CPU 버그 탐지를 위한 Directed Differential Fuzzing 방법 연구

곽영준[1], 문현곤[2]
[1]울산과학기술원 컴퓨터공학과 석사과정
[2]울산과학기술원 컴퓨터공학과 교수
kyj05137@unist.ac.kr, hyungon@unist.ac.kr

A Study on Directed Differential Fuzzing Method to Detect CPU Bugs

Yeongjun Kwak[1], Hyungon Moon[2]
[1]Dept. of Computer Science Engineering, UNIST
[2]Dept. of Computer Science Engineering, UNIST

요 약

CPU의 결함은 보안적인 측면뿐만 아니라 경제적인 측면에서도 큰 손해를 발생시킨다. 최신 연구로 Differential Fuzzing을 이용하여 CPU RTL의 버그를 탐지하는 방법이 소개되었지만 전반적인 CPU 코어의 모듈을 탐색하기에 패치 테스트, 버그 재생산 및 새로운 모듈만을 검사하기에는 시스템 자원이 많이 낭비될 수 있다. 이에 본 연구에서는 Differential Fuzzer의 ISA 시뮬레이터에 Directed Greybox Fuzzer를 적용하여 그 결과를 분석하고 시스템 자원을 효율적으로 사용할 수 있는 Directed Differential Fuzzing 방법을 제안한다.

1. 서론

CPU의 결함은 하드웨어 및 그 위에서 실행되는 소프트웨어의 동작에 중대한 영향을 미친다. 하지만 하드웨어의 특성상 시장에 출품되고 난 이후에는 이러한 결함을 패치하기 힘들다. CPU 결함의 일례로 Pentium FDIV bug[1]를 들 수 있다. Pentium FDIV bug의 경우 결함이 있는 프로세서의 교체로 475M 달러의 비용이 발생하였다. 이렇듯 CPU의 결함은 보안적인 측면에서뿐만 아니라 경제적인 측면에서도 큰 손해를 발생시키기 때문에 선제적으로 발견하는 것이 중요하다. 하지만 CPU RTL의 복잡성 및 소프트웨어와 달리 크래시를 거의 발생시키지 않는 특성으로 인해 CPU RTL에서 결함을 탐지하는 것은 매우 어려운 과제이다. 최신 연구인 DifuzzRTL[2]은 커버리지 메트릭으로써 control register를 사용하며, 동일한 입력값에 대한 ISA 시뮬레이터와 RTL 에뮬레이터 결과의 일치 여부를 통해 버그를 탐지하는 Differential Fuzzing 기법을 사용하였다. DifuzzRTL은 여러 가지 CPU RTL에서 버그를 발견하는 등 그 효과를 입증하였다. 하지만 테스트 케이스의 변이가 간단하고 모든 모듈을 대상으로 퍼즈 테스팅을 수행하기 때문에 버그 발생 가능성이 적은 영역에 시스템 자원을 많이 사용하는 경우가 발생할 수 있다.

본 연구에서는 시스템 자원을 효율적으로 사용하여 버그 발생 가능성이 높은 영역 등 특정 코드 영역을 타겟으로 퍼즈 테스팅을 수행하는 fuzzer의 구현을 목표로 한다. 이를 위해 Directed Greybox Fuzzer인 AFLGo[3]를 DifuzzRTL에 통합하였다. RISC-V ISA 시뮬레이터인 Spike[4]에 AFLGo의 코드 계측을 수행하였고, 이를 바탕으로 Spike의 타겟 영역에 더 가까워지는 테스트 케이스에 시스템 자원을 집중하여 DifuzzRTL의 퍼즈 테스팅을 수행하였다.

2. 배경지식

Directed Greybox Fuzzer (DGF): AFL[5]과 같은 Coverage-guided Greybox Fuzzer (CGF)들의 등장으로 소프트웨어에서의 많은 버그들을 효율적으로 탐지할 수 있게 되었다. CGF는 가벼운 코드 계측을 통해 테스트 케이스의 실행 경로를 추적하고, 새로운 경로를 탐색한 테스트 케이스를 코퍼스에 추가 및 변이를 적용하는 등의 방법을 통해 프로그램의 버그를 찾는다. 하지만 CGF는 프로그램 전반에 대해 테스트 케이스를 생성하고 실행하기 때문에 버그 발생 가능성이 적은 영역에도 많은 시스템 자원

을 사용한다. 시스템 자원을 더 효율적으로 사용하여 패치 테스트, 버그 재생산 등을 하기 위해 특정 영역을 타겟으로 하는 Directed Greybox Fuzzer가 제안되었다. 타겟까지의 거리가 더 가까운 테스트 케이스에 더 많은 변이 기회를 제공하기 위한 파워 스케쥴링 방법[3], 프로그램 분석을 통해 타겟에 도달할 수 없는 테스트 케이스를 프로그램 실행 전에 식별하는 방법[6], 타겟에 가까운 시드만 코퍼스에 포함하는 방법[7]을 사용하는 등 DGF 연구가 활발히 진행되고 있다. 특히 본 연구에서 사용한 AFLGo는 컴파일 타임에 call graph와 intra-procedural control-flow graph를 생성하여 각 basicblock으로부터 타겟까지의 거리를 계산하고 계측을 수행한다[3]. 타겟까지의 거리를 컴파일 타임에 계산하여 런타임에 적은 오버헤드로 프로세스를 실행할 수 있다.

