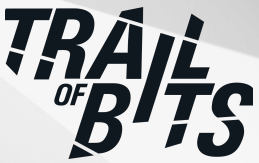

Offchain Labs, AIP 1.1 and 1.2

Security Assessment (Summary Report)

May 2, 2023

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

Prepared by: **Jaime Iglesias 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.
228 Park Ave S #80688
New York, NY 10003
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

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of its AIP 1.1 and 1.2 governance proposals. From April 24 to April 28, 2023, a team of two consultants conducted a security review of the client-provided source code, with two person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system, including access to the source code and documentation. We performed static analysis and manual testing of the target system.

Summary of Findings

The audit did not uncover any significant flaws or defects that could impact system confidentiality, integrity, or availability.

EXPOSURE ANALYSIS

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

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	1
Testing	1
Undefined Behavior	1

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

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

The following engineers were associated with this project:

Jaime Iglesias, Consultant
jaime.iglesiasbotas@trailofbits.com

Simone Monica, Consultant
simone.monica@trailofbits.com

Project Timeline

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

Date	Event
April 24, 2023	Pre-project kickoff call
May 1, 2023	Delivery of report draft
May 1, 2023	Report readout meeting
May 2, 2023	Delivery of final report
May 4, 2023	Delivery of final report with fix review appendix

Project Goals

The engagement was scoped to provide a security assessment of the AIP 1.1 and AIP 1.2 governance proposals for Offchain Labs. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the action contract developed for AIP 1.2 follow the action contract guidelines provided by Offchain Labs?
- Do the verification scripts perform all of the relevant checks?
- Can unvested funds become vested before they are intended to be?
- Are the threshold and constitution hash constants set correctly?
- Does the action contract perform all of the actions set in the proposal (i.e., changing the threshold and the constitution hash)?
- Can the L2 registry be altered after its deployment?

Project Targets

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

AIP 1.1

Repository	https://github.com/ArbitrumFoundation/governance
Version	`a2a9c8d01024a9888036eac167116a92a2d595f2`
Type	Solidity
Platform	Arbitrum

AIP 1.2

Repository	https://github.com/ArbitrumFoundation/governance
Version	`07d3e88c71c3fde922721837f947cef2868e6d67`
Type	Solidity
Platform	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:

- **`L2AddressRegistry`**
  This contract is meant to be the "single source of truth" for the addresses that governance actions may want to access. This contract will help minimize the error surface when developing governance actions, as developers will be able to query a trusted contract.

  Our focus during our review of this contract was to ensure the correctness of the code, ensure that the addresses in the registry cannot be changed once they are set, ensure that the registry is deployed with the right addresses, and review the available tests and scripts.

- **`AIP1Point2Action`**
  This action contract implements the tasks set out in AIP 1.2, namely changing the proposal thresholds for the core and treasury governor contracts to 1 million and changing the DAO constitution hash.

  Our focus while reviewing this contract was to understand the tasks set out by AIP 1.2, the governance proposal life cycle, the way action contracts should be developed, and the way the constitution hash is generated.

  Once we reviewed the documentation, we compared the code against the proposal specification, verified that action contract guidelines are being followed, and reviewed the correctness of the contract.

  Finally, we reviewed the available tests and deployment scripts.

- **`ArbitrumFoundationVestingWallet`**
  This contract implements a vesting wallet that is meant to lock the Arbitrum Foundation's funds under the supervision of the Arbitrum DAO.

  Our focus while reviewing this contract was to first understand the tasks set out by AIP 1.1 and to review the available documentation. Once we understood the purpose of the contract, we reviewed the available code to look for potential flaws in the vesting, release, and migration mechanisms and to verify that correct access controls are enforced and that the funds are delegated to the exclusion address.

  Finally, we reviewed the available tests, deployment scripts, and deployment verification scripts and verified that the proposal's specifications are enforced, such as the four-year vesting schedule, starting from the `start` timestamp.

