ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 49, No. 11, pp. 981-986, 2022. 11
https://doi.org/10.5626/JOK.2022.49.11.981

커맨드 라인 옵션을 변이 및 선택하여 테스팅 효과를 높이는 퍼징 기법

(Effective Fuzzing Technique with Command Line Option Mutation/Selection)

이 아 청 † 김 윤 호 †† 김 문 주 †††
(Ahcheong Lee) (Yunho Kim) (Moonzoo Kim)

요 약 커맨드 라인 인터페이스 프로그램(CLI program)에서 커맨드 라인 옵션은 해당 프로그램이 어떻게 작동하는지에 지대한 역할을 하며, 테스트를 수행할 때 어떤 커맨드 라인 옵션을 사용하는지에 따라 테스트의 성능(커버리지 및 오류 탐지)은 매우 크게 변화할 수 있다. 본 연구에서는 유용한 옵션을 자동으로 선택하여 테스트를 진행하는 퍼징 기술인 PAW(Program option-AWare fuzzer)를 제시한다. PAW는 전체 퍼징 시간의 첫 10% 동안 다양한 옵션을 변이해본 뒤 실행하여, 생성된 옵션들 중 새로이 분기 커버리지를 달성하는 옵션을 유용한 옵션으로 판정하여, 이후 90%의 시간 동안에는 해당 유용한 옵션에 집중하여 테스트를 진행하도록 한다. 최신 퍼저인 AFL++을 기반으로 PAW를 구현한 뒤, 10개의 실제 C 프로그램에서 최신 퍼저인 AFL++과 Angora와 비교하여 평가하였다. PAW는 10개의 실제 C 프로그램에서 AFL++과 Angora와 비교하여 높은 분기 커버리지를 달성하고, 보다 많은 크래시 오류를 탐지하였다.
키워드: 퍼징, 커맨드 라인 옵션, 테스트 케이스 생성 기법, 커버리지 향상, 소프트웨어 테스팅

Abstract Command line option determines how Command Line Interface (CLI) program executes. Therefore, selecting and using proper command line options determines the performance and effectiveness of testing in terms of coverage and bug detection power. In this paper, we propose PAW (Program option-AWare fuzzer) which automatically selects useful command line options to improve the effectiveness of fuzz testing. In the first 10% of fuzzing time, PAW generates diverse command line options by mutating existing options, and it decides which options are useful ones using branch coverage. For the remaining 90% of the time, PAW focuses on executing the selected, useful options for effective testing. We implemented PAW on top of AFL++ which is the state-of-the-art fuzzer, and we evaluated PAW on ten real-world C programs by comparing coverage with AFL++ and Angora. PAW showed higher branch coverage results and crash detection power on the ten C programs.
Keywords: fuzzing, command line option, test case generation, coverage improvement, software testing

· 이 논문은 2021 한국소프트웨어종합학술대회에서 ′유용한 커맨드 라인 옵션을 선택하여 커버리지와 오류 탐지 능력을 높이는 퍼징 기법′의 제목으로 발표된 논문을 확장한 것임
· 이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(NRF-2021R1A5A1021944, NRF-2021R1A2C2009384, NRF-2020R1C1C1013996)과 정보통신기획평가원의 지원(no. 2021-0-00905-001) 및 삼성전자의 지원을 받아 수행된 연구임

† 학생회원 : 한국과학기술원 전산학부 학생
ahcheong.lee@kaist.ac.kr
†† 종신회원 : 한양대학교 컴퓨터소프트웨어학부 교수
yunhokim@hanyang.ac.kr
††† 종신회원 : 한국과학기술원 전산학부 교수(KAIST)
moonzoo.kim@gmail.com
(Corresponding author임)

논문접수 : 2022년 3월 10일
(Received 10 March 2022)
논문수정 : 2022년 7월 29일
(Revised 29 July 2022)
심사완료 : 2022년 8월 13일
(Accepted 13 August 2022)

정보과학회논문지 제49권 제11호(2022. 11)

1. 서 론

커맨드 라인 인터페이스 프로그램(Command-Line Interface program, CLI program)에서 커맨드 라인 옵션은 해당 프로그램이 어떻게 작동할지를 결정하는데 매우 중요한 역할을 한다. 예를 들어, 파일의 목록을 표시해 주는 프로그램인 ls 프로그램[1]을 -l 옵션으로 사용하는 경우 전체 파일의 부가적인 정보를 같이 표시해 주는 동작을 하며, -a 옵션을 사용하는 경우 숨김 처리된 파일도 표시를 해주는 동작을 한다. 따라서, 프로그램을 테스트할 때 좋은 커맨드 라인 옵션을 선택하여 테스트할 때, 보다 더 커버리지를 높이고 오류를 많이 찾을 수 있다.

