입력 키워드 추출을 통한 뮤테이션 기반 Fuzzing의 성능 향상

조정인, 홍신

한동대학교 전산전자공학부
{21600689, hongshin}@handong.edu

Improving Mutation-based Fuzzing by Input Keyword Extraction

요 약

본 논문은 퍼징(fuzzing)을 통한 보다 효과적인 테스트 입력 생성을 위하여 퍼징에 사용되는 테스트 입력 값(seed input)에서 특정 분기 달성에 결정적인 역할을 하는 부분문자열을 입력 키워드로 자동 추출하고, 이를 향후 뮤테이션 과정에 사용하여 분기 커버리지를 향상하는 새로운 방법을 제안한다. 입력 키워드를 사용자가 제공하거나 혹은 테스트 대상 프로그램의 정적 정보를 활용하여 추출하는 이전 방식과 달리, 본 논문이 제안하는 방법은 동적 테인트 분석(dynamic taint analysis)을 통해 입력 텍스트로부터 입력 키워드를 자동 추출한다. 본 연구는 제안하는 기법을 Angora 퍼저에 구현하였고, djpeg을 대상으로 한 실험에서 17.6%의 문맥-분기 커버리지 향상을 확인할 수 있었다.

1. 서론

뮤테이션 기반 퍼징 기법(mutation-based fuzzing)은 주어진 프로그램 입력 값에 연속적으로 변형(뮤테이션 연산)을 적용시킴으로써, 다양한 프로그램 경로를 탐색하는 프로그램 입력 값을 생성하는 자동 테스팅 기법이다[1]. 프로그램의 동적 정보를 활용하여 전략적으로 뮤테이션을 적용하는 Greybox 퍼징 방식이 제안된 이후로 뮤테이션 기반 퍼징을 통한 오류 검출과 테스트 커버리지 달성 성능이 크게 향상되었으며[2], 최근에는 유닛 테스팅, 회기 테스팅 등 다양한 소프트웨어 테스팅 작업에 뮤테이션 기반 퍼징 기법이 널리 사용되기 시작하였다.

뮤테이션 통해 테스팅에 유용한 프로그램 입력을 도출하기 위해서는 (1) 다양한 방향으로 프로그램 입력 값을 변형하면서도, (2) 프로그램 실행경로 전환에 결정적인 값 변화를 발생시키는 효과적인 뮤테이션 연산자(mutator)가 사용되어야 한다. 이러한 두 가지 필요를 동시에 만족하기 위해서, 현재 널리 쓰이는 퍼징 도구의 경우(예: AFL, libFuzzer), 무작위적 뮤테이션 연산자와 입력 키워드(input keyword) 기반한 뮤테이션 연산자를 동시에 사용하는 방식으로 동작한다. 입력 키워드는 테스트 대상 프로그램의 특정 실행 조건에 연관성이 높은 프로그램 입력 값 요소를 별도로 입력 받거나, 테스트 대상 프로그램 코드 등의 정보로부터 수집하여 뮤테이션에 활용하는 방식이 제안되어 왔다.

본 논문은 테인트 분석(taint analysis)를 통해 프로그램 입력 값으로부터 입력 키워드를 자동으로 추출한 후, 이를 활용하여 뮤테이션을 수행하는 새로운 퍼징 기법을 제안한다. 제안하는 기법은 퍼징 과정에서 특정 분기 달성에 결정적인 것으로 추정되는 입력 키워드를 자동으로 추출함으로써, 테스트 입력 탐색 과정에서의 경험을 지식 형태로 표현하여 향후 퍼징 과정에 효과적으로 재활용할 수 있는 방법을 제공하며, 또한 입력 키워드를 사용자가 제공하거나 프로그램 코드로부터 추출하기 불가능한 상황에서도, 프로그램 입력으로부터 추출하여 사용하는 새로운 방법을 제공함으로써 기존의 입력 키워드 기반 뮤테이션 방식을 효과적으로 보완할 수 있다.

본 연구에서는 제안한 기법을 C/C++ 프로그램을 대상으로 하는 Angora 퍼징 도구[3]를 기반으로 활용하여 구현하였다. 특히, Angora 퍼저의 동적 테인트 분석 기능을 활용하여 특정 분기문 달성에 결정적인 역할을 하는 것으로 보이는 프로그램 입력 구간을 추론하여 입력 키워드로 추출하였다. 또한, 추출한 입력 키워드를 Angora 퍼저의 세 가지 뮤테이션 연산 방식에 각각 활용함으로써 프로그램 실행경로의 효율적인 탐색을 유도하였다. 제안한 기법을 Angora에 구현한 후 djpeg 프로그램을 대상으로 실험 평가를 수행한 결과, 제안한 기법의 사용이 17.6%의 문맥-분기 커버리지 성능 향상을 보임을 확인할 수 있었다.

