



Balancer Relayer Contracts

Security Assessment

June 23, 2022

Prepared for:

Nicolás Venturo

Balancer Labs

Prepared by:

Natalie Chin

Bo Henderson

Robert Schneider

Evan Sultanik

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 80+ 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

© 2022 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to Balancer Labs under the terms of the project statement of work and has been made public at Balancer Labs's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

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

About Trail of Bits	1
Notices and Remarks	2
Table of Contents	3
Executive Summary	4
Project Summary	5
Project Goals	6
Project Targets	7
Project Coverage	8
Codebase Maturity Evaluation	9
Summary of Recommendations	11
A. Vulnerability Categories	12
B. Code Maturity Categories	14
C. Token Integration Checklist	16
D. Code Quality Recommendations	21

Executive Summary

Engagement Overview

Balancer Labs engaged Trail of Bits to review the security of its relay contracts. From May 23 to May 27, 2022, 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 target system including access to all relevant source code and all available documentation. We performed static testing of the target system and its codebase, using both automated and manual processes.

Summary of Findings

The audit did not uncover any significant flaws or defects that could impact system confidentiality, integrity, or availability. One finding could allow an attacker to overwrite arbitrary storage slots; however, because of the time-boxed nature of the assessment, we were unable to determine whether an exploit of this issue is feasible. It is likely that such an attack would be as difficult as a keccak preimage attack.

EXPOSURE ANALYSIS

<i>Severity</i>	<i>Count</i>
Informational	4
Undetermined	1

CATEGORY BREAKDOWN

<i>Category</i>	<i>Count</i>
Access Controls	3
Data Validation	2

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:

Natalie Chin, Consultant
natalie.chin@trailofbits.com

Bo Henderson, Consultant
bo.henderson@trailofbits.com

Robert Schneider, Consultant
robert.schneider@trailofbits.com

Evan Sultanik, Consultant
evan.sultanik@trailofbits.com

Project Timeline

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

Date	Event
May 19, 2022	Pre-project kickoff call
May 31, 2022	Delivery of report draft
May 31, 2022	Report readout meeting
June 23, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the Balancer relayer system. Specifically, we sought to answer the following non-exhaustive list of questions:

- Could an attacker steal funds from the system?
- Could an attacker trap the system?
- Are there any denial-of-service attack vectors?
- Could users lose access to their funds?
- Does the system correctly validate function inputs?
- What are the risks associated with token and contract callbacks?

Project Targets

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

Balancer V2 Relay Contracts

Repository	https://github.com/balancer-labs/balancer-v2-monorepo
Version	8ac66717502b00122a3fcdf78e6d555c54528c3c
Type	Solidity
Platform	Ethereum

Project Coverage

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

- **pkg/standalone-utils/contracts/relayer/** contains the multicall proxy, composed of the BalancerRelayer and BaseRelayerLibrary contracts. We used static analysis and a manual review to test the behavior of the proxy.

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:

- The system's off-chain auditing controls were not considered in this assessment. We recommend leveraging blockchain-monitoring tools (if they are not already being used) and developing an incident response plan (if one does not already exist).
- The relayer has complex interactions with vaults, which were out of the scope of this assessment.
- This assessment covered only on-chain code; any user-facing interfaces for interacting with the relayer were out of scope. It would be beneficial to assess any such interfaces to ensure they provide adequate warnings to users about signing untrusted authorizations.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The relayer performs minimal arithmetic. Math libraries are used where appropriate. The codebase would benefit from automated testing such as fuzzing and symbolic execution.	Satisfactory
Auditing	The relayer itself does not emit any events; this is understandable, since emitting an event for each proxy call would result in significant gas costs, and presumably, the proxied functions themselves emit events. However, we recommend emitting events for less frequent interactions that could affect the system's security, such as those for receiving ETH and approving vaults and relayers.	Moderate
Authentication / Access Controls	Appropriate access controls are in place for privileged functionality. Although three informational-severity issues could allow attackers to bypass access controls, which could result in lost funds, these issues are not caused by flaws in the Balancer contracts; rather, they are either inherent to interacting with potentially untrusted third-party contracts or triggered by user error. We recommend clearly documenting these risks for users.	Satisfactory
Complexity Management	The codebase is quite complex, and understanding some interactions requires tracing their execution through several contracts. These layers of abstraction make it difficult to statically determine the intended interactions between contracts.	Moderate