퍼징(Fuzzing)은 테스트 생성 기법 중 하나로, 테스트 입력을 무작위로 변이한 뒤 실행하는 것을 계속해서 반복하는 것으로 새로운 커버리지를 달성하고 오류를 찾을 수 있는 유의미한 테스트 생성하는 것을 목표로 작동한다[2-4]. 프로그램을 논리적으로 분석하여 테스트를 생성하는 Concolic testing 기술[5,6]에 비해, 기술이 쉽게 적용이 가능하고, 매우 크고 복잡한 프로그램에 대해서도 빠르게 실행이 가능한 특징이 있다.

그러나 커맨드 라인 옵션이 퍼징의 테스트 성능에 지대한 영향을 줄 수 있음에도 불구하고, 대부분의 기존 퍼징 연구들은 단지 하나의 무작위로 선택한 옵션만을 사용하여 테스트 입력에 사용되는 파일만을 변이하도록 하였다.

AFL[11]과 AFLFast[12]에서는 옵션을 변이할 수 있는 기능을 실험적으로 제공하지만, 옵션을 바이너리로 보고 바이너리 레벨에서 무작위로 변이하기 때문에, 정상적으로 실행 될 수 있는 커맨드 라인 옵션을 만들기 어렵다.

본 연구에서는 커맨드 라인 옵션을 적극적으로 변이하여 유용한 커맨드 라인 옵션을 판별하여 높은 커버리지와 오류 탐지 능력을 보이는 새로운 퍼징 기술인 PAW (Program option-AWare fuzzer)를 제시한다. PAW는 전체 테스트 수행 시간의 첫 10% 동안에는 다양한 옵션을 변이 한 뒤 수행하여, 최대한 프로그램의 다양한 옵션을 찾는 것을 목표로 한다. 그 후, 찾은 옵션 중 유용한 옵션을 선별하여, 이후 90%의 시간 동안에는 해당 옵션에만 집중하여 테스트를 진행하여, 테스트 입력 파일을 변이하도록 한다.

높은 테스트 성능을 보이기 위해서는, 다양한 커맨드 라인 옵션을 사용하여 테스트를 수행함과 더불어, 중요하고 유용한 옵션을 선택하여 집중적으로 테스트를 진행하는 것이 중요하다. 예를 들어, 어떤 프로그램이 3개의 옵션 단어를 사용한다고 할 때, (-a, -b, -c), 가능한 옵션의 개수는(같은 단어를 여러 번 쓰지 않는다고 할 때)총 15개이다(-a, -b, -c, -a -b, -a -c, -b -a, -b -c, -c -a, -c -b, -a -b -c, -a -c -b, -b -a -c, -b -c -a, -c -a -b, -c -b -a). 하지만 중복된 역할을 하는 옵션이 많은 경우 이 모든 옵션이 프로그램의 커버리지를 높이는데 중요한 역할을 하지는 않을 것이며, 이 때 모든 옵션을 사용하여 테스트를 수행하는 경우 효율이 매우 떨어질 수밖에 없다. 따라서 PAW는 이 가능한 옵션 중 중요하고 유용한 옵션을 판별하여, 해당 옵션에 집중하여 테스트 입력값을 만드는 것에 집중하도록 하여, 보다 효율적으로 커버리지를 높이고 오류를 탐지할 수 있도록 하였다.

PAW를 최신 퍼저인 AFL++[7]과 Angora[8]를 10개의 실제 C 프로그램에서 퍼징을 수행하여 비교한 결과, 모든 프로그램에서 높은 분기 커버리지와 높은 오류 탐지 능력을 보이는 것을 볼 수 있었다.

