OSS-Fuzz 테스트 이력을 활용한 체계적인 프로젝트-교차 결함 사례의 수집

김지웅†, 강영빈‡, 이아청‡, 김문주‡, 홍신†
†충북대학교, ‡한국과학기술원
jeewoong@chungbuk.ac.kr, yeongbin.kang@kaist.ac.kr, acheong.lee@kaist.ac.kr, moonzoo.kim@gmail.com, hongshin@chungbuk.ac.kr

Systematically Collecting Cross-project Bug Cases from OSS-Fuzz Test History

Jeewoong Kim†, Yeongbin Kang‡, Ahcheong Lee‡, Moonzoo Kim‡, Shin Hong†
†Chungbuk National University, ‡KAIST

요약

본 논문은 OSS-Fuzz 프로젝트를 통한 오픈소스 프로젝트의 테스트 이력과 오픈소스 프로젝트 저장소를 활용하여 프로젝트-교차 결함(cross-project bug)으로 발생한 시스템 오류 사례를 체계적으로 수집하는 방법을 소개한다. 제안하는 방법은 OSS-Fuzz의 퍼징 이력과 오픈소스 프로젝트의 개발 이력 사이의 연관 관계를 체계적으로 검토함으로써, 향후 임상적 분석 연구의 실험 대상으로서 요구되는 다양한 결함 정보를 총체적으로 수집한다. 제안한 방법을 7개 오픈소스 프로젝트를 대상으로 적용한 결과, 78건의 OSS-Fuzz 결함 보고를 체계적으로 검토함으로써 실제 프로젝트-교차 결함에 해당하는 2건을 식별할 수 있었다.

Abstract

This paper presents a systematic approach to collect cross-project bug cases by utilizing OSS-Fuzz test history and open-source project repositories. The proposed approach collects diverse bug information required for future empirical analysis by systematically examining the correlation between OSS-Fuzz fuzzing history and the development history of open-source projects. The proposed approach identifies 2 cross-project bug cases by systematically examining 78 OSS-Fuzz bug reports on 7 open-source projects.

1. 서론

오픈소스 라이브러리 프로그램(library program)은 오늘날 소프트웨어 공급망의 핵심 자원으로, 이들의 안정성과 보안성을 검증하기 위한 협력적 노력이 이루어지고 있다. OSS-Fuzz[1] 프로젝트는 Greybox Fuzzing 기술을 활용해 1000개 이상의 주요 오픈소스 라이브러리 프로그램에 대한 지속적 자동 테스팅을 수행하여 다양한 오류와 보안취약점을 검출하여 보고하고 있다[2].

프로젝트-교차 결함(cross-project bug)이란 라이브러리 프로그램에 존재하는 결함으로, 해당 라이브러리의 검증 단계에서는 발견되지 않았으나, 이를 사용하는 클라이언트 프로그램(client program)에서 오류를 발생시키는 결함을 뜻한다[3, 4]. Python 프로젝트에서 발견되는 프로젝트-교차 결함에 대한 Wanwangying Ma와 동료들의 연구[3]에서는, 결함(틀린 코드)이 존재하는 라이브러리 프로젝트를 업스트림 프로젝트(upstream project), 오류가 발생하는 클라이언트 프로그램의 프로젝트를 다운스트림 프로젝트(downstream project)로 칭하였으며, 프로젝트-교차 결함이 유발하는 검증 및 유지보수 문제를 보고하였다[3]. 통상적 소프트웨어 결함과 달리, 프로젝트-교차 결함의 탐지 및 디버깅에는 다음과 같은 새로운 문제가 존재한다:

- **오류 재현의 어려움**: 프로젝트-교차 결함은 업스트림 프로젝트의 기존 검증 과정을 통해 발견에 실패한 결함으로, 오류 재현에 있어서도 기존 검증 체계(예: 유닛 테스트 케이스, 퍼징 드라이버 등)가 유용하지 않은 경우가 많다.
- **결함 분석의 어려움**: 오류가 발생한 코드와 오류의 원인이 되는 코드가 서로 다른 프로젝트에 존재하므로, 통상적인 디버깅 방법을 통해서 오류의 원인이 되는 결함 위치, 오류 해결을 위한 결함 수정 방법을 도출하기 어렵다.
- **결함 해결의 어려움**: 다운스트림 프로젝트의 오류 해결 과정이 외부 프로젝트(업스트림 프로젝트)의 코드 수정에 의존하므로, 즉각적이고 확실한 결함 수정을 만들어내기 어렵다.

