ULTI

Security Assessment (Summary Report)

December 27, 2024

Prepared for:
0xStef
ULTI

Prepared by: **Elvis Skoždopolj**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

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 consultant was associated with this project:

Elvis Skoždopolj, Consultant
elvis.skozdopolj@trailofbits.com

Project Timeline

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

Date	Event
December 13, 2024	Pre-project kickoff call
December 20, 2024	Delivery of report draft
December 20, 2024	Report readout meeting
December 23, 2024	Completion of fix review
December 27, 2024	Delivery of final summary report

Project Targets

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

ULTI Smart Contracts

Repository	https://github.com/ulti-org/ulti-protocol-contract
Version	`a03dd8890760ea2c517858fe35461619ccf58110`
Type	Solidity
Platform	Ethereum

Executive Summary

Engagement Overview

ULTI engaged Trail of Bits to review the security of the ULTI smart contracts. ULTI is an incentivized savings protocol through which users acquire ULTI tokens. ULTI creates a Uniswap v3 position consisting of the ULTI token and the input/underlying token. Users depositing into the protocol get a certain amount of ULTI minted to them depending on the current time-weighted average price (TWAP) ratio in the Uniswap pool, and are incentivized to keep depositing via the streak, referral, and top contributor bonuses. Top contributors are able to call a permissioned function to swap the reserves of the underlying token for ULTI.

One consultant conducted the review from December 16 to December 20, 2024, for a total of one engineer-week of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

The ULTI system consists of three smart contracts: the main `ULTI` contract, the `ULTIData` contract, which contains functions meant to be called off-chain, and the `ULTIShared` library, which contains the constants. The focus of this review was the main `ULTI` contract.

We reviewed the deposit and claim flow, looking for ways in which users could increase their ULTI allocation, ways in which users could decrease another user's allocation, arithmetic and rounding issues, and MEV risks. We reviewed how referrals and contributions are accounted for to investigate whether it is possible to loop or chain referrals in order to gain a larger ULTI bonus, whether referrals can be canceled or replaced, whether accounting values are properly removed when a top contributor is deleted, whether a top contributor can be unfairly removed or added, and whether the claim delays can be avoided. The library contracts in the `lib/uniswap` directory are copies of the Uniswap v3 libraries with some minor modifications. We compared these libraries against the original implementation and differentially fuzzed the `FullMath` library using Foundry. The deployment scripts were considered out of scope for this review.

We discovered one informational-severity finding in the modifications made to the `FullMath` function and one low-severity finding related to inconsistent precision when minting ULTI, which could result in less ULTI being minted. We also found that the Uniswap reentrancy lock can be used to force a silent failure of the `_tryIncreaseLiquidity` function by reentering into the `ULTI` contract's `deposit` function from a swap or mint callback, keeping the input token in the `ULTI` contract. However, this does not present a clear security issue at this time, so it is not disclosed as a finding in this report.

Additionally, we noted several code quality issues; addressing them would improve the code readability and maintainability.

Recommendations

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

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Implement a thorough fuzzing harness.** The system implements large delays (e.g., the bonus claim delay, which is 99 days), relies on interactions with a Uniswap pool, and requires at least 33 top contributors before top contributors are sorted. Testing the protocol for a longer time period (i.e., simulating time passing via fuzzing) will help to confirm that the protocol behaves as expected when all features are active.