Cryptography and Key Management	Off-chain key management was out of the scope of this assessment. A signature reuse issue exists but is inherent to EIP-2612 and can be mitigated only by increasing user awareness of the risks of signing arbitrary transactions.	Satisfactory
Decentralization	The system permits the use of untrusted third-party tokens that may be flawed, malicious, or problematic (e.g., tokens representing shares of EIP-4626 vault balances), which can affect the security of the Balancer contracts.	Moderate
Documentation	At a high level, Balancer has extensive, approachable user-facing documentation and adequate code comment coverage. However, developer documentation specific to the relayer is lacking.	Moderate
Front-Running Resistance	The system's use of troublesome EIPs like 2612 can permit front-running attacks.	Moderate
Low-Level Manipulation	Assembly is used in the relayer only to save gas when storing data to manually calculated storage slots. However, these storage slots are calculated based on user input. It is currently unclear whether an attacker could feasibly calculate the inputs necessary to invoke a write operation (SSTORE) to a chosen slot.	Moderate
Testing and Verification	The system has adequate unit test coverage, but more end-to-end test scenarios are needed. Additionally, we had difficulty in configuring the tests to run correctly and reproducibly across environments. Finally, the codebase would benefit from Echidna property tests.	Moderate

Summary of Recommendations

The Balancer relay system is a work in progress with multiple planned iterations. Trail of Bits recommends that Balancer Labs address the findings detailed in this report and take the following additional steps prior to deployment:

- Document all risks associated with phishing attempts and with using the `read` and `write` system calls.
- Integrate **Slither** into a continuous integration pipeline to detect common smart contract vulnerabilities.
- Identify invariants and write **Echidna** property tests to ensure the code behaves as expected.
- Improve the current unit tests by ensuring that they are reproducible, and implement a means of quantifying test coverage.
- Document and investigate mitigations for potentially risky interactions with third-party contracts and standards (e.g., EIP-4626).
- Ensure that all constructors check inputs for the zero address.

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 Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	Description
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Token Integration Checklist

The following checklist provides recommendations for interactions with arbitrary tokens. Every unchecked item should be justified, and its associated risks understood. For an up-to-date version of the checklist, see [crytic/building-secure-contracts](#).

For convenience, all [Slither](#) utilities can be run directly on a token address, such as the following:

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken --erc erc20
slither-check-erc 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d KittyCore --erc erc721
```

To follow this checklist, use the below output from Slither for the token:

