연속적 퍼징을 위한 유닛 테스트는 어떻게 변화하는가: OSS-Fuzz의 퍼징 테스트 케이스 변경 연구

김지웅, 김서예, 홍신

한동대학교 전산전자공학부

jeewoong@handong.ac.kr, 21800088@handong.ac.kr, hongshin@handong.edu

How Does a Unit-level Test Case for Continuous Fuzzing Evolves: An Empirical Study of Code Changes in OSS-Fuzz Projects

요약

본 논문에서는 연속적 퍼징을 위해 실제 개발자들이 어떠한 유지 보수 작업을 수행하고 있으며, 어떠한 기술적 어려움을 겪고 있는 지 이해하기 위해 OSS-Fuzz 프로젝트에서 퍼징 테스트 코드 변화를 조사 분석한 결과를 소개한다. 본 연구에서는 총 50개 프로젝트에서 수집한 690건의 퍼징 테스트 코드 변경 사례를 변경 대상, 변경 목적, 변경 방식을 기준으로 분류한 후, 각 분류별로 대표적인 유형이 무엇인 지 파악하여 정리하였다. 또한, 조사 결과를 바탕으로, 연속적 퍼징을 위한 테스트 케이스의 효과적인 유지보수를 지원하는 퍼징 기술과 퍼징 관행의 개선 방향을 논의한다.

1. 서론

연속적 퍼징(continuous fuzzing)[1]이란 검증대상 프로젝트의 코드 변경이 이루어져 새로운 버전이 등록될 때마다 자동으로 퍼저(fuzzer)를 구동하여 테스트 입력 생성을 통한 자동 테스팅을 지속적으로 수행하는 검증 방법론이다. 연속적 퍼징은 개발자가 제공한 퍼징 테스트 케이스와 이를 이용하는 퍼저(fuzzer)를 활용하여 코드 변경에 따라 새로운 테스트 입력을 자동으로 확충한다. 최근 Greybox 퍼징 기술의 발전에 따라 이를 이용한 연속적 퍼징이 새로운 검증 관행으로 자리잡고 있다[2]. 최근에는 오픈소스 라이브러리 프로그램에 대한 유닛 테스트 입력 생성을 위해 연속적 퍼징 방법이 널리 적용되어 많은 오류와 보안취약점을 성공적으로 검출하고 있다[2].

연속적 퍼징은 개발자가 코드 변경에 따른 신규 테스트 케이스를 직접 작성해야 하는 비용을 감축할 수 있는 반면, 개발자가 프로젝트의 변화에 맞춰 퍼징 테스트 케이스를 지속적으로 수정, 보완, 개선해야 하는 새로운 형태의 유지보수 비용을 발생시킨다. 유닛 테스트 퍼징의 경우, 개발자는 퍼저가 검증대상 함수를 구동할 수 있도록 연결하는 테스트 드라이버인 퍼징 타겟(fuzzing target)을 작성해야 한다. 효과적인 퍼징을 위한 퍼징 타겟 개발은 까다로운 작업으로, 이를 유지 보수하는데 많은 비용이 발생할 수 있다.

본 논문은 이와 같은 연속적 퍼징의 적용에 따른 새로운 형태의 프로젝트 유지보수 비용이 무엇인지 이해하기 위해, 실제로 연속적 퍼징을 적용한 프로젝트에서 어떠한 퍼징 테스트 케이스 변경이 발생하는 지를 조사 분석한 결과를 소개한다.

구체적으로, 본 논문은 OSS-Fuzz[3]에 등록된 50개 프로젝트로부터 퍼징 테스트 케이스의 코드를 변경한 690건의 커밋을 수집한 후(2장), 이들을 변경 대상(3.1절), 변경 목적(3.2절), 변경 방식(3.3절, 3.4절)에 따라 분류하고 관찰한 결과를 소개한다. 조사 분석 결과, 퍼징 타겟 코드는 오류 수정, 적응, 테스트 효용성 향상, 정리의 네 가지 목적을 이루기 위해 발생함을 파악할 수 있으며, 오류 수정과 테스트 효용성 향상을 위한 코드 변경에는 다양한 유형이 있음을 파악할 수 있다. 조사 결과를 바탕으로, 본 논문에서는 연속적 퍼징을 위한 테스트 케이스의 효과적인 유지보수를 지원하는 퍼징 기술과 퍼징 관행의 개선 방향을 논의한다(4장).