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:

- Third-party libraries, such as OpenZeppelin's contracts, are used by this system. Although we reviewed the target system's use of these libraries, we did not review the source code of the libraries themselves, as they were out of scope.

- Some of the scripts, particularly the configuration file for the foundation wallet deployment, contain some "TODO" comments, specifically regarding the `beneficiary` and `startTimestamp` initialization arguments; therefore, we were unable to verify their correctness.

Summary of Findings

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

ID	Title	Type	Severity
1	The owner can migrate vested tokens	Undefined Behavior	**Low**
2	Lack of deployment verification scripts for AIP 1.2	Testing	**Informational**
3	Lack of a contract-existence check on the fund-migration recipient	Data Validation	**Informational**

Detailed Findings

1. The owner can migrate vested tokens	
Severity: **Low**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-AIP-1
Target: `ArbitrumFoundationVestingWallet.sol`	

Description

When funds are migrated by the owner of the wallet from the vesting wallet to another address (presumably a different vesting wallet), both the vested and unvested funds are transferred.

The `ArbitrumFoundationVestingWallet` contract exposes two owner-protected functions that allow the caller to migrate the funds from the vesting wallet to a different address.

```
/// @notice DAO can migrate unvested (as well as vested but not yet claimed) tokens
to a new wallet, e.g. one with a different vesting schedule, as per AIP-1.1.
/// @param _token address of token to be migrated
/// @param _wallet address of wallet to receive tokens
/// Emits event TokenMigrated
function migrateTokensToNewWallet(address _token, address _wallet) public onlyOwner
{
    IERC20 token = IERC20(_token);
    uint256 tokenBalance = token.balanceOf(address(this));
    token.safeTransfer(_wallet, tokenBalance);
    emit TokenMigrated(_token, _wallet, tokenBalance);
}

/// @notice DAO can migrate unvested (as well as vested but not yet claimed) Eth to
a new wallet, e.g. one with a different vesting schedule, as per AIP-1.1.
/// @param _wallet address of wallet to receive Eth
/// Emits event EthMigrated
function migrateEthToNewWallet(address _wallet) public onlyOwner {
    uint256 ethBalance = address(this).balance;
    (bool success,) = _wallet.call{value: ethBalance}("");
    require(success, "ArbitrumFoundationVestingWallet: eth transfer failed");
    emit EthMigrated(_wallet, ethBalance);
}
```

Figure 1.1: The fund-migration functions in ArbitrumFoundationVestingWallet.sol#L127-L146

As shown in figure 1.1, these functions will transfer both vested and unvested funds from the wallet, which means that the owner (the Arbitrum DAO, in this case) can transfer funds that, in theory, belong to the beneficiary, as these funds have already gone through the vesting process.

Finally, note that the beneficiary can prevent the owner from transferring the already vested tokens by calling the `release` function just before the migration is executed, which means that, if this behavior is intended, the beneficiary will be able to completely prevent it.

Exploit Scenario

The DAO grants the Arbitrum Foundation 1 million ARB tokens that unlock linearly over a year. Whenever the foundation needs liquidity, it calls the `release` function to transfer the available vested tokens to the foundation.

Just before the one-year vesting period ends, the DAO decides that whatever is left unvested of the original 1 million ARB token grant should be locked for another year. This proposal is passed and executed; however, when the DAO calls the `migrateTokensToNewWallet` function, the DAO also transfers the vested tokens, effectively leaving the foundation without access to liquidity that, in theory, it should be allowed to access.

Recommendations

Short term, consider whether the ability of the owner to transfer vested tokens out of the wallet is intended behavior.

If it is not intended behavior, then include a call to the `release` function in both of the fund-migration functions, which will transfer the available vested tokens to the beneficiary just before the unvested tokens are migrated. This will prevent the owner from migrating vested funds.

