OpenZeppelin | security

Arbitrum Stylus SDK Pull Request #370 Audit

December 19, 2025

Table of Contents

Table of Contents 2

Summary 3

Scope 4

usertrace 5

replay 5

debug_hook 5

High Severity 6

H-01 External Contract Replay Is Never Invoked 6

Low Severity 6

L-01 Potential Execution Failure on Windows 6

Notes & Additional Information 7

N-01 Inconsistent Error Handling 7

N-02 Unaddressed TODO Comments 7

N-03 Hardcoded Extension in Error Messages 8

N-04 Redundant Contract Library Maps 8

N-05 Unused stable_rust Flag 8

N-06 Misleading Errors 8

Conclusion 10

Appendix 11

Issue Classification 11

Summary

Type	SDK
Timeline	From 2025-12-11 To 2025-12-15
Languages	Rust

Total Issues	8 (7 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (0 resolved)
Notes & Additional Information	6 (6 resolved)

Scope

OpenZeppelin audited the OffchainLabs/stylus-sdk-rs repository at commit 8e2202c.

The scope covered all the changes made in pull request #370 that had been merged into the main branch.

The following files were in scope:

```
├── cargo-stylus/src
│   ├── commands
│   │   ├── debug_hook.rs
│   │   ├── mod.rs
│   │   └── replay.rs
│   └── utils/hostio.rs
└── stylus-tools/src/core/tracing/mod.rs
```

System Overview

Pull request #370 adds the `usertrace` command and enhances the `replay` command with debugger options.

`usertrace`

The `usertrace` command is a debugging tool that allows developers to visualize the function call hierarchy within their Stylus contracts for a given transaction using the `stylusdb` debugger tool.

`replay`

The `replay` command and the `hostio` module introduce logic for debugging interactions between multiple contracts. Within `replay.rs`, when contract addresses are provided as input to the CLI command, `ContractRegistry` uses helper functions from `hostio.rs` to get the execution frame, after which it builds and loads the shared libraries. When an external call is detected during the replay, `ContractRegistry` loads and executes the code for the target contract, allowing for a continuous debugging session across all contracts involved in the transaction.

`debug_hook`

`debug_hook` defines a mechanism for the `cargo-stylus` tool to communicate with an external debugger, such as `stylusdb`. The `DebuggerHook` trait is an interface that defines a set of events that can occur during a transaction replay. `StylusDebuggerHook` is the implementation of the `DebuggerHook` trait. It works by creating a Unix socket and listening for a connection from the debugger. When an event occurs during the transaction replay (e.g., a call to another contract), `StylusDebuggerHook` sends a command over the socket to the debugger. This allows the debugger to stay in sync with the execution flow.

High Severity

H-01 External Contract Replay Is Never Invoked

The `relay` command registers a `ContractRegistry` instance via `set_external_contract_access` and implements `ExternalContractAccess` to delegate calls to other contracts during debugging.

However, `hostio` only defines `EXTERNAL_CONTRACT_ACCESS`. Functions such as `call_contract`, `delegate_call_contract`, and `static_call_contract` only read the input frame from the static trace via the `frame!` macro and do not check for or dispatch to the registered `ExternalContractAccess` object. As a result, the `ContractRegistry` registry is populated, but external calls are never dispatched to the loaded libraries. Hence, calls to external contracts in `ContractRegistry` cannot be executed, making the multi-contract debugging feature non-functional.

Within the functions in `hostio`, consider checking the `is_in_external_contract` flag and reading data from `EXTERNAL_CONTRACT_ACCESS` to properly debug external contract calls.

Update: *Resolved at commit 68f8ceb.*

Low Severity

L-01 Potential Execution Failure on Windows

In the `stylus_tools::verification` module, the `verify_os` function verifies that the OS on which a command is being executed is supported. However, this function is left unused throughout the codebase, leading to potential failures on Windows without the Windows Subsystem for Linux. For instance, when initiating a new `StylusDebuggerHook` struct, the `socket_path`'s format uses the Unix path syntax with a Unix-specific `tmp` folder. This could lead to issues when creating a connection to the socket.

Consider either implementing support for Windows throughout the codebase or implementing the `verify_os` function to provide better error messaging for Windows users.

Update: *Acknowledged, not resolved. The Walnut team stated:*

> Since Windows is staying out of focus for now for these features, we have addressed this issue by adding a TODO marker in `debug_hook.rs` on lines 49 to 52.

Notes & Additional Information

N-01 Inconsistent Error Handling

For most subcommands, the `exec` function handles results uniformly by returning a `CargoStylusResult`. However, for the `replay` and `usertrace` commands, the `exec` function returns a `eyre::Result<()>`. Within `mod.rs`, `.map_err(Into::into)` is used to convert the `eyre::Error` into a `CargoStylusError`.

To improve the overall quality of the codebase, consider having consistent returns and error handling in different command executions.

Update: *Resolved at commit 68f8ceb.*

N-02 Unaddressed TODO Comments

The TODO comment in line 73 of `debug_hook.rs` mentions functionality that has not yet been implemented. Another unaddressed TODO comment was identified in line 531 of `replay.rs`.

Consider addressing TODO comments that point to important unimplemented functionality. For other TODO comments, consider clearly linking them to their respective issue in the issues backlog.

Update: *Resolved at commit 68f8ceb.*

N-03 Hardcoded Extension in Error Messages

In the `cargo_stylus::commands::replay` module, the `.so` file extension is hardcoded in the error messages in lines 330, 336, 617, and 623.

Consider replacing the `.so` file extension with the value of the `extension` variable to clarify the error messages based on the operating system.

Update: *Resolved at commit 68f8ceb.*

N-04 Redundant Contract Library Maps

`ContractRegistry` maintains `libraries` and `library_paths` maps in parallel, which increases coordination risk and can cause them to diverge over time.

Consider merging these fields and updating call sites to read from the unified entry.

Update: *Resolved at commit 68f8ceb.*

N-05 Unused `stable_rust` Flag

The `Args` struct in `replay.rs` and `usertrace.rs` defines a `stable_rust` [1 2] flag. However, this flag is never used in the respective `exec` functions.

Consider either using or removing any unused flags to improve the clarity and maintainability of the codebase.

Update: *Resolved at commit 68f8ceb.*

N-06 Misleading Errors

Throughout the codebase, multiple instances of misleading errors were identified:

- The replay flow builds a `ContractRegistry` instance and then requires the top-level transaction's `to` address to have a corresponding source/library in the registry. If the source is not found, the flow bails in line 544 with the message “Use --contracts flag”. When `--contracts` is already provided but does not include the main address, this message is inaccurate and confusing. Consider improving the error by detecting whether `--contracts` was supplied, including the missing address and the set of provided addresses, and then suggesting a fix.

- The error message in line 486 of the `replay.rs` wrongly suggests that the error was thrown in the `trace` command.

Consider updating any misleading or incorrect error messages.

Update: *Resolved at commit 68f8ceb.*

Conclusion

Pull request 370 adds the `usertrace` command and extends the `replay` command with multi-contract debugging, aiming to give developers clearer end-to-end visibility into Stylus transaction execution. Overall, the new tooling moves the system toward richer cross-contract observability and debugger integration.

One high-severity issue was identified, which leaves the newly added multi-contract debugging non-functional. The codebase otherwise shows solid integration with only a few remaining issues that need to be addressed.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.