이와 같은 문제를 해결하기 위해서는, 다양한 실제 프로젝트 개발 상황에서 발생하는 프로젝트-교차 결함 사례에 대한 조사, 분석이 필요하다. 또한, 프로젝트-교차 결함 문제에 자동 테스팅 및 디버깅 기법을 적용할 수 있는 임상적 연구 환경이 필수적이다.

본 논문은 오픈소스 라이브러리에 대한 대규모 자동 테스팅이 수행되고 있는 OSS-Fuzz 프로젝트의 테스트 이력 데이터를 바탕으로 실제 프로젝트-교차 결함 사례를 체계적으로 수집하는 방법을 제시한다. 단일 프로젝트 내에서 코드 수정과 테스팅이 이루어지는 통상적인 결함

본 연구는 과학기술정보통신부의 재원으로 한국연구재단의 지원을 받아 수행한 연구임(No. 2021R1A5A1021944)

상황과 달리, OSS-Fuzz에서 발견된 프로젝트-교차 결함을 식별하기 위해서는 업스트림 프로젝트의 개발 이력, 다운스트림 프로젝트의 개발 이력은 물론 OSS-Fuzz 내에서 업스트림 프로젝트에 대해 수행하는 연속적 테스팅 이력과 다운스트림 프로젝트에 대한 연속적 테스팅 이력에 대한 연관 관계 분석이 필요하다.

본 연구에서는 이러한 네 개의 코드수정 및 테스팅 이력으로부터 프로젝트-교차 결함을 특정할 수 있는 조건을 정의하였으며(2장), 이를 수집하는 프로세스를 정의하였고, 실제 7개 오픈소스 프로젝트를 대상으로 실제 프로젝트-교차 결함을 발견하는데 성공하였다(3장). 이를 바탕으로 프로젝트-교차 결함에 대한 임상적 연구가 가능할 것으로 기대한다.

2. 연속적 테스팅과 코드 변경 이력을 바탕으로 한 프로젝트-교차 결함 사례 조건

프로젝트-교차 결함은 다운스트림 프로젝트의 오류와 이를 유발한 업스트림 프로젝트의 잘못된 커밋으로 정의할 수 있다. 본 장에서는 업스트림 프로젝트와 다운스트림 프로젝트가 OSS-Fuzz를 통해 연속적 퍼징이 수행되는 상황에서 프로젝트-교차 결함 상황을 식별할 수 있는 조건을 정의한다.

2.1. 배경. 퍼징을 통한 OSS-Fuzz의 연속적 테스팅은 주기적으로 검증대상 프로젝트의 코드 저장소(예: GitHub)로부터 최신 버전을 내려 받아 테스팅을 수행한다. 통상적으로 회귀 테스팅(regression testing)이 매 커밋 마다 수행되는 것과 달리, 많은 테스팅 시간이 소요되는 연속적 퍼징의 경우, 비교적 넓은 시간 간격을 두고 수행된다(예: 하루 한번[6]). 서로 다른 두 프로젝트의 경우, 연속적 테스팅이 각기 다른 스케쥴에 따라 분산적으로 수행된다.

