수집 결과의 파일을 분석하여 아톰별 오프셋 주소를 파악하고 Fig. 2.와 같이 Size 필드만을 변이한다. 아톰 Size 필드는 해당 아톰의 전체 크기를 결정짓기 때문에 실제 아톰 크기와 Size 필드간의 불일치 및 메모리 할당 문제를 일으킬 소지가 큰 부분이다.

파일 포맷 분석에는 파일 명세 파악에 시간 투자가 큰 단점을 극복하고자 MPEG-4 Parser 도구(AtomicParsley)를 활용한다. 또한 Size 필드를 변이할 시 Table 1.과 같이 heap overflow를 일으킬 확률이 높은 정수 값[7]으로 변이하여 효율을 극대화 한다.

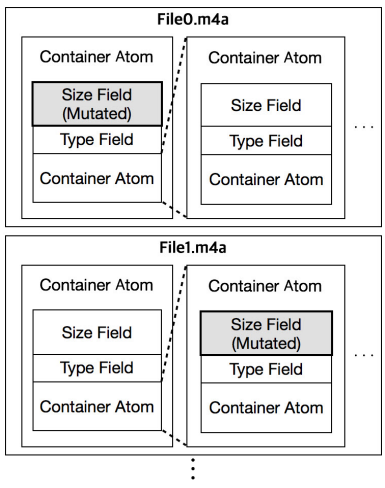


Fig. 2. Mutated test cases

Table 1. Mutation value

	Hexadecimal Value
1	0x000000FF
2	0x0000FF00
3	0x00FF0000
4	0xFF000000
5	0xFFFFFFFF
6	0x7FFFFFFF
7	0x80000000
8	0x20000000
9 ~ 19	0x00000000 ~ 0x0000000A

테스트 케이스 생성 이후에는 파일명과 아톰 타입 필드를 기록한 파일을 생성하여 크래시 결과 확인 시 파일명만으로 크래시 위치를 파악할 수 있도록 한다. 테스트 케이스 생성은 파이썬 코드로 구현하여 일련의 과정을 모두 자동화한다.

생성한 테스트 케이스는 퍼징에 사용하고 크래시 여부를 모니터링한다. 인터넷 브라우저를 통해 자동으로 모든 테스트 케이스를 열람하고 크래시를 기록할 수 있도록 쉘 스크립트로 구현한다.

IV. 실험 방법 및 결과

4.1 실험환경

파일 포맷 인식 기반의 퍼징 실험 환경은 Table 2.와 같이 테스트케이스를 생성하는 웹 서버와 퍼징을 수행할 스마트폰, iOS 최신버전 검증에 사용할 맥북으로 구성하였다.

Table 2. Environment of fuzzing system

Operating System	Device
Ubuntu 12.04.5 LTS	Web Server
OS X El Capitan 10.11.6	MacBook (Xcode)
iOS 7.1.2, 9.3.5	iPhone 4, iPhone 6s
Android 6.0.1	Nexus 5

4.2 실험절차 및 방법

Fig. 3.은 시드파일 수집부터 변이기반 퍼징을 거쳐 크래시 분석까지의 실험 전체 흐름도를 나타낸다. 파일 포맷 인식 관점에서 전체 실험이 이루어지도록 하기 위해 시드파일 검증, 테스트 케이스 생성, 크래시 분석에 파일 포맷 분석을 활용한다.