3. 디자인

DifuzzRTL은 control register를 커버리지 메트릭으로, 동일한 테스트 케이스에 대한 ISA 시뮬레이터와 RTL 에뮬레이터 결과의 일치 여부를 통해 버그를 탐지한다. 이 때 ISA 시뮬레이터는 C/C++로 구현된 소프트웨어이기 때문에 기존의 DGF를 적용할 수 있다. 우리는 DifuzzRTL에서 사용한 ISA 시뮬레이터인 Spike에 AFLGo를 적용하였다. 전체적인 디자인은 <그림 1>과 같다.

Instrumentation 단계에서는 각 basicblock에서 타겟까지의 거리를 계산하기 위해 Spike에 AFLGo의 코드 계측을 수행한다. Fuzzing 단계에서는 AFLGo와 DifuzzRTL, 두 프로세스를 동기적으로 실행한다. fuzz iteration은 다음과 같은 순서로 실행된다. 우선 fuzz iteration에 필요한 테스트 케이스를 생성한다. ISA 시뮬레이터와 RTL 에뮬레이터에는 동일한 테스트 케이스가 사용되어야 하고, RTL 에뮬레이터에 항상 올바른 문법의 테스트 케이스가 사용되어야 한다. 그러나 AFLGo의 변이 방법은 RTL 에뮬레이터에 적합하지 않은 테스트 케이스를 많이 생성하기 때문에 사용할 수 없다. 이를 해결하기 위해서 문법적으로 올바른 테스트 케이스를 사용하는 DifuzzRTL의 mutator를 사용하였다. 생성된 테스트 케이스는 파일로써 AFLGo에서 실행 중인 Spike에 전달된다. 생성된 테스트 케이스에서 타겟까지의 거리가 가까워졌는지 여부를 공유 메모리를 통해

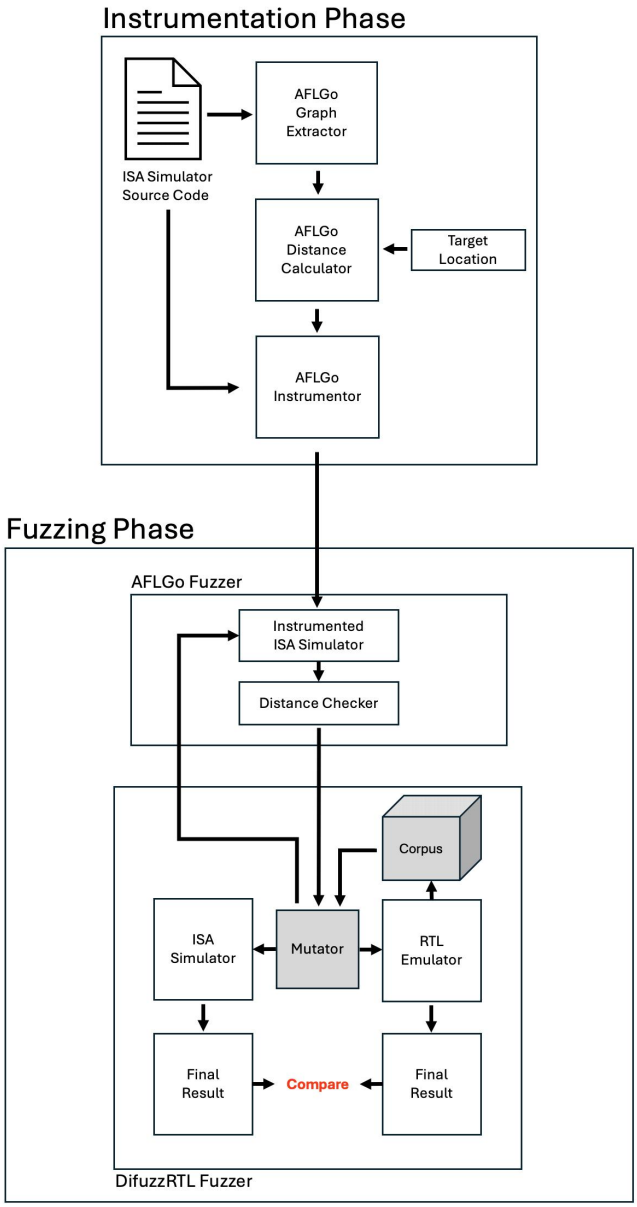

그림 1. Directed Differential Fuzzer의 전체 디자인

DifuzzRTL 프로세스로 전달한다. 만약 타겟까지의 거리가 가까워지지 않았다면 크로스 체크를 진행하지 않고 타겟까지의 거리가 가까워졌다면 크로스 체크를 진행하여 불일치 여부를 판단한다. 마지막으로 control register 커버리지가 증가했다면 코퍼스에 추가하고 다음 fuzz iteration을 시작한다.

