커버리지 기반 웹어셈블리 퍼저의 분류와 한계점

강하영 [1], 송수현 [2], 권동현 [3]
[1] 부산대학교 수학과 학부생
[2] 부산대학교 정보융합공학과 석사과정
[3] 부산대학교 정보컴퓨터공학부 교수(교신저자)

rkdgkdud12345@pusan.ac.kr, sshpnu@pusan.ac.kr, kwondh@pusan.ac.kr

The Classification and Limitation of Coverage-based WebAssembly Fuzzer

Ha-Young Kang[1], Su-Hyeon Song[2], Dong-Hyeon Kwon
[1]Dept. of mathematics, Pusan National University
[2]Dept. of Information Convergence Engineering, Pusan National University
[3]School of Computer Science and Engineering, Pusan National University

요 약

WebAssembly(Wasm)은 웹에서 네이티브에 가까운 속도로 실행 가능하고, 고성능 어플리케이션의 구현도 가능하기 때문에 브라우저 및 기타 플랫폼에서 활발히 사용되고 있다. 이로 인해 Wasm에 대한 보안성이 대두되고 있는데, 이때 취약점을 탐지하는 Fuzzing 기법을 적용한 연구들이 있다. Fuzzing 기법에 대한 분류 및 대표적인 도구를 소개하고 각 기법 간 차이점 및 한계점과 향후 연구 방향을 제시한다.

1. 서론

WebAssembly(Wasm)[1]은 빠르고 안전한 실행이 가능하고 수많은 소스 언어를 Wasm 으로 컴파일 할 수 있는 장점이 있는 바이트 코드 언어이다. 따라서 다양한 웹 어플리케이션과 브라우저에서 사용되고 있으며, 그 중요도가 계속해서 증가하고 있다. 그와 함께 Wasm 의 보안도 함께 중요해지고 있는데, 아직 많은 취약점을 가지고 있다. 이러한 취약점을 탐지해내는 방법으로써 퍼징이 존재하며 이는 대상 프로그램에 무작위 입력을 통해 비정상적인 동작이나 충돌을 탐지한다.

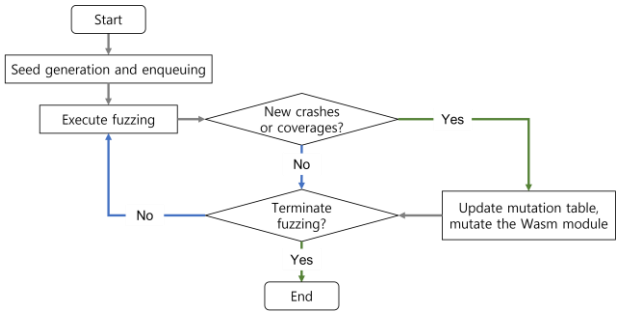
Wasm 에 퍼저 코드를 instrumentation 하는 방법에는 두가지로 나뉜다. 첫번째로는 “source-based instrumentation”으로, 이는 가장 널리 사용되는 방법으로 소스코드 내에 퍼저를 삽입하여 재컴파일하는 과정을 거쳐야 한다. 두번째로는 “binary-only instrumentation”으로, 소스코드에 접근할 수 없는 상황에서 binary 정보만을 가지고 fuzzer 에 유용한 정보를 얻어내는 방식이다. 본 논문에서는 각 퍼징 기법을 사용하는 대표적인 퍼저를 소개하고 각 기법의 한계점을 서술하고 향후 연구에 대한 방향성을 제시한다.

2. 본론

2.1 source-based fuzzing

Wasmfuzzer[2]는 Wasm virtual mahcine 을 위한 퍼징 도구이다. 이는 바이트 코드 수준에서 fuzzing 을 위한 seed 를 생성하고 mutation 전략을 통해 가장 적합한 mutation operator 를 선별한다. 이후 선택된 operator 를 기존 Wasm source code 에 적용한 후 새로운 wasm module 을 생성한다.

Wasmfuzzer 와 같은 source-based fuzzer 가 동작하는 방식은 다음과 같다.



(그림 1) Fuzzer 의 기본 동작

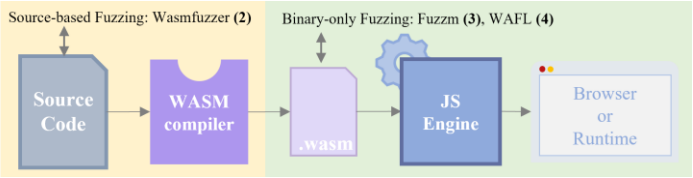
먼저, Wasm 모듈의 seed 를 생성하고 대기열을 만든다. 이후 가장 첫번째 Wasm 모듈부터 시행하는데 만약 새로운 커버리지나 충돌이 있으면 Wasmfuzzer

가 mutation 을 수행하여 새로운 Wasm 모듈을 만든다. 이 과정을 반복해서 수행하며, 만약 미리 정해둔 시간이 소진되면 중단한다.

이러한 source-based fuzzing 은 소스코드에 대한 높은 이해도를 기반으로 효과적인 input 을 생성하는 것에 유리하고 mutation 을 위한 다양한 정보(coverage, execution path, result, hash)를 추출하기에 유리하다. 다만 소스코드가 제공되지 않고 binary 만 이용가능한 환경에서는 fuzzing 을 시도할 수 없다는 한계를 가진다.