If it is intended behavior, then note in the documentation that there is no way, in the current design, to prevent the beneficiary from calling the `release` function just before the migration is executed, effectively preventing the owner from migrating vested tokens; therefore, the owner should assume that any vested tokens up to the point that the migration is executed can be transferred out by the beneficiary and should strive to plan token migrations in advance to minimize that effect.

Long term, thoroughly document the assumptions around token migrations and whether the owner should be able to migrate vested tokens as well.

2. Lack of deployment verification scripts for AIP 1.2	
Severity: **Informational**	Difficulty: **Low**
Type: Testing	Finding ID: TOB-AIP-2
Target: `scripts/proposals/AIP12`	

Description

AIP 1.2 does not contain scripts that verify whether the constants in the action contract have been set with the expected values; by contrast, AIP 1.1 includes in its deployment verification scripts checks that ensure the contracts were deployed using the correct arguments.

AIP 1.2 is tasked with deploying a governance action contract that will update the treasury and core governor contracts' proposal threshold and the DAO constitution hash.

Actions contracts are executed using `delegatecall` and allow the governance system to perform a "single-time arbitrary action". Because using `delegatecall` is risky, the Offchain Labs team documented recommendations for developing action contracts to minimize the risks.

One of the recommendations is to develop action contracts without state; otherwise, there would be a risk of storage collision between the governor executor contract and the action contract. Because action contracts should be developed without state, all data should be stored using constants or immutables. However, the use of constants or immutables does not guarantee that the values have been correctly set (e.g., developers could mistakenly set incorrect values). To ensure that the constants in the AIP 1.2 action contract are set correctly, and to keep consistent practices across AIPs, there should be deployment verification scripts that verify the constants' values.

Exploit Scenario

During the development of the action contract for AIP 1.2, the new DAO proposal threshold is incorrectly set with a very small value. Because there are no scripts to verify that the action contract was deployed with the correct values, this error goes unnoticed. As a result, users are able to create proposals when they should not be.

Recommendations

Short term, implement deployment verification scripts as part of AIP 1.2 and as part of future proposals.

Long term, thoroughly document the proposal development process and create proposal verification checklists for all present and future proposals; this will help create consistency among proposals and minimize the risk of faulty proposal implementations that go unnoticed.

3. Lack of a contract-existence check on the fund-migration recipient

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-AIP-3
Target: `ArbitrumFoundationVestingWallet.sol`	

Description

Due to a missing contract-existence check in the fund-migration functions, it is possible to migrate the wallet funds to an EOA.

The available NatSpec documentation for the `migrateEthToNewWallet` and `migrateTokensToNewWallet` functions specifies that these functions are meant to be used to migrate the funds to a new wallet.

```
/// @notice DAO can migrate unvested (as well as vested but not yet claimed) tokens
to a new wallet, e.g. one with a different vesting schedule, as per AIP-1.1.
/// @param _token address of token to be migrated
/// @param _wallet address of wallet to receive tokens
/// Emits event TokenMigrated
function migrateTokensToNewWallet(address _token, address _wallet) public onlyOwner
{
    IERC20 token = IERC20(_token);
    uint256 tokenBalance = token.balanceOf(address(this));
    token.safeTransfer(_wallet, tokenBalance);
    emit TokenMigrated(_token, _wallet, tokenBalance);
}

/// @notice DAO can migrate unvested (as well as vested but not yet claimed) Eth to
a new wallet, e.g. one with a different vesting schedule, as per AIP-1.1.
/// @param _wallet address of wallet to receive Eth
/// Emits event EthMigrated
function migrateEthToNewWallet(address _wallet) public onlyOwner {
    uint256 ethBalance = address(this).balance;
    (bool success,) = _wallet.call{value: ethBalance}("");
    require(success, "ArbitrumFoundationVestingWallet: eth transfer failed");
    emit EthMigrated(_wallet, ethBalance);
}
```

Figure 3.1: The fund-migration functions in `ArbitrumFoundationVestingWallet.sol#L127-L146`

However, as shown in figure 3.1, the code itself does not check whether the `_wallet` address is a contract, so it is possible to send the funds to an EOA, making the NatSpec documentation for these functions confusing.