4. 실험 결과 및 분석

Intel Xeon Gold 6136, 128GB DRAM을 가지고 있는 Ubuntu 20.04 서버에서 실험을 수행하였다. ISA 시뮬레이터로 RISC-V ISA 시뮬레이터인 Spike를 사용하였으며, DifuzzRTL이 Boom Core에서 발견한 버그(Issue #454, Issue #493,

CVE-2020-29561)를 재생산하기 위해 본 연구에서 구현한 Directed Differential Fuzzer를 적용하였다. 각각의 버그마다 12시간의 퍼즈 테스팅을 수행하였으며 그 결과는 <표 1>과 같다.

<표 1>을 통해 알 수 있듯이 현재의 Directed Differential Fuzzer를 통해서는 버그를 재현하기 어려움을 확인할 수 있다. 그 원인에 대해서 분석한 결과, 대부분의 fuzz iteration이 조기에 종료됨을 발견하였다. Directed Differential Fuzzer는 RTL 에뮬레이션 과정에서 사용되는 시스템 자원을 효율적으로 사용하기 위하여 테스트 케이스가 Spike의 타겟과 더 가까워 질 경우에만 RTL 에뮬레이션을 수행하도록 설계되었다. 많은 양의 fuzz iteration에서 타겟까지의 거리가 가까워지지 않은 테스트 케이스를 생성하였고, 그 결과 조기에 중단되었다. 타겟까지의 거리가 가까워지지 않은 테스트 케이스가 많이 생성된 원인은 Directed Differential Fuzzer의 코퍼스 추가 조건 및 DifuzzRTL의 mutator에 있다. DifuzzRTL의 mutator는 일련의 랜덤한 명령어들을 생성하거나, 코퍼스에서 테스트 케이스를 랜덤하게 선택하여 몇몇 명령어를 변경, 또는 코퍼스에서 랜덤하게 선택한 2개의 테스트 케이스를 병합하는 등 간단한 변이를 적용하게 된다. 코퍼스에는 타겟에 가까워지는 테스트 케이스만 추가한다. 이로 인해 코퍼스에는 매우 적은 수의 테스트 케이스만이 존재하여 다소 제한적인 테스트 케이스들이 생성된다. 그 결과 타겟에 가까워지는 테스트 케이스를 만들기 어려워지고 중단되는 fuzz iteration이 많이 발생한다.

중단되는 fuzz iteration을 줄이고 시스템 자원을 효율적으로 사용하기 위해서는 타겟에 더 가까워지는 테스트 케이스만 ISA 시뮬레이터와 RTL 에뮬레이터의 결과를 비교하기보다 1) 타겟에 더 가까운 테스트 케이스에게 더 많은 변이 기회를 제공하거나 2) 코퍼스에서 테스트 케이스를 선택할 때 타겟과 더 가까운 테스트 케이스가 선택될 확률을 높이는 방법, 3) mutator가 더 복잡한 테스트 케이스를 생성하도록 하는 등의 방법을 적용해 볼 수 있을 것이다.

5. 결론

본 연구에서는 CPU RTL의 패치 테스트 및 버그 재생산, 새로운 모듈 검사를 위해 시스템 자원을 효율적으로 사용하기 위한 방법을 고안하였으며 그 결과를 분석하였다. 이번 연구의 결과를 바탕으로 후속 연구에서는 추가적인 방법을 적용하여 효율적으로 CPU RTL의 특정 영역을 검사할 수 있는 Fuzzer를 구현할 수 있을 것으로 기대된다.

Target Bug ID	Reproduced	# of test case in corpus
Issue #454	X	31
Issue #493	X	22
CVE-2020-29561	X	87

표 1. 각 버그에 대한 12시간의 퍼즈 테스팅 결과

Acknowledgement

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터사업의 연구결과로 수행되었음(IITP-2024-2021-0-01817).

참고문헌

[1] Intel, 1994 - Annual Report, https://www.intel.com/content/www/us/en/history/history-1994-annual-report.html

[2] Hur, J., Song, S., Kwon, D., Baek, E., Kim, J., & Lee, B. (2021, May). Difuzzrtl: Differential fuzz testing to find cpu bugs. In*2021 IEEE Symposium on Security and Privacy (SP)*(pp. 1286-1303). IEEE.

[3] Böhme, M., Pham, V. T., Nguyen, M. D., & Roychoudhury, A. (2017, October). Directed greybox fuzzing. In*Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*(pp. 2329-2344).

[4] riscv-isa-sim, https://github.com/riscv-software-src/riscv-isa-sim

[5] American Fuzzy Lop, https://lcamtuf.coredump.cx/afl/

[6] Huang, H., Guo, Y., Shi, Q., Yao, P., Wu, R., & Zhang, C. (2022, May). Beacon: Directed grey-box fuzzing with provable path pruning. In*2022 IEEE Symposium on Security and Privacy (SP)*(pp. 36-50). IEEE.

[7] Canakci, S., Matyunin, N., Graffi, K., Joshi, A., & Egele, M. (2022, May). Targetfuzz: Using darts to guide directed greybox fuzzers. In*Proceedings of the 2022 ACM on Asia conference on computer and communications security*(pp. 561-573).