협업 퍼징 기법에 대한 연구

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

A Survey on Collaborative Fuzzing Techniques

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

요 약

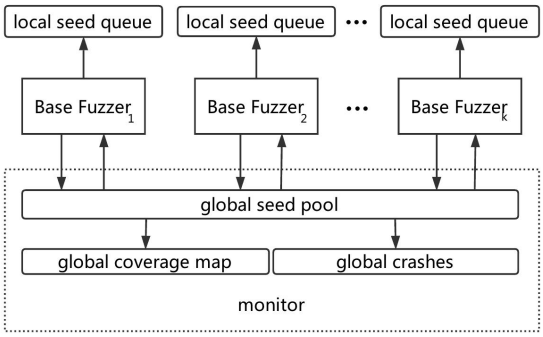
본 논문에서 여러 종류의 퍼저들을 주기적으로 동기화하여 효율적으로 퍼징을 수행하는 협업 퍼징 기법 연구들을 소개한다. 정적으로 효율적인 퍼저의 조합을 찾거나 동적으로 각 퍼저에 리소스를 효율적으로 할당하는 협업 퍼징 기법들 3가지를 정리하고, 향후 연구 방향을 조망하고자 한다.

1. 서론

퍼징 (fuzzing) 기법[1]은 특정 소프트웨어 내의 버그 (bug)를 자동으로 찾아내는 소프트웨어 검증 기법의 하나이다. 해당 기법은 적법한 시드 입력값 (seed)을 기반으로 다양한 입력값들을 생성하여 소프트웨어에 입력하며, 소프트웨어가 실행되었을 때 얻을 수 있는 여러 정보를 활용하여 버그를 트리거할 수 있는 특별한 입력값들을 생성한다. 효율적으로 이러한 버그를 트리거하는 입력값들을 찾아내기 위해 소프트웨어의 정적 정보, 동적 정보를 활용하는 수많은 퍼징 기법들이 개발되어 왔다. 이러한 퍼징 기법들은 특정 소프트웨어에 대해서는 퍼징 성능이 높지만, 다른 소프트웨어에 대해 퍼징 성능이 낮을 가능성이 있다. 이에 따라 특정 소프트웨어에 대해 어떤 퍼저 (fuzzer)를 사용해야 효율적으로 버그를 트리거하는 입력값들을 찾을 수 있는지 알아내는 챌린지가 존재한다. 협업 퍼징 기법 (collaborative fuzzing)은 동시에 여러 개의 서로 다른 퍼저를 구동하여 해당 챌린지를 해결하고자 하며, 적절한 퍼저의 조합을 선택하거나 각 퍼저에 대해 효율적인 리소스 할당을 수행한다. 본 논문에서는 퍼저를 사전에 조합하여 구동하는 정적 협업 퍼징 기법 2개와 동적으로 리소스를 할당하는 동적 협업 퍼징 기법에 대해 소개하고, 이를 기반으로 향후 관련 연구 방향을 탐색하고자 한다.

2. 정적 협업 퍼징 기법

2-1. EnFuzz [2]



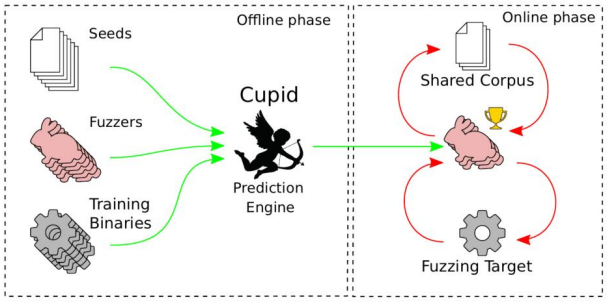
(그림 1) EnFuzz 구조도

Enfuzz는 다양한 퍼저들의 특성을 분류하여 적절한 퍼저의 조합을 탐색하며, 각 퍼저들의 입력 큐 (queue)들에 들어가는 시드 입력값을 동기화하여 효율적인 퍼징을 수행한다. 먼저 각 퍼저들을 코드 커버리지 (coverage) 정보 세분성 (granularity), 입력값 생성 전략, 시드 입력값 선택 및 입력값 변형 (mutation)의 다양성에 따라 분류하였다. 그 다음 최대한 다양성이 높도록 퍼저들의 조합을 선정하였고 AFL, AFLFast [3], FairFuzz [4], QSYM [5], libFuzzer [6], Radamsa [7] 를 선택하여 협업 퍼징을 진행하였다. 소프트웨어 내에 여러 개의 프로그램 실행 경로 (program execution path) 가 존재할 때 각 퍼저는 각자의 방식을 따라 퍼징을 수행하고,