OSS-Fuzz는 널리 사용되는 중요한 오픈소스 프로젝트에 대해 클라우드 자원을 활용해 연속적 퍼징을 제공하는 프로젝트다. 최근 OSS-Fuzz 프로젝트가 검출한 오류와 결함에 대한 임상적 연구가 소개된 바 있다[4]. 본 연구는 OSS-Fuzz의 퍼징 테스트 케이스의 변화에 집중함으로써 기존 연구와 차별화된 새로운 관점으로 OSS-Fuzz의 연속적 퍼징 관행을 연구하였다.

본 연구는 한국연구재단(NRF-2020R1C1C1013512, NRF-2021R1A5A1021944)과 소프트웨어중심대학지원사업(2017-0-00130)의 지원을 받아 수행되었음.

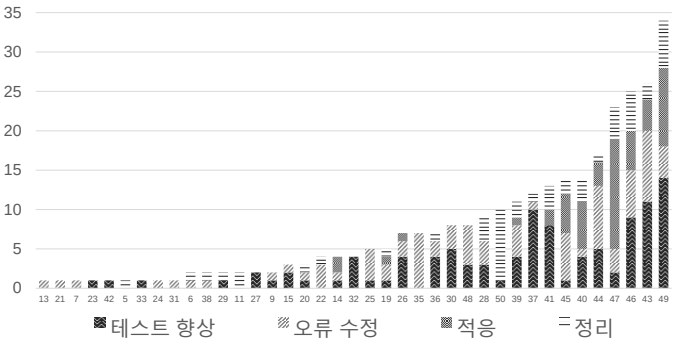
2. 조사 대상과 조사 방법

2.1. 조사대상 프로젝트. 연속적 퍼징이 적용되는 실제 프로젝트 상황을 이해하기 위하여 본 연구는 OSS-Fuzz에 등록된 검증대상 프로젝트 중 총 50개를 조사대상으로 사용하였다. OSS-Fuzz 코드 저장소에서는 580개 이상의 오픈소스 프로젝트에 대한 퍼징 테스트 케이스가 등록되어 있다.본 연구에서는 조사의 편의를 위해 C/C++로 작성되었고 libFuzzer[1]를 통해 연속적 퍼징을 수행하는 프로젝트 중 가능한 다양한 종류가 포함되도록 임의의 50개 프로젝트를 조사대상으로 선택했다.

2.2. 조사대상 커밋. 선택된 50개 프로젝트에서 발생한 코드 변화를 분석하기 위해 다음의 절차를 거쳐 총 690건의 코드 변화를 정의하였다. 먼저, 각 조사대상 프로젝트에 대해 OSS-Fuzz에 등록된 퍼징 테스트 케이스를 바탕으로 검증대상 프로젝트의 코드 저장소를 파악하고, 그 중 연속적 퍼징에 연관된 파일을 하나 이상 수정한 커밋으로 수집하였다. 단, 각 파일이 최초로 등록된 커밋은, 코드 변경과는 무관하므로 조사대상에서 제외하였다. 조사대상 프로젝트는 최소 1개, 최대 51개, 프로젝트 당 평균 14개의 조사대상 커밋을 가진다.

2.3. 조사대상 코드변경. 각 조사대상 커밋이 변경하는 대상을 파악하여 세 가지 경우로 분류한 후(3.1절), 각각을 조사대상 코드변경으로 수집하였다. 이 때, 하나의 커밋에 두 개 이상의 독립된 변경이 같이 있는 경우 각각을 별개의 변경으로 정의하였다. 퍼징 타겟에 대한 코드변경의 경우, 총 294개의퍼징타겟 코드변경을 수집하였고, 각 코드변경의 커밋 메시지, 코드 변경 내역, 관련 깃헙 토론과 이슈 리포트를 종합적으로 검토하여 연구질문에 따라 세부적으로 분류하였다(3.2절~3.4절). 조사대상 코드변경의 검토, 분류와 연구질문별 판별은 퍼징 기법에 대한 이해와 개발경험이 있는 저자 간의 합의를 통해 결정하였다. 또한, 주어진 정보로부터 변경 내용을 이해하기 어려운 30개 코드변경은 조사대상에서 배제했다. 본 연구에서 조사한 총 294개 퍼징 타겟 코드 변경에 대한 정보는 아래 저장소에 공개한 데이터를 통해 확인할 수 있다:

http://github.com/arise-handong/oss-fuzz-study



<그림 1. 프로젝트별 퍼징타겟 코드변경 개수>

3. 연구질문과 조사결과

본 연구에서는 퍼징 테스트 케이스가 변경되는 원인과 형태를 이해하기 위해 네 개의 연구질문을 구상하여 조사를 수행했다.

