Journal of the Korea Academia-Industrial cooperation Society
Vol. 20, No. 6 pp. 541-547, 2019

https://doi.org/10.5762/KAIS.2019.20.6.541
ISSN 1975-4701 / eISSN 2288-4688

바이너리 취약점의 자동 탐색을 위한 동적분석 정보 기반 하이브리드 퍼징 연구

김태은[1*], 전지수[1], 정용훈[2], 전문석[2]
[1]숭실대학교 컴퓨터학과, [2]한국인터넷진흥원

A Study on Hybrid Fuzzing using Dynamic Analysis for Automatic Binary Vulnerability Detection

Taeeun Kim[1*], Jeesoo Jurn[1], Yong Hoon Jung[2], Moon-Seog Jun[2]
[1]Korea Internet & Security Agency
[2]Dept. of Computer Science, SoongSil University, Seoul, South Korea

요 약 최근 자동화 되는 해킹 및 분석 기술의 발전으로 인하여 수많은 소프트웨어 보안 취약점이 빠르게 발표되고 있다. 대표적인 취약점 데이터베이스인 NVD(National Vulnerability Database)에는 2010년부터 2015년까지 보안 취약점(CVE: Common Vulnerability Enumeration) 약 8만 건이 등록되었으며, 최근에도 점차 증가하고 있는 추세이다. 보안 취약점은 빠른 속도로 증가하고 있는 반면, 보안 취약점을 분석하고 대응하는 방법은 전문가의 수동 분석에 의존하고 있어 대응 속도가 느리다. 이런 문제점을 해결하기 위해 자동화된 방법으로 보안 취약점을 탐색하고, 패치하여 악의적인 공격자에게 공격 기회를 줄 수 있는 보안 취약점을 사전에 대응 할 수 있는 기술이 필요하다. 본 논문에서는 복잡도 분석을 통해 취약점 탐색 대상 바이너리의 특징을 추출하고, 특징에 적합한 취약점 탐색 전략을 선정하여 취약점을 자동으로 탐색하는 기술을 제안한다. 제안 기술은 AFL, ANGR, Driller 도구와 비교 검증 하였으며 코드 커버리지는 약 6% 향상, 크래시 개수는 약 2.4배 증가, 크래시 발생율 약 11% 향상 효과를 볼 수 있었다.

Abstract Recent developments in hacking technology are continuing to increase the number of new security vulnerabilities. Approximately 80,000 new vulnerabilities have been registered in the Common Vulnerability Enumeration (CVE) database, which is a representative vulnerability database, from 2010 to 2015, and the trend is gradually increasing in recent years. While security vulnerabilities are growing at a rapid pace, responses to security vulnerabilities are slow to respond because they rely on manual analysis. To solve this problem, there is a need for a technology that can automatically detect and patch security vulnerabilities and respond to security vulnerabilities in advance. In this paper, we propose the technology to extract the features of the vulnerability-discovery target binary through complexity analysis, and select a vulnerability-discovery strategy suitable for the feature and automatically explore the vulnerability. The proposed technology was compared to the AFL, ANGR, and Driller tools, with about 6% improvement in code coverage, about 2.4 times increase in crash count, and about 11% improvement in crash incidence

Keywords : Software Vulnerability, Vulnerability Analysis, Fuzzing, Symbolic Execution, Binary

본 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임
(No.2017-0-00184, 자기학습형 사이버 면역 기술 개발)
*Corresponding Author : Taeeun Kim(KISA)
Tel: +82-2-820-1273 email: tekim31@kisa.or.kr
Received May 8, 2019 Revised June 5, 2019
Accepted June 7, 2019 Published June 30, 2019