2.2 binary-only fuzzing

Wasm 어플리케이션의 binary-only fuzzing 은 Wasm 의 두가지 특징으로 인해 그 장점을 가진다. 첫번째, Wasm 어플리케이션은 다양한 소스 언어(C, C++, Rust, GO)로 개발되고, Wasm 컴파일러를 통해 타겟 머신 아키텍쳐에 적합한 Wasm 바이너리를 생성한다. 따라서 소스코드를 기반으로 하는 퍼징은 개발에 사용된 언어 따라 다른 퍼징 도구를 필요로 한다. 두번째, Wasm 어플리케이션은 컴파일 된 Wasm 바이너리 파일만 배포하고, 소스코드는 배포하지 않는 것이 일반적이다. 따라서 소스코드에 접근하지 못하는 경우가 존재하는데, binary-only fuzzing 은 이러한 상황에서도 효율적인 fuzzing 을 수행할 수 있다.



(그림 2) Source-based fuzzing 과 binary-based fuzzing 이 wasm 동작과정에서 영향을 미치는 단계를 나타낸 과정이다.

Fuzzm[3]은 웹어셈블리 바이너리전용으로 가장 처음 제안된 퍼저이다. 널리 사용되는 AFL 퍼저와 검증된 input generation 을 기반으로 한 그레이박스 퍼징 접근법을 사용한다. 시드 입력부터 시작하여 프로그램을 반복적으로 실행하여 커버리지 피드백을 수집하고, 이러한 피드백은 충돌을 트리거할 때까지 입력을 변형하는 데 사용한다. 또한 이는 coverage guided fuzzer 로서, 코드 커버리지 피드백을 통해 fuzzing 을 수행한다. 코드 커버리지 정보를 수집하기 위해 바이너리 instrumentation 을 통해 모든 분기마다 트리거 여부를 확인하고 커버리지 정보를 수집한다. 이를 통해 소스코드에 액세스 할 수 없는 상황에서도 fuzzing 에 필요한 정보를 획득한다.

Wafl[4]은 웹어셈블리 바이너리를 위한 퍼저로써 주로 C/C++같은 저급언어에서 버그를 찾아내는데 사용된다. 소스코드로 접근하지 않고, AFL++ 퍼저 호환 커버리지 정보를 얻기 위해 WAVM runtime 을 수정한다. 또한 여러 테스트 케이스에 대해 매번 새로운 프로세스를 수행하는 것 아닌 스냅샷을 통해 전반적인 fuzzing 성능을 개선한다. 또한 standalone Wasm runtime 인 WAVM 을 기반으로 디자인 되었기에, 브라우저와 standalone runtime 을 동시에 지원한다는 장점을 가진다.

Binary-only fuzzing 은 binary 기반 instrumentation 을 통해 source code 가 필요 없다는 강점을 가지나, source 의 부재로 인해 여러 단점을 가진다. Binary 만으로는 기존 코드의 동작에 대한 높은 이해도를 달성하기 어려우므로 적합한 input 을 생성하는 것이 까다로우며, 그러한 동작이해도 없이 오직 instrumentation 을 통해 mutation 에 사용될 유용한 정보를 획득하는 데에는 시간적, 기술적 어려움이 있다. 또한 결점이 발생하더라도 근본 원인을 제거하기 위해서는 여전히 Wasm 어플리케이션에 대한 소스코드가 필요하다.

3. 결론

WebAssemlby 의 취약점 탐지를 위해 다양한 fuzzer 들이 개발되고 있으며, Wasm 의 특징을 반영한 binary-only fuzzing 기법 또한 제안되어 왔다. 그러나 이 경우에도 다양한 fuzzing 기법의 적용에 사용할 유용한 정보를 추출하는 모니터링 코드 instrumentation 이 제한적이다. 따라서 퍼징 효율성을 나타내는 중요한 지표인 높은 코드 커버리지를 달성하기 어렵다는 단점이 여전히 존재한다. 이를 극복하기 위해 다양한 execution path 확보를 위한 추가 연구가 필요할 것으로 보인다.

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학 ICT 연구센터육성지원사업의 연구결과로 수행되었음 (IITP-2023-2020-0-01797)

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 융합보안핵심인재양성사업의 연구 결과로 수행되었음 (IITP-2023-2022-0-01201)

참고문헌

[1] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., ... & Bastien, J. F., “Bringing the web up to speed with WebAssembly.”, In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 185-200.,2017

[2] Jiang, B., Li, Z., Huang, Y., Zhang, Z., & Chan, W. K., “WasmFuzzer: A Fuzzer for WebAssembly Virtual Machines.”, In 34th International Conference on Software Engineering and Knowledge Engineering, SEKE 2022, KSI Research Inc.., pp. 537-542., 2022

[3] Lehmann, D., Torp, M. T., & Pradel, M. “Fuzzm: Finding memory bugs through binary-only instrumentation and fuzzing of webassembly.”, arXiv preprint arXiv:2110.15433. , 2021

[4] Haßler, K., & Maier, D. “Wafl: Binary-only webassembly fuzzing with fast snapshots.”, In Reversing and Offensive-oriented Trends Symposium ,pp. 23-30,2021