테스팅 과정에서, 이전 버전의 테스팅에서 보고되지 않은 새로운 오류가 발견되는 경우, OSS-Fuzz 이슈 트래커(issue tracker)에 해당 오류를 재현할 수 있는 테스트 입력과 함께 등록한다. 악의적 사용자가 OSS-Fuzz의 이슈를 악용하는 경우를 막기 위하여, 오류 보고는 검증대상 프로젝트의 이슈 트래커가 아닌 별도의 OSS-Fuzz 이슈 트래커에 비공개로 이루어지며, 이슈 등록 이후 한달이 지난 후에 일반에 공개된다. 이러한 관행으로 인해, 검증대상 프로젝트의 개발 이력과 OSS-Fuzz의 테스팅 이력 사이에 추적성(traceability)을 유추하는 작업이 필요하다[7]. 이러한 특성은, 개발자가 코드 변경과 테스팅 이력 사이의 추적성을 면밀히 생성하는 통상적인 개발 관행과 달라 프로젝트-교차 결함 식별을 위한 추가적인 노력을 요구한다.

OSS-Fuzz 서비스는, OSS-Fuzz 이슈 트래커에 새로운 오류가 등록되면, 그 이후 연속적 테스팅 수행 과정에서 오류 재현 테스트 케이스가 계속해서 실패를 발생시키는지를 추적하여 그 정보를 제공한다. 특정 커밋에서 실패가 사라지는 경우, 해당 오류를 해결(resolve)로 표시하며 실패가 처음으로 사라진 커밋을 해결 시점으로 기록한다.

2.2. 프로젝트-교차 결함 상황. 프로젝트-교차 결함 상황은 다운스트림 프로젝트에서 발견된 오류 e와 이를 유발하는 업스트림 코드 변경 b의 쌍, $\langle e, b\rangle$로 정의할 수 있다. OSS-Fuzz 테스팅 이력과 업스트림 프로젝트, 다운스트림 코드 변경 이력을 바탕으로 $\langle e, b\rangle$ 를 정의하기 위해 다음의 커밋(코드 변경), 테스트 수행 이벤트를 정의하였다:

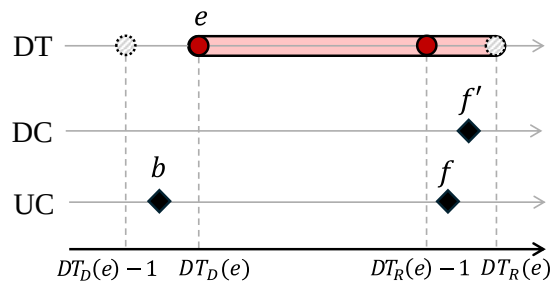
- $DT_D(e)$: 다운스트림 프로젝트에 대한 OSS-Fuzz의 테스트 수행으로 오류 e를 신규로 탐지하여 OSS-Fuzz 이슈 트래커에 등록하는 테스트 수행.
- $DT_D(e)-1$: 다운스트림 프로젝트에 대한 OSS-Fuzz의 테스트 수행으로 $DT_D(e)$ 직전의 테스트 수행.
- $DT_R(e)$: 다운스트림 프로젝트에 대한 OSS-Fuzz 이슈 트래커에서 e 가 해결(resolved)로 최초로 확인한 테스트 수행.
- $DT_R(e)-1$: 다운스트림 프로젝트에 대한 OSS-Fuzz의 테스트 수행 중 $DT_R(e)$ 직전의 테스트 수행. 즉, $DT_D(e)$ 이후 e가 재현된 마지막 테스트 수행.
- b: $DT_D(e)-1$ 시점과 $DT_D(e)$ 시점 사이에 발생한 업스트림 프로젝트의 코드 수정 커밋.
- f: $DT_R(e)-1$ 시점과 $DT_R(e)$ 시점 사이에 발생한 업스트림 프로젝트의 코드 수정 커밋.

