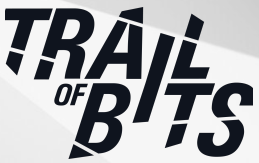

Offchain BoLD Fixes

Summary Report

December 26, 2024

Prepared for:
Harry Kalodner, Steven Goldfeder, and Ed Felten
Offchain Labs

Prepared by: **Gustavo Grieco, Josselin Feist, and Simone Monica**

Notices and Remarks

Copyright and Distribution

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@trailofbits.com

The following consultants were associated with this project:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Josselin Feist, Consultant
josselin@trailofbits.com

Simone Monica, Consultant
simone.monica@trailofbits.com

Project Timeline

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

Date	Event
December 20, 2024	Delivery of report draft
December 20, 2024	Report readout meeting
December 26, 2024	Delivery of final summary report

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of several pull requests (PRs) on the Nitro contracts repository.

A team of three consultants conducted the review from December 12 to December 16, 2024, for a total of four engineer-days of effort. Our testing efforts focused on the changes made by the PRs. With full access to source code and documentation, we performed a manual review of the PRs. Additionally, on December 19, 2024, a team of two consultants reviewed the BoLD upgrade actions payload and deployed contracts.

Observations and Impact

We identified no significant issues during the PR review. Most of the PRs have a small scope, and their changes are properly identified. We found one issue that affects the deployed contracts for executing the BoLD upgrade actions, highlighting that a more accurate approach to deploying and verifying the contract should be taken.

Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that Offchain Labs take the following step:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	0
Low	1
Informational	2
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	2
Undefined Behavior	1

Project Targets

The engagement involved a review and testing of several PRs.

Nitro contracts

Repository	https://github.com/OffchainLabs/nitro-contracts
Version	Upgrade instructions, 325, 266, 214, 262, 263, 250, 230, 270, 265, 275
Type	Solidity
Platform	Ethereum, Arbitrum

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Upgrade instructions and PR 325**. We assessed whether the actions are executed in the correct order and whether the new configuration values are as expected, that the BoLD proposal payload data matches the script's output, and that the deployed action contracts are well configured.
- **PR 266**. We checked the effects on the validators to, for example, determine the state after the proposal execution (e.g., if users can withdraw). We also checked how the old rollup is used and its interactions with the upgraded and new contracts.
- **PR 214**: We looked at the overall contracts for flaws that would allow an attacker to steal funds, withdraw more than deposited, bypass the access controls, or compromise the bidding process.
- **PR 262**: We reviewed the refactoring and looked for unexpected semantic changes.
- **PR 263**: We reviewed the improvements and the additional checks for correctness.
- **PR 250**: We reviewed the need for zero transfer checks in the other contracts.
- **PR 230**: We reviewed the gas optimization to assess correct variable usage.
- **PR 270**: We reviewed the refactoring. While we briefly examined the impact of removing the checks, we have not assessed the impact on the broader codebase.
- **PR 265**: We reviewed the salt addition to determine if it prevents the risk of front-running.
- **PR 275**: We assessed whether the delay proof is disabled and not required, and whether it can still be submitted.

Summary of Findings

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

ID	Title	Type	Severity
1	Out-of-order event risk due to reentrancy	Data Validation	**Informational**
2	Express lane design is unnecessarily complex	Data Validation	**Informational**
3	Validator added to Arb1 instead of Nova	Undefined Behavior	**Low**

Detailed Findings

1. Out-of-order event risk due to reentrancy	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-NITRO-PR-001
Target: `express-lane-auction/ExpressLaneAuction.sol`	

Description

A reentrancy can occur when tokens are transferred, and may cause the express lane to emit events in the wrong order.

A recurring pattern in the express lane contracts is to transfer tokens, and then emit an event:

```
function finalizeWithdrawal() external {
    uint256 amountReduced =
        _balanceOf[msg.sender].finalizeWithdrawal(roundTimingInfo.currentRound());
    biddingToken.safeTransfer(msg.sender, amountReduced);
    emit WithdrawalFinalized(msg.sender, amountReduced);
}
```

Figure 1.1: `ExpressLaneAuction.sol#L312-L317`

If the bidding token has a callback mechanism on transfer, this pattern may allow a malicious actor to reenter the contract and perform some actions that will emit events. This will cause the overall order of the event emission to be incorrect.

While there is no direct on-chain risk, it may complicate third-party integration.

