함수 관련도를 이용한 퍼징의 커버리지 향상

이아청 ○ 김윤호 김문주

한국과학기술원

dkcjd2000@gmail.com yunho.kim03@gmail.com moonzoo@cs.kaist.ac.kr

Function Relevance based Fuzzing for High Coverage

Ahcheong Lee○ Yunho Kim Moonzoo Kim

Korea Advanced Institute of Technology and Science

요 약

커버리지 기반 퍼징 (Coverage Guided Fuzzing)은 테스트 케이스 생성 기법으로, 기법 자체가 간단하고, 큰 소프트웨어에도 적용이 가능했기 때문에 널리 이용되고 있다. 본 논문에서는 커버리지 기반 퍼징의 커버리지 향상을 위해 함수 관련도를 기반으로 변이할 바이트를 선택하는 새로운 변이 휴리스틱을 제시한다. 두 함수 간의 함수 관련도는 두 함수가 같이 실행되는 테스트 케이스의 개수로 정의 되며, 높은 함수 관련도는 두 함수가 서로 높은 의존성을 띄고 있음을 나타낸다. 어떤 한 타겟 함수의 커버리지 향상을 위해, 이 새로운 휴리스틱은 그 타겟 함수와 관련도가 높은 함수들이 읽고 쓰는 바이트를 변이하여 커버리지 향상을 꾀한다. 이 바이트들은 타겟 함수의 변수들과 데이터 의존성이 높기 때문에, 이 바이트를 변이함으로써 타겟 함수의 커버리지를 높일 기회를 더 얻을 수 있다.

본 논문에서는 제시한 휴리스틱을 최신 퍼저 (Fuzzer)인 Angora를 기반으로 구현하였고, LAVA-M data set에 포함된 4개의 실제 C 프로그램에서 평가하였다. 제시된 휴리스틱은 경로 커버리지를 9.4%, 버그 탐지에서 15.7%만큼의 성능 향상을 보였다.

1. 서 론

커버리지 기반 퍼징 (Coverage Guided Fuzzing, CGF)은 테스트 케이스 생성 기술로, 그 기술 자체가 쉽게 적용이 가능하고, 큰 단위의 소프트웨어에도 적용이 가능한 특징이 있다. CGF는 각 테스트 케이스를 단순한 바이트의 나열로 읽은 후, 무작위의 바이트를 무작위 하게 변이 시켜 새로운 테스트 케이스를 생성한다. 가장 유명한 CGF 중 하나인 AFL (American Fuzz Lop) 은 무작위로 바이트를 선택하여 해당 바이트를 변이하여 새로운 테스트 케이스를 생성하며, 다른 CGF 인 Angora[3]와 Vuzzer[4]는 taint analysis를 적용하여 어떤 바이트를 변이할 것인지 선택한다. 더 자세하게, Angora는 먼저 달성하지 못한 (uncovered) 분기 하나를 타겟으로 지정한 뒤 taint analysis를 통해 해당 분기와 데이터 의존성을 띄는 바이트를 구하여 해당 바이트만을 변이한다. 이러한 바이트만을 선택하여 변이하는 것은 테스트 케이스 전체 바이트를 변이하는 것에 비해 짧은 시간 안에 새로운 커버리지를 달성하는 테스트 케이스를 생성할 수 있다. 하지만, taint analysis는 해당 분기를 달성하는데 필요한 바이트를 모두 구하지 못하기 때문에, taint analysis를 통해 구한 바이트만을 변이할 때는 해당 분기를 달성하는데 실패할 수 있다.

따라서, 본 논문에서는 변이할 바이트를 선택하는 새로운 변이 휴리스틱을 제안한다. 해당 타겟 분기가 포함된 함수를 f_t라 하였을 때, 1. f_t에 포함된 모든 분기를 타겟 분기와 높게 관련된 분기라 한다. 2. f_t와 관련도가 높은 함수들에 포함된 모든 분기도 타겟 분기와 높게 관련된 분기라 한다. 3. 타겟 분기 뿐만 아니라 타겟 분기와 높게 관련된 분기들에도 taint analysis를 적용하여 해당 분기들에서 사용하는 바이트를 모두 구하여 해당 바이트도 포함하여 테스트 케이스를 변이한다. 타겟 분기와 높게 관련된 분기들은 타겟 분기의 변수들을 읽고 쓸 가능성이 높기 때문에 해당 타겟 분기의 커버리지를 향상시키는데 사용될 수 있다.