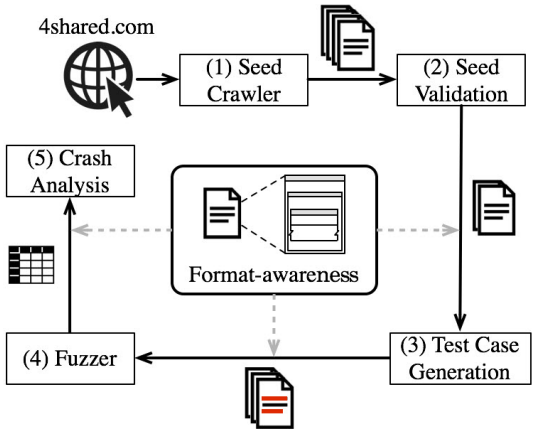


Fig. 3. File Format-aware Fuzzing Architecture

4shared.com 웹사이트에서 마우스/키보드 매크로를 활용한 크롤러를 구현하고 수집한 시드파일이 유효 아톰을 포함하는지 검증한다. 이후 테스트 케이스 및 변이 테이블 생성이 종료되면 iOS에서는 Safari, Android는 Chrome 브라우저로 테스트 케이스를 순차적으로 열람하고 크래시를 기록하도록 모든 절차를 자동화한다. 퍼징이 종료된 이후에는 테스트 케이스 생성 단계에서 저장한 변이 테이블을 참고하여 크래시 아톰을 파악하고 해당 아톰만을 활용하여 더 큰 취약점을 공략한다. 그리고 본 연구에서 제안한 퍼징 기법의 성능을 비교하기 위해 상용 파일 퍼징 도구인 FileFuzz[8]와 ZZUF[9]로 똑같은 시드파일를 이용해 같은 수의 테스트 케이스를 생성하고 퍼징을 수행하였다. FileFuzz 2.0은 파일 퍼징의 성능 비교를 위해 사용되는 가장 기본적인 퍼져로 시드파일의 오프셋 0부터 순차적으로 4바이트씩 0x000000FF로 변이한다. ZZUF 0.15는 bit-flipping 방식으로 기본 변이비율 값인 0.4로 테스트 케이스를 생성한다.

특히 iOS의 경우 iPhone 4 기기 (iOS 7.1.2)에서 실험을 하였는데 iPhone 4의 최신 업데이트 버전이 iOS 7.1.2 이고 해당 버전이 실험환경을 만들기 위해 필요한 탈옥을 자유롭게 할 수 있기 때문이다.

4.3 실험결과

4.3.1 시드파일 수집 결과

4shared.com 웹사이트의 m4a/mp4 파일을 크롤링한 결과 5초에 1개꼴로 다운로드가 가능했으며 1시간 동안의 수집을 통해 8개의 시드파일로 94개 유효 아톰 중 총 58개의 아톰을 얻을 수 있었다. Fig. 4.에서는 시간에 따른 시드파일 수집 결과와 성능 비교를 위한 YouTube 크롤링 결과를 볼 수 있다. 약 2,600초 이후에는 58개 이상의 아톰을 얻을 수 없었는데 나머지 36개의 아톰은 MPEG-4 Part 14 파일에서 사용하지 않는 다른 포맷의 아톰인 것을 파악했다. 한편, YouTube는 webm 포맷에서 m4a/mp4 포맷으로 변환할 때의 컨버터 설정에 따라 아톰이 다양하지 못하고 항상 동일한 것을 확인할 수 있다.

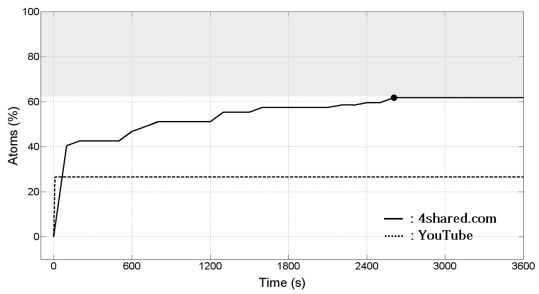


Fig. 4. Collected atoms over time for two websites

4.3.2 제안 퍼징 기법 결과

6시간 동안 1,102개의 테스트 케이스로 실험한 결과 iOS 9.3.5에서는 3개(mvhd, trak, udta), Android 6.0.1에서 15개(stbl, trak, mean, name, data, cprt, covr, albm, gnre, perf, titl, yrrc, auth, dinf, mp4a)의 크래시 아톰을 얻었다. iOS는 7.1.2 버전에서 실험한 이후 iOS 9.3.5를 Xcode에 연결하고 수작업으로 크래시 파일을 열람하여 크래시 여부를 파악하였다. Table 3.에서는 모든 유효 아톰 목록과 퍼징을 수행한 아톰, 크래시 아톰을 확인할 수 있다.

Table 4. Comparison between fuzzers. (* indicates crash files)

Seed Type	iOS 9.3.5			Android 6.0.1		
	Ours	File Fuzz	ZZUF	Ours	File Fuzz	ZZUF
m4a	3	2	0	14	0	9*
mp4	0	0	0	1	0	1*

4.3.3 상용퍼저 결과

Table 4.에서 상용퍼저의 결과를 확인할 수 있듯이 iOS 9.3.5에서 FileFuzz는 2개, ZZUF는 0개의 크래시 아톰을, Android 6.0.1에서는 FileFuzz가 0개, ZZUF는 10개의 크래시 파일을 얻었다. 특히 ZZUF의 결과가 크래시 아톰이 아닌 크래시 파일인 이유는 bit-flipping 방식의 변이이기 때문에 크래시 파일의 어느 부분이 문제를 일으켰는지 파악하는데 한계가 있기 때문이다.