1. 서론

최근 해킹 기술이 고도화되면서 취약점 수가 지속적으로 증가하고 있다. 대표적인 취약점 DB인 CVE (Common Vulnerability Enumeration)에 2010년부터 2015년까지 약 8만 건의 취약점이 등록 되었으며 2019년에만 1.1만 건 이상 등록 중이다[1]. 또한 최근에는 취약점 탐색 자동화 도구를 활용하여 제로데이 건수가 늘어나고 있는 추세이다. 기존에는 취약점 탐색을 위해 보안 전문가가 직접 소프트웨어를 분석하고 취약점을 발견하여 패치를 하였다. 그러나 소프트웨어 분석 및 취약점 탐색에 많은 시간이 소요되고, 전문가에 따라 취약점을 분석하는 속도의 차이가 있기 때문에 빠른 대응이 어렵다[2]. 최근, 이러한 전문가의 기술 의존도를 해결하고 취약점 탐색에 들어가는 비용을 줄이기 위해 취약점 자동 탐색 기술 및 도구가 등장하고 있다. 보안 취약점 자동 탐색의 대표적인 기술로는 퍼징 및 기호실행이 있다. 퍼징(Fuzzing)은 대상 프로그램에 임의의 입력 값을 반복적으로 생성하고 응용 프로그램이 충돌을 발생하도록 하는 기술이다. 기호실행(Symbolic Execution)은 입력 값을 기호로 지정하고 프로그램의 취약점이 있는 경로에 도달할 수 있는 값을 찾는 방법 이다. 또한 DARPA에서 자동화 해킹 방어 대회인 Cyber Grand Challenge(CGC)를 개최하였다[3]. 이 대회에서도 취약점 탐색을 위해 두 기술이 대표적으로 활용되어 주목을 받았다. 그러나 현재 국내에서는 바이너리 대상으로 취약점 자동 탐색에 대한 연구는 활발하게 진행되지 않으며, 퍼징과 기호실행 기술 모두 한계점이 존재한다. 퍼징의 경우, 빠른 속도로 충돌(Crash)을 일으킬 수 있으나, 대상 바이너리에 대한 실행 경로를 고려하지 않기 때문에 논리적인 취약점을 탐색하기에는 어려움이 있다. 이 문제를 해결하기 위해 최근에는 퍼징과 기호실행을 결합하여 각각의 장점만을 활용하는 연구가 발표되고 있다. 그러나 퍼징과 기호실행을 결합하는 과정에 있어, 대상 바이너리의 특징을 고려하지 않고 퍼징과 기호실행 간의 단순한 전환이 이루어진다. 본 논문에서는 대상 바이너리의 복잡도를 기반으로 퍼징과 기호실행을 수행하여 취약점을 자동 탐색하는 기술을 제안한다.

2. 관련 연구

2.1 퍼징(Fuzzing)

'퍼징 (Fuzzing)'라는 단어는 1988 년 위스콘신 대학(University of Wisconsin)의 Miller 교수 프로젝트에서 최초로 발표되었다[4]. 이 논문의 저자는 전화 회선을 사용해 워크스테이션에 로그 시도 중 폭풍의 영향을 받아 임의의 입력 값이 워크스테이션에 전달되는 것을 확인하였다. 이 경험은 현재 퍼징의 개념으로 이어졌으며 90 개의 유닉스 유틸리티 중 24 % 이상이 퍼징으로 인해 충들을 일으킬 수 있었다. 퍼징의 기본 개념은 테스트 케이스를 무작위로 생성하여 대상 소프트웨어에 입력하는 것이다. 퍼징 기술의 초기 단계에서는 대상 SW를 분석하지 않고 입력 값을 무작위로 생성하여 테스트하는 Dumb Fuzzing이 대부분이었다. 대표적인 Dumb Fuzzer로는 zzuf가 있다. 그 이후 발전된 모델로는 대상 SW 및 입력 값의 구조를 분석하여 소프트웨어에 적합한 입력 값을 생성하는 스마트 퍼징 프레임워크가 개발되었다. 대표적인 도구로는 Peach, Sulley, SPIKE 등이 있다. 최신 기술로는 Evolutionary Fuzzing이 있으며, 대표적인 도구로는 AFL이 있다[5-6]. Evolutionary Fuzzing 기술은 SW 코드 커버리지를 측정하기 위해 계측 (Instrumentation) 환경에서 실행된다. 계측 환경 위에서 유전 알고리즘 기반 입력 값 변이를 통해 코드 커버리지를 높일 수 입력 값을 반복적으로 생성한다. 최근 CGC 자동 해킹 방어대회에서도 대표적으로 활용된 도구이며, 바이너리 취약점 탐색 부분에서 최고의 성적을 기록했다.