Note that low-level calls (like the one used in `migrateEthToNewWallet`) will always succeed if the target is an EOA or a contract that does not exist; therefore, if the DAO proposal targets a wallet contract that does not exist, the migration will be deemed "successfully executed," which could lead to confusion and/or errors.

Exploit Scenario

The owner of the wallet (the Arbitrum DAO, in this case) triggers an ETH migration to a new wallet contract; however, because of a mistake during the proposal execution, the migration is triggered before the creation of the wallet contract, leading to a successfully executed migration that did not transfer any tokens.

Recommendations

Short term, consider whether funds can be migrated from the wallet to an EOA or whether the target address should always be a contract.

If the target should always be a contract, then add a contract-existence check to prevent errors. Otherwise, consider including other mitigation strategies, such as having different migration flows depending on whether the target is a contract (e.g., by adding a flag to the migration functions that will perform a contract-existence check when set to `true` and will not perform additional checks when set to `false`).

Long term, thoroughly document the expected behavior of the migration functions and the aforementioned risks.

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 Findings

- The `IERC20VotesUpgradeable` import is unused.

```
import {IERC20VotesUpgradeable} from "./Util.sol";
```

Figure B.1: The unused `IERC20VotesUpgradeable` import in `ArbitrumFoundationVestingWallet.sol#L8`

- OpenZeppelin's vesting wallet implementation uses the `SafeERC20Upgradeable` library (figure B.2), while Arbitrum's uses the `SafeERC20` library (figure B.3), even though Arbitrum's contract extends OpenZeppelin's. Note that, as of this writing, these libraries have effectively the same functionality; however, there is no guarantee that these libraries will always have the same functionality. Developers should be careful when choosing different libraries from those used by the code they import.

```
import "../token/ERC20/utils/SafeERC20Upgradeable.sol";
```

Figure B.2: The `SafeERC20Upgradeable` import in OpenZeppelin's `VestingWalletUpgradeable.sol#L5`

```
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

Figure B.3: The `SafeERC20` import in `ArbitrumFoundationVestingWallet.sol#L6`

- The import in figure B.4 includes code that is not needed. Replace it with the interfaces for only the `L2ArbitrumGovernor`, `ArbitrumDAOConstitution`, and `L2AddressRegistry` contracts.

```
import "../address-registries/L2AddressRegistry.sol";
```

Figure B.4: The `L2AddressRegistry` import in `AIP1Point2Action.sol#L4`

- The `ArbitrumFoundationVestingWallet` contract has to re-declare the `_beneficiary` state variable to fit the requirements of the AIP; however, there is no explicit inline documentation explaining why this is the case, which could lead to confusion.

```
address private _beneficiary;
```

Figure B.5: The `_beneficiary` state variable in `ArbitrumFoundationVestingWallet.sol#L20`

- The second error message shown in figure B.6 is missing a `:` character after `L2AddressRegistry`.

```
require(
    _treasuryWallet.owner() == _treasuryGov.timelock(),
    "L2AddressRegistry: treasury gov timelock must own treasuryWallet"
);
require(
    _arbitrumDAOConstitution.owner() == _coreGov.owner(),
    "L2AddressRegistry DAO must own ArbitrumDAOConstitution"
);
```

Figure B.6: The constructor code in `L2AddressRegistry.sol#L18-L25`

- The error message shown in figure B.7 contains the word "gov", which should be removed.