2. PAW : 유용한 옵션 선택 퍼징 기술

그림 1에 PAW의 기본적인 프로세스를 나타내었다. PAW는 일단 테스트 대상 프로그램의 문서로부터(help 메시지나, 매뉴얼 등) 자동으로 프로그램 옵션에 사용되는 단어(ex. -l, -a, -o 등)를 탐지하여 해당 단어를 사용하여 이후 퍼징을 반복하여 새로운 커맨드 라인 옵션(ex. -a -l, -a -o, 등)을 찾을 수 있도록 한다.

PAW는 다음 3개의 단계로 구성되었다.

1. 탐색 단계: 전체 테스트 생성 시간 중의 첫 10%의 시간 동안에는 테스트 대상 프로그램이 사용하는 다양한 커맨드 라인 옵션을 최대한 탐색하는 것을 목표로 한다.
2. 옵션 선택 단계: 탐색 단계에서 탐색된 옵션들 중에서, 테스트에 유용한 옵션을 선택하여 이후 90%의 시간 동안에는 해당 옵션들에 대해서만 테스트를 진행하도록 한다. PAW는 탐색 단계에서 새로이 분기 커버리지를 달성한 옵션들을 이후 테스트에서 유용한 옵션으로 판별한다.
3. 파일 퍼징 단계: 옵션을 변이하는 것을 멈추고, 옵션 선택 단계에서 선택된 유용한 옵션만을 사용하여, 이에 해당하는 테스트 입력 파일을 계속해서

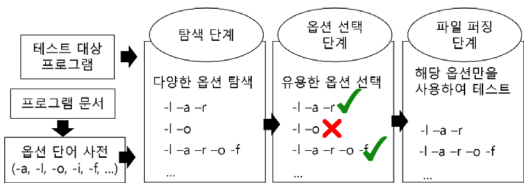


그림 1 PAW 프로세스

Fig. 1 The process of PAW

변이하며 생성하여 보다 다양한 입력 값을 만들 수 있도록 한다.

2.1 PAW 탐색 단계

PAW는 첫 10%의 시간 동안에는 테스트 프로그램이 사용하는 커맨드 라인 옵션을 최대한 많이 찾아내는 것을 목표로 한다. 그러기 위해 테스트 입력 파일을 변이함과 동시에, 사전 기반 변이 기법[9,10]을 적용하여, 새로운 경로 커버리지를 달성하는 옵션들을 찾아내어 저장하도록 한다. 문서로부터 추출한 단어들을 계속해서 조합하는 방식으로, 새로운 옵션을 계속해서 만들 수 있도록 한다.

2.2 PAW 옵션 선택 단계

PAW는 입력 파일과 커맨드 라인 옵션을 모두 변이하기 때문에, 매우 큰 탐색 공간을 가지고 있다. 커맨드 라인 옵션과 입력 파일이 요구하는 탐색 공간의 크기가 다르기 때문에, PAW는 10%의 시간 정도를 다양한 커맨드 라인의 행동을 보기에 충분하다고 보고, 이때 발견된 커맨드 라인 중 테스트의 집중이 필요하다고 생각되는 커맨드 라인을 선택하여 이후 90%의 시간 동안에는 해당 커맨드 라인에 집중할 수 있도록 한다.

유용한 커맨드 라인 옵션의 선택은 첫 10%의 시간 동안 새로운 커맨드 라인 옵션이 얼마나 많은 분기 커버리지를 높일 수 있었는가를 기준으로 판별한다. 즉, 각 커맨드 라인마다 생성된 테스트 입력 파일 중, 새로 분기 커버리지를 올리는 데 성공한 테스트 입력 파일의 비율을 계산하여, 평균값 보다 높은 경우, 유용하다고 판단하도록 하였다.

2.3 파일 퍼징 단계

커맨드 라인 변이를 같이 수행하였던 이전 단계와는 다르게, 선택된 커맨드 라인에 집중하여 파일 변이만을 반복하도록 한다. 이 단계에서는 선택된 유용한 커맨드 라인만을 이용하여 테스트를 진행하기 때문에, 정해진 시간 내에서 더 효율적으로 테스팅을 진행할 수 있다.

