유용한 커맨드 라인 옵션을 선택하여 커버리지와 오류 탐지 능력을 높이는 퍼징 기법

이아청[O] 김윤호 김문주
KAIST 한양대 KAIST/브이플러스랩
ahcheong.lee@kaist.ac.kr yunhokim@hanyang.ac.kr moonzoo.kim@gmail.com

PAW: Useful option selection fuzzing technique to Improve Coverage and Bug Detection Ability

Ahcheong Lee[O] Yunho Kim Moonzoo Kim
KAIST Hanyang University KAIST/V+Lab

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 Angora를 기반으로 PAW를 구현한 뒤, 6개의 실제 C 프로그램에서 최신 퍼저인 AFL++과 Angora와 비교하여 평가하였다. PAW는 6개의 실제 C 프로그램에서 모두 AFL++과 Angora와 비교하여 월등히 높은 분기 커버리지를 달성하고, 보다 많은 크래시 오류를 탐지하였다.

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 표시해주는 프로그램인 `ls` 프로그램을 `-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.[1] 프로그램을 논리적으로 분석하여 테스트를 생성하는 Concolic testing 기술[2][3]에 비해, 기술 자체가 적용이 쉽게 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.

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

이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(NRF-2021R1A5A1021944, NRF-2021R1A2C2009384, NRF-2020R1C1C1013996)과 정보통신기획평가원의 지원(no. 2021-0-00905-001)을 받아 수행된 연구임.

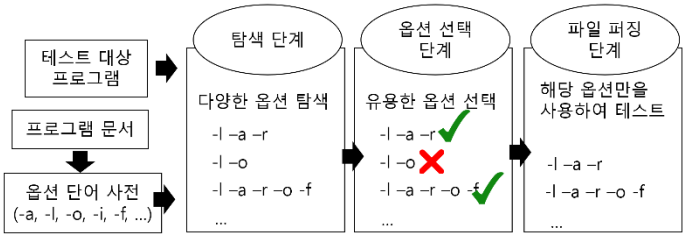

그림 1 PAW 프로세스

않는다고 할때)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++[4]과 Angora[5]를 6개의 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. 테스트 입력 파일을 변이함과 동시에, 테스트 대상 프로그램에서 문서로부터 추출한 단어들을 다양하게 조합하는 사전 기반 변이 기법[6][7]을 적용하여, 새로운 경로 커버리지를 달성하는 옵션들을 찾아내어 저장하도록 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.

표 1. 각 프로그램 정보

프로그램	LoC	# prog. Opt.
jhead	3,864	33
sam2p	23,429	37
tcptrace	49,058	100
tic	24,368	42
tiff2ps	29,550	35
xmllint	127,384	67

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

프로그램	AFL++		Angora		PAW	
	br. cov.	# bug	br. cov.	# bug	br. cov.	# bug
jhead	409.4	0	521.7	0	954.3	5
sam2p	2942.6	0	1583.6	0	4403.1	4
tcptrace	1587.4	0	3596.5	3	2646.4	7
tic	2114.7	0	1641.5	0	5103.6	3
tiff2ps	2964.2	0	2019.7	0	4037.6	0
xmllint	6552.7	0	5778.6	0	12462.9	4

표 3. PAW와 다른 설정을 하였을 때 평균 분기 커버리지와 탐지한 오류의 개수

프로그램	PAW-all		PAW-rnd		PAW	
	br. cov.	# bug	br. cov.	# bug	br. cov.	# bug
jhead	859.1	0	816.5	0	954.3	5
sam2p	4357.8	2	4277.6	3	4403.1	4
tcptrace	2526.8	6	2506.6	5	2646.4	7
tic	4764.2	2	4765	3	5103.6	3
tiff2ps	3832.3	0	3746.7	0	4037.6	0
xmllint	12203.3	4	12313.2	4	12462.9	4

3. 파일 퍼징 단계: 옵션을 변이하는 것을 멈추고, 옵션 선택 단계에서 선택된 유용한 옵션만을 사용하여, 이에 해당하는 테스트 입력 파일을 계속해서 변이하며 생성하여 보다 다양한 입력값을 만들 수 있도록 한다.

