퍼징 기반 근본 원인 분석 기술에 연구

김현준[1], 윤수빈[1], 백윤흥[1]
[1]서울대학교 전기정보공학부, 반도체공동연구소
hjkim@sor.snu.ac.kr, sbyun@sor.snu.ac.kr, ypaek@snu.ac.kr

A Survey on Fuzzing-based Root Cause Analysis Techniques

Hyun-Jun Kim[1], Subin Yun[1], Yun-Heung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

요 약

퍼징(fuzzing) 기술은 소프트웨어의 취약점을 자동으로 탐지하는 데 있어 강력한 도구로 자리잡았지만, 퍼징으로 발견된 크래시(crash)의 근본 원인(root cause)을 규명하는 과정은 여전히 수작업에 크게 의존하고 있다. 이를 해결하기 위해 퍼징 기반 Root Cause Analysis(RCA)를 자동화하려는 다양한 연구들이 제안되고 있다. 본 논문에서는 대표적인 퍼징 기반 RCA 기법인 AURORA, RACING, BENZENE을 소개한 다음, 향후 연구 방향을 제안한다. 본 연구는 퍼징 기반 RCA 기술의 발전 현황을 정리하고, 완전 자동화된 취약점 근본 원인 분석으로 나아가기 위한 기반을 제공하고자 한다.

1. 서론

퍼징(fuzzing) 기술은 프로그램 입력값을 변형하며 프로그램 내에 존재하는 취약점에 의해 발생하는 버그를 트리거하는 입력값들을 찾아내는 기술이다. 퍼징을 수행하는 퍼저(fuzzer)가 실질적으로 포착하는 것은 프로그램의 비정상적인 종료인 크래시(crash)에 불과하며, 해당 크래시를 유발한 근본 원인(root cause)까지 파악할 수는 없다. 취약점의 근본 원인을 해결하는 패치(patch)를 생성하려면 보안 전문가의 분석이 필수적이다. 그러나 크래시를 역추적하여 근본 원인을 파악하는 근본 원인 분석(root cause analysis, RCA)은 난이도가 높고 많은 시간이 소요된다. 실제로 크래시 하나를 수동으로 분석하는 데 며칠이 소요될 수 있으며, 퍼징으로 수집된 수백 건의 크래시를 사람이 모두 분석하는 것은 비효율적이다.

이에 따라 효율적인 RCA를 수행하기 위해 RCA에 필요한 프로세스를 최대한 자동화하려는 연구들이 활발히 진행되고 있다. 초기에는 역추적 디버깅(reverse debugging)이나 오염 분석(taint analysis) 등을 활용하여 크래시로부터 취약점을 역으로 찾아가는 도구들이 제안되었으나, 명시적 데이터 흐름이 없는 취약점 (ex. 타입 혼동 오류 등)을 분석하는 데에는 한계가 있었다. 최근에는 통계적 디버깅(statistical debugging) 기법을 퍼징과 결합하여 근본 원인 파악에 필요한 다양한 입력값들을 생성하는 방식의 연구들이 발표되었다. 본 논문에서는 퍼징 기반 RCA 연구들을 소개하고, 향후 어떤 방향으로 RCA 연구를 진행할 수 있을지 조망하고자 한다.

2. AURORA[1]

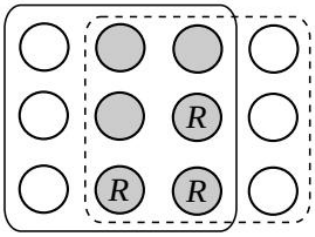


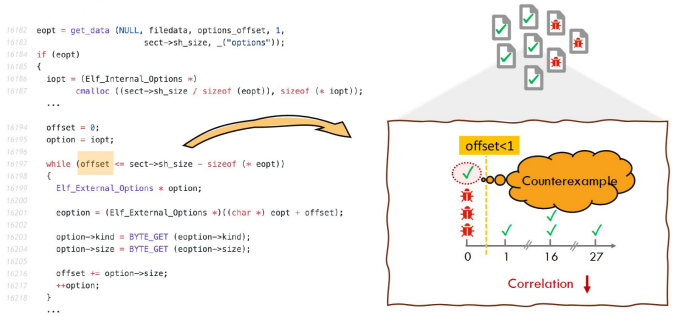
그림 1. 입력과 근본 원인

AURORA는 퍼징 등을 통해 생성한 크래시를 유발하는 초기 입력을 기반으로 추가 퍼징을 수행한다. 이를 통해 여전히 크래시를 일으키는 입력값들과, 크래시를 유발하지 않지만 크래시를 유발하는 입력들과 실행 경로가 유사한 입력값들을 생성한다. 이 과정 중에 레지스터 값, 메모리 쓰기 값의 최소/최대값이나 제어 흐름 등 프로그램 상태를 추적하여 실행 데이터로 저장해둔다. 이 데이터를 기반으로 간단한 불린(boolean) 조건식인 프레디케이트