3.1. 퍼징 테스트케이스의 어떤 요소가 수정되는가? 총 690 개의 조사대상 커밋에서 변경된 구성 요소를 파악한 결과, 검증대상 프로젝트의 빌드 및 실행 환경 구성 스크립트를 변경한 경우가 419 개로 가장 많았으며, 퍼징 타겟 코드 변경이 262 개, 초기 테스트 입력 변경이 9 개로 파악되었다. 각 유형별에 대한 관찰은 다음과 같다:

- 빌드 및 실행 환경 구성 스크립트를 변경한 커밋 중 29 개는 퍼징 타겟의 실행에서 오류 검출을 위한 동적 분석기(예: AddressSanitizer)에 대한 설정을 변경하는 경우다. 그 외는 퍼징 타겟의 동작과 무관하게 검증대상 프로젝트의 코드 변화를 반영하거나, 외부 라이브러리 또는 테스트 실행 환경의 구성의 변경에 해당한다.
- 퍼징 타겟을 수정한 커밋은 총 39 개 프로젝트에서 1 건 이상 존재한다. 테스트 입력 생성을 직접적으로 조정하기 위한 경우로, 이에 대한 자세한 내용은 3.2 절의 연구질문을 통해 분석하였다.
- 초기 테스트 입력을 변경한 9 개는 검증대상이 허용하는 다양한 종류의 입력에 대해서 퍼징을 수행하기 위해 새로운 종류의 초기 테스트 입력을 추가한 경우이다.

3.2. 퍼징 타겟 코드 변경은 어떠한 목적을 달성하기 위해 이루어졌는가? 조사대상 퍼징 타겟 코드변경을 분석한 결과 다음 네 가지 주요 변경 목적을 파악할 수 있었다:

- 오류 수정(86 개): 퍼징 타겟 자체의 오류로 인해 의도한 테스팅을 올바르게 하지 못하는 문제를 해결하는 변경
- 적응(50 개): 검증대상 코드의 기능이나 인터페이스가 변경됨에 따라 원래 의도한 테스팅 목적이 계속해서 달성되도록 유지하고자 코드 변경
- 테스트 향상(110 개): 새로운 테스팅 목표를 추가하거나 테스팅 효과를 향상하기 위한 변경
- 정리(48 개): 테스팅 목적이나 동작과 관련 없이 퍼징 타겟 코드 관리의 편의를 위한 편집

그림 1 은 프로젝트별 퍼징타겟 코드변경의 개수와 이들이 네 가지 목적에 따른 분류의 비중을 나타내는 그래프다. 각 변경 목적 별 세부 분류와 관찰은 다음과 같다:

- **오류 수정**: 검증대상의 동작에 대한 이해 부족으로 잘못된 테스트를 수행하는 경우가 33 개로 가장 많은 비율을 차지하였고, 퍼징 타겟 작성 중 발생한 프로그래밍 실수를 해결하기 위한 변경이 24 개 존재하였다. 퍼징 타겟의 빌드 문제를 해결하기 위한 변경은 22 개가 발견되었다. 이들은 검증대상 코드와 퍼징 타겟 코드가 별도로 개발되다가 이를 통합하는 상황에서 발생한 것으로 추정된다. 마지막으로, 연속적 퍼징의 실행환경을 올바로 고려하지 못해 발생한 오류를 해결하기 위한 코드변경이 7 개 존재한다.
- **적응**: 39 개의 코드변경은 검증대상 프로젝트의 코드와 퍼징 타겟의 관련된 부분을 동일 커밋에서 갱신한 경우로, 검증대상 코드와 퍼징 타겟 코드가 통합적으로 관리되는 경우로 볼 수 있다. 8 개는 검증대상 프로젝트의 코드 변경이 먼저 이루어지고, 그 이후 별도의 커밋에서 퍼징 타겟이 이에 대한 변경이 뒤늦게 이루어지는 경우로, 퍼징 타겟이 검증대상과 별도로 관리되는 상황으로 보인다. 총 5 개의 프로젝트에서 이와 같은 경우가 발견되었으며, 퍼징 타겟이 수정되기 전까지 퍼징 타겟은 빌드 오류 혹은 경고(warning)이 발생하는 문제가 평균 5 일 간 발생하였다. 남은 3 건은 검증대상 코드에서 탐지된 오류가 오랫동안 수정되지 않아, 퍼징 타겟이 해당 오류를 발생시키는 테스트를 회피하도록 조정한 경우다.
- **테스트 향상**: 퍼징 적용 대상을 늘리기 위한 경우가 68 개, 동일한 검증대상에 대한 테스트 효용성을 높이기 위한 경우가 20 개, 퍼징의 효율성을 향상하기 위한 경우가 22 개로 파악되었다.
- **정리**: 37 개의 코드변경은 단순 표현 개선(예: 주석 변경), 테스트 동작과 상관없는 불필요 요소(예: 중복된 헤더)를 제거하기 위한 경우다. 나머지 11 개는 여러 퍼징 타겟에서 공통으로 사용하는 코드를 함수로 만들거나, 퍼징 타겟 코드의 위치 변경 등 퍼징 타겟 코드의 관리를 개선하기 위한 변경이다.

3.3. 오류수정을 목적으로 퍼징 타겟 코드변경은 어떻게 오류를 수정하였나? 오류수정을 위한 퍼징타겟 코드변경은 해당 오류의 유형에 다음의 6 가지 유형으로 분류할 수 있다. 각 유형별 코드변경 개수와 세부 분류는 다음과 같다:

- **퍼징 타겟에 존재하는 일반적인 결함을 수정함(21 개)**: 검증대상과 무관하게 퍼징 타겟을 작성하는 중에 발생한 결함을 수정하는 경우로, 결함 종류에 따라 buffer overrun 2 건, null-pointer dereference 가 3 건, integer overflow 가 5 건, memory leak 이 10 건, uninitialized memory 가 1 건 존재한다.
- **퍼징 타겟이 받는 입력의 크기를 축소함(15 개)**: 퍼저로부터 받는 입력이 너무 커서 테스트 실행시간이 초과하는 문제 해결이 12 개, 메모리 용량 초과 문제해결이 3 개 있다. 이에 해당하는 코드변경은 퍼징 타겟이 받는 최대 입력 크기를 축소하였다.
- **테스트 환경에 대한 불필요한 종속성 제거(7 개)**: 퍼징 타겟이 특정 실행환경을 협소하게 가정하여 실제 연속적 퍼징 실행 환경에서 올바른 테스팅이 이루어지지 않는 문제를 해결하는 경우다. OSS-Fuzz 실행 환경을 추가로 고려하도록 수정한 경우가 4 개, 실행 환경의 다양성을 고려하도록 고정된 설정을 다변화한 경우가 3 개 존재한다.
- **검증대상 함수를 올바른 방식으로 실행하지 않는 문제를 해결함(12 개)**: 검증대상 함수를 비현실적인 입력 값이나 상황에서 실행시켜 오경보(실제 오류와 상관없는 테스트 실패[5])가 생기는 문제를 해결한 경우다. 검증대상 함수 입력의 유효성 검사를 추가하는 경우가 9 개, 검증대상 함수를 잘못된 방식으로 호출하는 결함을 수정한 경우가 2 개, 올바른 동작이지만 실행 시간이 너무 긴 경우를 배제하도록 수정하는 경우가 1 개 있다.

- **검증대상 함수의 결과 값을 잘못 처리하는 오류를 수정함(7 개)**: 검증대상 함수가 의도한 예외(exception)를 발생시키거나 에러코드를 반환하는 경우가 있으나, 이를 퍼징 타겟에서 잘못 해석해 발생한 오경보를 해결하는 경우다. 검증대상 함수의 리턴 값을 처리하는 실행 경로를 수정한 경우가 2 개, 검증대상 함수가 발생시킨 예외를 처리하는 실행 경로(예: try-catch)를 수정한 경우가 4 개, 퍼징 타겟의 리턴값을 수정한 경우가 1 개 있다.
- **빌드 오류를 해결함(24 개)**: 퍼징 타겟 코드의 컴파일 실패 등 빌드 오류를 야기하는 실수를 제거하는 코드 변경으로, 각 코드의 상황에 따라 다양한 경우가 존재하였고, 특정한 패턴이 발견되지는 않았다.

