L1-L3 Teleporter

Security Assessment (Summary Report)

March 18, 2024

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

Prepared by: **Troy Sargent 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

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:

Troy Sargent, Consultant
troy.sargent@trailofbits.com

Simone Monica, Consultant
simone.monica@trailofbits.com

Project Timeline

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

Date	Event
February 9, 2024	Pre-project kickoff call
February 20, 2024	Delivery of report draft
February 20, 2024	Report readout meeting
March 18, 2024	Delivery of summary report

Project Targets

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

L1-L3 Teleporter

Repository	https://github.com/OffchainLabs/l1-l3-teleport-contracts
Version	`6a764526843965ace519c6e066fc8d90e9d43fbe`
Type	Smart contract
Platform	EVM

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of the L1-L3 Teleporter contracts. The L1-L3 Teleporter provides a way to perform ERC20 transfers from Ethereum mainnet to Orbit chains via Arbitrum Nitro.

A team of two consultants conducted the review from February 12 to February 16, 2024, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the contracts, using automated and manual processes.

Observations and Impact

Despite the underlying complexity of the inbox, token bridge, and ArbOS support for retryable transactions, the L1-L3 teleporter contracts enabled the interactions necessary to perform L1-L3 token transfers without exposing users to much additional risk. Specifically, the L1-L3 teleporter design segregates each transfer into a unique contract for each sender-receiver pair and supports pausing transfers, limiting the amount of funds that could be lost should a vulnerability be found. We focused on issues that could allow the theft of funds, cause funds to be trapped in the smart contracts, or result in charging incorrect fees.

Our review did not uncover any severe vulnerabilities or design flaws. In this report, we provide a few suggestions that will increase that quality of the code and make it more maintainable for future versions of the protocol.

Recommendations

We recommend performing additional integration testing and manual testing on a developer/test network prior to deployment.

Summary of Findings

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

ID	Title	Type	Severity
1	The _setupRole function is deprecated	Patching	**Informational**
2	Vacuous unit tests	Testing	**Informational**
3	Suggested refactorings to make precedence explicit and simplify code	Undefined Behavior	**Informational**
4	Undocumented struct fields	Undefined Behavior	**Informational**
5	Teleport function should document that contract callers should be able to create retryable tickets	Configuration	**Informational**

Detailed Findings

1. The _setupRole function is deprecated	
Severity: **Informational**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-ARBTEL-1
Target: `contracts/L1Teleporter.sol`	

Description

The `L1Teleporter` contract inherits the Openzeppelin's `AccessControl` contract. In the `constructor` function, the contract uses the `_setupRole` function to give the `DEFAULT_ADMIN_ROLE` to the `_admin` address and `PAUSER_ROLE` to the `_pauser` address (figure 1.1). However, the `_setupRole` function is deprecated in favor of the `_grantRole` function (figure 1.2).

```
constructor(address _l2ForwarderFactory, address _l2ForwarderImplementation, address
_admin, address _pauser)
     L2ForwarderPredictor(_l2ForwarderFactory, _l2ForwarderImplementation)
 {
     _setupRole(DEFAULT_ADMIN_ROLE, _admin);
     _setupRole(PAUSER_ROLE, _pauser);
 }
```

Figure 1.1: The `constructor` function (`L1Teleporter.sol#L24-L29`)

```
* NOTE: This function is deprecated in favor of {_grantRole}.
*/
function _setupRole(bytes32 role, address account) internal virtual {
    _grantRole(role, account);
}
```

Figure 1.2: The `_setupRole` function (`AccessControl.sol#L204-L208`)

Recommendations

Short term, use the `_grantRole` function instead of the `_setupRole` function.

Long term, when using third-party libraries, make sure to accurately review the documentation and follow recommendations when using the libraries.

2. Vacuous unit tests

Severity: **Informational**	Difficulty: **Low**
Type: Testing	Finding ID: TOB-ARBTEL-2
Target: `test/Teleporter.t.sol`	

Description

The tests for the fee logic of the `L1Teleporter` contract are tautological and assert that the calculation of the function being tested is equivalent to the same calculation. While this may increase code coverage, it does not specify what is correct, but rather defines the implementation as correct. Concrete values should be used instead of reperforming the calculation.

```
    assertEq(
        standardEth,
        standardCosts.l1l2TokenBridgeCost + standardCosts.l2ForwarderFactoryCost
            + standardCosts.l2l3TokenBridgeCost,
        "standardEth"
    );

    // we only check RetryableGasCosts once because it'll be the same for all modes
    assertEq(
        standardCosts.l1l2FeeTokenBridgeCost,
        gasParams.l1l2FeeTokenBridgeGasLimit * gasParams.l2GasPriceBid
            + gasParams.l1l2FeeTokenBridgeMaxSubmissionCost,
        "l1l2FeeTokenBridgeCost"
    );
    assertEq(
        standardCosts.l1l2TokenBridgeCost,
        gasParams.l1l2TokenBridgeGasLimit * gasParams.l2GasPriceBid +
gasParams.l1l2TokenBridgeMaxSubmissionCost,
        "l1l2TokenBridgeCost"
    );
    assertEq(
        standardCosts.l2ForwarderFactoryCost,
        gasParams.l2ForwarderFactoryGasLimit * gasParams.l2GasPriceBid
            + gasParams.l2ForwarderFactoryMaxSubmissionCost,
        "l2ForwarderFactoryCost"
    );
    assertEq(
        standardCosts.l2l3TokenBridgeCost,
        gasParams.l2l3TokenBridgeGasLimit * gasParams.l3GasPriceBid +
gasParams.l2l3TokenBridgeMaxSubmissionCost,
        "l2l3TokenBridgeCost"
    );
```

```
}
```

