Observability of innovative sandboxed container technology: Detection and monitoring of WebAssembly Workloads

Supervisors

Dhouha AYED, PhD, security architect at Thales

Quentin MICHAUD, PhD CIFRE student at Thales and Télécom SudParis[1]

Contact:
- `dhouha.ayed@thalesgroup.com`
- `quentin.michaud@thalesgroup.com`

Description

WebAssembly [1], [2], or Wasm for short, has been created as a fast and secure-by-design answer to the always increasing need for complex computation in browsers, such as 3D rendering, 3D model parsing, gaming, hardware emulation or physics workloads (e.g. computational fluid dynamics) [3]. The success of WebAssembly as a portable Instruction Set Architecture (ISA) and binary format has prompted its adoption in many applications besides browsers. Today, we can find WebAssembly in smart contracts, embedded devices [4], in secure plugins [5], [6], in Function as a Service (FaaS) platforms [7], or as a standalone runtime [8]. The latter has a huge impact on the cloud world and the computing world in general. Some see in the flexibility of WebAssembly a universal binary format that could be distributed seamlessly across operating systems and hardware architectures. It also appears in various cloud-related projects and is considered as an alternative to Linux-based containers [9], promising to be more portable, lightweight and secure.

While WebAssembly provides sandboxing and performance benefits, its standalone use cases in cloud, edge computing, and embedded systems raise new challenges concerning security and observability. The cloud-native community has a strong experience of observing the behavior of containers. With several methods such as eBPF[2] [10], [11], it is possible to collect performance metrics, uncover bugs, but also to detect both successful and unsuccessful attacks. While WebAssembly is reputed more secure than containers, it can still be impacted by attacks [12], [13].

Goals

The main goal of this project is to investigate techniques for detecting, analyzing, and monitoring WebAssembly workloads in real-time. The candidate will study the security landscape of standalone (i.e. using WASI [14]) WebAssembly workloads, analyze vulnerability types and potential attacks, and explore tools and methods for collecting relevant metrics and identifying workload characteristics and anomalies. The candidate will analyze and evaluate various approaches to monitor wasm workloads:

- Adding monitoring hooks to a programmable Wasm runtime (example: Wasmtime)
- Add instrumentation at the Wasm binary level to generate precise events or counters if needed
- Implement monitoring techniques leveraging Linux eBPF technology to provide complementary system-level observability without modifying the Wasm runtime or binaries

A proposition of steps for conducting this project would be:

[1]https://www.theses.fr/s374883
[2]https://ebpf.io/

1. Conduct a comprehensive analysis of known vulnerabilities and attack vectors targeting WebAssembly modules and runtimes, especially in standalone contexts. Investigate sandbox escape possibilities, malicious bytecode patterns, side-channel attacks, runtime exploits, and risks related to import/export interfaces.
2. Review existing techniques to monitor WebAssembly workloads, including runtime hooks, binary instrumentation tools (e.g., Binaryen[3], WABT[4]), and observability frameworks (e.g., OpenTelemetry[5], Prometheus[6]) and identify gaps in current approaches where system-level monitoring can add value.
3. Explore the use of extended Berkeley Packet Filter (eBPF) technology to trace system calls, resource utilization (CPU, memory, I/O), and network interactions of Wasm runtimes. Develop or adapt eBPF scripts to extract operational metrics and anomalous behavior indicators without modifying Wasm code or runtime internals.
4. Evaluate how to integrate the collected metrics into observability pipelines (e.g., Prometheus exporters, Grafana dashboards) for real-time monitoring.
5. Test the monitoring solution on typical standalone Wasm workloads and demonstrate detection capabilities for security-relevant events.
6. Produce a comprehensive report summarizing the security assessment and monitoring strategy, providing recommendations for implementing ongoing security monitoring and potential runtime hardening measures.

Bibliography

[1] A. Haas *et al.*, "Bringing the web up to speed with WebAssembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, in PLDI 2017. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 185–200. doi: 10.1145/3062341.3062363.

[2] A. Rossberg, "WebAssembly Core Specification," Dec. 2019. [Online]. Available: https://www.w3.org/TR/wasm-core-1/

[3] M. Sakuta, "Computational Fluid Dynamics simulation with Webassembly and Rust." Accessed: Apr. 17, 2024. [Online]. Available: https://github.com/msakuta/cfd-wasm

[4] R. Gurdeep Singh and C. Scholliers, "WARDuino: a dynamic WebAssembly virtual machine for programming microcontrollers," in *Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*, Athens Greece: ACM, Oct. 2019, pp. 27–36. doi: 10.1145/3357390.3361029.

[5] "Extism - make all software programmable. Extend from within. | Extism - make all software programmable. Extend from within.." Accessed: Apr. 17, 2024. [Online]. Available: https://extism.org/

[6] S. Narayan, T. Garfinkel, S. Lerner, H. Shacham, and D. Stefan, "Gobi: WebAssembly as a Practical Path to Library Sandboxing." Accessed: Apr. 17, 2024. [Online]. Available: http://arxiv.org/abs/1912.02285

[3] https://github.com/WebAssembly/binaryen
[4] https://github.com/WebAssembly/wabt
[5] https://opentelemetry.io/
[6] https://prometheus.io/

[7] V. Kjorveziroski and S. Filiposka, “WebAssembly Orchestration in the Context of Serverless Computing,” *Journal of Network and Systems Management*, vol. 31, no. 3, p. 62, Jul. 2023, doi: 10.1007/s10922-023-09753-0.

[8] L. Clark, “Standardizing WASI: A system interface to run WebAssembly outside the web – Mozilla Hacks - the Web developer blog.” Accessed: Apr. 17, 2024. [Online]. Available: https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface

[9] “wasmCloud.” Accessed: Apr. 17, 2024. [Online]. Available: https://wasmcloud.com/

[10] D. Soldani *et al.*, “eBPF: A New Approach to Cloud-Native Observability, Networking and Security for Current (5G) and Future Mobile Networks (6G and Beyond),” *IEEE Access*, vol. 11, no. , pp. 57174–57202, 2023, doi: 10.1109/ACCESS.2023.3281480.

[11] C. Cassagnes, L. Trestioreanu, C. Joly, and R. State, “The rise of eBPF for non-intrusive performance monitoring,” in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–7. doi: 10.1109/NOMS47738.2020.9110434.

[12] D. Lehmann, J. Kinder, and M. Pradel, “Everything Old is New Again: Binary Security of WebAssembly,” 2020, pp. 217–234. Accessed: Sep. 14, 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/lehmann

[13] Q. Stiévenart, C. De Roover, and M. Ghafari, “Security risks of porting C programs to webassembly,” in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, in SAC '22. New York, NY, USA: Association for Computing Machinery, May 2022, pp. 1713–1722. doi: 10.1145/3477314.3507308.

[14] “Introduction | WASI.dev.” Accessed: Sep. 24, 2024. [Online]. Available: https://wasi.dev/