3.4. 테스트 향상을 위한 코드 변경은 퍼징 타겟을 어떻게 변경하였는가? 테스트 향상을 위해 퍼징 타겟을 변경한 110 개의 코드 변경의 형태와 방법을 조사한 결과, 7 가지 유형을 정의할 수 있었다. 각 유형에 대한 설명과 해당 코드변경의 개수, 그리고 세부사항에 대한 논의는 다음과 같다:

- **새로운 검증대상에 연속적 퍼징을 적용하기 위해 새로운 퍼징 타겟을 추가함(56 개)**: 가장 많은 수의 코드변경은 새로운 검증대상에 연속적 퍼징을 적용하기 위해 새로운 퍼징 타겟을 개발하는 경우다. 50 개 프로젝트 중 25 개의 프로젝트에서 이러한 변경이 관찰되었는데, 이들은 초기에 제한된 범위의 검증대상에만 퍼징을 적용하다가, 퍼징의 효용성을 확인한 후 점진적으로 퍼징의 적용 범위를 늘린 것으로 추정된다. 이들 중 4 개의 코드변경은 발견된 오류를 수정한 후, 이에 대한 지속적인 검증을 위해 해당 함수에 대한 연속적 퍼징을 적용한 경우다.
- **기존 퍼징 타겟을 합치거나 분리하거나 제거함(5 개)**: 기존의 퍼징 타겟을 합치거나, 분리하거나, 제거함으로써 퍼징 타겟의 수를 조정한 경우가 5 개 존재한다. 이 중 3 건의 변경은 기존에 하나의 퍼징 타겟에서 수행한 테스트를 2 개의 퍼징 타겟으로 분리한 경우로, 해당 검증대상에 대해 더 많은 자원을 배정한 것으로 보인다. 나머지 2 건의 코드변경은 퍼징의 효율성을 높이기 위해 중복적인 테스팅을 수행하는 퍼징 타겟을 제거하였다.
- **기존 퍼징 타겟이 검증대상을 호출하는 테스트 시나리오를 변경함(12 개)**: 기존의 퍼징 타겟의 테스트 효용성을 향상하기 위해 퍼징 타겟의 동작을 변경한 경우는 12 건이다. 이들 중 4 개는 기존의 퍼징 타겟에서 새로운 함수를 호출함으로써 실행 경로를 확장하였고, 다른 3 개는 입력에 따라 선택적으로 실행될 수 있는 실행 경로를 추가하였으며, 또다른 2 개는 예외처리(exception handling)에 관한 실행경로를 추가하는 방식으로 퍼징 타겟을 변경하였다. 다른 3 개는 검증대상 함수를 호출할 때 사용하는 값이나 상태를 변경하여 더 다양한 상황에 대한 테스트가 수행되도록 한 변경이다.
- **테스트 입력 값 검사 조건을 변경함(8 개)**: 퍼저가 생성한 입력을 선택적으로 테스트하도록 프로그램된 퍼징 타겟에서 입력 검사 조건을 변경한 경우가 8 개 존재한다. 이 중 4 개의 변경은 입력이 올바른 문법을 갖추어졌는지에 대한 검사조건을 강화하여 문법이 틀려서 테스트 효용성이 낮은 실행을 방지하였다. 반대로, 다른 4 개의 변경은 입력 검사조건을 약화하여 예외적인 입력에 대해 검증대상이 올바르게 대응하는 지 검사하는 테스트를 추가하였다.
- **테스트 입력 크기를 조정함(6 개)**: 테스트 입력의 크기를 선택하거나 조정하는 변경은 6 개가 있었다. 이 중 특정 크기 이상의 입력을 배제하는 경우가 2 개, 일정 크기 이하를 배제하는 경우가 2 개가 있었다. 나머지 2 개의 경우, 퍼저가 제공한 입력을 다시 가공해 입력 크기를 통제하도록 퍼징 타겟을 변경하였다.
- **불필요한 연산, 명령, 데이터를 제거함(18 개)**: 퍼징의 효율성을 높임으로써 테스트 효과를 개선하기 위한 변경이 18 개 있다. 이들 중 퍼징 타겟의 불필요한 연산 제거가 12 개, 불필요한 함수 호출 제거가 3 개, 대기시간 설정을 줄인 경우가 1 개다. 또한, 데이터에 불필요한 부분을 제거하는 변경이 2 개 있다.
- **퍼징 수행 관찰을 위한 로그생성을 추가함(5 개)**: 퍼징 성능 개선에 활용 하기 위해 퍼징이 생성하는 테스트 입력, 퍼징 중간 결과를 로그로 출력하는 명령을 추가하였다.