2.4 구현

본 연구에서 제시한 PAW 기법을 AFL++ 위에서 구현하였다. AFL++은 기존 가장 널리 사용된 퍼저 중 하나인 AFL[11]에서 AFLFast[12], REDQUEEN[13] 등의 다양한 휴리스틱을 추가하여 개선된 버전으로, 현재 가장 강력한 퍼저 중 하나이기 때문에, PAW를 구현하는데 사용하였다.

3. 평가 및 분석

3.1 실험 설정

본 연구에서 제시한 옵션 선택 퍼징 기술 PAW를 AFL++ 위에서 구현하였다. 그리고 이를 기존 퍼징 연구에서 많이 사용된 10개의 실제 C 프로그램에서 평가하였다. 표 1에서 대상 프로그램에 대한 정보를 나타나 있다. LoC (Lines of code)는 테스트 대상 프로그램의 크기를, # prog. opt.는 PAW가 해당 테스트 프로그램의 문서를 분석하여 입수한 옵션 단어(ex. -l, -a)의 개수를 의미한다. Klees et al.[14]에서 제시하였듯이, 퍼징 자체의 무작위성을 고려하여[15], 모든 실험은 5번을 반복 수행하여 평균값을 보고하였으며, 모든 퍼징 실험은 24시간 동안 수행하였다. 각 퍼징 기술을 평가하기 위해 각 퍼징 기술로 달성한 평균 분기 커버리지와, 오류를 탐지하는 능력을 평가하기 위해 5번의 반복 시험에서 발견된 크래시 오류의 개수를 보고하였다. 평가를 위해 다음 4개의 연구 문제를 설정하여 평가하였다.

표 1 평가에 사용된 테스트 대상 프로그램 정보
Table 1 Test subject programs

Program	LoC	# prog. opt
bison	102,910	54
cflow	37,107	45
cjpeg	10,499	37
jasper	29,498	16
nasm	103,626	33
nm	437,342	55
sam2p	25,624	48
tiff2pdf	8,234	35
tiff2ps	5,646	27
xmllint	11,285	94

RQ1. 기존 한 개의 옵션만을 사용하여 변이하는 최신 퍼징 기술에 비해 높은 커버리지 성능과 오류 탐지 능력을 보였는가?

RQ2. 유용한 옵션만을 선택하여 해당 옵션에만 집중하여 퍼징을 진행하였을 때, 모든 옵션을 사용하여 퍼징을 진행하는 것보다 높은 커버리지 성능과 오류 탐지 능력을 보였는가?

RQ3. 무작위로 같은 개수의 옵션을 선택하여 해당 옵션에 집중하여 퍼징을 진행하였을 때 보다 높은 커버리지 성능과 오류 탐지 능력을 보였는가?

RQ4. 유용한 옵션을 선택하여 해당 옵션에만 집중하여 퍼징을 진행하였을 때, 선택하지 않고 계속 옵션을 변이하는 것 보다 높은 커버리지 성능과 오류 탐지 능력을 보였는가?

RQ1에 대해 평가하기 위해, 10개의 테스트 프로그램을 최신 퍼저인 AFL++과 Angora를 이용하여 퍼징을 수행한뒤, 분기 커버리지와 탐지한 오류의 개수를 PAW와 비교하였다. 해당 최신 퍼저을 이용하여 테스트를 생성할 때는 기존 연구들에서 많이 사용된 옵션을 사용하여 퍼징을 수행하도록 하였다.

RQ2를 평가하기 위해, PAW의 옵션 선택 단계에서 유용한 옵션을 선택하지 않고 전체 옵션을 사용하여 이후 파일 퍼징 단계를 진행하는 PAW-all을 구현하여 PAW와 분기 커버리지 및 탐지한 오류의 개수를 비교하였다.

RQ3를 평가하기 위해, PAW의 옵션 선택 단계에서 PAW와 같은 개수의 옵션을 무작위로 선택하는 PAW-rnd을 구현하여 PAW와 분기 커버리지 및 탐지한 오류의 개수를 비교하였다.

마지막으로 RQ4를 평가하기 위해, 옵션 선택 단계를 거치지 않고 (즉, 옵션을 선택하지 않고) 계속해서 옵션을 변이하는 PAW-no-sel을 구현하여 PAW와 분기 커버리지 및 탐지한 오류의 개수를 비교하였다.