이와 같은 e, b, f가 존재할 때, 다음 두 조건이 동시에 만족한다면 $\langle e, b\rangle$ 를 프로젝트-교차 결함 상황으로 볼 수 있다(충분 조건):

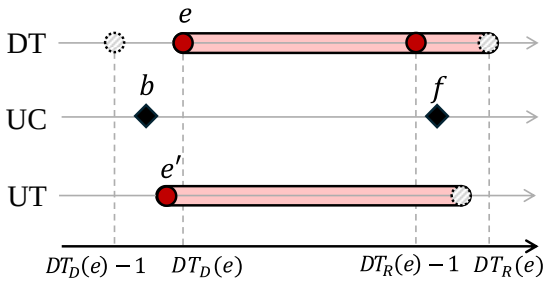
- (조건 1) 다운스트림 프로젝트에서, $DT_R(e)-1$ 시점과 $DT_R(e)$ 시점 사이에 코드 수정 커밋이 없어야 함. 즉, $DT_R(e)-1$ 가 테스팅하는 커밋부터 $DT_R(e)$이 테스팅하는 커밋 직전까지 오류 e는 계속 재현되어야 함.
- (조건 2) 업스트림 프로젝트에 대한 OSS-Fuzz의 테스트 수행에서 $DT_D(e)-1$ 시점과 $DT_D(e)$ 시점 사이에 신규로 등록되며 $DT_R(e)-1$과 $DT_R(e)$ 사이에 해결되는 업스트림 프로젝트 오류가 없어야 함.
- (조건 3) f 에 대한 업스트림 프로젝트 이력(커밋 메시지, 관련 이슈 등)에서는 b 혹은 e를 f의 코드 수정을 수행하게 된 원인으로 명시적으로 언급한다.

첫 번째 조건은 업스트림 프로젝트의 코드 변경 f 가 다운스트림 프로젝트의 테스팅에서 발견된 오류 e 를 해결하는 원인이어야 함을 기술하는 충분 조건이다. 실제 프로젝트-교차 결함 상황에서는 오류 e 를 회피하도록 다운스트림 코드를 수정할 수 있는데(예: work-around patch), 이 경우 첫 번째 조건을 만족하지 않을 수 있다.

그림 1-(a)는 첫 번째 조건이 만족하지 않는 상황으로 다운스트림 프로젝트(DC)에서 f' 이 존재하여 오류 e 가 해결된 원인이 불분명한 상황이다.



(a) 조건 1을 만족하지 않는 상황의 예



(b) 조건 2를 만족하지 않는 상황의 예

그림 1. 프로젝트−교차 결함 조건 판별 상황

두 번째 조건은 코드 수정 b로 인한 오류가 OSS-Fuzz의 업스트림 프로젝트에 대한 테스트 실행에서 탐지되지 않아야 함을 기술하는 충분 조건이다. 이 조건에서는 오류의 최초 탐지되는 시점과 오류가 최초로 해결되는 시점이 같으면 동일한 원인의 오류로 판별했다[7]. 그림 1-(b)는 두 번째 조건이 만족하지 않은 상황을 나타낸다. 업스트림 프로젝트의 테스팅 기록(UT)에서는 b로 인해 등장하고 f로 인해 해결된 업스트림 프로젝트 오류 e'를 e와 동일한 원인에 의한 오류로 판별한다.

세 번째 조건은 e, b, f의 관계를 명확히 하기 위해서 개발자가 남긴 명시적인 증거를 확인한다. 단, 실제 프로젝트-교차 결함 상황이라도 개발자가 이와 관련된 OSS-Fuzz 테스트 수행 정보, 혹은 결함 해결을 위한 코드 수정 정보를 명시하지 않을 수 있다.

3. 프로젝트-교차 결함 사례 수집

3.1. 수집 대상. 제안한 조건을 바탕으로 실제 OSS-Fuzz에서 탐지된 결함 중 프로젝트-교차 결함 사례를 수집하는 사례 연구를 수행했다. 수집대상으로는 OSS-Fuzz 프로젝트의 연속적 퍼징을 수행 중인 C/C++ 프로젝트 중 7쌍을 선정했다. 프로젝트 선정 기준은 다음과 같다: (1) OSS-Fuzz GitHub에 등록된 다운스트림 프로젝트의 Dockerfile에서 최소 2개 이상의 다른 라이브러리를 다운받아야 하며 (총 535개 프로젝트 중 93개에 해당), (2) 최근 3개월 이내에 오류에 대한 상태 변화(예: New에서 Verified)가 있어야 하며, (3) 선정한 프로젝트 사이에 중복이 없도록 선정했다.