```
slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER]
slither [target] --print human-summary
slither [target] --print contract-summary
slither-prop . --contract ContractName # requires configuration, and use of Echidna
and Manticore
```

General Considerations

- ❑ **The contract has a security review.** Avoid interacting with contracts that lack a security review. Check the length of the assessment (i.e., the level of effort), the reputation of the security firm, and the number and severity of the findings.
- ❑ **You have contacted the developers.** You may need to alert their team to an incident. Look for appropriate contacts on [blockchain-security-contacts](#).
- ❑ **They have a security mailing list for critical announcements.** Their team should advise users (like you!) when critical issues are found or when upgrades occur.

Contract Composition

- ❑ **The contract avoids unnecessary complexity.** The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither's [human-summary](#) printer to identify complex code.
- ❑ **The contract uses SafeMath.** Contracts that do not use SafeMath require a higher standard of review. Inspect the contract by hand for SafeMath usage.
- ❑ **The contract has only a few non-token-related functions.** Non-token-related functions increase the likelihood of an issue in the contract. Use Slither's [contract-summary](#) printer to broadly review the code used in the contract.

- ❑ **The token has only one address.** Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g., `balances[token_address][msg.sender]` may not reflect the actual balance).

Owner Privileges

- ❑ **The token is not upgradeable.** Upgradeable contracts may change their rules over time. Use Slither's `human-summary` printer to determine whether the contract is upgradeable.
- ❑ **The owner has limited minting capabilities.** Malicious or compromised owners can abuse minting capabilities. Use Slither's `human-summary` printer to review minting capabilities, and consider manually reviewing the code.
- ❑ **The token is not pausable.** Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pausable code by hand.
- ❑ **The owner cannot blacklist the contract.** Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand.
- ❑ **The team behind the token is known and can be held responsible for abuse.** Contracts with anonymous development teams or teams that reside in legal shelters require a higher standard of review.

ERC20 Tokens

ERC20 Conformity Checks

Slither includes a utility, `slither-check-erc`, that reviews the conformance of a token to many related ERC standards. Use `slither-check-erc` to review the following:

- ❑ **Transfer and transferFrom return a boolean.** Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail.
- ❑ **The name, decimals, and symbol functions are present if used.** These functions are optional in the ERC20 standard and may not be present.
- ❑ **Decimals returns a uint8.** Several tokens incorrectly return a `uint256`. In such cases, ensure that the value returned is below 255.
- ❑ **The token mitigates the known ERC20 race condition.** The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens.

Slither includes a utility, `slither-prop`, that generates unit tests and security properties that can discover many common ERC flaws. Use `slither-prop` to review the following:

- ❑ **The contract passes all unit tests and security properties from slither-prop.** Run the generated unit tests and then check the properties with **Echidna** and **Manticore**.

Risks of ERC20 Extensions

The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ❑ **The token is not an ERC777 token and has no external function call in transfer or transferFrom.** External calls in the transfer functions can lead to reentrancies.
- ❑ **Transfer and transferFrom should not take a fee.** Deflationary tokens can lead to unexpected behavior.
- ❑ **Potential interest earned from the token is taken into account.** Some tokens distribute interest to token holders. This interest may be trapped in the contract if not taken into account.

Token Scarcity

Reviews of token scarcity issues must be executed manually. Check for the following conditions:

- ❑ **The supply is owned by more than a few users.** If a few users own most of the tokens, they can influence operations based on the tokens' repartition.
- ❑ **The total supply is sufficient.** Tokens with a low total supply can be easily manipulated.
- ❑ **The tokens are located in more than a few exchanges.** If all the tokens are in one exchange, a compromise of the exchange could compromise the contract relying on the token.
- ❑ **Users understand the risks associated with a large amount of funds or flash loans.** Contracts relying on the token balance must account for attackers with a large amount of funds or attacks executed through flash loans.
- ❑ **The token does not allow flash minting.** Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token.

ERC721 Tokens

ERC721 Conformity Checks

The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ❑ **Transfers of tokens to the 0x0 address revert.** Several tokens allow transfers to 0x0 and consider tokens transferred to that address to have been burned; however, the ERC721 standard requires that such transfers revert.
- ❑ **safeTransferFrom functions are implemented with the correct signature.** Several token contracts do not implement these functions. A transfer of NFTs to one of those contracts can result in a loss of assets.
- ❑ **The name, decimals, and symbol functions are present if used.** These functions are optional in the ERC721 standard and may not be present.
- ❑ **If it is used, decimals returns a uint8(0).** Other values are invalid.
- ❑ **The name and symbol functions can return an empty string.** This behavior is allowed by the standard.
- ❑ **The ownerOf function reverts if the tokenId is invalid or is set to a token that has already been burned.** The function cannot return 0x0. This behavior is required by the standard, but it is not always properly implemented.
- ❑ **A transfer of an NFT clears its approvals.** This is required by the standard.
- ❑ **The token ID of an NFT cannot be changed during its lifetime.** This is required by the standard.

Common Risks of the ERC721 Standard

To mitigate the risks associated with ERC721 contracts, conduct a manual review of the following conditions:

- ❑ **The onERC721Received callback is taken into account.** External calls in the transfer functions can lead to reentrancies, especially when the callback is not explicit (e.g., in `safeMint` calls).

- ❑ **When an NFT is minted, it is safely transferred to a smart contract.** If there is a minting function, it should behave similarly to `safeTransferFrom` and properly handle the minting of new tokens to a smart contract. This will prevent a loss of assets.
- ❑ **The burning of a token clears its approvals.** If there is a burning function, it should clear the token's previous approvals.

D. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- The solc pragma `^0.7.0` is old and non-constant. Monitor the [known solc bugs](#) to ensure nothing in the `0.7.x` range applies to the codebase, and consider switching to a newer pragma.
- `BaseRelayerLibrary.approveVault` and `BaseRelayerLibrary.getEntrypoint` are public but not used internally. Consider marking them as `external` instead.
- A variable in [the loop](#) in `BaseRelayerLibrary._pullTokens` is implicitly initialized to zero. This initialization should be made explicit (e.g., replace `uint256 i;` with `uint256 i = 0;`) to enhance readability.