3.2 적용 결과

표 2는 최신 퍼저와 PAW 가 달성한 평균 분기 커버리지와 탐지한 오류의 개수를 비교하였으며, 표 3은 PAW, PAW-all, PAW-rnd에서 다른 설정을 하였을 때의 평균 분기 커버리지와 찾은 오류의 개수, 마지막으로, 표 4에서 PAW와 PAW-no-sel이 달성한 평균 분기 커버리지와 탐지한 크래시 오류의 개수를 비교하였다.

표 2 최신 퍼저와 PAW가 달성한 평균 분기 커버리지와 탐지한 오류의 개수

Table 2 The average branch coverage and the total number of crashes detected by PAW and the state of the art fuzzers)

Program	AFL++		Angora		PAW	
	Br. Cov	# bug	Br. Cov	# bug	Br. Cov	# bug
bison	4880.8	1	3726.2	0	6258.4	3
cflow	1299.8	2	1234.8	2	1523.2	7
cjpeg	3317.2	0	3050.0	0	3741.8	0
jasper	1767.8	0	1653.6	0	3610.5	0
nasm	6175.8	1	5370.4	0	6434.2	6
nm	11056.8	6	5189.0	9	10173.8	20
sam2p	175.0	0	175.0	0	3795.0	0
tiff2pdf	4203.4	0	4001.8	0	4335.2	0
tiff2ps	3356.6	0	3276.0	0	3300.4	0
xmllint	8458.0	0	2451.4	0	13343.2	0
Average/sum	4469.1	10	3012.8	11	**5651.6**	**36**

표 3 PAW, PAW-all, PAW-rnd이 달성한 평균 분기 커버리지와 탐지한 오류의 개수

Table 3 The results of PAW, PAW-all, and PAW-rnd)

Program	PAW-all		PAW-rnd		PAW	
	Br. Cov	# bug	Br. Cov	# bug	Br. Cov	# bug
bison	6492.8	3	5817.2	0	6258.4	3
cflow	1517.0	4	1522.0	5	1523.2	7
cjpeg	3547.8	0	3733.8	0	3741.8	0
jasper	3538.4	0	2800.6	0	3610.5	0
nasm	5867.3	3	6445.0	3	6434.2	6
nm	10482.8	18	11098.0	18	10173.8	20
sam2p	4017.2	0	3512.8	0	3795.0	0
tiff2pdf	3905.8	0	3808.4	0	4335.2	0
tiff2ps	2356.4	0	2589.0	0	3300.4	0
xmllint	10670.6	0	12113.0	0	13343.2	0
Average/sum	5239.6	28	5344.0	26	**5651.6**	**36**

표 4 PAW와 PAW-no-sel이 달성한 평균 분기 커버리지와 탐지한 오류의 개수

Table 4 The results of PAW and PAW-no-sel

Program	PAW-no-sel		PAW	
	Br. Cov	# bug	Br. Cov	# bug
bison	6232.4	0	6258.4	3
cflow	1506.0	2	1523.2	7
cjpeg	2528.6	0	3741.8	0
jasper	3843.0	0	3610.5	0
nasm	5956.0	2	6434.2	6
nm	4667.8	0	10173.8	20
sam2p	3828.2	0	3795.0	0
tiff2pdf	2954.8	0	4335.2	0
tiff2ps	1951.2	0	3300.4	0
xmllint	12148.6	0	13343.2	0
Average/sum	4561.7	4	**5651.6**	**36**

RQ1. PAW를 최신 퍼징 기술인 AFL++과 Angora와 비교하였을 때, 10개의 테스트 대상 프로그램에서 평균적으로 더 높은 분기 커버리지를 달성하였으며, 크래시 오류도 많이 찾아내었다. AFL++에서 10개, Angora에서는 11개의 크래시 오류만을 찾은 것에 대비하여, PAW는 총 36개의 크래시 오류를 탐지하였다. 이는 테스트를 수행할 때 다양하고 유용한 옵션을 사용하는 것이 중요함을 의미한다.

RQ2. 유용한 옵션을 선택하지 않고 전체 옵션을 사용하여 테스트를 진행하는 PAW-all에 비해, PAW는 평균적으로 더 높은 분기 커버리지를 달성하고, 더 많은 28.6% ((36-28)/28) 더 많은 크래시 오류를 탐지하였다. 중요하고 유용한 옵션을 선택하여 해당 옵션에 테스트를 집중하였을 때 보다 효율적으로 커버리지를 달성하고 오류를 많이 탐지할 수 있음을 의미한다.

