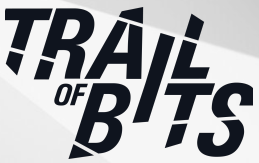

Offchain Labs ArbOS 31

Security Assessment (Summary Report)

July 24, 2024

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

Prepared by: **Gustavo Grieco and Simone Monica**

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

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

Simone Monica, Consultant
simone.monica@trailofbits.com

Project Timeline

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

Date	Event
July 1, 2024	Delivery of report draft
July 1, 2024	Report readout meeting
July 24, 2024	Delivery of summary report

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of a number of Nitro pull requests (PRs) related to Stylus and BoLD, as described in Project Coverage.

A team of two consultants conducted the review from June 24 to July 1, 2024, for a total of two engineer-weeks of effort. With full access to source code and documentation a manual code review processes.

Observations and Impact

We reviewed a number of small changes in the Nitro codebase related to Stylus cache costs and the safe handling of return memory from EVM contracts. Additionally, from the BoLD side, we reviewed a PR related to the Sepolia deployment and the addition of fast confirmation support for AnyTrust chains.

We found two issues: a low-severity issue related to an incorrect usage of the Stylus program version, and a medium-severity issue related to an unexpected failure of AnyTrust fast node confirmations. Offchain provided fixes for both issues; these were included in the scope and reviewed during the audit.

Project Targets

The engagement involved a review and testing of the targets listed below.

Nitro

Repository	https://github.com/OffchainLabs/bold
Version	`dd8cf656831ecb25c9e9001fc65148c362cb5c5d`
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 PRs:

- #2423 and #194 change how to specify programs to cache using their addresses instead of the codehashes.
- #2424 adds a cost for returning data in external calls similar to EVM.
- #2425 and #333 define a Stylus v2 and prepare the code for an upgrade.
- #2426, #2430 and #2436 separate the cached cost from the init cost when using the program cache.
- Individual commits from #193 were also covered:
  - Commit ec065286c6ff04d3854f3c1d4483e17527dca845 specifies Sepolia config values.
  - Commit d6e62dfb7473ecea69a2faecf3e50eccad84efbe includes specific Sepolia config addresses.
  - Commit c60647e83c34bf9955275026807cb1dbf741da6b increases the number of transactions required to wait for confirmation during Sepolia deployment.
- #187 allows `anyTrustFastConfirmer` role to immediately create and confirm an assertion.

We reviewed this code for usual flaws in Solidity code as well as any issues that would allow a malicious to block, delay, or disrupt Stylus or Solidity program execution inside an Arbitrum rollup. We also reviewed for potential consensus issues introduced by the ArbOS upgrade that enable these Stylus changes. We also checked for possible misconfiguration of the new parameters and features in the modified parts of either Stylus and BoLD code.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We have not reviewed in detail how the Nitro or ArbOS codebases evolved. Instead, we used the diff/PRs provided by Offchain Labs to bound the scope of this assessment.

1. Outdated stylus programs can be cached

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STY-1
Target: `arbos/programs/programs.go`	

Description

A Stylus program can be cached to spend less gas the next time it gets called. However, the property that only updated programs can be cached does not hold for Stylus version 2.

The `SetProgramCached` function performs the actual caching, and one of the preconditions is that the caching reverts if the program version is 0 and the cache flag is set to `true`. While this works correctly for Stylus version 1, when it is upgraded to version 2, the precondition will not catch outdated Stylus programs whose versions are set to 1. This makes it possible for outdated programs to be cached.

```
func (p Programs) SetProgramCached(
        emitEvent func() error,
        db vm.StateDB,
        codeHash common.Hash,
        address common.Address,
        cache bool,
        time uint64,
        params *StylusParams,
        runMode core.MessageRunMode,
        debug bool,
) error {
        program, err := p.getProgram(codeHash, time)
        if err != nil {
                return err
        }
        expired := program.ageSeconds > am.DaysToSeconds(params.ExpiryDays)

        if program.version == 0 && cache {
                return ProgramNeedsUpgradeError(0, params.Version)
        }
        ...
}
```

Figure 1.1: Snippet of the `SetProgramCached` function (`arbos/programs/programs.go#L370-L389`)

Exploit Scenario

Multiple Stylus programs with version 1 are cached, consuming unnecessary space.

