Wasmi security assurance

Security audit of the WebAssembly interpreter used for substrate smart contracts

v1.3 – Dec 20, 2023

Louis Merlin	louis@srlabs.de
Regina Bíró	regina@srlabs.de

Abstract. This study describes the results of a thorough, independent security assurance review of Wasmi, a WebAssembly interpreter used by the contract pallet built on Substrate.

No vulnerabilities were identified related to Wasmi during the audit timeframe. As a result, the auditors concluded that the Wasmi project was implemented according to security best practices. Further recommended maturity steps include continuously fuzzing the Wasmi codebase.

1 Overview

1.1 Motivation & Scope

Wasmi was written by Parity Technologies in order to create an efficient WebAssembly interpreter with low overhead, that can be used in embedded environments. Currently, Wasmi is used as the execution engine for smart contracts written in ink! and deployed by the contracts pallet from Substrate. Wasmi is run within the Substrate runtime that is a WebAssembly environment itself.

This audit assesses the existing protection of Wasmi's core logic implementation against a variety of **likely hacking scenarios** and **identifies the most relevant weaknesses**, all with the goal of improving the protection capabilities of the functionality and the whole blockchain system that relies on the wasmi execution engine.

The Wasmi logic is implemented in Rust, the code that was considered the scope of the audit can be found in its own repository on Github [1]. This review was performed via 2 main workstreams on version v0.31.0 [2]:

1. **Manual code review**: The Wasmi logic was assessed by the auditors via performing manual code review. Parts of the logic flow was already assessed during the contracts pallet and initial ink! audit performed in early 2023.

2. **Dynamic fuzz-testing**: The Wasm interpreter was run extensively against another reference project (Wasmtime, also used by Parity as part of Substrate). The audit team created two new differential fuzzing harnesses for this review. The first one was tasked with verifying the correctness of execution for a given Wasm blob. The second one examined the gas usage of the same code execution for discrepancies.

Both of these workstreams were carried out taking into account the restricticted utilization of Wasmi features by the contracts pallet.

2 Findings

No issues were uncovered during this dedicated audit workstream for the Wasmi Rust implementation.

3 Evolution suggestions

While the manual and tool-based assessment on the current version of the wasmi codebase did not uncover any weaknesses, it is further recommended to periodically perform audits on major changes to the core logic. Additionally, SRLabs advises ramping up continuous fuzzing of the codebase, complementing the fuzzing already implemented as part of the CI/CD pipeline.

4 References

[1] [Online]. Available: https://github.com/paritytech/wasmi.

[2] [Online]. Available: https://github.com/paritytech/wasmi/releases/tag/v0.31.0.