각 퍼저가 특화된 실행 경로를 따라서 퍼징을 수행하고 전체 코드 커버리지를 효율적으로 증가시킨다. (그림 1)은 EnFuzz의 구조도로, 각 퍼저는 고유의 로컬 시드 입력 큐 (local seed queue)를 보유하고 있으며 각 퍼저들 간에 특별 입력값을 전역 시드 풀 (global seed pool)을 통해 공유한다. 특별 입력값은 이전에 나타나지 않은 새로운 프로그램 경로를 지나가게 하여 버그를 일으킬 확률을 높아지게 하는 입력값 혹은 버그를 발생시켜서 다른 버그를 발생시킬 확률을 높아지게 하는 입력값에 해당한다. 전역 시드 풀은 각 퍼저로부터 공급되는 특별 입력값을 저장해두고, 주기적으로 저장된 특별 입력값들을 모든 퍼저들에게 전송한다. 각 퍼저들은 공유된 특별 입력값을 기반으로 새로운 특별 입력값들을 효율적으로 생성할 수 있다.

2-2. Cupid [8]

Cupid는 각 퍼저들을 구동해서 얻은 결과를 바탕으로 해당 퍼저들 간에 협업 퍼징을 수행할 경우 어떤 퍼저들이 서로 상호보완적으로 도움을 줄 수 있는지 자동으로 알아내는 연구를 진행하였다.



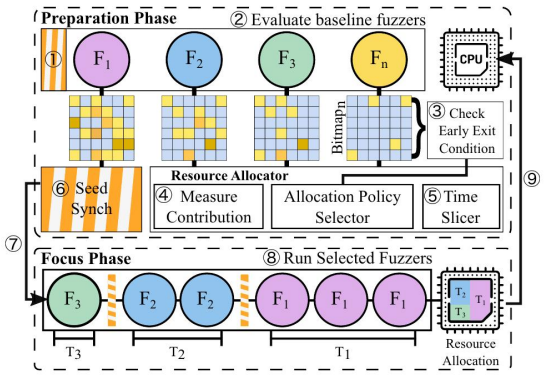
(그림 2) Cupid 구조도

(그림 2)는 Cupid의 구조도로, 학습 바이너리들을 활용하여 각 퍼저의 특성을 나타내는 확률 모델을 학습하여 해당 퍼저를 구동했을 때 확률적으로 특정 경로가 커버될 확률을 추론한다. 이는 학습 바이너리들에 대해 여러 번 퍼징을 수행해서 여러 개의 분기 경로 중 하나를 결정하는 분기문 (branch)이 실제로 얼마나 커버되는지 개수를 세어 실제 확률을 계산한다. 이를 바탕으로 특정 프로그램에 대해 두 개의 퍼저를 구동했을 때 두 퍼저가 커버하는 프로그램 경로가 최대한 다르게 채워지도록 최적의 퍼저 조합을 선택한다. 이에 따라 동일한 시간에 새로운 경로를 최대한 많이 발견할 수 있다.

3. 동적 협업 퍼징 기법

3-1. autofz [9]