4.3.4 성능평가 및 분석

퍼징 실험을 통해 iOS에서 원격 임의코드 실행 취약점을 발견할 수 있었다. 발견한 취약점은 Apple에 제보하여 CVE-2016-4702 CVSS

Table 3. Experiment Results of each FOURCC atoms (Shade area indicates 58 fuzzed atoms. 17 atoms in bold are crash atom)

©alb	3g2b	**covr**	edts	h263	isom	meta	mp4v	s263	sinf	stz2	trex
©ART	3gp4	cpil	elst	hdlr	keys	mfra	MSNV	saio	**stbl**	tenc	trkn
©day	aART	**cprt**	enca	hev1	mdat	minf	mvex	saiz	stco	text	trun
©gen	**albm**	ctts	encv	hvc1	mdhd	moof	**mvhd**	samr	stsc	tfhd	tx3g
©nam	**auth**	d263	esds	hvcC	mdia	moov	**name**	sawb	stsd	**titl**	**udta**
©wrt	avc1	**data**	frma	ID32	mdta	mp41	**perf**	sbtl	stss	tkhd	**yrrc**
©xyz	avcC	**dinf**	ftyp	ilst	**mean**	mp42	pssh	schi	stsz	traf	
3g2a	co64	disk	**gnre**	iso2	mehd	**mp4a**	qt	sidx	stts	**trak**	

Score 10.0을 얻었다. 퍼징 대상 선정의 장점에서 언급하였듯이 해당 취약점은 iOS 뿐만 아니라 공통적으로 미디어 서버 데몬을 운용하는 tvOS, Mac OS, watchOS 모두에 적용되는 것을 확인하였다. 반면 Android에서는 CHECK() 종류의 함수에서 아톰 Size 필드를 검증하여 크래시가 발생하는데 이는 DoS 이상의 취약점을 막고자 하는 일종의 방어 메커니즘인 것을 확인할 수 있었다.

상용 퍼져와의 비교 결과를 파일 포맷 인식 관점에서 분석하면 제안한 퍼징 기법이 같은 수의 테스트 케이스 대비 가장 많은 아톰의 크래시 여부를 파악할 수 있었다. 다른 상용 퍼져로 더 다양한 아톰의 변이 여부를 확인하기 위해서는 훨씬 많은 수의 테스트 케이스가 필요하기 때문에 제안한 퍼징 기법의 효율성을 검증하였다.

V. 관련연구

퍼징 분야는 1990년 위스콘신 대학교의 바턴 밀러에 의해 생겨났다. 초기 연구는 구조화되지 않고 무작위 값을 입력하여 7개 버전의 UNIX 운영체제에서 9개 유틸리티 프로그램을 테스트하기 위한 기초적인 연구였다[10]. 퍼징은 소프트웨어 내부 정보의 유무에 따라 블랙박스 퍼징, 화이트박스 퍼징, 그레이박스 퍼징으로 분류되며, 데이터 생성 방법에 따라 생성기반 퍼징과 변이기반 퍼징으로 분류할 수 있다[11].

퍼징은 소프트웨어에 무작위 값을 입력하는 형태를 시작으로 1999년 PROTOS test suite로 시작된 프로토콜 분야[12]와 2004년 Microsoft JPEG 취약점(MS04-028)이 발견된 이후 활발해지기 시작한 파일 퍼징 두 분야로 초기 흐름을 설명할 수 있다. 이후 지속적으로 발전한 퍼징 분야의 최근 흐름은 아래와 같이 시드파일 선택부터 변이 전략까지 더욱 발전된 형태로 나타난다.

2014년 A. Rebert 등은 퍼져가 버그를 잘 발견한다고 보여주는 것 외에 어떻게 퍼징하는 것이 적절한지 과학적으로 이해하려는 체계적인 노력은 많이 없다고 지적한다. 퍼징을 진행하는 동안 발견할 수 있는 총 버그 개수를 최대화하기 위해 어떻게 하면 시드 파일을 가장 잘 선택할 수 있을까에 대해 수학적으로 공식화하고 추론하는 방법에 대해 중점적으로 다룬다[13].

2015년 Cha 등은 블랙박스 변이기반 퍼징의 버그 발견 효율을 극대화시키기 위해 프로그램-시드 쌍이 주어졌을 때 파일의 입력 비트 의존성을 파악하여 최적의 변이율을 찾는 연구가 진행되었다. 프로그램마다 최적의 bit-flipping 변이율을 다르게 적용하기 때문에 같은 시간동안 진행한 다른 퍼져보다 성능이 더 우수함을 보였다[14].