(predicate)를 대량으로 생성한다. 생성된 조건식은 “특정 레지스트의 최소값이 0x10보다 작다” 혹은 “특정 분기가 항상 수행된다” 등으로 구성된다. 각 프레디케이트와 크래시 발생 간의 상관관계를 통계적으로 분석하여 순위를 매긴다.

그림 1의 예시에서는 특정 입력을 실행하면 실선으로 표시된 블록 안의 프레디케이트들이 참(true)이 되고, 다른 입력을 실행하면 점선으로 표시된 블록 안의 프레디케이트들이 참이 된다. 두 실행 모두 크래시를 유발한다면, 해당 크래시를 유발하는 근본 원인에 해당하는 프레디케이트들(R로 표시)이 교집합 상에 존재하게 된다. 반대로 크래시가 발생하지 않는 입력값들은 이러한 근본 원인 프레디케이트를 포함하지 않을 것이다. 따라서 다양한 입력값들에 대한 실행 데이터를 분석하여 통계적으로 크래시와 높은 관련성을 가지는 프레디케이트를 식별하고 우선순위를 매길 수 있다. AURORA는 25개의 샘플 중 18개의 샘플에서 실제 근본 원인이 Top-10 안에 포함됨을 보였다. 그러나 퍼징에 많은 시간이 소요되고 정확도가 실사용 수준에 미치지 못해 실제 도입에는 제약이 있다는 한계가 있다.

3. RACING[2]



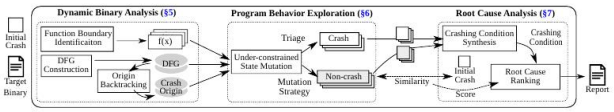
(그림 2) RACING에서 제시된 반례 예시

RACING은 기존 퍼징 기반 RCA 기법인 AURORA의 속도 문제를 대폭 개선한 연구이다. AURORA는 크래시를 일으킨 입력값을 기반으로 퍼징을 수행하면서 새로 생성된 입력값들이 원본 입력값에 편향되는 샘플링 특성을 가지게 되고, 동일한 코드 영역만 대부분 반복 실행하게 되어 RCA에 필요한 다양한 샘플들을 얻는 데 많은 시간이 소요되었다. 이를 극복하기 위해 RACING은 강화학습(reinforcement learning)을 통해 반례(counterexample)를 찾아내는 RCA 특화 퍼징 전략을 도입하였다.

여기서 반례란 취약점과 관련된 것으로 의심되는 코드의 상관관계 추정치를 크게 변동시키거나 (CoR, counterexample for ranking), 특정 프레디케이트를 위반하는 (CoP, counterexample for predicate) 테스트 입력값들을 뜻한다. 이러한 반례는 근본 원인을 판별하는 데에 중요한 단서를 제공할 수 있다. 그림 2에서는 offset이라는 변수가 0인데도 버그가 발생하지 않는 반례를 찾아서 offset이 취약점과 관계성이 상대적으로 낮다는 것을 보여준다.

RACING은 퍼징 과정 중에 이러한 반례에 해당되는 입력값을 찾았을 때 보상(reward)을 주어 강화학습 모델이 반례를 더 효과적으로 탐색할 수 있는 방향으로 퍼징 프로세스를 조정하도록 학습시킨다. 이를 통해 서로 다른 실행 경로를 커버하는 입력값들을 효율적으로 수집할 수 있으며, 수집된 크래시/비크래시 입력값들의 실행 결과를 통계적으로 비교하여 크래시와 직접적으로 관련된 코드들을 빠르게 식별할 수 있다. 실험 결과 RACING은 기존 AURORA 대비 평균 13.22배 빠르게 RCA를 완료했으며, 테스트한 30개 샘플 중 22개에서 실제 근본 원인이 Top-10 안에 포함되었다. 또한 AURORA보다 평균적으로 3.8만큼 근본 원인 랭크를 더 높게 책정하여 근본 원인 식별 성능이 향상되었음을 보였다.