autofz는 정적으로 퍼저들의 조합을 선택해서 동시에 퍼저들을 구동하는 기존 연구들과 달리 실시간으로 퍼저들에게 할당하는 리소스를 조정하여 효율적으로 퍼징을 수행한다. 이는 단일 CPU 코어에 대해 라운드 로빈 (round robin) 방식으로 나누어 할당할 수 있고, 혹은 여러 개의 CPU 코어들에 나누어 할당할 수 있다. (그림 3)은 autofz의 구조도로, 해당 연구에서 개발한 알고리즘을 설명하고 있다. autofz는 적절한 리소스 분배를 위해 각 퍼저들의 동작에 대한 모니터링을 수행한다. 준비 페이즈 (preparation phase)에서 모든 퍼저들에게 동일한 리소스를 할당하고 어떤 경로가 커버되었는지를 나타내는 비트맵 (bitmap)을 수집한다. 이 중 각 퍼저가 고유하게 기여한 비트 (bit) 수를 측정하여 각 퍼저에 할당할 리소스를 결정한다. 만약 기여도가 가장 높은 퍼저와 기여도가 가장 낮은 퍼저간 기여 비트 수 차이가 크면 초기 탈출 (early exit)를 시도하여 준비 페이즈를 바로 종료하고 가장 기여도가 높은 퍼저에 모든 리소스를 할당한다. 만약 초기 탈출을 하지 않고 정해진 시간이 지나서 준비 페이즈를 종료하게 되면 기여 비트 수에 비례해서 퍼저들에게 리소스를 할당한다. 집중 페이즈 (Focus Phase)에서는 리소스가 할당된 이후에 EnFuzz와 같은 방식으로 협력 퍼징을 수행한다. 이후 준비 페이즈와 집중 페이즈를 반복하며 효율적으로 퍼징을 수행한다.



(그림 3) autofz 구조도

4. 차후 연구 방향

autofz는 준비 페이즈에서 각 퍼저들에 동일한 리소스를 할당하기 때문에 비효율성이 존재하며, 퍼징 후반부에는 준비 페이즈에서 새로운 비트가 추가되지 않는 경우가 많아져서 (준비 페이즈에서 새로운

경로를 찾지 못한 경우) 효율적인 리소스 할당을 실패하게 된다. 위와 같은 상황이 발생하여 autofz의 리소스 할당 효율성이 떨어질 경우, 딥러닝 학습 기법을 적용하여 효율성이 떨어지는 것을 완화할 수 있을 것으로 기대된다. 현재 입력값에 대한 프로그램 실행 정보 혹은 코드 내부 정보를 딥러닝 모델로 학습하여 현재 상황에 알맞게 퍼저들에 대해 리소스를 할당하는 것이 가능할 것으로 예상된다.

5. 결론

본 논문에서 다양한 퍼저들을 조합해서 효율적인 퍼징을 수행하는 협업 퍼징 연구들의 동향을 살펴보았다. 차후에 협업 퍼징에 사용되는 퍼징들의 스케쥴링 과정에 있어 효율성을 더 높일 수 있을 것으로 기대된다.

6. Acknowledgement

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(RS-2023-00277326)과 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(IITP-2023-RS-2023-00256081)과 2024년도 정부(과학기술정보통신부)의 재원으로
정보통신기획평가원의 지원(No.2020-0-01840,스마트폰의 내부데이터 접근 및 보호 기술 분석)을 받았으며, 2024년도 BK21 FOUR 정보기술 미래인재 교육연구단의 지원을 받았으며, 반도체 공동연구소 지원을 받아 수행된 연구임.

참고문헌

[1] McNally, Richard, et al. "Fuzzing: The State of the Art." 2012.

[2] Chen, Yuanliang, et al. "{EnFuzz}: Ensemble fuzzing with seed synchronization among diverse fuzzers." 28th USENIX Security Symposium (USENIX Security 19). 미국. 2019.

[3] Böhme, Marcel, Van-Thuan Pham, and Abhik Roychoudhury. "Coverage-based greybox fuzzing as markov chain." Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 오스트리아. 2016.

[4] Lemieux, Caroline, and Koushik Sen. "Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage." Proceedings of the 33rd ACM/IEEE international conference on automated software engineering. 프랑스. 2018.

[5] Yun, Insu, et al. "{QSYM}: A practical concolic execution engine tailored for hybrid fuzzing." 27th USENIX Security Symposium (USENIX Security 18). 미국. 2018.

[6] Serebryany, Kosta. "Continuous fuzzing with libfuzzer and addresssanitizer." 2016 IEEE Cybersecurity Development (SecDev). 미국. 2016.

[7] A. Helin. "radamsa." https://gitlab.com/akihe/radamsa. 2019. (접속 일자: 2024.04.17).

[8] Güler, Emre, et al. "Cupid: Automatic fuzzer selection for collaborative fuzzing." Proceedings of the 36th Annual Computer Security Applications Conference. 온라인. 2020.

[9] Fu, Yu-Fu, Jaehyuk Lee, and Taesoo Kim. "autofz: automated fuzzer composition at runtime." 32nd USENIX Security Symposium (USENIX Security 23). 미국. 2023.