2.2 기호실행(Symbolic Execution)

1975 년 King이 처음 도입 한 기호실행은 프로그램이 특정 속성을 위반했는지 여부를 확인하는 소프트웨어 테스팅과 관련하여 널리 사용되는 프로그램 분석 기술이다[7]. 기호실행의 주요 아이디어는 입력 값을 기호로 설정하고 대상 프로그램의 여러 경로를 동시에 탐색하는 것이다[8]. 그러나 다양한 경로가 동시에 검색되므로 기호실행은 높은 자원 소비를 필요로 하고 경로 폭발을 야기한다. 기호실행의 초기 연구단계에서는 자식 프로세스를 생성하여 모든 경로를 탐색하는 Online Symbolic Execution 방식을 활용하였다. 그러나 자원 활용의 효율성을 높이기 위하여 Depth First Search, Breadth First Search, Buggy Path Search 등 각종 Heuristic 탐색이 연구되기 시작하였다. 최근에는, 특정 경로를 선택하여 구체적인 입력 값을 가진 프로그램을 실행하고 분기문에서는 기호실행을 수행하는 Concolic Execution

연구되고 있으며 기호실행 기술 중 가장 많이 사용되어진다.

2.3 혼합형 퍼징(Hybrid Fuzzing)

퍼징과 기호실행 기술은 각각 장단점이 존재한다. 퍼징 기술의 경우 빠른 속도로 크래시를 발생시킬 수 있으나, 깊은 경로를 탐색하여 취약점을 발생시키기에는 한계점을 가지고 있다. 기호실행의 경우 깊은 경로를 탐색하여 취약점을 발생시킬 수 있으나 프로그램 규모에 따라 실행 속도가 느려지며 자원 소모가 크다는 단점이 있다. 이러한 문제를 해결하기 위하여 얕은 경로에서는 퍼징을 활용해 취약점을 빠르게 탐색하고, 더 이상 경로 탐색이 어려운 경우 기호실행을 활용하여 취약점을 탐색하는 Hybrid 방식의 퍼징 기술이 발표되었다. 취약점 탐색 도구들의 비교는 Table 1과 같다[9].

Table 1. Compare Vulnerability Exploration Tools

Skill	Tool	Technique	Strategy	Target
Fuzzing	AFL	Blackbox	Genetic Algorithm	Binary
	Peach	Blackbox	Format Modeling	Binary
	Sulley	Blackbox	Format Modeling	Binary
Symbolic Execution	Angr	Whitebox	Stepping	Binary
	KLEE	Whitebox	Random Path	Source Code
	CREST	Whitebox	Search Heuristics	Source Code
	Cloud9	Whitebox	Parallel	Source Code
Hybrid Fuzzing	Driller	Greybox	Selective	Binary

3. 동적 분석 기반 하이브리드 퍼징 기술

기존 연구에서도 하이브리드 퍼징 기술을 활용하여 취약점을 탐색하는 연구가 발표되었다. 그러나 퍼징과 기호실행 간의 스위칭 여부를 판단할 때, 퍼징의 테스트 케이스가 더 이상 상태 전환을 일으키지 못하면 스위칭이 일어나는 형태로 단순 휴리스틱이 적용되어있다. 본 논문에서는 단순한 휴리스틱이 아닌 바이너리 복잡도를 분석하여, 바이너리의 특징을 기반으로 취약점을 탐색하는 기법을 제안한다.

Fig. 1과 같이 동적 분석 기반 하이브리드 퍼징은 취약점 탐색, 탐색 경로 분석, 취약점 탐색 제어로 구성된다.

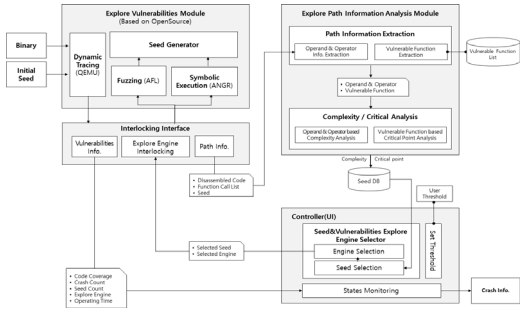


Fig. 1. Structure of Hybrid Fuzzing using Dynamic Analysis

'취약점 탐색'은 동적 Tracing 모듈에서 입력되는 바이너리 및 Seed에 대하여 실행 경로 정보를 추출하고, 이에 대한 '경로 정보 분석' 및 'Seed/엔진 선정' 결과에 따라 퍼징 및 기호실행 엔진을 활용하여 지속적으로 취약점 및 경로를 탐색하고 취약점이 탐색된 결과를 Crash 정보로 기록한다.

