Uniswap Foundation Staking Infrastructure

Security Assessment (Summary Report)

February 23, 2024

Prepared for:
Erin Koen
Uniswap Foundation

Prepared by: **Richie Humphrey and Robert Schneider**

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

This report is considered by Trail of Bits to be public information; it is licensed to Uniswap Foundation under the terms of the project statement of work and has been made public at Uniswap Foundation'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.

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

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:

Sam Greenup, Project Manager
sam.greenup@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:

Richie Humphrey, Consultant
richie.humphrey@trailofbits.com

Robert Schneider, Consultant
robert.schneider@trailofbits.com

Project Timeline

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

Date	Event
February 8, 2024	Pre-project kickoff call
February 20, 2024	Delivery of report draft
February 20, 2024	Report readout meeting
February 23, 2024	Delivery of summary report

Executive Summary

Engagement Overview

Uniswap Foundation engaged Trail of Bits to review the security of the Unistaker protocol, which is a system for staking governance tokens and distributing rewards. A team of two consultants conducted the review from February 12 to February 16, 2024, for a total of two engineer-weeks of effort. Our testing efforts focused on the pools and related contracts. With full access to the source code and documentation, we performed static and dynamic testing, using automated and manual processes.

Observations and Impact

The Unistaker contracts are well-designed, with a clear focus on security and simplicity. The code quality is high, with most functions serving a singular purpose and the presence of comprehensive NatSpec comments throughout the codebase.

The test suite implements fuzzing, providing more comprehensive coverage than unit tests alone. Our mutation testing indicates that the current tests have excellent coverage. However, there is a lack of stateful invariant tests. Such tests could have identified issues like TOB-UNIFEE-1, which we found using Echidna.

The protocol includes an interesting permissionless mechanism for collecting fees from various pools, converting them to the reward token, and sending them to the staking contract. Outsourcing this process reduces the complexity of the contract and pushes the associated risks onto an MEV searcher or other fee claimer.

The project currently lacks any external documentation. User-facing documentation is necessary for stakers, and developer documentation is helpful for integrators building on Uniswap.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Uniswap Foundation take the following steps:

- **Remediate the finding disclosed in this report.** There is one finding that should be considered for remediation, TOB-UNIFEE-1.

- **Enhance external documentation.** Create technical documentation for developers and integrators, as well as user-facing documentation to educate and inform on the staking contract.

- **Enhance testing.** Consider adding invariant tests to the current test suites (see appendix C) to ensure that important system properties hold. Additionally, if future development is planned, consider incorporating mutation testing to ensure maintaining the highest-quality tests.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description
slither-mutate	A static analysis framework that can statically verify algebraic relationships between Solidity variables
Echidna	A smart contract fuzzer that can rapidly test security properties via malicious, coverage-guided test case generation

We also used Slither for static analysis of the codebase, but it did not identify any security issues.

Test Results

The results of this focused testing are detailed below.

`UniStaker.sol`

This contract is used to stake governance tokens and distribute reward tokens. We developed a stateful invariant testing harness for the contracts to be used with Echidna. Aside from the property reported in TOB-UNIFEE-1, all other properties held during a testing session that completed over 4,000,000 runs using a sequence length of 350.

Property	Tool	Result
1. The contract's `totalStaked` variable is equal to the sum of all deposits less the sum of all withdrawals.	Echidna	**Passed**
2. The sum of the governance token balances of all the surrogate contracts is equal to the sum of all deposits less the sum of all withdrawals.	Echidna	**Passed**
3. The sum of all users' `depositorTotalStaked` amounts is	Echidna	**Passed**

equal to the value of `totalStaked`.		
4. The sum of all users' `deposits` balances is equal to the sum of all deposits less the sum of all withdrawals.	Echidna	**Passed**
5. The sum of the amounts delegated to delagatees is equal to the contract's `totalStaked` variable.	Echidna	**Passed**
6. The sum of all amounts applied to beneficiaries is equal to the contract's `totalStaked` variable.	Echidna	**Passed**
7. The sum of all beneficiaries' `earningsPower` amounts is equal to the sum of all deposits less the sum of all withdrawals.	Echidna	**Passed**
9. The sum of the increases in beneficiaries' reward token balances is equal to the rewards distributed by the system.	Echidna	**Passed**
10. The sum of the increases in beneficiaries' reward token balances plus the reward token balance of the UniStaker contract is equal to the sum of all rewards notified, plus the sum of all reward token donations, less the sum of rewards claimed, less the sum of all rewards not transferred in during reward notification	Echidna	**Passed**
11. The reward token balance is greater than or equal to the remaining reward payable.	Echidna	TOB-UNIFEE-1
12. The UniStaker contract's reward token balance is equal to the sum of all rewards notified, plus the sum of all reward token donations, less the sum of rewards claimed, less the sum of all rewards not transferred in during reward notification	Echidna	**Passed**
13. The `lastCheckpointTime` variable is greater than or equal to the previous value.	Echidna	**Passed**
14. The `rewardPerTokenAccumulatedCheckpoint` amount is greater than or equal to the previous amount.	Echidna	**Passed**

slither-mutate: The following table displays the portion of each type of mutant for which all unit tests passed. The presence of valid mutants indicates that there are gaps in test coverage because the test suite did not catch the introduced change.