```
assertEq(
    address(arbOneAddressRegistry.coreGov()),
    address(coreGov),
    "Invalid coreGov gov address"
);
```

Figure B.7: The error message in `test/gov-actions/L2AddressRegistry.t.sol#L10-L14`

C. Mutation Testing

This appendix outlines how we conducted mutation testing for the AIP 1.1 and 1.2 code.

At a high level, mutation tests make several changes to each line of a target file and re-run the test suite for each change. Changes that result in test failures indicate adequate test coverage, while changes that do not result in test failures indicate gaps in test coverage. Although mutation testing is a slow process, it allows auditors to focus their review on areas of the codebase that are most likely to contain latent bugs, and it allows developers to identify and add missing tests.

We used Necessist to identify insufficient, incorrect, or broken test cases. Broken test cases provide a false sense of confidence in the code's correctness; therefore, it is important to identify and fix these tests to prevent security incidents.

```
necessist --framework foundry
```

Figure C.1: An example command to run Necessist on all tests

```
necessist test/ArbitrumFoundationVestingWallet.t.sol --framework foundry
```

Figure C.2: An example command to run Necessist only on the `ArbitrumFoundationVestingWallet` tests

Figure C.3 shows Necessist's output for the `ArbitrumFoundationVestingWallet` tests. Necessist found four possible mutations that, if removed, would still allow the tests to pass. To triage the results, it is necessary to understand the codebase and the purpose of the given test. For example, the first result `` `.release()` `` is not an issue because the test is trying to assert that no changes to balances occur when some vested amount is released before the start time. On the other hand, the `` `.initialize(...)` `` results could mean that the initialization of that contract is not necessary on that test or that some checks regarding correct contract initialization may be missing.

```
22 candidates in 1 test file
test/ArbitrumFoundationVestingWallet.t.sol: dry running
test/ArbitrumFoundationVestingWallet.t.sol: mutilating
test/ArbitrumFoundationVestingWallet.t.sol:131:32-131:42: `.release()` passed
test/ArbitrumFoundationVestingWallet.t.sol:189:18-189:93: `.initialize(beneficiary,
startTime, newWalletVestingDuration, address(gov))` passed
test/ArbitrumFoundationVestingWallet.t.sol:208:18-208:93: `.initialize(beneficiary,
startTime, newWalletVestingDuration, address(gov))` passed
test/ArbitrumFoundationVestingWallet.t.sol:227:18-227:93: `.initialize(beneficiary,
startTime, newWalletVestingDuration, address(gov))` passed
```

Figure C.3: Necessist output for the `ArbitrumFoundationVestingWallet` tests

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On May 4, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Offchain Labs team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the three issues described in this report, Offchain Labs has resolved two and has not resolved the remaining issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	The owner can migrate vested tokens	Low	Unresolved
2	Lack of deployment verification scripts for AIP 1.2	Informational	Resolved
3	Lack of a contract-existence check on the fund-migration recipient	Informational	Resolved

Detailed Fix Review Results

TOB-AIP-1: The owner can migrate vested tokens

Unresolved. The client provided the following context for this finding's fix status:

> *Acknowledged, won't-fix, on the grounds that in the case of a migration proposal from the DAO, the foundation has time to release their vested funds at whatever point is deemed appropriate.*

TOB-AIP-2: Lack of deployment verification scripts for AIP 1.2

Resolved in PR #52. A script that verifies whether the action contract was deployed with the correct values was added. This will help identify faulty deployments; however, the Offchain Labs team should consider making the following adjustments to the script:

- Instead of hard-coding the expected values in the script file, have the script read the values from a configuration file. This will ensure that the verification script and the deployment script are using the same "source of truth" and will prevent situations in which a value is updated in one place and not in the other.
- Update the assertion error messages to be more verbose. They currently do not provide any information regarding what went wrong.
- Consider having the script calculate the expected constitution hash in-place or read it from a configuration file rather than using a hard-coded value. This will reduce the risk of human error.

TOB-AIP-3: Lack of a contract-existence check on the fund-migration recipient

Resolved in PR #50. A contract-existence check was included in both of the fund-migration functions. The unit tests were expanded to verify that the execution will revert when the target wallet is not a contract or does not exist.