'탐색 경로 분석'은 동적 Tracing 모듈에서 추출된 정보 중 복잡도 및 위험도 분석에 요구되는 정보를 추출하며, 추출된 정보를 기반으로 경로 정보를 분석한다. Tracing 모듈에서 추출된 정보는 명령어 정보, 호출된 함수 정보이며 명령어 정보에서 연산자 및 피연산자 수를 추출하여 복잡도를 산정한다. 또한, 호출된 함수 정보에서 취약한 함수 정보를 추출하여 사전에 정의해 놓은 취약한 함수 위험도를 누적하여 위험도를 산정한다. 복잡도 및 위험도는 각 Seed별로 파일형태로 기록한다.

'취약점 탐색 제어'는 '탐색 경로 정보 분석'에서 분석된 정보를 바탕으로 Seed 및 취약점 탐색 엔진을 선정한다. 위험도 분석을 통해 위험도가 가장 높게 측정된 Seed를 선정하고, 복잡도 분석을 통해 복잡도가 사용자가 선정한 임계값 보다 낮으면 기호실행 엔진을 선정하고, 높으면 퍼징 엔진을 선정하여 취약점을 탐색한다.

3.1 취약점 탐색

'취약점 탐색'은 오픈소스 기반으로 개발하였다. 동적 Tracing과 퍼징 및 기호실행 기능 모두 오픈소스를 활용한다. 동적 Tracing은 Binary Instrumentation을 통해 원하는 정보를 활용하며, 퍼징 및 기호실행을 통해 Crash 및 다음 Iteration에서 활용될 Seed 값을 탐색하는데 활용한다.

'동적 Tracing'은 대상 바이너리를 실행하고, Binary Instrumentation 기능을 통해 명령어 정보 및 호출된 함수 정보를 추출한다.

'퍼징'은 대상 바이너리 및 Seed를 입력받아, Seed값을 지속적으로 변이해 가면서 퍼징을 수행한다. 퍼징 도구는 AFL을 활용하며, 퍼징 방법은 먼저, Seed값 변이(2bit 단위 변이, 4bit 단위 변이 등)을 통해 코드 커버리지를 확대(State Transition)시키는 새로운 Seed 모니터링 하고, State Transition을 일으킨 새로운 Seed를 생성한다. Crash가 일어날 경우, Crash 정보를 생성한다.