3.2. 수집 방법. 실제 프로젝트-교차 결함 사례 쌍을 체계적으로 수집하기 위해 다음 6단계를 거쳤다.

단계 1. OSS-Fuzz 결함 보고 수집: OSS-Fuzz 이슈 트래커의 검색 엔진의 AND 조건을 사용하여 프로젝트 쌍의 이름이 하나의 OSS-Fuzz 결함 보고서에 모두 등장하는 사례를 수집했다. OSS-Fuzz 결함 보고 제목은 “퍼징 대상 프로젝트 이름: 결함 보고 요약”으로 구성되고, 본문은 오류 정보(예: 오류가 발생한 함수의 스택 트레이스, fuzzing target 이름)와 개발자 메세지가 작성되어 있다. 두 프로젝트와 관련된 OSS-Fuzz 결함 보고를 수집하기 위해 검색 항목 중 Title과 All fields 항목을 서로 다른 프로젝트 이름으로 설정함으로써, OSS-Fuzz 결함 보고서의 제목과 본문에서 다른 프로젝트의 이름이 언급되는 $DT_D(e)$를 수집했다.

단계 2. 업스트림 프로젝트의 연속적 퍼징 여부 확인: $DT_D(e)$ 가 등록된 날짜를 기준으로 OSS-Fuzz GitHub에 업스트림 프로젝트의 빌드 스크립트가 존재하는지 확인했다. 연속적 퍼징을 업스트림 프로젝트에 대해서 수행하기 이전에 등록된 $DT_D(e)$ 는 업스트림 프로젝트의 기존 검증 체계의 한계점 파악과 무관하므로 제외하였다.

단계 3. 오류 검증 상태 평가: OSS-Fuzz 결함 보고의 오류 검증 상태인 Status 항목과 개발자 메세지를 활용하여 동일한 오류가 전혀 재현되지 않거나 불규칙적으로 재현되는 오류(Flaky)에 대한 $DT_D(e)$는 제외했다. OSS-Fuzz 서비스는 주기적으로 오류의 발생 여부를 Status 항목에 갱신한다(예: Verified 또는 Open). 또한, 개발자가 해당 오류의 원인이 다른 프로젝트의 결함(즉, 프로젝트-교차 결함)이라고 판단한 경우 Won’t fix 로 Status를 변경한다.

단계 4. 업스트림 프로젝트의 결함 수정 추적성 점검(조건3): OSS-Fuzz 결함 보고 본문에서 구체적인 업스트림 프로젝트의 결함 수정 정보(예: GitHub 커밋)가 언급되는지 확인했다. 오류가 해결되었지만 구체적인 결함 수정 추적성을 찾지 못했거나 아직 결함 수정이 이뤄지지 않은 $DT_D(e)$ 는 업스트림 프로젝트 결함으로 인한 오류로 특정하기 어려워 제외하였다. 또한, 구체적인 언급이 있더라도 수정된 코드 라인 수가 100줄 이상인 경우에는 해당 수정이 리팩토링 또는 기능 변경이 포함되어 있을 수 있기 때문에 제외하였다.

단계 5. 중복 탐지된 업스트림 프로젝트 결함 제외(조건2): 업스트림 프로젝트에 대한 OSS-Fuzz 테스트 수행 기록에서 b 시점에 등장하고 f 시점에 사라진 업스트림 오류 e'이 있는 지 확인한다. 만약 이러한 e' 이 존재한다면, e'은 e와 동일한 업스트림 결함으로 인한 오류를 발견한 것일 가능성이 높으므로 수집 대상에서 제외하였다. 이 때, OSS-Fuzz가 통상적으로 하루에 한 번 퍼징을 수행하는 주기를 고려하여, b 시점과 f 시점의 ±2일 구간을 확인하였다.