본 논문에서는 이 휴리스틱을 최신 CGF 중 하나인 Angora에 적용하여 구현하였다. LAVA-M[5] data set에 포함된 4개의 실제 C 프로그램에서 평가하였으며, 4개 프로그램 모두에서 높은 경로 커버리지 (path coverage)와 버그 탐지능력을 보였다.

2. 커버리지 기반 퍼징 프로세스

그림 1은 커버리지 기반 퍼징의 일반적인 프로세스를 나타내고 있다. 0. 사용자가 기존에 가지고 있는 테스트 케이스 파일을 퍼저(Fuzzer)에게 전해주면, 퍼저는 이 테스트 케이스를 퍼저 자체의 시드 뭉치 (seed corpus)에 보관한다. 1. 시드 뭉치에서 하나의 테스트 케이스 파일 선택한 뒤, 2. 이를 무작위로 변이하여 여러 개의

이 논문은 과학기술정보통신부의 재원으로 한국연구재단의 지원(NRF-2019R1A2B5B01069865), 과학기술정보통신부의 재원으로 한국연구재단-차세대 정보 컴퓨팅기술개발사업의 지원(NRF-2017M3C4A7068177), 교육부의 재원으로 한국연구재단의 지원(NRF-2017R1D1A1B03035851)을 받아 수행한 연구임

새로운 테스트 케이스 파일을 생성한다. 그 후 3. 생성된 테스트 케이스 파일을 실행하여 각 테스트 케이스의 경로 커버리지를 구한 뒤, 4. 새로운 커버리지를 달성한 테스트 케이스만을 시드 뭉치에 넣는다. 이후 시드 뭉치에서 새로운 테스트 케이스를 선택하여 1번 단계부터 해당 프로세스를 반복한다.

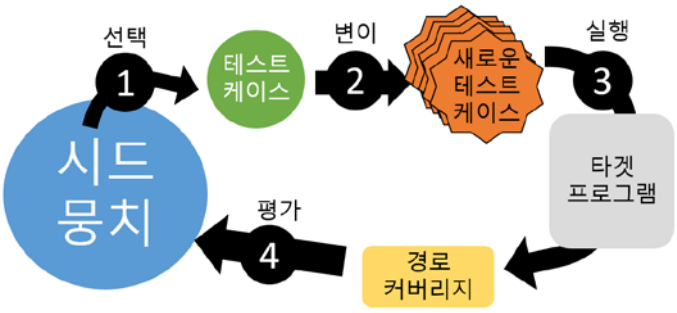

그림. 1. 커버리지 기반 퍼징 프로세스

단순한 프로세스이지만 구체적인 전략에 따라 CGF의 성능이 달라진다. 1단계에서 시드 뭉치에서 어떤 테스트 케이스를 선택하여 변이할 것인지 결정해야하며, 2단계에서 어떤 테스트 케이스의 어떤 바이트를 선택하여 어떤 값으로 변이 할 것인지에 따라 새로 생성한 테스트 케이스가 커버리지를 높일 수 있는지가 결정된다. 본 논문에서는 2단계에서 테스트 케이스의 어떤 바이트를 선택하여 변이할 것인지에 대한 새로운 전략을 제안한다.

3. 함수 관련도 기반 퍼징

3.1 함수 관련도 정의

어떤 한 함수 f_1 에 대한 다른 함수 f_2의 함수 관련도를 다음 수식(1)과 같이 정의한다.[6]

$$\mathrm{Rel}(f_1 \leftarrow f_2) = \frac{n_{f_1,f_2}}{n_{f_1}} \quad \dots(1)$$

n_{f_1} 는 시드 뭉치에 존재하는 테스트 케이스 중 f_1을 실행하는 테스트 케이스의 수, n_{f_1,f_2}는 시드 뭉치에 존재하는 테스트 케이스 중 f_1과 f_2를 모두 실행하는 테스트 케이스의 수를 의미한다.

CGF가 기본적으로 사용하는 커버리지 정보를 이용하여 어떤 함수가 실행되었는지 쉽게 알아낼 수 있으며, 따라서 함수 관련도의 계산도 간단하게 가능하다. 이는 다양한 CGF에 모두 적용이 가능함을 의미한다.

