**24.01.15 보고서**

**서강대학교 20191244 김현승**

* **Fuzzing 개요[[1]](#footnote-1)**

Fuzzing은 소프트웨어에 예상치 못한 입력을 주어 프로그램이 어떻게 반응하는지 관찰하며 보안 취약점을 찾아내는 자동화된 테스트 방법이다. Fuzzing은 원격 코드 실행 또는 권한 상승 등의 치명적인 보안 취약점을 찾아낼 수 있는 강력한 수단이다. 하지만 여전히 많은 취약점들은 특정 코드 경로[[2]](#footnote-2)를 벗어나거나 Fuzzing의 범위를 벗어나고 있다. 이런 문제점을 해결하기 위해 여러 시도들이 있었는데, 초기 접근 중 하나는 Travis Ormandy의 *Corpus distillation*이다. 이 방법은 커버리지[[3]](#footnote-3) 신호에 의해 대규모의 고품질 코퍼스로부터 흥미로운 시드(좋은 입력 데이터 세트들)을 선택해 퍼징에 사용하는 전략이다. Corpus distillation은 잘 작동하지만 위와 같은 코퍼스들을 사용가능하도록 준비해야하며, 블록 커버리지 측정 방식은 장기적 측면에서 퍼징 과정에 적절하지 않을 수 있는 문제가 있다.

* **AFL-Fuzz**

AFL(American Fuzzy Lop)은 instrumentation-guided genetic algorithm과 결합한 브루트 포스 fuzzer이다. 프로그램 실행 흐름의 미묘하고 지엽적인 변화를 쉽게 감지할 수 있는 변형된 엣지 커버리지를 사용한다. 전체적인 알고리즘의 흐름은 다음과 같다 :

1. 사용자가 제공한 초기 테스트 케이스를 queue에 로드.
2. queue에서 다음 입력 파일을 가져옴.
3. 프로그램의 측정된 행동을 변경하지 않는 가장 작은 사이즈로 테스트 케이스를 줄임.
4. 다양한 전통적인 퍼징 전략을 이용해 반복적으로 파일을 변형.
5. 생성된 변형이 instrumentation[[4]](#footnote-4)에 의해 기록된 새로운 상태 전환을 일으켰다면 변형된 출력을 큐에 새로운 항목으로 추가.
6. 2번으로 돌아가 반복.

여기서 발견된 테스트 케이스들은 새롭고 높은 커버리지가 발견되면 주기적으로 제거된다.

1. <https://github.com/google/AFL> 중 README.md [↑](#footnote-ref-1)
2. 프로그램이 입력에 따라 어떤 명령을 실행하고, 어떤 조건문을 거치며, 어떤 함수를 호출하는지를 나타내는 프로그램의 실행 흐름 [↑](#footnote-ref-2)
3. 테스트 과정에서 프로그램의 어떤 부분이 실행되었는지 나타내는 지표. 넓은 코드 커버리지는 퍼저가 프로그램의 많은 부분을 검사하고 많은 잠재적 취약점을 발견할 수 있다는 것을 의미. [↑](#footnote-ref-3)
4. 계측. 프로그램이 실행될 때 특정 정보를 수집하거나 추가적인 동작을 수행하도록 추가적인 코드를 삽입하는 과정. fuzzing 도중 프로그램의 행동을 모니터링하고, 프로그램이 어떻게 반응하는지 관찰하는데 사용. [↑](#footnote-ref-4)