Figure 2.1: Test reimplementing contract's calculation
(`l1-l3-teleport-contracts/test/Teleporter.t.sol#201–231`)

Recommendations

Short term, add unit tests that explicitly specify expected values, and perform integration testing against a devnet deployment of Arbitrum Nitro.

Long term, create and implement testing plans as part of the design and development of new features.

3. Suggested refactorings to make precedence explicit and simplify code

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ARBTEL-3
Target: `contracts/L1Teleporter.sol`, `contracts/L2Forwarder.sol`	

Description
In the `L1Teleporter` and `L2Forwarder` contracts, some calculations chain arithmetic operations and rely on the implicit precedence of the operators instead of making the desired precedence syntactically explicit with parentheses (figures 3.1 and 3.2). In addition, some calculations are redundant, and the values can be reused to clarify that the values are expected to be equivalent (figures 3.2 and 3.3). Below are alternative implementations that are more explicit.

```
diff --git a/contracts/L1Teleporter.sol b/contracts/L1Teleporter.sol
index 52d440f..f27e9f9 100644
--- a/contracts/L1Teleporter.sol
+++ b/contracts/L1Teleporter.sol
@@ -219,13 +219,13 @@ contract L1Teleporter is Pausable, AccessControl,
L2ForwarderPredictor, IL1Telep
         returns (RetryableGasCosts memory results)
     {
         results.l1l2FeeTokenBridgeCost =
gasParams.l1l2FeeTokenBridgeMaxSubmissionCost
-             + gasParams.l1l2FeeTokenBridgeGasLimit * gasParams.l2GasPriceBid;
+             + (gasParams.l1l2FeeTokenBridgeGasLimit * gasParams.l2GasPriceBid);
         results.l1l2TokenBridgeCost =
-             gasParams.l1l2TokenBridgeMaxSubmissionCost +
gasParams.l1l2TokenBridgeGasLimit * gasParams.l2GasPriceBid;
+             gasParams.l1l2TokenBridgeMaxSubmissionCost +
(gasParams.l1l2TokenBridgeGasLimit * gasParams.l2GasPriceBid);
         results.l2ForwarderFactoryCost =
gasParams.l2ForwarderFactoryMaxSubmissionCost
-             + gasParams.l2ForwarderFactoryGasLimit * gasParams.l2GasPriceBid;
+             + (gasParams.l2ForwarderFactoryGasLimit * gasParams.l2GasPriceBid);
         results.l2l3TokenBridgeCost =
-             gasParams.l2l3TokenBridgeMaxSubmissionCost +
gasParams.l2l3TokenBridgeGasLimit * gasParams.l3GasPriceBid;
+             gasParams.l2l3TokenBridgeMaxSubmissionCost +
(gasParams.l2l3TokenBridgeGasLimit * gasParams.l3GasPriceBid);
     }
```

Figure 3.1: Suggested change to make desired precedence explicit (`l1-l3-teleport-contracts/contracts/L1Teleporter.sol#220-229`)

```
diff --git a/contracts/L2Forwarder.sol b/contracts/L2Forwarder.sol
index b250e51..d617efa 100644
--- a/contracts/L2Forwarder.sol
+++ b/contracts/L2Forwarder.sol
@@ -96,7 +96,8 @@ contract L2Forwarder is IL2Forwarder {

         // create retryable ticket
         uint256 maxSubmissionCost =
IERC20Inbox(params.routerOrInbox).calculateRetryableSubmissionFee(0, 0);
-        uint256 callValue = tokenBalance - maxSubmissionCost - params.gasLimit *
params.gasPriceBid;
+        uint256 totalFeeAmount = maxSubmissionCost + (params.gasLimit *
params.gasPriceBid);
+        uint256 callValue = tokenBalance - totalFeeAmount;
         IERC20Inbox(params.routerOrInbox).createRetryableTicket({
             to: params.to,
             l2CallValue: callValue,
@@ -109,7 +110,7 @@ contract L2Forwarder is IL2Forwarder {
             data: ""
         });

-        emit BridgedToL3(callValue, maxSubmissionCost + params.gasLimit *
params.gasPriceBid);
+        emit BridgedToL3(callValue, totalFeeAmount);
     }
```