본 논문의 나머지 부분에서는, 먼저 테인트 분석에 기반한 퍼징 도구인 Angora를 소개한 후(2.1절), 본 논문이 제시하는 입력 키워드 추출과 이를 이용한 뮤테이션 방법을 살펴본다(2.2절). 그 후 제안한 기법을 평가하기 위한 실험의 구성과 결과를 소개하고(3장), 향후 연구를 논의함으로써 논문을 마무리한다(4장).

본 연구는 소프트웨어중심대학 지원사업(2017-0-00130)과 한국연구재단의 지원(2020R1C1C1013512, 2017M3C4A7068179)을 받아 수행됨.

2. 입력 키워드 추출을 통한 뮤테이션

2.1. 배경: Angora 퍼저

Angora는 C/C++ 프로그램을 대상으로 한 뮤테이션 기반, Greybox 퍼징 기법이다. Angora는 효과적인 분기 커버리지 달성을 위한 여러 가지 기법이 적용되었는데, 그 중 세 가지 주요 기법의 원리는 다음과 같다:

(1) 세분화된 커버리지인 문맥-분기 커버리지 사용: Angora는 각 분기(branch) 커버리지를 이에 도달하는 당시의 함수 호출 스택(call stack) 경우에 따라 분화한 문맥-분기 커버리지를 달성 목표로 사용하여 퍼징을 수행한다. 문맥-분기 커버리지의 사용은, 실제 프로그램에서, 특정 분기의 달성 조건이 해당 분기를 실행하는 경로에 따라 상이한 경우가 많은 특징에 기인한다. Angora는 다양한 문맥-분기를 도달하도록 퍼징을 수행함으로써, 더 많은 실행 경로를 탐색하게 되고, 이를 통해 더 효과적인 오류 검출 및 커버리지 달성이 이루어지도록 유도한다.

(2) 동적 테인트 분석을 통한 목표 커버리지 대상 뮤테이션 수행: Angora는 데이터 흐름 분석의 일종인 동적 테인트 분석을 활용하여, 도달 목표 분기의 판별 조건 검사에 영향을 미치는 프로그램 입력 오프셋 집합을 파악한 후 해당 오프셋을 대상으로만 뮤테이션을 수행함으로써, 커버리지 목적 달성에 효율적인 테스트 입력 생성을 시도한다. Angora는 동적 테인트 분석의 런타임 오버헤드를 고려하여, 이전에 달성하지 못한 분기를 새로 커버하게 되는 실행에 대해서만 동적 테인트 분석을 수행한다.

(3) 다양한 뮤테이션 방법의 혼합적 사용: Angora는 커버리지 대상 뮤테이션 연산자와 무작위 연산자를 함께 사용함으로써 지향적 탐색과 탐사적 탐색을 복합적으로 수행한다. Angora는 내부적으로 다음 세 가지 종류의 뮤테이션 연산자를 사용한다:

- Exploit: 커버리지 지향적 뮤테이션 연산자로, 동적 테인트 분석을 통해 목표 분기에 연관된 것으로 추정되는 오프셋에 임의의 값 혹은 인근 값을 대입하는 변형을 발생시킨다.
- Explore: Exploit과 유사하게 동적 테인트 분석에서 지정한 오프셋을 임의의 값 혹은 인근 값으로 변형하는데, 이 때 분기 조건식과 적합도가 높은 방향으로 변형을 반복한다.
- AFL: 무작위적 뮤테이션 연산자로, 임의의 입력 오프셋을 지정하여 값을 여러 가지 무작위적 변형을 발생시킨다.

2.2. 제안하는 기법

Angora의 뮤테이션 연산자가 분기 달성에 결정적인 입력 키워드를 사용할 경우, 순차적 혹은 임의적 값 변형보다 효과적이며 효율적인 경로 탐색이 가능하다는 점에 착안하여, 입력 키워드 추출과 추출한 키워드를 활용한 뮤테이션 방법을 다음과 같이 제안한다:

(1) 동적 테인트 분석을 통한 입력 키워드 추출: Angora가 동적 테인트 분석을 통해 특정 분기에 연관된 것으로 추정한 입력 오프셋에 위치한 입력 값을 입력 키워드로 추출한다. 이 때 연관된 입력 오프셋이 불연속적으로 지정된 경우, 연속된 부분 구간별로 입력 키워드를 추출하고, 이에 더하여 가장 이른 오프셋으로부터 가장 늦은 오프셋까지의 전체 구간을 또다른 입력 키워드로 추가로 추출한다. 예를 들어, 프로그램 입력이 “ABCDEFGHIJK”이고 특정 분기에 연관된 오프셋이 {0, 1, 2, 4, 5, 7, 8]로 주어질 경우, “ABC”, “EF”, “HI”, 그리고 “ABCDEFGHI”의 4가지의 입력 키워드가 추출된다.