2016년 Bohme 등이 제안한 CGF (Coverage-based Greybox Fuzzing)는 프로그램 분석을 하지 않은 채 시드 입력을 약간 변이시켜 테스트를 진행한다. 테스트가 새로운 경로를 찾게 되면 시드 집합에 추가되고 그렇지 않으면 버린다. 대부분의 테스트가 특정 경로를 높은 빈도로 도달하는 것을 확인하였고 이를 보완하기 위해 Markov chain model을 사용하여 빈도가 낮은 경로로의 탐색을 수행한다. 이로 인해 같은 수의 테스트로 더 많은 경로를 탐색할 수 있도록 하였다[15].

최근에는 특히 프로그램의 특징을 고려한 그레이박스 퍼징이나 symbolic execution 기반의 연구가 주를 이루고 있지만 실제 대부분의 원격 임의코드 실행 취약점은 프로그램의 특징을 고려하지 않은 퍼징에 의해 발견된다[16]. 이러한 유형의 퍼징은 여전히 유효하지만 이를 체계적으로 연구한 사례는 부족한 편이다. 아래에는 PC 플랫폼에서 파일 포맷 분석을 통해 악성 페이로드를 시도한 사례를 소개한다.

5.1 포맷 인식 기반의 퍼징

2007년 Lewis 등은 미디어 플레이어와 같은 입력 파일 포맷 검증이 까다로운 소프트웨어의 취약점을 발견하고자 기존의 포맷을 고려하지 않는 퍼징을 변형시켜 진행하였다. 최소한의 포맷팅 요구사항을 지키지 않고 완전히 랜덤 데이터로 퍼징을 수행하면 미디어 플레이어가 재생을 거부하는 경우가 잦아 취약점을 얻기 위해서는 실제 유효한 미디어 파일 형식을 갖춰야 한다고 지적하였다[3].

2008년 Thiel은 기존의 퍼징 도구로는 미디어 플레이어나 코덱에서 버그를 찾는 것에 한계가 있어 이를 극복하고자 미디어 파일 포맷을 상세히 분석하여 분석된 프레임 헤더를 퍼징 대상으로 삼았다. 정확한 퍼징 대상 선정으로 인하여 단순한 퍼져에서는 찾을 수 없는 버그를 찾을 수 있었고 퍼징 시간을 고려했을 때 효율적인 실험이 가능하였다[4].

5.2 스마트폰 환경에서의 퍼징

2011년 Klein은 iOS 3.1.3 이전 버전에서 오디오 파일 퍼징을 수행했다. 시드파일의 오프셋 0부터 999까지 차례차례 오프셋까지 255의 값으로 변이하여 테스트 케이스를 생성하였다. 그 다음 모바일 사파리를 통해 테스트 케이스를 실행하여 미디어 서버 데몬(mediaserverd) 프로세스 취약점을 발견하였다[17].

2015년 Lee는 iOS와 Android 두 운영체제에서 GIF, JPEG, PNG, MP3, MP4를 사용한 퍼징을 수행하였다. 시드파일을 분석한 후 서로 겹치지 않고 질 좋은 시드파일을 선택할 수 있는 Seed File Selection 알고리즘을 고안하고 발견된 크래시를 자동으로 분류할 수 있도록 CACE (Crash Automatic Classification Engine) 도구를 이용하였다. 그 결과 iOS과 Android에서 약 1900개의 크래시와 서로 다른 7개의 유니크 버그들을 찾아내며 ZZUF 퍼져보다 더 좋은 결과를 얻었다[18].

VI. 결 론

본 연구에서는 MPEG-4 Part 14 파일 분석을 통해 파일 포맷 인식 기반의 퍼징을 구현하였고 iOS와 Android에서 다수의 크래시 아톰을 발견할 수 있었다. 파일 포맷 인식 관점에서 스마트폰이 실제 MPEG-4 파일 처리에 사용하는 아톰을 수집하고 수집한 모든 아톰의 크래시 여부를 판단할 수 있는 시드파일 수집과 변이기반 퍼징 기법을 설계하였다. 성능확인을 위해 테스트 케이스 개수 대비 수집한 아톰의 크래시 여부를 상용 파일 퍼징 도구와 비교한다. 마지막으로 퍼징으로 발견한 취약점이 미디어 서버 데몬을 운용하는 다른 플랫폼에도 적용되는지 확인하여 본 연구의 우수성을 검증하였다.