RQ3. PAW와 같은 개수의 옵션을 무작위로 선택하는 PAW-rnd와 비교하여, 평균적으로 보다 높은 분기 커버리지를 달성하였으며, 38.5% ((36-26)/26) 더 많은 크래시 오류를 탐지하였다. PAW가 분기 커버리지를 이용하여 선택한 유용한 옵션을 사용하는 것이 무작위로 선택한 옵션보다 더 효율적으로 테스트를 진행하는데 도움이 되었음을 의미한다.

RQ4. PAW와 달리 옵션 선택을 하지 않고 계속해서 옵션을 변이하는 PAW-no-sel과 비교하였을 때, 평균적으로 보다 높은 분기 커버리지를 달성하였으며, 매우

많은 오류를 더 많이 탐지할 수 있었다. 입력 파일과 커맨드 라인 옵션의 탐색 공간을 고려하여, 대부분의 시간을 입력 파일의 변이에 할당하는 전략이 유효하였음을 의미한다.

4. 결 론

본 연구에서는 유용한 커맨드 라인 옵션을 분기 커버리지 메트릭을 이용, 자동으로 판별하여, 보다 효율적인 테스트를 진행하여 커버리지와 오류 탐지 능력을 향상시키는 퍼징 기술인 PAW를 제시하였다

기존 퍼징 도구가 한 개의 커맨드 라인 옵션만을 입력으로 받아 실행이 가능했던 것에서 확장하여, 복수의 커맨드 라인을 변이하고 실행해 더 높은 테스팅 효과를 볼 수 있었으며(RQ 1), 유용한 옵션만을 선택하여 퍼징을 진행하였을 때 전체 옵션을 그대로 사용하는 것보다 더 높은 성능을 보는 것을 알 수 있었다(RQ 2). 또한 분기 커버리지를 이용하여 선택하는 것이 맞았는지를 확인 하기 위한 실험에서도, 그 유용성을 확인할 수 있었으며(RQ 3), 마지막으로 계속해서 옵션을 선택하지 않고 계속하였을 때와의 비교에서도 우위를 보이는 것을 볼 수 있었다(RQ 4).

향후 연구로는, 커맨드 라인 옵션뿐 아니라, 초기 입력 파일에 따라서도 퍼징의 성능에 지대한 영향을 줄 수 있기 때문에[16-18], 퍼징에 유용한 초기 입력값을 자동으로 판별할 수 있는 기술에 대한 연구를 진행할 것이다. 또한, 분기 커버리지 외에 다양한 프로그램의 정적/동적 성질[19,20]을 사용하여 보다 더 유용한 옵션을 선택하는 방법도 연구할 계획이다.

References

[1] ls program : A part of GNU coreutils [Online]. Available: https://www.gnu.org/software/coreutils/ (accessed 2022, July 29)

[2] V. J. M. Manes, H. Han, C. Han, S. K. Cha, M. Egele, E. J. Schwartz, and M. Woo. "The art science and engineering of fuzzing: A survey," *IEEE Transactions on Software Engineering*, Vol. 47, pp 2312-2331, Nov. 2021.

[3] H. Liang, X. Pei, X. Jia, and W. Shen, "Fuzzing: State of the Art," *IEEE Transactions on Reliability*, Vol. 67, No. 3, pp. 1199-1218, June 2018.

[4] X. Zhu, S. Wen, S. Camtepe, and Y. Xiang. "Fuzzing: A Survey for Roadmap," *ACM Computing Surveys*. Just Accepted, Jan. 2022.

[5] C. Cadar, D. Dunbar, and D. Engler. "KLEE: Unassissted and Automatic Generation of High-Coverage Tests for Complex Systems Programs," *Proc. of the 8th USENIX conference on Operating systems design and implementation (OSDI 2008)*, pp. 209-224, 2008.

[6] Y. Kim, Y. Choi, and M. Kim. "Precise concolic unit testing of c programs using extended units and symbolic alarm filtering," *Proc. of the 40th international Conference on Software Engineering (ICSE 2018)*, pp. 315-326, 2018.