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 arithmetic formulas are straightforward, mostly dealing with price conversion and percentage calculations. All arithmetic formulas are thoroughly documented in the code. Unchecked arithmetic is used only in the modified third-party libraries. However, we did discover one issue in which unchecked arithmetic should be used (finding 2), and ULTI could benefit from more consistency in the arithmetic precision and the selection of scaling factors (finding 1).	**Moderate**
Auditing	Most state-changing functions emit events, although having `_updateTopContributors` emit an event would be beneficial to monitor the correct replacement of top contributors. We are not aware of an incident response plan or monitoring strategy; however, the system is fully decentralized, and the owner has no power to modify it after it is deployed. Due to this decentralization, monitoring is not as important for this particular system.	**Satisfactory**
Authentication / Access Controls	The `ULTI` contract contains only two permissioned functions, one for the owner, which is used only once to launch the system, and one for the top contributors. No access control issues were found. The system is fully decentralized.	**Satisfactory**
Complexity Management	The system consists of only one main smart contract and two utility contracts. The contracts and libraries used by the system are inherited from Uniswap and OpenZeppelin or are copied and modified from Uniswap. The API is limited, with only a couple of actions available to users. The most complex parts of the system are the interactions of the referral, streak, and contributor	**Satisfactory**

	bonuses. Selecting a single scaling factor for arithmetic calculations would help reduce some complexity of the arithmetic.	
Decentralization	The system is fully decentralized after launch. The owner is only able to launch the system, after which the ownership is renounced. The only other privileged role, the guardians, is selected based on contributions to the protocol.	**Strong**
Documentation	The protocol has strong documentation in the form of a litepaper, a whitepaper, and very extensive inline code and NatSpec documentation. It would be beneficial to create additional internal documentation detailing the possible scenarios in which the `pump` function might not work or might lead to larger slippage.	**Strong**
Low-Level Manipulation	The system does not use low-level manipulation except in the modified third-party library contracts.	**Satisfactory**
Testing and Verification	The codebase has unit tests written in JavaScript that cover most expected scenarios. The testing suite does not test interactions with the Uniswap pool outside the protocol. Defining invariants and testing them using smart contract fuzzing would be beneficial in order to confirm the system behaves as expected under heavy load, during external interactions with the Uniswap pool, and after the predefined delays have passed. The testing suite was only briefly reviewed; this area requires additional investigation.	**Further Investigation Required**
Transaction Ordering	The system is potentially susceptible to MEV risks; however, these risks seem to be properly mitigated via the use of slippage parameters.	**Satisfactory**

Summary of Findings

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

ID	Title	Type	Severity
1	Users can receive less ULTI for deposits due to precision loss	Data Validation	**Low**
2	Missing unchecked block in FullMath	Data Validation	**Informational**

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
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 Findings

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

- The codebase uses a number of different scaling factors and denominators depending on the calculations performed. This creates unnecessary cognitive overhead and makes the codebase harder to review and maintain. We recommend that a single precision factor and denominator be chosen and used for calculations throughout the codebase.

- The `name` and `symbol` variables used in the `ULTI` contract's constructor shadow the ERC-20 state variables. The variables should be renamed in order to avoid variable name shadowing.

- The `nextDepositOrClaimTimestamp` mapping is checked twice in the `_deposit` function. The second occurrence can be safely removed since it was already checked in the if-revert statement.

```
if (block.timestamp < nextDepositOrClaimTimestamp[msg.sender]) revert
DepositCooldownActive();

// 2. Auto-claim pending ULTI if requested
bool autoClaimed;
if (autoClaim && block.timestamp >= nextDepositOrClaimTimestamp[msg.sender] &&
claimableUlti[msg.sender] > 0) {
```

Figure C.1: `ULTI.sol#L894–L898`

- The `maxBonusPlusOne` local variable is the result of an expression using only constants and literals. This expression can be rewritten as a constant.

```
uint256 maxBonusPlusOne = (ULTIShared.STREAK_BONUS_MAX_PERCENTAGE + 1) *
ULTIShared.PRECISION_FACTOR_1E6 / 100; // 0.34 scaled by 1e6
```