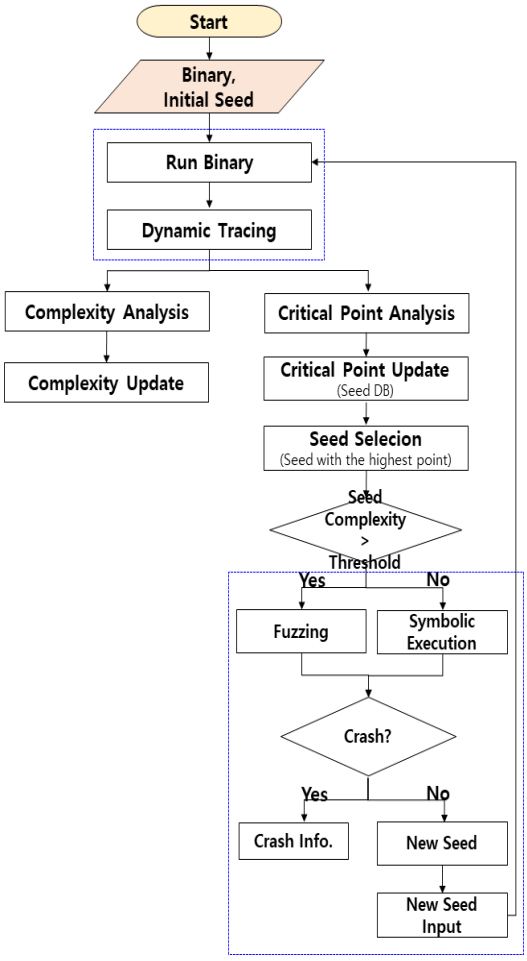


Fig. 2. Flow Chart of Vulnerabilities Explore Module

Table 2. Definition of Crash Informations

Crash Info.	Content
Target Binary	Binary to find vulnerabilities
Input Value	Binary input value to find vulnerabilities
Input Format	Standard input, Option value, File et al
Error Info.	Segmentation fault et al Error signal

기호실행은 대상 바이너리 및 Seed를 입력받아, Seed로 인해 실행된 경로와 다른 경로를 탐색할 수 있는 경로식을 해결하여 새로운 경로를 탐색한다. 기호실행 도구는 ANGR을 활용하며, 기호실행 방법은 먼저 바이너리를 중간언어(Intermediate Language) 변환한다. 변환 후, Symbol을 설정하여 Symbol을 입력값을 생성된 경로식을 풀이한다. 경로식 풀이에는 SMT-Solver인 Z3를 활용한다. Crash가 일어날 경우, Crash 정보를 생성한다. 각 모듈에서 생성하는 정보의 정의는 Table 2와 같으며 동작 과정은 Fig. 2와 같다.

3.2 탐색 경로 분석

탐색 경로 분석은 Fig. 3과 같이 대상 바이너리에 대해 동적 Tracing된 결과를 활용하여, 경로 정보를 추출을 하여 복잡도 및 위험도를 분석해 Seed와 맵핑시키는 기능을 한다. 복잡도 분석을 위해 피연산자 및 연산자 수를 추출하여 복잡도 분석식인 Halstead Metrics에 대입하여 복잡도를 분석하고, 위험도 분석을 위해 취약 함수를 사전 정의하여 취약 함수의 위험도 Score를 누적해 위험도를 산정한다. 산정된 결과는 파일 형태로 Seed와 맵핑하여 저장한다.

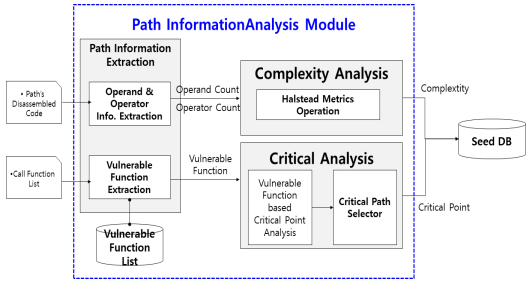


Fig. 3. Structure of Path Information Analysis Module

'경로 정보 추출'에서는 동적 Tracing된 결과 중 복잡도 분석 및 위험도 분석에 필요한 정보를 추출하는 기능을 한다. 동적 Tracing 된 명령어 및 함수 중 피연산자/연산자 수, 취약한 함수를 추출해낸다.

'복잡도 분석'에서는 Tracing 모듈에서 추출된 Disassembled된 코드 중 연산자 및 피연산자를 분류하고 이를 통해 복잡도 분석을 수행한다. 복잡도 분석은 먼저 Disassembled Code 내 연산자 및 피연산자 분류하고, 연산자 및 피연산자 사용 횟수 Count한다. 이후, 복잡도 분석식(Halstead Metrics)을 활용하여 복잡도를 분석하고, Seed와 맵핑하여 파일 형태로 저장한다.

Table 3. Complexity Analysis Variables and Equations

	Variable Name	Disc.
Variables	n1	Number of Operator types
	n2	Number of Operand types
	N1	Number of Total Operator
	N2	Number of Total Operand
	n	n1 + n2
	N	N1 + N2
Equations	$V = N^* \log_2 n$	Volume of Program
	$D = \frac{n_1}{2} * \frac{N_2}{n_2}$	Complexity of a Program
	$E = D^* V$	Effort
	$B = \frac{E^{\frac{2}{3}}}{3000}$	Bug Estimate

'위험도 분석'에서는 동적 Tracing 모듈에서 추출된 호출된 함수 리스트 중 사전에 정의해 놓은 취약 함수를 분류하고 이를 통해 위험도 분석을 수행한다. 위험도 분석은 먼저 호출된 함수 리스트 중 취약 함수를 추출하고, 취약한 함수별 위험도를 산정한다. 이후, 추출된 경로에 대하여 위험도를 산정하고 이 결과를 Seed와 맵핑하여 파일 형태로 저장한다. 위험도 분석에 활용되는 취약한 함수 리스트는 Table 4와 같다.