[7] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse. "AFL++ : Combining incremental steps of fuzzing research," *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[8] P. Chen and C. Hao. "Angora: Efficient Fuzzing by principled search," *Proc. of 2018 IEEE Symposium on Security and Privacy (SP)*, pp. 711-725, 2018.

[9] J. Wang, B. Chen, L. Wei and Y. Liu. "Superion: Grammar-aware greybox fuzzing," *Proc. of the 41st International Conference on Software Engineering (ICSE 2019)*, pp. 724-735, 2019.

[10] V. T. Pham, M. Bohme, A. E. Santosa, A. R. Caciulescu, A. Roychoudhury, "Smart greybox fuzzing," *IEEE Transactions on Software Engineering*, Vol. 47, pp. 1980-1997, Sept. 2021.

[11] M. Zalewski. American fuzzy lop (afl) fuzzer [Online]. Available: http://lcamtuf.coredump.cx/afl/ (accessed 2022, July 29).

[12] M. Bohme, V. T. Pham, and A. Roychoudhury. "Coverage-Based Greybox Fuzzing as Markov Chain," *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1032-1043, 2016.

[13] C. Aschermann, S. Schumilo, T. Blazytko, and R. Gawlik, "REDQUEEN: Fuzzing with Input-to-State Correspondence," *Proc. of 2019 Network and Distributed System Security Symposium*, 2019.

[14] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating fuzz testing," *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 2018)*, pp. 2123-2138, 2018.

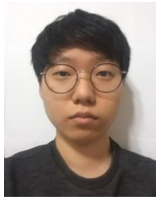
[15] M. Bohme, D. Liyanage, and V. Wustholz. "Estimating residual risk in greybox fuzzing," *Proc. of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pp. 230-241, 2021.

[16] Z. Xie, Z. Cui, J. Zhang, X. Liu, and L. Zheng, "CSEFuzz: Fuzz Testing Based on Symbolic Execution," *IEEE Access*, Vol. 8, pp. 187564-187574, Oct. 2020.

[17] T. Yue, Y. Tang, B. Yu, P. Eang, and E. Wang. "LearnAFL: Greybox Fuzzing With Knowledge Enhancement," *IEEE Access*, Vol. 7, pp. 117029-117043, Aug. 2019.

[18] A. Herrera, H. Gunadi, S. Magrath, M. Norrish, M. Payer, and A. L. Hosking, "Seed Selection for Successful Fuzzing," *Proc. of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pp 230-243, 2021.

[19] Y. Kim, S. Mun, S. Yoo, and M. Kim, "Precise Learn-to-Rank Fault Localization using Dynamic and Static Features of Target Programs," *ACM Transactions on Software Engineering and Methodology*, Vol. 28, No. 23, pp. 1-34, Oct. 2019.
[20] J. Petke, S. O. Haraldsson, M. Harman, and W. B. Langdon, "Genetic Improvement of Software: A Comprehensive Survey," *IEEE Transactions on Evolutionary Computation*, Vol. 22, No. 3, pp. 415-432, April 2017.



이 아 청

2019년 KAIST 전삭학부 학사. 2021년 KAIST 전산학부 석사. 2021년~현재 KAIST 전산학부 박사 과정. 관심 분야는 퍼징, Concolic 테스팅

김 윤 호

2007년 KAIST 전산학과 학사. 2009년 KAIST 전산학과 석사. 2017년 KAIST 전산학부 박사. 2018년~2020년 KAIST 전산학부 연구교수. 2020년~현재 한양대 컴퓨터소프트웨어학부 조교수. 관심 분야는 자동화된 Concolic 유닛 테스팅, 변이 테스팅, 자동 오류 위치 추정, 정형검증, 퍼징

김 문 주

1995년 KAIST 전산학과 학사. 2001년 Univ. of Pennsylvania 박사. 2002년~2004년 SECUi.COM 차장. 2004년~2006년 POSTECH 연구원. 2006년~2012년 KAIST 전산학과 조교수. 2012년~현재 KAIST 전산학부 부교수. 관심분야는 Concolic testing, 자동 오류 위치 추정, Concurrency 테스팅, 변이 테스팅, 정형 검증, 내장형 소프트웨어, 퍼징