4. 논의사항

본 연구의 관찰 결과로 미루어 볼 때, 다음과 같은 기술적 개선이 있을 경우 퍼징 테스트 케이스의 유지보수 관행을 효과적으로 개선할 수 있을 것으로 기대된다:

- **퍼징 타겟의 유효성 검증을 목표로 한 입력 생성**: 검증대상 테스트와는 별개로, 개발자가 작성한 퍼징 타겟의 오류나 비효율성을 탐지하기 위해 퍼저가 다양한 크기, 특수한 값을 가지는 입력을 생성하여 퍼징 타겟 검증의 오류를 효율적으로 탐지할 수 있을 것으로 보인다.
- **탐지한 오류의 오경보 여부 예측**: 퍼징 타겟 변화 변경 유형에서 알 수 있듯이(3.3~3.4 절), 퍼저가 발견한 오류는 퍼징 타겟의 결함이나 검증대상과 퍼징 타겟 사이의 불일치로 인한 오경보일 가능성이 있다. 오경보가 발생하는 유형을 분석해 오경보 여부를 자동으로 예측하는 기술이 연속적 퍼징 운용에 필요할 것으로 보인다.
- **퍼징 파라미터의 전략적 탐색**: 조사 결과로 볼 때 퍼징 타겟의 입력 크기는 퍼징 효용성을 결정하는 중요한 파라미터이므로 입력 크기와 같이 퍼징 성능에 영향을 주는 파라미터를 전략적으로 탐색하기 위해 테스트 입력을 시험적으로 수행하는 기법이 유용할 것으로 예상된다.
- **퍼징 테스트 케이스 빌드 오류 수정 기법**: 본 연구 결과를 통해 알 수 있듯이(3.1 절, 3.3 절), 실제 프로젝트 상황에서 개발자가 검증대상 프로그램의 수정에 맞추어 퍼징 테스트 케이스를 갱신하지 못해 연속적 퍼징이 중단되는 경우(예: 빌드 에러)가 때때로 발생한다. 이러한 문제를 해결하기 위해 검증대상 코드의 변경 정보를 활용해 퍼징 타겟의 코드를 자동 수정하는 기술이 유용할 것으로 보인다.

5. 결론

본 논문에서는 연속적 퍼징을 수행하고 있는 50개 오픈소스 프로젝트의 퍼징 테스트 케이스 변경을 총체적으로 조사하여 퍼징 테스트 케이스 변경의 주요 원인과 코드 변화 유형을 정리하고, 이와 관련된 다양한 관찰 결과를 소개하였다. 향후 연구에서는 연속적 퍼징의 유지보수에 필요한 개발 활동에 대한 이해를 바탕으로, 이를 효과적으로 지원하는 자동화 기법을 연구하고자 한다.

참조문헌

[1] K. Serebryany, Continuous Fuzzing with libFuzzer and AddressSanitizer, IEEE Cybersecurity Development, Nov. 2016

[2] V. Manes et al., The Art, Science, and Engineering of Fuzzing: A Survey, IEEE Transactions on Software Engineering, Oct. 2019

[3] K. Serebryany, OSS-Fuzz: Google's Continuous Fuzzing Service for Open Source Software, USENIX Security, 2017

[4] Z. Ding, C. Le Gouses, An Empirical Study of OSS-Fuzz Bugs, International Conference on Mining Software Repository, 2021

[5] Q. Luo et al., An Empirical Analysis of Flaky Tests, International Symposium on Foundation of Software Engineering, Nov. 2014