4. BENZENE [3]



(그림 3) BENZENE 구조도

BENZENE은 크래시를 일으킨 “실행” 자체를 직접 변형하여 근본 원인을 빠르고 정확하게 찾아내는 새로운 방법론을 제안하였다. 해당 방법론은 Under-Constrained State Mutation(USM) 이라고 하며, 프로그램이 실제로 크래시가 나기 직전의 상태를 부분적으로 변형하여 ‘거의 동일하지만 정상 종료되는’ 비크래시 실행을 대량으로 만들어낸다. 이를 위해 BENZENE은 프로그램에 대한 데이터 플로우 분석(data flow analysis)을 수행하여 데이터 플로우 그래프 (data flow graphm, DFG)를 생성하거, 이를 역추적해 함수 기준 외부값(함수 인자, 전역 변수 등)에 의존하는 피연산자들만 퍼징 변이 대상으로 삼는다. 그 다음, 해당 함수 진입 직전의 프로그램 상태(state)를 저장해두고 포크(fork)하여 제한적으로 변이(mutation)한 뒤, 크래시 조건에 해당하는 일부

조건문이나 메모리 값을 조작한 후에 (일종의 퍼징) 실행하여 크래시 및 비크래시 실행 결과들을 수집한다. 이렇게 얻은 크래시 경로와 매우 유사하지만 정상 종료된 실행들과, 크래시가 발생한 실행들을 비교 분석함으로써 어떤 지점이 근본원인에 해당하는지를 효과적으로 식별할 수 있다. BENZENE은 테스트한 샘플들 60개 중 93.3%의 샘플들에서 실제 근본 원인을 Top-10 내에 포함시켰다. 또한 Aurora보다 평균적으로 속도가 8.1배 빠르고, 9.1배 적은 메모리 사용량을 기록하였다.

5. 차후 연구 방향

앞서 살펴본 연구들은 퍼징 기법을 응용하여 자동으로 크래시의 근본 원인을 찾아주는 도구들을 제시하였다. 특히 RACING에서 강화학습을 활용했던 것처럼, 향후 AI(artificial intelligence)를 이용한 효율적인 RCA가 큰 잠재력을 가질 것으로 보인다. 최근에는 대규모 언어 모델(LLM)을 기반으로 한 에이전트 시스템이 다양한 소프트웨어 분석 분야에 적용되고 있으며, 이는 기존에 보안 전문가의 경험과 직관에 의존하던 분석 프로세스를 LLM의 지식 기반으로 대체할 수 있음을 보여준다.

이러한 에이전트는 외부 도구 및 환경과 상호작용하여 코드를 실행하고 실시간으로 디버깅 정보(로그, 메모리 덤프 등)를 수집하거나, 코드를 분석하여 다양한 정보를 획득할 수 있을 것으로 기대된다. 또한 기존 퍼징 기반 RCA 기법을 에이전트가 효과적으로 운용하여 자율적으로 RCA를 수행할 수 있을 것이다. 궁극적으로는 기존 기법들의 랭크 결과를 기반으로 분석을 각 predicate에 대해 직접 수행하여 RCA 과정을 완전 자동화하는 것도 가능할 것으로 전망된다.

6. 결론

최근에 제안된 다양한 퍼징 기반 RCA 연구들은 통계적 분석, 강화 학습, 실행 중 프로그램 상태 변형 등 저마다의 혁신적 아이디어로 퍼징에서 발견된 크래시 유발 입력값에 대해 퍼징을 재적용하여 근본 원인을 파악하는 데 많은 도움을 주는 툴을 개발하였다. 이러한 연구들의 성과로 인해 RCA 프로세스 중 많은 부분을 자동화할 수 있게 되었으며, 차후에 사람의 개입이 필요없는 지능형 RCA 에이전트 개발이 기대된다.

7. Acknowledgement

이 논문은 2025년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(RS-2023-00277326)과 2025년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(IITP-2023-RS-2023-00256081)을 받았으며, 2025년도 BK21 FOUR 정보기술 미래인재 교육연구단의 지원을 받았으며, 반도체 공동연구소 지원을 받아 수행된 연구임.

참고문헌

[1] Blazytko, Tim, et al. "{AURORA}: Statistical crash analysis for automated root cause explanation." 29th USENIX Security Symposium (USENIX Security 20). 2020.

[2] Xu, Dandan, et al. "Racing on the Negative Force: Efficient Vulnerability {Root-Cause} Analysis through Reinforcement Learning on Counterexamples." 33rd USENIX Security Symposium (USENIX Security 24). 2024.

[3] Park, Younggi, et al. "Benzene: A practical root cause analysis system with an under-constrained state mutation." 2024 IEEE Symposium on Security and Privacy (SP). IEEE, 2024.