Table 4. Vulnerable Function List

Classification	Function List
Dangerous (0.5)	scanf, fscanf, vscanf, vsscanf, sscanf, vfscanf, snprintf, vsnprintf, strtok, wcstok, itoa
Banned (1.0)	strcpy, wcscpy, stpcpy, wcpcpy, strecpy, memcpy, strcat, wcscat, streadd, strtrns, sprintf, vsprintf, vprintf, vfprintf, gets, getwd, realpath, syslog, vsyslog, fprintf, printf, sprintf, vfprintf, vprintf, vsprintf, vasprintf, asprintf, vdprintf, dprintf

3.3 취약점 탐색 제어

'취약점 탐색 제어'는 Fig. 4와 같이 취약점 탐색 분석 결과를 활용하여 다음 취약점 탐색을 위해 Seed 및 엔진을 선정하는 기능을 한다. Seed 선정은 현재 생성된 Seed 중 위험도가 가장 높은 Seed를 선정하며, 엔진 선정은 분석된 복잡도를 임계값과 비교하여 임계값보다 낮으면, 기호실행 엔진을 선정하고, 임계값보다 높으면 퍼징 엔진을 선정한다.

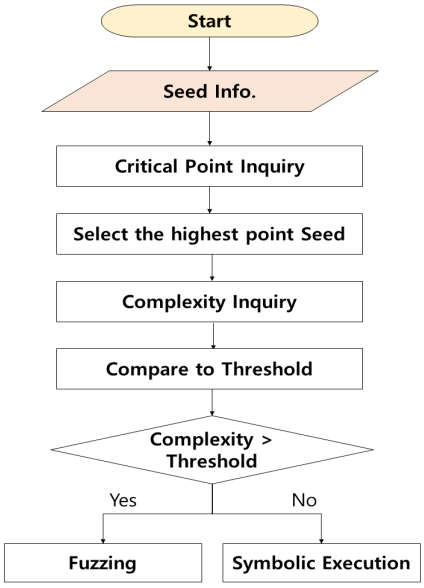


Fig. 4. Flow Chart of Vulnerabilities Explore Control Module

'Seed 선정'은 위험도 분석 모듈에서 분석된 결과를 조회하여 가장 높은 위험도를 가진 Seed를 선정하고 이 Seed를 취약점 탐색 모듈에 넘겨주는 기능을 한다. Seed 선정은 먼저 Seed 파일 중 최고 위험도를 오름차순으로 정열 한다. 최고 위험도를 가진 Seed를 선정 후, 선정된 Seed를 취약점 탐색 엔진에 입력 값으로 넣어준다.

'엔진 선정'은 복잡도 분석 모듈에서 분석된 복잡도를 활용하여 사용자가 사전에 정의한 임계값과 비교하여, 퍼징 혹은 기호실행 엔진을 선정하는 기능을 한다. 엔진 선정 순서는 먼저, 선정된 Seed 파일의 복잡도를 조회하고, 선정된 Seed의 복잡도와 사용자가 정의한 임계값을 비교한다. 임계값보다 복잡도가 낮을 시, 퍼징 엔진을 실행하고, 임계값보다 복잡도가 높을 시, 기호실행 엔진을 실행한다.

4. 성능 실험 결과

4.1 실험 환경

실험에 사용되는 Dataset은 Darpa에서 주최한 Cyber Grand Challenge 자동 해킹방어 대회에서 활용된 CGC 바이너리 120종을 선정하여 실험했다. CGC 대회는 Decree라는 특수한 환경 위에서 동작하게 설계되어 있으므로, 이를 Linux 운영체제에서 동작할 수 있도록

ELF 파일로 모두 변환하여 테스트하였다. 실험 환경은 Ubuntu 64bit, Intel Xeon Gold 6136 12Core, 256GB Mem 환경에서 검증하였으며, 바이너리 한 개당 한 시간의 제한시간을 두고 테스트를 진행하였다.

