Stylus Emergency Fixes Review

Security Assessment (Summary Report)

October 23, 2024

Prepared for:
Offchain Labs

Prepared by: **Gustavo Grieco and Jaime Iglesias**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

© 2024 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to Offchain Labs under the terms of the project statement of work and has been made public at Offchain Labs' request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Jaime Iglesias, Consultant
jaime.iglesias@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 6, 2024	Pre-project kickoff call
September 16, 2024	Delivery of report draft
September 16, 2024	Report readout meeting
September 20, 2024	Delivery of report draft
October 23, 2024	Delivery of final summary report

Project Targets

The engagement involved a review and testing of the following targets.

Stylus Emergency Fixes

Repository	`nitro-private` (fixes)
Version	`0cc80646cff240d3f86cb8d6f6e8b9c83be9266b`
Type	Go, Rust
Platform	Arbitrum

Governance ArbOS 32 Actions

Repository	`governance-private` (PR #1)
Version	`2d42f247d0cc2ab4b8eb47f49dd2b322be9f9d1f`
Type	Solidity
Platform	Arbitrum

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of fixes for a number of high-severity issues found after Stylus was deployed.

A team of two consultants conducted the review from September 11 to September 19, 2024, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed a manual review of the affected code and the fixes.

Observations and Impact

During the engagement, we verified that the proposed fixes completely mitigate the high-severity issues found after Stylus was deployed, including an activation edge case that was found during this engagement (TOB-STYLUS-FIX-1); note that this issue was also found by Offchain Labs during a refactor of the affected code.

Additionally, we reviewed a set of governance actions to upgrade ArbOS to version 32 (`L1ModuleRootArbOneAction`, `L1ModuleRootNovaAction`, and `L2ArbOS32Action`) and two emergency actions that will be executed if Stylus needs to be disabled (`SetInkPriceOneAction`, `SetWasmMaxStackDepthZeroAction`).

The governance actions will upgrade ArbOS to version 32 and set the new module root hash on both Nova and ArbOne; additionally, they will revert any changes made by either of the emergency actions, effectively re-enabling Stylus if either of the emergency actions was executed; otherwise (i.e., if the emergency actions were not executed), they will simply upgrade the ArbOS version and the module root.

All the aforementioned action contracts are correct; additionally, we verified that the correct module root hash, `0x184884e1eb9fefdc158f6c8ac912bb183bf3cf83f0090317e0bc4ac5860baa39`, is being employed in all the actions and confirmed that it is the root hash generated by the reviewed Nitro changes.

Recommendations

Perform an ArbOS upgrade with the reviewed actions, either as an emergency or nonemergency measure, depending on whether the issues are actively exploitable and whether it is likely that the issues will be identified by third parties.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Importing forward__set_trap produces a failure in WASM initialization	Data Validation	High

Detailed Findings

1. Importing forward__set_trap produces a failure in WASM instantiation	
Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STYLUS-FIX-1
Target: `forward_stub.wat`	

Description

Users can deploy a specially crafted Stylus program that will cause ArbOS to panic after the activation checks are passed.

Users can access a variety of imports to call in their Stylus programs. These are included in the `forward_stub.wat` file:

```
    ;; allows user_host to request a trap
    (global $trap (mut i32) (i32.const 0))
    (func $check unreachable)
    (func (export "forward__set_trap") unreachable)
    ;; user linkage
    (func (export "vm_hooks__read_args") (param i32) unreachable)
    (func (export "vm_hooks__write_result") (param i32 i32) unreachable)
    …
```

Figure 1.1: Part of the `forward_stub.wat` file

The imports available to users from the `vm_hooks` module are properly documented and seem to work correctly, but there is an additional import available called `forward__set_trap`. Using this import will produce a valid Stylus program that will correctly pass the activation checks.