향후 본 연구를 바탕으로 시드파일 타입을 MPEG-4 Part 14 컨테이너가 아닌 다른 디지털 컨테이너 포맷으로 확장한다면 더 다양한 결과를 얻을 것이다. 또한 프로그램 커버리지 기반의 퍼징에 본 연구의 변이전략을 접목하여 보다 개선된 성능을 기대할 수 있다.

References

[1] "CVE Details," https://cve.mitre.org/.

[2] P. Oehlert, "Violating assumptions with fuzzing," In Proc. the IEEE Security & Privacy (S&P), vol. 3, no. 2, pp. 58-62, Mar. 2005.

[3] C. Lewis, B. Rhoden, and C. Sturton, "Using Structured Random Data to Precisely Fuzz Media Players," Project Report, UC Berkeley, Dec. 2007.

[4] D. Thiel, "Exposing Vulnerabilities in Media Software," BlackHat EU, Mar. 2008.

[5] "AtomicParsley," http://atomicparsley.sourceforge.net/.

[6] "Quicktime File Format Specification," https://developer.apple.com/library/mac/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html.

[7] M. Sutton and A. Greene, "The Art of File Format Fuzzing," in BlackHat USA, Jul. 2005.

[8] M. Sutton, "FileFuzz," http://osdir.com/ml/security.securiteam/2005-09/msg00007.html.

[9] C. Labs, "ZZUF," http://caca.zoy.org/wiki/zzuf.

[10] B. P. Miller, L. Fredriksen, and B. So, "An Empirical Study of the Reliability of UNIX Utilities," Communications of the ACM, vol. 33, no. 12, pp. 32-44, Dec. 1990.

[11] M. Sutton, A. Greene, and P. Amini, "Fuzzing: Brute Force Vulnerability Discovery," Addison-Wesley Professional, Jul. 2007.

[12] Oulu University Secure Programming Group, "PROTOS," https://www.ee.oulu.fi/research/ouspg/Protos.

[13] A. Rebert, S. K. Cha, T. Avgerinos, J. Foote, D. Warren, G. Grieco, and D. Brumley, "Optimizing Seed Selection for Fuzzing," In Proc. the USENIX Conference on Security Symposium,, pp. 861-875, Aug. 2014.

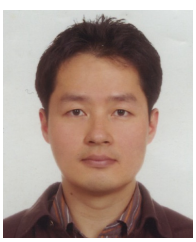
[14] S. K. Cha, M. Woo, and D. Brumley,

"Program-adaptive Mutational Fuzzing," In Proc. the IEEE Symposium on Security and Privacy (S&P), pp. 725-741, May. 2015.
〔15〕 M. Bohme, P. Van-Thuan, and R. Abhik, "Coverage-based Greybox Fuzzing as Markov Chain," In Proc. the ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1032-1043, Oct. 2016.
〔16〕 "Symbolic Execution in Vuln Research," https://lcamtuf.blogspot.kr/2015/02/symbolic-execution-in-vuln-research.html.
〔17〕 T. Klein, "A Bug Hunter's Diary: A Guided Tour Through the Wilds of Software Security," No Starch Press, Nov. 2011.
〔18〕 W. H. Lee, M. Srirangam Ramanujam, and S. Krishnan, "On Designing an Efficient Distributed Black-box Fuzzing System for Mobile Devices," In Proc. the ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 31-42, Apr. 2015.

〈저자소개〉

신 민 식 (MinSik Shin) 학생회원
2016년 2월: 연세대학교 컴퓨터공학과 졸업
2016년 3월~현재: 연세대학교 정보보호연구실 석사과정
〈관심분야〉 시스템 보안, 네트워크 보안, 스마트폰 보안 등

유 정 빈 (JungBeen Yu) 학생회원
2016년 2월: 대구카톨릭대학교 정보보호학 졸업
2016년 3월~현재: 연세대학교 정보보호연구실 석사과정
〈관심분야〉 시스템 보안, 악성코드 탐지



권 태 경 (Taekyoung Kwon) 종신회원
1992년 2월: 연세대학교 컴퓨터과학과 학사
1995년 2월: 연세대학교 컴퓨터과학과 석사
1999년 8월: 연세대학교 컴퓨터과학과 박사
1999년~2000년: U.C. Berkely Post-Doc
2001년~2013년 8월: 세종대학교 컴퓨터공학과 교수
2007년~2008년 Univ. Maryland at College Park 교환교수
2013년 9월~현재: 연세대학교 정보대학원 교수
〈관심분야〉 암호 프로토콜, 네트워크 프로토콜, 사물인터넷 보안, HCI 보안 등