3.2 함수 관련도 기반 바이트 선택

AFL은 단순히 테스트 케이스의 무작위 바이트를 선택하여 변이하지만, Angora는 일단 어떤 분기 (branch) 달성을 목표로 변이할지 선택한 뒤 taint analysis를 통해 테스트 케이스의 어떤 바이트가 해당 분기와 데이터 의존성을 보이는지 알아낸 뒤, 이 바이트만을 변이하여 새로운 테스트 케이스를 생성한다. 테스트 케이스의 모든 바이트를 변이하는 것에 비해 짧은 시간 안에 커버리지를 높이는 테스트 케이스를 생성할 확률은 높지만, taint analysis로 구한 바이트만을 변이하는 것이 해당 분기를 달성할 수 있다는 것을 보장하지 못한다. 따라서 taint analysis로 구한 바이트만을 계속해서 변이하는 것보다, 해당 분기를 달성할 수 있는 다른 바이트를 더 구해서 변이할 때 커버리지를 더 높일 수 있다.

본 논문에서는 함수 관련도를 이용하여 바이트를 선택하는 기법을 다음과 같이 제시한다.

1. 기존의 시드 뭉치 안의 테스트 케이스로 도달되었지만 (reached), 달성되지 않은 (uncovered) 분기 하나를 타겟으로 선택한다. 해당 분기에 reach하는 테스트 케이스의 어떤 바이트를 변이할 것인지 선택하기 위해 다음 단계를 수행한다.
2. 해당 타겟 분기를 포함하는 함수를 f_{target} 이라 하였을 때, 관련도가 높은 함수들의 집합 $rel(f_{target})$을 구한다. $rel(f_{target})$는 수식 (2)와 같이 정의 된다.

$$rel(f_{target}) = \{f \mid Rel(f_{target} \leftarrow f) > \alpha\} \quad \dots(2)$$

α는 사용자가 선택하는 역치 값으로, 본 논문에서는 0.8을 사용하였다.

3. $rel(f_{target})$ 의 모든 함수의 모든 분기를 타겟 분기와 관련도가 높은 분기라 하며, taint analysis를 관련도가 높은 분기들에 적용하여 해당 분기들에 데이터 의존성을 띄는 바이트를 모두 구한다.
4. 구한 바이트만을 변이하여 새로운 테스트 케이스를 생성한다.

기존 Angora는 해당 타겟 분기만을 선택하여 구한 바이트만을 변이했지만, 제시된 프로세스에서는 타겟 분기와 관련된 분기들에 대해서도 taint analysis를 적용하므로 더 많은 바이트를 확장하여 변이하게 된다.

관련도가 높은 분기만을 골라 바이트를 확장하는 이유는 관련도가 높은 분기들이 타겟 분기와 같이 실행되는 경우가 많고, 따라서 타겟 분기와의 데이터 의존성이 높기 때문이다. 타겟 분기만을 taint analysis하는 경우에는 타겟 분기를 달성할 수 있는 바이트를 구하지 못할 수 있는데, 이때 놓친 바이트를 관련된 분기에도 taint analysis를 적용함으로써 구할 수 있으며, 동시에 테스트 케이스의 무작위 바이트를 변이하는 것 보다 짧은 시간 안에 해당 분기를 달성할 수 있다.

4. 평가 및 분석

4.1 실험 설정

본 논문에서는 제시한 함수 관련도 휴리스틱을 Angora를 수정하여 구현하였으며, 이를 Angora-FR이라 하였다. 이를 버그가 삽입된 4개의 실제 C 프로그램으로 이루어진 LAVA-M data set에서 평가하였다. 표 1에서 대상 프로그램에 대한 정보를 나타내었다. LoC는

코드 라인 수, Listed bugs는 LAVA-M 제작자가 LAVA-M에 삽입한 버그의 개수를 의미한다. 퍼징 자체에 무작위성이 존재하기 때문에 10번 수행을 반복한 뒤 평균값을 보고하였으며, 다음 실험 문제 (Research Question) 1번과 2번에 대해 답변하기 위해 각 퍼저가 달성한 평균 경로 커버리지 (path coverage)와 찾은 버그의 개수를 구하여 나타냈다.

RQ1. 함수 관련도를 이용하여 바이트를 선택하였을 때 더 높은 커버리지를 달성하는가?