- Uncaught revert mutants replace a given expression with a `revert` statement and indicate that the line is not executed during testing.
- Uncaught comment mutants comment out a given expression and indicate that the effects of this line are not checked by any assertions.
- Uncaught tweak mutants indicate that the expression being executed features edge cases that are not covered by the test suite.

The `scopelift/src` subdirectory is the root for all target paths listed below. Targets that are out of scope (e.g., governance contracts, timelock, Uniswap pools) or that produced zero analyzed mutants (e.g., interfaces) were omitted from mutation testing analysis.

Target	Uncaught Reverts	Uncaught Comments	Uncaught Tweaks
`UniStaker.sol`	0%	0%	10%
`V3FactoryOwner.sol`	0%	0%	10%
`DelegationSurrogate.sol`	0%	0%	0%

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 codebase does not rely heavily on arithmetic. The tests appear to cover all significant operations. The only rounding issue we observed was a 1 wei difference when dividing rewards among recipients, which is unavoidable due to fixed-point math. The rounding we observed was always in favor of the protocol, so no free tokens were given out.	**Satisfactory**
Auditing	All changes to state variables and critical operations correctly generate events. However, the event system could be improved with the use of a technical specification and better documentation. Currently, there is no incident response plan in place, although it is planned for the future.	**Moderate**
Authentication / Access Controls	The access controls in place are adequate. The system could be improved by documenting the list of privileged actors and description of their roles. Also, we recommend using a two-step process for changing ownership (see the code quality appendix).	**Satisfactory**
Complexity Management	The contracts are written with emphasis on sustainability and simplicity. The functions are single-purpose with little branching and low cyclomatic complexity. The protocol includes a novel mechanism for collecting fees and sending them to the staking contract that offers an incentive for this work to be done by outside actors,	**Satisfactory**

	thereby removing the associated complexity.	
Decentralization	The contracts are not upgradeable. Ownership is a timelock contract. Privileged actors are not able to unilaterally move funds out of the contract. Critical configuration parameters are immutable once deployed.	**Satisfactory**
Documentation	The NatSpec is mostly complete for all external functions, and there are helpful inline comments throughout. However, there currently is no external documentation for users or integrators. Additionally, some user-facing documentation does not identify the risks and nuances of the staking contract, which is important. Technical developer documentation would also be helpful for integrators or MEV searchers interested in collecting fees.	**Weak**
Transaction Ordering Risks	The developers identified one low-risk permit front-running issue during the course of the audit that we've mentioned in the code quality appendix.	**Satisfactory**
Low-Level Manipulation	No low-level manipulation is used in this codebase.	**Not Applicable**
Testing and Verification	The code includes a comprehensive fuzz-testing suite with close to 100% code coverage. Our mutation testing also confirmed that the tests have excellent coverage. We did note a lack of stateful invariant tests. Such tests could have identified issues like TOB-UNIFEE-1, which we found using Echidna.	**Moderate**

Summary of Findings

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

ID	Title	Type	Severity
1	notifyRewardAmount() can be called without transferring tokens	Data Validation	**Medium**

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
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
Transaction Ordering	The system's resistance to transaction-ordering attacks

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. Code Quality Recommendations

We recommend considering the following recommendations to improve code quality.

- **Permit front-running**

  During the review, the Uniswap development team alerted us to a possible issue related to the use of `permit` in the `permitAndStake` and `permitAndStakeMore` functions. The fundamental issue is that someone could front-run the call to `permit`, which would cause these functions to revert. We agree with the team's suggested mitigation: to wrap each call to permit in a try/catch statement.

- **Implement ownable two-step**

  Consider using a two-step process for transferring ownership of the `V3FactoryOwner` contract.

- **Add zero address checks**

  Consider adding a check for zero address when setting `REWARD_RECEIVER` in the constructor of `V3FactoryOwner`.

- **Add missing return value**

  As a convenience to integrators, consider having the `UniStaker.claimRewards` function return the amount of rewards distributed.

- **Add complete NatSpec**

  Consider adding complete NatSpec for all external functions. We found these issues using natspec-smells.

  - `UniStaker:lastTimeRewardDistributed  src/UniStaker.sol:219`
    `@return` missing for unnamed
  - `UniStaker:rewardPerTokenAccumulated src/UniStaker.sol:227`
    `@return` missing for unnamed
  - `UniStaker:unclaimedReward src/UniStaker.sol:238`
    `@param _beneficiary` is missing
    `@return` missing for unnamed