Exploit Scenario

The bidding token allows for arbitrary callbacks on transfers. Eve calls `finalizeWithdrawal`. During the token transfer, the execution reenters, and Eve makes a deposit and then initiates a new withdrawal. As a result of the transaction execution, the contract emits the following events:

- `Deposit`
- `WithdrawalInitiated`
- `WithdrawalFinalized`

The last event is associated with the withdrawal created in the previous block. This out-of-order event emission perturbs the monitoring tool.

Recommendations

Short term, emit the event before the token transfer or document the associated risks for the bidding token.

Long term, document the known reentrancy risks and the constraints associated with the tokens the contracts interact with.

2. Express lane design is unnecessarily complex	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-NITRO-PR-002
Target: `express-lane-auction/ExpressLaneAuction.sol`	

Description

The express lane contracts have a lot of unnecessary complexity. This complexity increases the risks and the likelihood of mistakes in case of code updates. Below we provide general design recommendations.

- **Use a special value for uninitialized rounds.** Currently, the zero value has a double meaning: it is either the round zero or the uninitialized state. This dual state could complicate third-party integration. For example, `resolvedRounds` will return the round zero as being already resolved when the system is deployed:

```
/// @inheritdoc IExpressLaneAuction
function resolvedRounds() external view returns (ELCRound memory, ELCRound memory) {
    return latestResolvedRounds[0].round > latestResolvedRounds[1].round
        ? (latestResolvedRounds[0], latestResolvedRounds[1])
        : (latestResolvedRounds[1], latestResolvedRounds[0]);
}
```

Figure 2.1: `ExpressLaneAuction.sol#L561-L566`

- **Do not use a signed type for `offsetTimestamp`.** This makes the manipulation more complex than it should be. Given that it is a direct offset (and not a sliding value), there is no need for a negative value.

- **Consider merging `resolveSingleBidAuction` and `resolveMultiBidAuction`**. The trust assumption is that `AUCTIONEER_ROLE` is trusted, so there is no need to put the second-highest bid on-chain. If the user with the `AUCTIONEER_ROLE` is malicious, he can set up a fake account with any second bid value. While having two functions to resolve the bid eases the monitoring/tracking, it increases the attack surface. A different design would be to have one function, and let the auctioneer specify the second-highest bid directly (which would be the reserve in case of a single bid).

Recommendations

Short term, consider implementing the recommendations highlighted above.

Long term, carefully review what information needs to be put on-chain and what are the trust assumptions before implementing any contract.

3. Validator added to Arb1 instead of Nova

Severity: **Low**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-NITRO-PR-003
Target: BoLD upgrade actions	

Description

The BoLD upgrade actions are a set of actions that will be eventually executed by a privileged address to upgrade BoLD.

Each action is a smart contract with a `perform` function that implements one well-defined job. In particular, the last action is to add a new validator (0x0fF813f6BD577c3D1cDbE435baC0621BE6aE34B4) for Nova (figure 3.1). The action contract is deployed at address 0x2f845d909058200e4E56855C2735975a004a4922.

```
pragma solidity 0.8.16;

import "../address-registries/interfaces.sol";

contract SetValidatorsAction {
    IRollupGetter public immutable addressRegistry;

    constructor(IRollupGetter _addressRegistry) {
        addressRegistry = _addressRegistry;
    }

    function perform(address[] calldata _validators, bool[] calldata _values)
external {
        IRollupAdmin(address(addressRegistry.rollup())).setValidator(_validators,
_values);
    }
}
```

Figure 3.1: The `SetValidatorsAction` contract

The `perform` function is simple and will call the `setValidator` function of the rollup address returned by the `addressRegistry`. The problem is with the address that was set as the `addressRegistry` variable during deployment. The address is 0xd514C2b3aaBDBfa10800B9C96dc1eB25427520A0, which represents the address registry for Arb1 and not Nova; this means that the validator will be added to Arb1 instead of Nova.

Exploit Scenario

The BoLD upgrade actions payload is proposed to the DAO and passes. It gets executed, and the validator is added to Arb1 instead of Nova.

Recommendations

Short term, deploy a new `SetValidatorsAction` with the correct `addressRegistry` variable set to Nova (Nova address registry: 0x2F06643fc2CC18585Ae790b546388F0DE4Ec6635). Otherwise, reuse a deployed `SetValidatorsAction` contract with the `addressRegistry` already set to Nova at the following address: 0xbf94afebfbf062a88615bc012da39d0822670aba.

Long term, due to the sensitivity of these actions, at least two people should review the contracts to ensure that they have the correct configuration set, even when they are deployed through script.

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.