WebAssembly Binary 보안 기법 비교 분석 및 향후 연구

한고원[1], 신채원[2], 권동현[3]
[1]부산대학교 컴퓨터공학과 학부생
[2]부산대학교 정보컴퓨터공학부 학부생
[3] *부산대학교 정보컴퓨터공학부 교수(교신저자)
gowonisgood@pusan.ac.kr, sinchaewon@pusan.ac.kr, kwondh@pusan.ac.kr

Comparative Analysis of WebAssembly Binary Security Techniques

Han-Go Won[1], Chae-won Shin[2], Dong-hyun Kwon[3]*
[1]Dept. of Computer Science and Engineering, Pusan National University
[2]Dept. of Computer Science and Engineering, Pusan National University
[3]School of Computer Science and Engineering, Pusan National University
(corresponding author)

요 약

WebAssembly(WASM)은 다양한 환경에서 높은 성능을 제공하는 바이트코드 형식의 언어이다. 본 논문에서는 WASM 바이너리 보안을 강화하기 위한 동적 분석 기법인 Fuzzm, Wasabi, Taint Assembly를 분석하며, 각 기법의 장점과 한계점을 다룬다. 또한, 동적 분석과 정적 분석을 융합한 새로운 기법의 필요성을 기반으로 향후 진행되어야 할 연구 방향에 대해 살펴본다.

1. 서론

WebAssembly(WASM)는 C, C++, Rust와 같은 언어로 작성된 코드를 저수준의 바이트코드로 컴파일하여 웹 브라우저 등 다양한 실행 환경에서 높은 성능으로 실행할 수 있도록 설계된 언어이다. WASM은 웹 어플리케이션에서 기존에 사용되던 자바스크립트보다 빠른 실행 속도를 제공하여 성능이 중요한 웹 환경에서 두드러진 장점을 보인다. 또한, WASM은 샌드박스 환경과 선형 메모리 모델을 통한 자체적인 보안 메커니즘을 적용하고 있어, 보안성 측면에서도 강점을 가진다. 이러한 빠른 실행 속도와 보안상의 이점 덕분에 WASM의 활용 범위는 계속해서 확대되고 있다.[1]

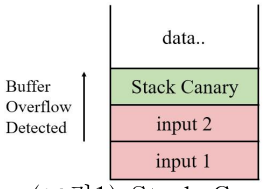
하지만 이러한 기본적인 보안 메커니즘에도 불구하고 WASM은 여러 보안 취약점에 노출될 수 있다. 예를 들어, WASM은 외부로부터 선형 메모리를 격리하지만, 내부 메모리 보호가 부족해 버퍼 오버플로우나 스택 오버플로우 같은 취약점에 여전히 노출된다. 이를 해결하기 위해 다양한 보안 기술들이 연구되고 있다.

본 논문에서는 WASM 바이너리에 적용된 보안 기술들을 비교·분석하고, 이를 바탕으로 WASM 바이너리 보안을 강화할 수 있는 연구 방향을 제시한다.

2. 관련 연구

2.1 Fuzzm

Fuzzm은 WASM 바이너리 전용 퍼징 도구로 소스코드 없이 동작한다. Fuzzm은 스택과 힙에 카나리 값을 삽입해 실행 중 값이 변경되었는지 확인하고 이를 통해 메모리 손상을 감지한다. 또한, AFL의 입력 생성 알고리즘을 통해 생성한 다양한 입력값으로 프로그램을 수 회 실행하며, 실행 도중 오류가 감지되면 프로그램을 중단하고 이를 사용자에게 알린다. Fuzzm은 평균적으로 1,232개의 고유 경로를 탐색하고 40개의 충돌을 감지하며, 기존 WASM 대비 약 1.1배의 낮은 오버헤드를 보인다. 따라서 Fuzzm은 메모리 취약점을 탐지하고 보호하는 데 탁월한 도구이다. 그러나, 논리적 결함이나 타입 혼동과 같은 다른 형태의 취약점에서는 카나리 값으로는 한계가 있어 탐색 범위의 제한이 발생할 수 있다.[2]



