연속적 퍼징을 위한 유닛테스트는 어떻게 변화하는가? OSS-Fuzz의 퍼징 테스트케이스 변경 연구

김지웅, 김서예, 홍신 (한동대학교)

연구 문제와 조사 대상

- Continuous fuzzing은 코드변경에 따라 fuzzing engine이 fuzzing test case를 활용해 테스트 입력 생성/실행을 지속적으로 수행
- Continuous fuzzing을 효과적으로 유지하기 위해 어떠한 비용이 발생하는 지 실제적, 구체적 이해 필요
  - fuzzing TC는 어떠한 이유로 변경되는가?
  - fuzzing TC 개발에 있어 어떠한 어려움이 발생하는가?
  - fuzzing TC의 유지보수를 지원하기 위해 어떠한 기술이 필요한가?
- Google OSS-Fuzz에 등록된 580개 프로젝트 중, libFuzzer로 unit-level fuzzing을 수행하고 있는 C/C++ 프로젝트 50개
- 지난 6년간의 커밋 중 continuous fuzzing에 연관된 파일을 하나 이상 수정한 690개 커밋 수집 (단, 최초 등록 커밋 제외)
- 코드 변경, 커밋 메시지, 이슈 트래커 기록을 종합적으로 검토하여 변경 대상, 변경 목적, 변경 방식에 따라 분류하고 관찰
  - 데이터 공개: https://github.com/arise-handong/oss-fuzz-study
- fuzzing TC의 효과적인 유지보수를 지원하는 기술과 fuzzing 관행 개선방향 논의

조사 결과와 발견

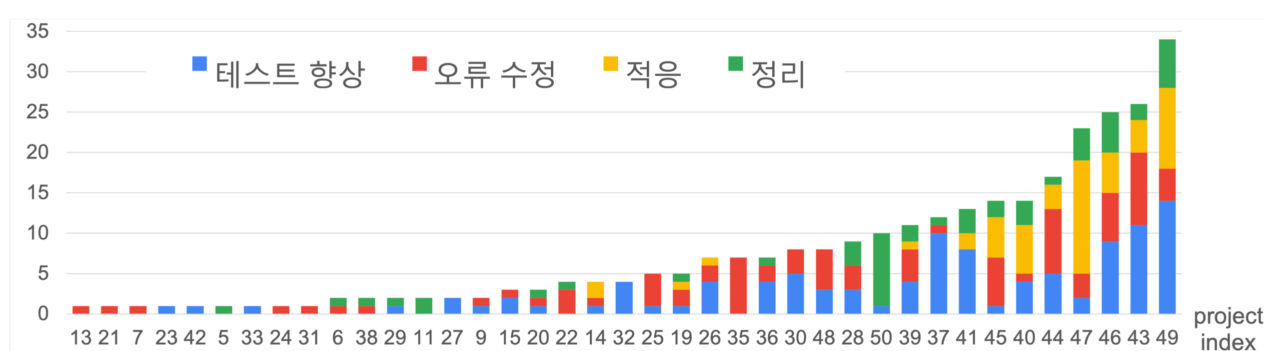
RQ1. fuzzing TC의 **어떤 요소**가 수정되는가?

- 조사대상 커밋의 fuzzing TC 구성요소 별 분포

fuzzing target	fuzzing 환경설정	입력 코퍼스 초기값
262 (38.0%)	419 (60.7%)	9 (1.3%)

RQ2. fuzzing target은 **어떤 이유로 변경**되었는가?

- 오류 수정 (29%): fuzzing target 자체에 존재하는 결함 제거
- 테스트 목표 향상 (36%): 새로운 테스트 대상 추가, 테스트 효과 향상
- 변화에 적응 (18%): 검증대상 코드 변화나 퍼징 경험 반영
- 단순 정리 (17%): 테스팅과 관련 없는 변경, 편리성 향상을 위한 편집



RQ3. fuzzing target 변경은 **어떤 오류**를 고쳤는가?

- 거짓 테스트 실패 (false positive) : 86.04%
  - fuzzing target에 존재하는 일반적인 결함 (21/86)
    - memory leak, null pointer dereference, integer overflow 등
  - 과도하게 큰 입력으로 인한 시간/메모리 초과 (15/86)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02   ...
++  if (input.size() < min_sz || input.size() > max_sz) return 0;
03   ...
04  parser.Parse(input.c_str());
```

- fuzzing 진행 중 특정 크기 이상의 data 값을 받는 경우, fuzzing 환경에서 설정한 제한시간을 초과하여 timeout 오류로 잘못 판별

> 논의사항 1. fuzzing target의 boundary condition을 검사함으로써 fuzzing target 자체가 가지고 있는 결함을 우선적으로 탐색하는 기법이 유용할 것으로 기대됨

- 테스트 환경에 대한 불필요한 종속성 (7/86)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
++    TEST_EQ(true, TestFileExists(f_name));
03    TEST_EQ(true, flatbuffers::LoadFile(f_name, &schemafile);
```

- 테스트를 위한 파일이 존재한다는 가정과 실제 환경의 차이로 인한 오경보

- 검증 대상의 결과 값을 올바르게 처리하지 않음 (7/86)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
03    catch(jsoncons::ser_error e) {}
++    catch(jsoncons::runtime_error<std::runtime_error> e2) {}
```

- 예외 처리 구문 누락, 잘못된 예외 처리 등의 문제를 해결

> 논의사항 2. 검증대상에서 발생 가능한 예외를 검사함으로써 fuzzing target이 검증대상을 올바르게 사용하는지 확인하는 기법이 유용할 것으로 기대됨

- 빌드 오류 (24/86)
- 테스트 미수행 (false negative) : 13.96%
  - 검증 대상을 올바른 방식으로 실행하지 않음 (12/86)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02   ...
03   Dwarf_Debug dbg = 0; _dwarf_get_debug();
04   Dwarf_Error *errp = 0; &err;
05   ...
06   dwarf_init_path(filename, ... , &dbg, 0, 0, 0, errp);
```

- input argument를 올바르게 주지 않아 검증대상 함수의 실행이 조기에 종료되며, 이로 인해 테스트가 수행되지 않음

> 논의사항 3. fuzzing 과정에서 커버리지, 동적 정보를 바탕으로 이상현상을 탐지하여 fuzzing target의 오류 가능성을 경고하는 기술이 유용할 것으로 기대됨

RQ4. **테스트 향상**을 위해 fuzzing target은 어떻게 변경되었는가?