(2) 입력 키워드를 뮤테이션에 적용: 퍼징 과정에서 추출한 입력 키워드를 저장하여 커버리지 지향 및 무작위 뮤테이션에서 다음과 같이 활용한다:

- Exploit & Explore: 동적 테인트 분석을 통해 목표 분기에 연관된 것으로 추정한 오프셋에 무작위 혹은 인근 값으로 변형하는 뮤테이션에 앞서, 오프셋과 길이가 유사한 키워드를 찾아 대입한다. 오프셋이 연속적이지 않더라도 입력 키워드의 문자열을 순서를 맞추어 대입한다.
- AFL: 기존의 무작위적 뮤테이션 연산자에 추가하여 특정 위치에 추출한 입력 키워드를 삽입하거나, 혹은 특정 위치의 값을 추출한 입력 키워드와 치환하는 변경을 수행한다.

제안한 기법은 특정 초기 입력 값 (initial seed) 혹은 퍼징 과정 가운데 발견된 새로운 커버리지를 달성하는 특정 입력에서 존재하는 고유한 정보(특정 분기 달성에 결정적인 입력 조건)가 다른 여러 뮤테이션 대상 입력에 전파될 수 있는 채널을 제공하는데 목적이 있다. 이를 통해, 퍼징 과정의 특정 시점에 획득한 프로그램 입력 값에 대한 지식이 이후 퍼징 과정에 재활용됨으로써 퍼징의 효율성이 향상되는 효과를 의도하였다.

3. 실험

3.1. 실험 셋팅

본 연구에서는 제안한 기법을 Angora를 기반으로 구현한 후, 기존 Angora와의 성능을 비교하는 실험을 수행하였다. 실험을 위해 제안한 기법을 Angora version 1.2.2에 349 LoC의 Rust 코드를 추가함으로써 구현하였다. 실험대상으로는 퍼징 기법 실험 평가(예: [3][4])에 자주 사용되는 오픈소스 JPEG 압축해제 프로그램인 프로그램으로는 djpeg[1]을 사용하였다. djpeg은 229 LoC의

[1] Independent JPEG Groups. https://github.com/libjpeg-turbo/ijg

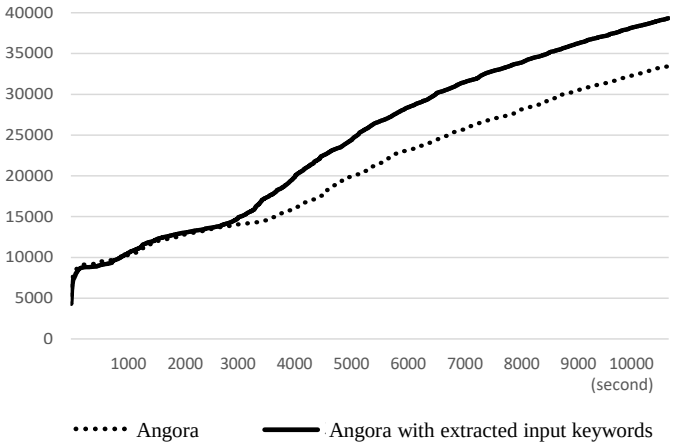

그림 1. 문맥-분기 커버리지 달성

C 코드로 구성되어 있으며, 총 153 개의 분기가 있다. 초기 테스트 입력(initial seed input)으로는 프로젝트 내에 있는 총 4개의 JPEG 이미지 예제 파일 중 2개를 임의로 선택하여 사용하였다.

실험은 기존 Angora와 본 논문이 제안한 기법을 구현한 Angora에 djpeg을 대상으로 각각 180분 동안 퍼징을 수행하는 방식으로 진행하였다. 실험 결과는 최종적으로 달성한 문맥-분기의 수를 측정하였는데, 이는 Angora가 더 높은 문맥-분기 커버리지 달성을 목표로 동작하는 특성을 반영한 것이다. 퍼징의 확률적 동작을 고려하여, 본 실험에서는 180분 간의 퍼징을 총 20회 반복한 후, 최종 커버리지 달성의 평균 값을 비교하였다.

3.2. 실험 결과

총 180분의 퍼징 테스팅 결과, 기존의 Angora는 평균 33440개의 문맥-분기 커버리지를 달성한데 반하여, 본 논문이 제안한 기법은 평균 39341개의 문맥-분기 커버리지를 달성하여 기존 Angora보다 약 17.6% 높은 커버리지 달성을 보였다.