(그림1) Stack Canary

(그림2) Heap Canary

2.2 Wasabi

Wasabi는 WASM을 동적으로 분석하기 위한 프레임워크이다. Wasabi는 WASM 바이너리 명령어 사이에 분석 함수를 호출하는 코드를 삽입하여 프로그램의 동작을 추적한다. 분석 함수는 명령어 실행 횟수를 기록하거나, 메모리 접근을 추적하는 등의 특정 분석 작업을 수행한다. 이를 통해 WASM 프로그램의 실행 흐름, 메모리 접근, 함수 호출 등 프로그램의 동작을 추적하는 데 두드러진 장점을 보인다. 또한, 선택적 계측을 통해 필요한 부분만 분석하여 성능 오버헤드를 최소화하였다. 하지만, 분석 코드 삽입으로 인해 바이너리 크기가 크게 증가하며, 삽입된 함수가 자바스크립트로 작성되어 고성능 분석 작업에는 부적합하다는 단점이 있다. [3]

2.3 Taint Assembly

Taint Assembly는 WASM에서 동적 taint 추적을 제공하는 도구로, 데이터에 태그를 부여해 흐름을 추적할 수 있도록 한다. Taint Assembly는 WASM 런타임인 V8 엔진을 수정하여 기능을 구현하였다. Taint Assembly는 WASM 값에 taint 레이블(태그)을 추가하고, taint의 메타데이터를 shadow memory에 저장한다. 이를 통해 데이터 흐름을 분석하여 보안성을 효과적으로 강화할 수 있다. 또한, 오버헤드를 줄이기 위해 프로파게이션 확률에 따라 taint 값을 확률적으로 유지하였다. 이를 통해 기존 WASM과 비교하여 약 5~12%까지 오버헤드를 줄였다. 그러나 Taint Assembly의 확률적 추적으로 인해 비트 연산과 난수 생성 등의 추가적인 연산에서는 성능 저하가 발생할 수 있다는 단점이 있다. [4]

3. 결론 및 향후 연구

본 논문에서는 WASM 바이너리의 보안 강화를 위해 사용되는 다양한 동적 분석 기법을 살펴보았다. Fuzzm, Wasabi, Taint Assembly는 각각 메모리 안전성, 프로그램 동작 추적, 데이터 흐름 분석 등에서 독자적인 장점을 제공하며, WASM 바이너리의 잠재적 취약점을 효과적으로 탐지할 수 있다. 이러한 도구들은 동적 환경에서 발생하는 보안 문제점을 실시간으로 분석하고 대응할 수 있다는 강력한 이점을 지닌다. 특히, 실행 경로 탐색이나 메모리 오류 감지 측면에서는 매우 유용하게 작동하며, WASM의 성능과 보안성 간의 균형을 유지할 수 있다.

그러나 이러한 분석 기법들에도 몇 가지 한계가 존재한다. 동적 분석은 실제 프로그램 실행 시에만 작동하기 때문에, 모든 가능한 경로를 고려하기 어렵고, 실행되지 않는 경로에서의 잠재적 취약점을 발견하지 못한다는 단점이 있다. 또한 런타임 상의 오버헤드가 발생할 수 있어, 성능 저하를 완전히 배제할 수 없다. WASM 바이너리를 보호하기 위한 정적 분석 도구도 존재하지만, 정적 분석은 프로그램 실행 전 모든 경로를 분석하는 장점이 있는 반면에 실행 환경에서 발생하는 동적인 문제를 파악하는 데는 한계가 있다.

따라서, 동적 분석과 정적 분석의 장점을 결합한 새로운 분석 기법이 필요하다. 정적 분석은 실행되지 않는 경로의 취약점을 발견해 동적 분석의 한계를 보완할 수 있으며, 두 기법을 융합하면 WASM 바이너리의 취약점을 더 포괄적으로 분석하면서 성능 저하를 최소화하고 보안성을 강화할 수 있다.

Acknowledgement

ITRC (제로트러스트클라우드)
"본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터사업의 연 구결과로 수행되었음" (IITP-2024-RS-2023-00259967)

참고문헌

[1] https://webassembly.org/

[2] Lehmann, Daniel, Martin Toldam Torp, and Michael Pradel. "Fuzzm: Finding memory bugs through binary-only instrumentation and fuzzing of webassembly." arXiv preprint arXiv:2110.15433 (2021).

[3] Lehmann, Daniel, and Michael Pradel. "Wasabi: A framework for dynamically analyzing webassembly." Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019.

[4] Fu, William, Raymond Lin, and Daniel Inge. "Taintassembly: Taint-based information flow control tracking for webassembly." arXiv preprint arXiv:1802.01050 (2018).