Figure C.2: `ULTI.sol#L1593`

- A number of constants are assigned to the result of an expression that uses other constants. This type of constant is evaluated at runtime instead of the result being saved in bytecode, resulting in a slightly higher gas cost. The constants can be precomputed and directly assigned to a value in order to reduce the gas costs.

```
/// @notice Cycle interval in seconds (33 days)
uint256 public constant CYCLE_INTERVAL = (ULTI_NUMBER * 1 days);
```

```
/// @notice Minimum time interval for Time-Weighted Average Price (TWAP) calculation
(18 minutes and 9 seconds)
uint32 public constant MIN_TWAP_INTERVAL = uint32((ULTI_NUMBER * 33 seconds));

/// @notice Interval between all bonuses claims (99 days)
uint256 public constant ALL_BONUSES_CLAIM_INTERVAL = (ULTI_NUMBER * 3 days);

// Liquidity pool constants
/// @notice Percentage of contributions allocated to liquidity pool
uint256 public constant LP_CONTRIBUTION_PERCENTAGE = ULTI_NUMBER / 11; // 3%

/// @notice Maximum allowed slippage for adding liquidity in basis points: 99 BPS
(0.99%)
uint256 public constant MAX_ADD_LP_SLIPPAGE_BPS = 3 * ULTI_NUMBER;

/// @notice Maximum allowed slippage for swaps in basis points: 132 BPS (1.32%)
uint256 public constant MAX_SWAP_SLIPPAGE_BPS = 4 * ULTI_NUMBER;

// Bonus-related constants
/// @notice Percentage of contributions allocated to top contributors
uint256 public constant TOP_CONTRIBUTOR_BONUS_PERCENTAGE = ULTI_NUMBER / 11; // 3%

/// @notice Interval between pump actions: 3300 seconds (55 minutes)
uint256 public constant PUMP_INTERVAL = (ULTI_NUMBER * 100 seconds);

/// @notice Minimum number of pumps required to be classified as an active pumper
uint256 public constant MIN_PUMPS_FOR_ACTIVE_PUMPERS = ULTI_NUMBER / 3;

/// @notice Maximum number of pumps allowed per user per cycle (33)
uint256 public constant MAX_PUMPS_FOR_ACTIVE_PUMPERS = ULTI_NUMBER;

/// @notice Percentage bonus for active pumpers (3% of the top contributor bonus)
uint256 public constant ACTIVE_PUMPERS_BONUS_PERCENTAGE = ULTI_NUMBER / 10;
```

Figure C.3: `ULTIShared.sol`

- The `inputTokenDecimals` state variable is set in the constructor but is never used. It can be removed.
- The `_isTopContributor` function of the `ULTIData` contract fetches a list of contributors and loops over it in order to determine whether the user address is a part of the list. Since the `topContributors` mapping is an `EnumerableMap` type, this information can be more easily fetched by using the `contains` function.
- The `_isActivePumper` function of the `ULTI` contract could be made public in order to avoid having to reimplement it in the `ULTIData` contract.
- The `depositedInCurrentCycle` variable could be removed and the check could be performed directly in the `if` statement, since the variable is used only once.

```
bool depositedInCurrentCycle = currentCycleDeposits > 0;
if (!depositedInCurrentCycle) {
    streakCount = ulti.streakCounts(cycle - 1, user);
}
```

Figure C.4: ULTIData.sol#L238–L241

- The `_updateTopContributors` function should emit an event when a new contributor is added or a previous contributor is removed from the set. This will help with monitoring the contract for unexpected behavior.
- The `getUserData` function of the `ULTIData` contract returns zero as the minimum bound instead of the actual minimum for the first cycle. This should be fixed.

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 December 23, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the ULTI team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, ULTI has resolved both of the issues disclosed in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Users can receive less ULTI for deposits due to precision loss	Low	Resolved
2	Missing unchecked block in FullMath	Informational	Resolved

Detailed Fix Review Results

TOB-ULTI-1: Users can receive less ULTI for deposits due to precision loss
Resolved in commit `579d372`. The order of operations in the `_allocateDeposit` function's arithmetic was updated to perform multiplication before division in order to preserve the precision of the operation.

TOB-ULTI-2: Missing unchecked block in FullMath
Resolved in commit `648f79f`. Unchecked arithmetic was added to the `FullMath` library's `mulDiv` function.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.