그림 1은 시간에 따라 기존의 Angora (점선)와 본 논문에서 제안한 기법(실선)이 도달한 서로 다른 문맥-분기 커버리지의 총 개수의 증가 추세를 나타낸다. 이 결과를 통하여, 대부분의 시점에서 본 논문이 제안한 기법이 기존의 Angora보다 높은 커버리지 성능을 보임을 확인할 수 있다. 본 논문이 제안한 기법은 총 퍼징 시간동안 평균 1770개의 입력 키워드를 추출하였다. 추출한 입력 키워드의 평균 길이는 124바이트이며, 이 중 16바이트 미만의 입력 키워드의 비중은 평균 23.2%이며, 127바이트 미만의 입력 키워드의 비중은 평균 43.2%였다. 그림 2는 퍼징 과정에서 시간에 따라 추출한 입력 키워드의 평균 개수를 나타낸다. 그림 3은 각 퍼징에서 뮤테이션 대상 입력으로 사용하는 입력 값(favored seed input)의 개수, 즉 퍼징 과정에서 고유한 문맥-분기를 신규로 도달한 입력 개수가 시간에 따라 어떻게 증가하는 지를 나타낸다. 그림 3의 결과로 볼 때,

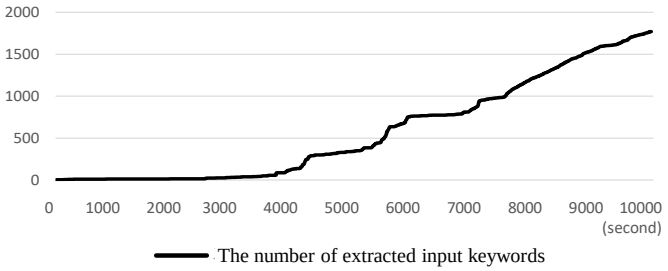

그림 2. 추출한 입력 키워드의 수

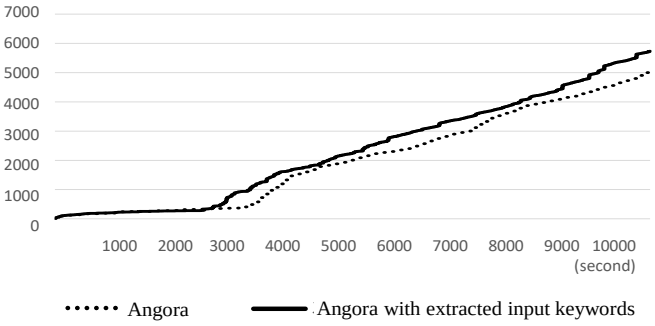

그림 3. 뮤테이션 대상 입력의 수

본 논문이 제안한 기법이 Angora보다 더 많은 문맥-분기에 도달하기 때문에, 퍼징 과정에서 더 많은 수의 뮤테이션 대상 입력을 활용하여 퍼징을 수행하였음을 확인할 수 있다.

4. 결론 및 향후연구

본 논문에서는 효과적인 뮤테이션 기반 퍼징을 위해 동적 테인트 분석을 통해 입력 키워드를 자동으로 추출하여 사용하는 방법을 제안하고, 이를 C/C++ 프로그램 대상 Fuzzing 도구인 Angora를 활용해 구현한 결과를 소개하였다. djpeg을 대상으로 한 실험 결과, 본 논문에서 제안한 방법은 Angora의 문맥-분기 커버리지 달성 성능을 17.6% 향상함을 확인할 수 있었다.

향후 연구에서는 다수의 입력 키워드가 추출되는 상황을 고려하여 입력 키워드를 선택적으로 저장하고 활용하는 방법을 개발하고자 한다. 특별히, 입력 키워드를 추출할 당시 연관된 분기문을 파악함으로써 뮤테이션 향상에 활용할 수 있을 것으로 기대한다. 또한, 동적 테인트 분석 이외의 방식을 통해[5], 보다 효율적으로 입력 키워드를 추출하는 방법에 대해서도 연구하고자 한다.

참조문헌

[1] P. Godefroid, Fuzzing: Hack, Art and Science, Communications of the ACM, 63 (2), Jan. 2020
[2] V. J. M. Manes et al., The Art, Science, and Engineering of Fuzzing: A Survey, IEEE Transactions on Software Engineering (early access)
[3] P. Chen and H. Chen, Angora: Efficient Fuzzing by Principled Search, IEEE Symposium on Security and Privacy (SP), 2018
[4] S. Rawat et al., VUzzer: Application-aware Evolutionary Fuzzing, NDSS Symposium, 2017
[5] S. Gan et al., GREYONE: Data Flow Sensitive Fuzzing, USENIX Security, 2020