3. 평가 및 분석

3.1 실험 설정

본 연구에서 제시한 옵션 선택 퍼징 기술 PAW를 Angora 위에서 구현하였다. 이를 기존 퍼징 연구에서 많이 사용된 6개의 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. [8] 에서 제시하였듯이, 퍼징 자체의 무작위성을 고려하여, 모든 실험은 10번을 반복

수행하여 평균값을 보고하였으며, 모든 퍼징 실험은 24시간 동안 수행하였다. 각 퍼징 기술을 평가하기 위해 각 퍼징 기술로 달성한 평균 분기 커버리지와, 탐지한 크래시 오류의 개수를 보고하였다. 평가를 위해 다음 3개의 연구 문제를 설정하여 평가하였다.

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

RQ2. 유용한 옵션을 선택하여 해당 옵션에만 집중하여 퍼징을 진행하였을 때 보다 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?

RQ1에 대해 평가하기 위해, 6개의 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.

3.2 적용 결과

표 2는 최신 퍼저와 PAW 가 달성한 평균 분기 커버리지와 탐지한 오류의 개수를 나타내며, 표 3는 PAW와, PAW에서 다른 설정을 하였을 때의 평균 분기 커버리지와 찾은 오류의 개수를 의미한다.

RQ1. PAW를 최신 퍼징 기술인 AFL++과 Angora와 비교하였을 때, 모든 6개의 테스트 대상 프로그램에서 월등히 높은 분기 커버리지를 달성하였으며, 크래시 오류도 많이 찾아내었다. AFL++에서는 전혀 크래시 오류를 찾지 못하였으며, Angora에서는 1개의 프로그램에서만 크래시 오류를 찾을 수 있었지만, PAW는 5개의 프로그램에서 총 23개의 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 전체 6개의 프로그램에서 높은 분기 커버리지를 달성하고, 더 많은 64.3% ((23-14)/14) 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 무작위 하게 선택하는 PAW-rnd와 비교하여, 6개의 모든 프로그램에서 보다 높은 분기 커버리지를 달성하였으며, 53.3% ((23-15)/15) 더 많은 크래시 오류를 탐지하였다. PAW가 분기 커버리지를 이용하여 선택한 유용한 옵션을 사용하는 것이 무작위 하게 선택한 옵션보다 더 효율적으로 테스트를 진행하는데 도움이 되었음을 의미한다.

4. 결론

본 연구에서는 유용한 옵션을 자동으로 판별하여, 보다 효율적인 테스트를 진행하여 커버리지와 오류 탐지 능력을 향상시키는 퍼징 기술인 PAW를 제시하였으며, 실험 결과 PAW가 최신 퍼저인 AFL++과 Angora보다 매우 높은 테스트 성능을 보이는 것을 입증할 수 있었다.

향후 연구로는, 커맨드 라인 옵션 뿐 아니라, 테스트에 지대한 성능 영향을 줄 수 있는 초기 입력값을 자동으로 판별할 수 있는 기술에 대한 연구를 진행할 것이다.

참고문헌

[1] Manes, V. JM et al. The art science and engineering of fuzzing : A survey. IEEE Transactions on Software Engineering. (2019).

[2] Cadar, C. et al. KLEE: Unassissted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008). (2008).

[3] Kim, Y. et al. Precise concolic unit testing of c programs using extended units and symbolic alarm filtering. Proceedings of the 40th international Conference on Software Engineering. ACM. (2018).

[4] A. Fioraldi et al. AFL++ : Combining incremental steps of fuzzing research. 14th USENIX Workshop on Offensive Technologies (WOOT 20). (2020)

[5] Chen, P., & Hao C. Angora: Efficient Fuzzing by principled search. 2018 IEEE Symposium on Security and Privacy (SP). IEEE, (2018)

[6] Wang, J. et al. Superion: Grammar-aware greybox fuzzing. Proceedings of the 41st International Conference on Software Engineering (ICSE 2019). (2019).

[7] Pham, T. et al. Smart greybox fuzzing. IEEE Transactions on Software Engineering. (2019).

[8] Klees, G. et al. Evaluating fuzz testing. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM (2018)