Recommendations

Short term, update the condition to error if the program version is different from the current Stylus version.

Long term, consider avoiding hard-coding constants when possible. This would remove the requirement for them to be changed when an upgrade is executed. Instead, use variables where they need to be changed in a single place.

2. AnyTrust fast confirmation will fail if the confirmer address is not a validator

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-STY-2
Target: `RollupUserLogic.sol`, `RollupAdminLogic.sol`	

Description

The AnyTrust confirmation uses certain validator-only functions that will fail unless the confirmer address is added as a validator.

AnyTrust confirmation allows privileged users to fast confirm a state of a rollup, if the AnyTrust assumptions are met. In order to do that, it is necessary to define a special address as the AnyTrust fast confirmer using the following function:

```
    /**
     * @notice set the anyTrustFastConfirmer address
     *         must also call `setValidator` to set the same address as a validator to work
     *         old fast confirmer need to be removed from the validator list manually
     * @param _anyTrustFastConfirmer new value of anyTrustFastConfirmer
     */
    function setAnyTrustFastConfirmer(address _anyTrustFastConfirmer) external {
        anyTrustFastConfirmer = _anyTrustFastConfirmer;
        emit OwnerFunctionCalled(31);
    }
```

Figure 2.1: The `setAnyTrustFastConfirmer` function

This special user should be able to confirm nodes using the `fastConfirmNextNode` and `_confirmNextNode` functions:

```
    /**
     * @notice This allow anyTrustFastConfirmer to confirm next node regardless of deadline
     *         the anyTrustFastConfirmer is supposed to be set only on an AnyTrust chain to
     *         a contract that can call this function when received sufficient signatures
     */
    function fastConfirmNextNode(bytes32 blockHash, bytes32 sendRoot) external whenNotPaused
{
        require(msg.sender == anyTrustFastConfirmer, "NOT_FAST_CONFIRMER");
        _confirmNextNode(blockHash, sendRoot, true);
    }

    function _confirmNextNode(
        bytes32 blockHash,
```

```
        bytes32 sendRoot,
        bool isFastConfirm
    ) internal {
        requireUnresolvedExists();

        uint64 nodeNum = firstUnresolvedNode();
        Node storage node = getNodeStorage(nodeNum);

        if (!isFastConfirm) {
            // Verify the block's deadline has passed
            node.requirePastDeadline();
        }

        // Check that prev is latest confirmed
        assert(node.prevNum == latestConfirmed());

        Node storage prevNode = getNodeStorage(node.prevNum);
        if (!isFastConfirm) {
            prevNode.requirePastChildConfirmDeadline();
        }

        removeOldZombies(0);
        ...
```

Figure 2.2: Snippet of the `_confirmNextNode` and the `fastConfirmNextNode` functions

However, the `removeOldZombies` function still contains a check that will allow only validators to execute it:

```
    function removeOldZombies(uint256 startIndex) public onlyValidator whenNotPaused {
        uint256 currentZombieCount = zombieCount();
        uint256 latestConfirmedNum = latestConfirmed();
        for (uint256 i = startIndex; i < currentZombieCount; i++) {
            while (zombieLatestStakedNode(i) < latestConfirmedNum) {
                removeZombie(i);
                currentZombieCount--;
                if (i >= currentZombieCount) {
                    return;
                }
            }
        }
    }
```

Figure 2.3: The `removeOldZombies` function

Exploit Scenario

A AnyTrust fast confirmer attempts to call `fastConfirmNextNode` but gets a revert because its address was not previously added as a validator.

Recommendations

Short term, either modify the `fastConfirmNextNode` function to properly add or remove the AnyTrust fast confirmer addresses, or clearly document the requirement.

Long term, make sure to add tests for each new or modified code before deployment.

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.