단계 6. 다운스트림 프로젝트의 코드 수정 확인(조건1): e가 최초로 해결된 시점에 다운스트림 프로젝트에서 코드 수정 커밋이 존재하는지 확인한다. 이러한 코드 수정이 존재한다면, e 는 다운스트림 프로젝트의 코드 수정으로 해결된 오류일 가능성이 높으므로 수집 대상에서 제외한다. OSS-Fuzz의 통상적인 퍼징 주기를 고려하여, $DT_R(e)$시점의 ±2일구간 다운스트림 프로젝트의 개발 이력을 확인하였다.

표 1. 프로젝트-교차 사례 수집 과정별 OSS-Fuzz 이슈 개수

Downstream	Upstream	# OSS-Fuzz issues					
		I (단계 1)	단계 2	단계 3	단계 4	단계 5	단계 6
Imagemagick	Libheif	11	1	1	1	1	0
Libtiff	Zlib	1	1	0	0	0	0
Libecc	Botan	14	14	14	1	1	1
Bitcoin-core	Libevent	3	3	3	1	0	0
Libvips	Libjxl	27	19	18	10	5	1
Xmlsec	Libxml2	1	1	0	0	0	0
Ffmpeg	Libvpx	21	17	15	1	1	0
Sum		78	56	51	14	8	2

3.2. 수집 결과. 표 1은 사례 수집을 위해 선정한 7개 프로젝트에 대한 수집 단계 별 OSS-Fuzz 결함 보고 개수이다. Downstream열은 다운스트림 프로젝트 이름이고 Upstream열은 다운스트림 프로젝트에서 사용하는 업스트림 프로젝트 이름이다. I 는 단계 1을 통해 다운스트림과 업스트림 프로젝트가 동시에 언급된 OSS-Fuzz 결함 보고를 수집한 개수이고, 단계 2~6은 각 단계를 거쳐 수집된 OSS-Fuzz 결함 보고 개수를 나타낸다.

수집한 78개 중 업스트림 프로젝트가 연속적 퍼징을 수행하기 이전에 보고된 22개 $DT_D(e)$ 는 두번째 단계에서 제외하였다. 단계 3에서 OSS-Fuzz 서비스가 flakiness가 발생하는 것으로 보고한 5개를 제외하였다. OSS-Fuzz 결함 보고 본문의 결함 수정 추적성을 검사했을 때, 업스트림 프로젝트의 결함 수정과 관련된 내용(예: 업스트림 프로젝트 GitHub issue/PR)이 없는 28개 OSS-Fuzz 결함 보고와 결함 수정 정보가 명시되었지만 변경된 라인이 100줄 이상인 9개를 제외하였다. 단계 5에서 업스트림 프로젝트의 결함이 수정된 시점에 업스트림 오류 e' 이 OSS-Fuzz 테스트 수행 기록에 존재하는 6개 사례를 제외하였다. 끝으로, 다운스트림 프로젝트의 코드 변경으로 해결된 가능성이 높은 e를 제외하기 위해 $DT_R(e)$ 시점에서 코드 변경이 발생한 6개 사례를 제외하여 최종적으로 2개 사례를 수집하였다.