- 커버리지 향상 : 70.90%
  - 새로운 검증대상 함수에 대해 fuzzing target 추가 (66/110)
  - 검증대상을 호출하는 테스트 시나리오 변경 (12/110)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
++    uint32_t compression_arr[8] = { ccJPEG, ... , ccLZW };
++    for (int c = 0; c < 8; c++) {
04       ...
++       writer2.WriteTIFF(..., compression_arr[c]);
++    }
```

- 다양한 동작을 탐색하기 위해 새로운 조건으로 검증대상 함수 실행

- 효율성 향상 : 24.55%
  - 기존 퍼징 타겟을 합치거나 분리하거나 제거 (5/110)
  - 테스트 입력 값 검사 조건을 변경 (8/110)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
03    pj = simdjson::build_parsed_json(Data, Size);
++    if (!pj.is_valid()) throw 1;
04    ...
05    pj.is_ok() ;
```

- 프로그램 동작 탐색에 유용하지 않은 입력을 실행 초기 단계에 배제하여 보다 의미 있는 실행 공간을 탐색하도록 유도

> 논의사항 4. 커버리지 측면에서 의미가 없는 실행 경로 조건식을 유추함으로써 테스트 효율성을 개선할 수 있는 fuzzing target 수정기법이 유용할 것으로 기대됨

- 테스트 입력 크기를 조정 (6/110)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
++    if (size < 500) return 0 ;
```

- 실제 실행 환경과 무관한 크기의 입력을 제외함으로써 현실성 있는 크기의 입력에 대해서만 테스트하도록 수정

- 불필요한 연산, 명령, 데이터를 제거 (8/110)

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
--    h3 = H3_EXPORT(stringToH3)(data) ;
++    memcpy(&h3, data, sizeof(H3Index) ;
03    ...
04    h3NeighborRotations(h3, ... , &rotations) ;
```

- fuzzing engine에 의해 생성된 입력을 특정 포맷으로 가공하는 명령을 제거함으로써 의도하지 않은 입력에 대해서도 테스트하도록 수정

> 논의사항 5. fuzzer가 생성한 입력과 특정 포맷으로 가공된 입력의 커버리지 달성률을 측정하여 효율적인 입력 형태를 제안하는 기법이 유용할 것으로 기대됨

- 분석 능력 향상 : 4.55%
  - 퍼징 수행 관찰을 위한 로그 생성 추가 (5/110)

RQ5. **적응**을 위한 fuzzing target 변경은 언제 발생하는가?

- 검증대상에서 함수 이름, 타입 등의 코드 변화에 맞춰 수정 : 76.0%
  - 검증대상 프로젝트의 코드와 fuzzing target이 동일 커밋에서 갱신되지 않고, fuzzing target의 변경이 뒤늦게 이루어진 경우가 있음

> 논의사항 6. 검증대상 프로젝트의 코드 변경 발생 시, 변경 정보를 활용해 fuzzing target의 코드를 자동 수정하는 기법이 유용할 것으로 기대됨

- 검증대상에 추가된 코드를 탐색하기 위해 fuzzing target 확장 : 12.0%

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
03    psl_is_public_suffix2(psl, domain, PSL_TYPE_ICANN);
++    psl_is_public_suffix2(psl, domain, PSL_TYPE_NO_STAR_RULE);
```

- 검증대상 함수에 추가된 실행 경로를 테스트하기 위해 파라미터를 조정함

> 논의사항 7. 검증대상 함수의 코드 변경 정보를 활용한 파라미터 탐색을 통해 검증대상에 추가된 코드를 실행하는 기법이 유용할 것으로 기대됨

- 검증대상에서 삭제된 코드에 대해 fuzzing target에서 배제 : 6.0%
- 이전 fuzzing 실행 결과를 바탕으로 오류 실행을 회피 : 6.0%

```
01 int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
02    ...
++    parser.opts.json_nested_flatbuffers = false ;
03    TEST_ASSERT( GenerateText(parser, … ) ) ;
```

- 아직 발견되지 않은 버그를 우선 탐색하도록 이미 보고된 버그에 대한 테스트 회피

> 논의사항 8. 버그 정보를 활용하여 버그 발생과 무관한 실행 경로를 우선 탐색하도록 유도하는 기법이 유용할 것으로 기대됨

향후 연구

- fuzzing target의 변경 빈도, fuzzing 실행 결과 등의 동적 정보, 버그 탐지 개수 등을 반영하여 더 많은 패턴 조사 분석 필요
- 연구 결과를 바탕으로 fuzzing TC의 결함, 성능 향상 가능성을 탐지하는 새로운 프로그램 분석 기술에 대해 연구할 계획

HANDONG GLOBAL UNIVERSITY

STAAR SOFTWARE DISASTER RESEARCH CENTER AGILE & AUTOMATED RESPONSE & REPAIR

ARISE Advancing Reliability, Safety & Security in Software Engineering https://arise.handong.edu