Figure 3.2: Suggested change to perform fee calculation once
(l1-l3-teleport-contracts/contracts/L2Forwarder.sol#99–112)

```
diff --git a/contracts/L1Teleporter.sol b/contracts/L1Teleporter.sol
index 52d440f..df545a8 100644
--- a/contracts/L1Teleporter.sol
+++ b/contracts/L1Teleporter.sol
@@ -133,14 +133,14 @@ contract L1Teleporter is Pausable, AccessControl,
L2ForwarderPredictor, IL1Telep

         teleportationType = toTeleportationType({token: params.l1Token, feeToken:
params.l3FeeTokenL1Addr});

+        ethAmount = costs.l1l2TokenBridgeCost + costs.l2ForwarderFactoryCost;
         if (teleportationType == TeleportationType.Standard) {
-            ethAmount = costs.l1l2TokenBridgeCost + costs.l2ForwarderFactoryCost +
costs.l2l3TokenBridgeCost;
+            ethAmount += costs.l2l3TokenBridgeCost;
             feeTokenAmount = 0;
         } else if (teleportationType == TeleportationType.OnlyCustomFee) {
-            ethAmount = costs.l1l2TokenBridgeCost + costs.l2ForwarderFactoryCost;
             feeTokenAmount = costs.l2l3TokenBridgeCost;
         } else {
-            ethAmount = costs.l1l2TokenBridgeCost + costs.l1l2FeeTokenBridgeCost +
costs.l2ForwarderFactoryCost;
+            ethAmount += costs.l1l2FeeTokenBridgeCost;
             feeTokenAmount = costs.l2l3TokenBridgeCost;
```

```
        }
    }
```

Figure 3.3: Suggested change to emphasize base fee amount for all types (`l1-l3-teleport-contracts/contracts/L1Teleporter.sol#136–147`)

Recommendations

Short term, apply the refactorings suggested above.

Long term, prefer explicit precedence and reuse values where they are expected to be identical rather than recomputing them.

4. Undocumented struct fields	
Severity: **Informational**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-ARBTEL-4
Target: `contracts/interfaces/IL1Teleporter.sol`	

Description

The fields of the structures, `RetryableGasParams` and `RetryableGasCosts`, are undocumented, and it would be helpful to specify how and where these values are to be used for reviewers and developers, especially considering how similar they are.

```
struct RetryableGasParams {
    uint256 l2GasPriceBid;
    uint256 l3GasPriceBid;
    uint64 l2ForwarderFactoryGasLimit;
    uint64 l1l2FeeTokenBridgeGasLimit;
    uint64 l1l2TokenBridgeGasLimit;
    uint64 l2l3TokenBridgeGasLimit;
    uint256 l2ForwarderFactoryMaxSubmissionCost;
    uint256 l1l2FeeTokenBridgeMaxSubmissionCost;
    uint256 l1l2TokenBridgeMaxSubmissionCost;
    uint256 l2l3TokenBridgeMaxSubmissionCost;
}

/// @notice Total cost for each retryable ticket.
struct RetryableGasCosts {
    uint256 l1l2FeeTokenBridgeCost;
    uint256 l1l2TokenBridgeCost;
    uint256 l2ForwarderFactoryCost;
    uint256 l2l3TokenBridgeCost;
}
```

Figure 4.1: Structs with undocumented fields
(`l1-l3-teleport-contracts/contracts/interfaces/IL1Teleporter.sol#32–51`)

Recommendations

Short term, document the structures' fields.

Long term, require documentation as part of pull requests.

5. Teleport function should document that contract callers should be able to create retryable tickets

Severity: **Informational**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-ARBTEL-5
Target: `contracts/interfaces/IL1Teleporter.sol`	

Description

The `L1Teleporter`'s documentation of its teleport function does not mention that callers of `teleport` may need to create retryable tickets to call `rescueFunds` on the `L2Forwarder` should "teleporting" to L3 fail. If a contract is immutable and does not have the capability to create retryable tickets, the sender's funds may be irrecoverable.

```
/// @notice Start an L1 -> L3 transfer. msg.value sent must equal the total ETH cost
of all retryables.
///         Call `determineTypeAndFees` to calculate the total cost of retryables in
ETH and the L3's fee token.
///         If called by an EOA or a contract's constructor, the L2Forwarder will be
owned by the caller's address,
///         otherwise the L2Forwarder will be owned by the caller's alias.
/// @dev    2 retryables will be created: one to send tokens and ETH to the
L2Forwarder, and one to call the L2ForwarderFactory.
///         If TeleportationType is NonFeeTokenToCustomFeeL3, a third retryable will
be created to send the L3's fee token to the L2Forwarder.
///         ETH used to pay for the L2 -> L3 retryable is sent through the
l2CallValue of the call to the L2ForwarderFactory.
function teleport(TeleportParams calldata params) external payable;
```

Figure 5.1: Natspec of the `teleport` function
(`l1-l3-teleport-contracts/contracts/interfaces/IL1Teleporter.sol#76–83`)

Recommendations

Short term, document that contracts using the teleporter should include functionality to create retryables in case they need to call `rescueFunds` on the L2.

Long term, review and implement user-facing documentation and SDKs for validations and recommendations to make integration less error-prone.

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.