4. 논의 사항

본 논문의 프로젝트-교차 결함 조건은 다음과 같은 이유로 인해 실제 프로젝트-교차 결함의 존재를 놓칠 수 있다. 먼저, 다운스트림 프로젝트의 코드 수정 커밋이 $DT_R(e)-1$ 과 $DT_R(e)$ 시점 사이에 없어야 하는 조건은 코드 수정의 내용을 고려하지 않은 제한적인 조건이다. 이는 다운스트림 프로젝트의 코드 수정에 따른 오류 재현 여부를 검사함으로써 조건을 완화할 수 있다. 그리고, 업스트림 프로젝트에 대한 OSS-Fuzz 테스트 이력이 일정 기간 없어야 하는 조건은 다운스트림에 영향을 주지 않는 업스트림의 다중 결함 상황을 고려하지 않은 비현실적인 조건이다. 이러한 상황을 반영하기 위해 프로젝트 간 결함 위치 추적 연구가 필요하다. 끝으로, 조건 3은 f에 대하여 b 혹은 e 의 명확한 언급을 요구하지만, 이는 비공개로 작성되거나 누락될 수 있기 때문에 보수적인 조건이다. 그러므로, 변경된 코드로 인해 오류가 발생하는 지점의 상태가 바뀌는지 검사를 통한 개선이 필요하다.

5. 관련 연구

프로젝트-교차 결함으로 인한 다운스트림 프로젝트에 미치는 영향을 분석하는 연구들이 있다. Wanwangying Ma와 동료들의 연구[4]는 프로젝트-교차 결함으로 판단되는 버그 리포트를 활용하여 기호 실행 경로 조건(symbolic execution path constraints)을 추출함으로써 다운스트림 프로젝트에서 업스트림 프로젝트의 결함 실행 가능성을 확인하는 기법을 제안했다. Yulun Wu와 동료들의 연구[5]는 보안 취약점(CVE)으로 보고된 업스트림 프로젝트의 결함을 수집하여, 다운스트림 프로젝트에서 해당 결함이 포함된 함수의 실행 여부를 함수 호출 그래프를 사용하여 분석했다. 본 논문은 OSS-Fuzz의 테스트 이력과 프로젝트 개발이력을 바탕으로 프로젝트-교차 결함 상황을 특정하여 업스트림 프로젝트에 대한 연속적 퍼징으로 탐지되지 않은 결함으로 발생한 다운스트림 오류 사례를 수집하는 체계를 제안한 점에서 관련 연구와 차별성을 갖는다.

6. 결론 및 향후 연구

본 논문에서는 OSS-Fuzz 이슈 트래커를 활용하여 프로젝트-교차 결함으로 인한 다운스트림 프로젝트의 오류 사례를 체계적으로 수집하는 방법을 제시하였고, OSS-Fuzz 에 등록된 실제 C/C++ 프로젝트 7 개를 대상으로 프로젝트-교차 결함 사례 2 건을 수집하였다. 향후 연구에서는 수집한 결함 사례를 바탕으로 다운스트림 프로젝트의 퍼징으로 탐색한 경로를 추출하여 업스트림 프로젝트의 테스팅을 개선하는 방향을 연구할 계획이다.

참조문헌

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

[2] V. Manes, H. Han, C. Han, S. Cha, M. Egele, E.S, and M. Woo, The Art, Science, and Engineering of Fuzzing: A Survey, IEEE Transactions on Software Engineering (TSE), 47(11), Oct. 2019

[3] W. Ma, L. Chen, X. Zhang, Y. Zhou, and B. Xu, How do Developers Fix Cross-project Correlated Bugs? A Case Study on the GitHub Scientific Python Ecosystem, International Conference on Software Engineering (ICSE), 2017

[4] W. Ma, L. Chen, X. Zhang, Y. Feng, Z. Xu, Z. Chen, Y. Zhou, and B. Xu, Impact Analysis of Cross-project Bugs on Software Ecosystems, International Conference on Software Engineering (ICSE), 2020

[5] Y. Wu, Z. Yu, M. Wen, Q. Li, D. Zou, and H. Jin, Understanding the Threats of Upstream Vulnerabilities to Downstream Projects in the Maven Ecosystem, International Conference Software Engineering (ICSE), 2023

[6] Google OSS-Fuzz - Setting up a new project, https://google.github.io/oss-fuzz/getting-started/new-project-guide/#build_frequency

[7] J. Kim and S. Hong, BugOss: A Benchmark of Real-world Regression Bugs for Empirical Investigation of Regression Fuzzing Techniques, Journal of Systems and Software, Vol. 216, 2024