4.2 성능 평가

최근 취약점을 자동 탐색하는데 대표적으로 활용되는 도구인 AFL, ANGR, Driller 도구와 비교하여 검증하였다. 성능 평가 항목은 코드 커버리지, 크래시 개수, 크래시 발생율 총 3가지 항목을 비교하였다. 세 가지 기준으로 비교하였다. 각 항목에 대하여 최대값, 평균값을 비교하였다.

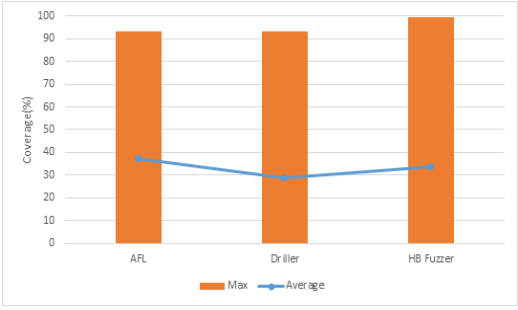


Fig. 5. Compare of Code Coverage

'코드 커버리지'는 바이너리 전체 분기 주소를 기록하고, 동적 탐색된 분기 주소를 기록하여, 전체 분기 주소 대비 동적 탐색된 분기 주소를 측정하여 이를 비교하였다. 결과는 Fig. 5와 같으며 Driller 도구 대비 6.24% 향상된 결과를 보였다.

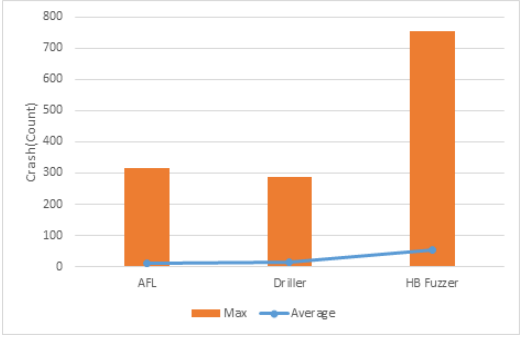


Fig. 6. Compare of Crash Counts

'Crash 개수'는 제한 시간 내 크래시 발생 개수를 기록하여 시간당 크래시 개수를 비교하였다. 결과는 아래 Fig. 6과 같았으며 위험도를 기반으로 취약점을 탐색해 AFL 도구보다 단위 시간 내에서 2.38배 많은 755개의 크래시를 발생하였다.

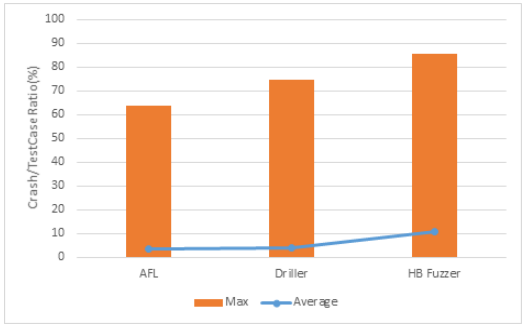


Fig. 7. Compare of Crash Ratio

'Crash 발생율'은 생성된 테스트케이스를 기록하고 크래시 발생 개수를 기록하여 크래시를 유발한 테스트케이스 수 대비 생성된 테스트케이스 수를 계산하여 크래시 발생률을 비교하였다. 결과는 Fig. 7과 같으며 Driller 도구 대비 11% 향상된 크래시를 발생 효율을 보였다.

5. 결론 및 향후 과제

본 논문에서는 자동화 된 취약성 탐지의 대표적인 기술인 퍼징 및 기호실행의 방법론, 한계 및 Hybrid 퍼징 기술 설계 방안에 대해 제안하였다. 퍼징은 취약성을 신속하게 탐색 할 수 있지만 대부분 얕은 경로만을 탐색한다. 반면 기호실행은 실행 경로가 깊고 고유한 크래시를 탐색할 수 있지만 리소스 사용이 제한되고 취약성 감지 속도가 너무 느리다. 자동화 된 취약점 탐지 기술은 각 제한점을 보완하기 위해 결합된 형태로의 연구가 필요하다. 이를 위해 바이너리의 복잡도를 분석하여 퍼징과 기호실행을 결합해 취약점을 탐색하는 방안을 제안하였다. 추후에는, 취약한 경로정보를 학습하여 취약점을 탐색하는 고도화 방안에 대해서 연구 할 예정이다.