However, the imports defined for WASM instantiation are only the ones from the `vm_hooks` module.

```
 define_imports!(
     "vm_hooks" => host {
         read_args, write_result, exit_early,
         storage_load_bytes32, storage_cache_bytes32, storage_flush_cache,
transient_load_bytes32, transient_store_bytes32,
         call_contract, delegate_call_contract, static_call_contract, create1,
create2, read_return_data, return_data_size,
         emit_log,
         account_balance, account_code, account_codehash, account_code_size,
         evm_gas_left, evm_ink_left,
         block_basefee, chainid, block_coinbase, block_gas_limit, block_number,
block_timestamp,
         contract_address,
         math_div, math_mod, math_pow, math_add_mod, math_mul_mod,
         msg_reentrant, msg_sender, msg_value,
         tx_gas_price, tx_ink_price, tx_origin,
         pay_for_memory_grow,
         native_keccak256,
     },
 );
 ...
 let instance = Instance::new(&mut store, &module, &imports)?;
```

Figure 1.2: WASM instantiation in ArbOS using the Wasmer API

Trying to activate a Stylus program that imports `forward__set_trap` will produce a panic in the ArbOS code:

```
thread '<unnamed>' panicked at stylus/src/util.rs:19:5:
encountered fatal wasm: init failed

Caused by:
    Error while importing "forward"."set_trap": unknown import. Expected
Function(FunctionType { params: [], results: [] })

Location:
    stylus/src/native.rs:207:24
```

Figure 1.3: Part of the panic error when trying to activate a Stylus program that imports `forward__set_trap`

Note that this issue was independently discovered by the Offchain Labs team as well as by Trail of Bits, during this review.

Exploit Scenario

Eve deploys a specially crafted Stylus program that imports `forward__set_trap`, meaning that it will halt the chain once it is activated.

Recommendations

Short term, remove the `forward__set_trap` export from `forward_stub.wat`.

Long term, consider creating a complete specification of the allowed WASM properties enforced during WASM activation and include additional test cases.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

General

- Consider using the `must_use` Rust hint in functions such as `flush_storage_cache` to mitigate the possibility that the caller will ignore the return value.

arbitrator/wasm-libraries/user-host-trait/src/lib.rs

- Consider expanding the documentation of each function in the file to include the minimum and maximum value of ink and gas that needs to be charged for every call. Additionally, clearly note whether an internal function handles the computational cost of each operation. This will help to prevent computational costs from being undercharged or overcharged.

- Related to the previous recommendation, add unit tests to cover average and corner cases in the ink/gas usage for each function.

C. WASM Security Measures for Stylus

This appendix lists the current security measures and constraints checked during WASM activation. These checks guarantee that WASM activation produces Stylus programs that are secure and execute correctly.

Imports

WASM imports have the following constraints:

- In general, only the function imports available in the `forward_stub.wat` file are available to use.
- Only function imports are valid. WASM programs importing other entities such as tables or memories will immediately fail validation.
- The use of imports with valid names but incorrect signatures will immediately fail validation.
- Any use of imports with the reserved prefix `stylus_` will immediately fail validation.

However, we must note the following:

- Imports can be duplicated as long as they are consistent and the number of duplicates is not directly limited.
- Imports can have special characters such as the null character (`\00`).

Exports

In general, users are free to export any function they need. However, there are number of constraints:

- The `user_entrypoint` export must be available, and it should have a specific signature.
- Export names cannot shadow any import names, either the raw name (e.g., `pay_for_memory`) or the full canonical name (e.g., `module__pay_for_memory`).
- Any use of exports with the reserved prefix `stylus_` will immediately fail validation.
- Some special global exports will be added to support ink metering (e.g., `stylus_ink_left`). These exports are correctly added at the end of the global index space and cannot be accessed by users directly.
- Exports can have special characters such as the null character (`\00`).

Additional Internal Functions

A number of internal functions are added into WASM programs during activation. These functions should never be used directly by the user. Fortunately, calling them is not possible since user WASM programs are validated to make sure the function indices are correct, so adding more internal functions is safe as long as they are added at the end of the function index space.