RQ2. 함수 관련도를 이용하여 바이트를 선택하였을 때 버그를 더 많이 잡을 수 있는가?

또한 다음의 RQ3에 대해 평가하기 위해서 Angora-random을 구현하였다.

RQ3. 함수 관련도를 이용하여 바이트를 추가로 선택하는 것이 무작위로 바이트를 선택하여 추가하는 것 보다 좋은 성능을 보이는가?

일단 Angora-FR을 각 C 프로그램에 실행하였을 때, 타겟 분기만을 taint analysis를 통해 바이트를 구하는 것에 더해서 함수 관련도를 이용하여 어느 만큼의 바이트를 선택하는지 구한 다음, Angora-random에서 해당 바이트를 무작위를 추가로 선택하여 변이하도록 하였다.

4.2 LAVA-M 적용 결과

표 2은 각 퍼저를 이용하여 달성한 평균 경로 커버리지를 나타내며, 표 3는 각 퍼저를 이용하여 평균적으로 찾은 버그의 개수를 나타낸다.

RQ1. Angora-FR은 4개 모든 프로그램에서 가장 높은 경로 커버리지를 달성하였으며, 평균적으로 Angora에 비해 커버리지는 9.4% 향상하였다.

RQ2. Angora-FR은 가장 많은 버그를 검출하는 것에도 성공하였다. 평균적으로 찾은 버그 개수는 15.7%만큼 향상하였다. 특히 4개의 프로그램 중 가장 큰 프로그램인 *who*에서 커버리지를 크게 높이며 거의 모든 버그를 검출하였다. 나머지 3 프로그램에서는 이미 Angora가 대부분의 버그를 검출하여 더 많은 버그를 찾지 못했지만, 경로 커버리지를 더 높이는데 성공하였다.

RQ3. Angora-FR은 Angora-random 보다도 더 높은 성능을 보이는 것에 성공했으며, 이는 실제로 함수 관련도를 이용하여 바이트를 선택하였을 때 더 가치 있는 바이트를 선택할 수 있었음을 의미한다.

5. 결론

본 논문에서는 커버리지 기반 퍼징에서 커버리지를 높이기 위해 함수 관련도를 이용하여 바이트를 선택하는 휴리스틱을 제안하였다. 이을 최신 퍼저인 Angor를 수정하여 구현하였으며, 실제 C 프로그램으로 이루어진 LAVA-M data set에서 평가하였다. 제시된 휴리스틱을 이용해 커버리지를 평균 9.4% 높이는데 성공하였으며 동시에 15.7 % 더 많은 버그를 검출할 수 있었다.

본 논문에서는 어떤 바이트를 선택하여 변이할 것인지에 대해서만 함수 관련도를 사용하였다. 하지만, 어떤 테스트를 골라서 변이할지, 각 테스트 케이스는 얼마나 많이 변이를 시도할지, 그리고 어떤 값으로 변이를 시도할지 등 많은 전략이 가능하기 때문에, 함수 관련도를 이용한 전략에 대해 향후 연구를 진행 할 것이다.

표 1. 각 프로그램 정보

프로그램	LoC	# of function	Listed bugs
base64	8936	75	44
uniq	9107	86	28
md5sum	9231	76	57
who	20132	79	2136

표 2. 각 프로그램에서 달성한 평균 경로 커버리지

프로그램	Angora	Angora-random	**Angora-FR**
base64	233.9	215.8	**266.5**
uniq	124.2	127.3	**142.1**
md5sum	429.4	412.4	**468.4**
who	4360.2	4326.0	**4757.1**

표 3. 각 프로그램에서 평균적으로 찾은 버그의 개수

프로그램	Angora	Angora-random	**Angora-FR**
base64	45	44	**45**
uniq	29	29	**29**
md5sum	55	56	**56**
who	1684	1894	**1968**

참고문헌

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

[2] Böhme, M. et al. Directed greybox fuzzing. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM (2017)

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

[4] Rawat, S. et al. VUzzer: Application-aware Evolutionary Fuzzing. *NDSS*. Vol. 17. (2017)

[5] Dolan-Gavitt, B. et al. Lava: Large-scale automated vulnerability addition. *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, (2016).

[6] 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.

[7] Manes, V. JM et al. The art science and engineering of fuzzing: A survey. *arXiv: 1812.00140* (2018).