References

[1] U.S. National Vulnerability Database. Available online: http://cve.mitre.org/cve/ (accessed April 30, 2019).

[2] S.H. Oh, T.E. Kim, H.W. Kim, "Technology Analysis on Automatic Detection and Defense of SW Vulnerabilities",

Journal of the Korea Academia-Industrial cooperation Society, Vol. 18, No. 11, pp. 94-103, 2017.
DOI: https://doi.org/10.5762/KAIS.2017.18.11.94

[3] Defense Advanced Research Projects Agency(DARPA), Program, DARPA, c2016, From: https://www.darpa.mil/program/cyber-grand-challenge, (accessed Oct., 11, 2017).

[4] Miller, B.P.; Fredriksen, L.; So, B. "An empirical study of the reliability of UNIX utilities", *Commun. ACM 1990*, 33, 32.44.

[5] Bekrar, S.; Bekrar, C.; Groz, R.; Mounier, L. "A taint based approach for smart fuzzing". *In Proceedings of the IEEE Fifth International Conference on Software Testing, Verification and Validation, Montreal*, QC, Canada, 17-21 April 2012; pp. 818-825.

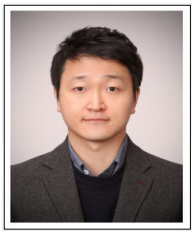
[6] American Fuzzy Lop. Available online: http://lcamtuf.coredump.cx/afl/ (accessed April 30, 2018).

[7] King, J.C. "Symbolic execution and program testing". *Commun. ACM 1976*, 19, 385-394.

[8] Cha, S.K.; Avgerinos, T.; Rebert, A.; Brumley, "D. Unleashing mayhem on binary code". *In Proceedings of the IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, 20-23 May 2012; pp. 380-394.

[9] Stephens, N.; Grosen, J.; Salls, C.; Dutcher, A.;Wang, R.; Corbetta, J.; Shoshitaishvili, Y.; Kruegel, C.; Vigna, G. "Driller: Augmenting Fuzzing through Selective Symbolic Execution". *NDSS 2016*, 16, 1-16.

김 태 은(Taeeun Kim) [정회원]



- 2005년 2월 : 백석대학교 정보통신학부 (공학사)
- 2007년 2월 : 숭실대학교 컴퓨터공학과 (공학석사)
- 2007년 3월 ~ 현재 : 숭실대학교 컴퓨터공학과 박사과정
- 2013년 7월 ~ 현재 : 한국인터넷진흥원 책임연구원

〈관심분야〉
정보보안, 네트워크 보안, 융합 보안

전 지 수(Jeesoo Jurn) [정회원]

- 2016년 2월 : 숭실대학교 컴퓨터학부 (공학사)
- 2015년 6월 ~ 현재 : 한국인터넷진흥원 주임연구원

〈관심분야〉
정보보안, 소프트웨어 취약점 분석

정 용 훈 (Yong Hoon Jung) [종신회원]

- 2006년 8월 : 숭실대학교 컴퓨터공학과 (공학석사)
- 2010년 2월 : 숭실대학교 컴퓨터공학과 (공학박사)
- 2011년 3월 ~ 2014년 2월 : 서일대학교 조교수
- 2018년 8월 ~ 현재 : 바스랩 연구소장
- 현) 한국산학기술학회 상임이사

〈관심분야〉
네트워크 보안, 융합 보안

전 문 석(Moon-Seog Jun) [정회원]

- 1989년 2월 : University of Maryland Computer Science (공학박사)
- 1989년 3월 ~ 1991년 2월 : New Mexico State University Physical Science Lab. 책임연구원
- 1991년 3월 ~ 현재 : 숭실대학교 컴퓨터학과 정교수

〈관심분야〉
네트워크 보안, 생체 인증