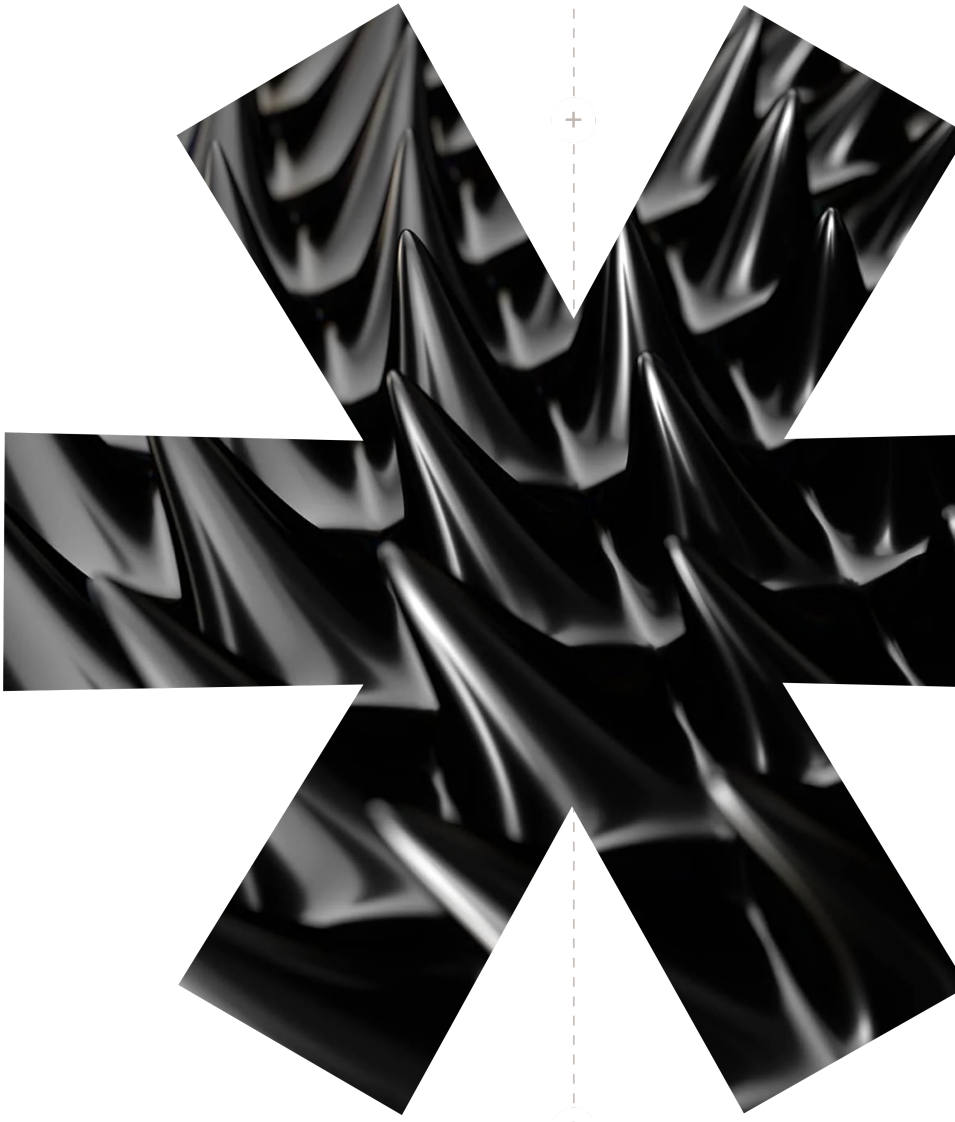
thesisdefense

Security Audit Report

Zest Protocol

Zest Protocol Smart Contracts

Initial Report // May 1, 2024
Final Report // May 7, 2024



Team Members

Jehad Baeth // Senior Security Auditor
Wanas Elhassan // Senior Security Auditor
Bashir Abu-Amr // Head of Delivery

Table of Contents

About Thesis Defense

Thesis Defense serves as the auditing services arm within Thesis, Inc., the venture studio behind tBTC, Fold, Taho, Etcher, and Mezo. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of technologies including smart contracts, wallets and browser extensions, bridges, node implementations, cryptographic protocols, and dApps. We offer our services within a variety of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos / Cosmos SDK, NEAR and more.

Thesis Defense will employ the Thesis Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

`Section_1.0`

Scope

Technical Scope

- **Repository:** https://github.com/Zest-Protocol/zest-contracts
- **Audit Commit:** 3a7d995287ee397cd0afb76196d0a3e2b62e5e37
- **Verification Commit:** b922e41231f8717d9941f26ed672f6b939a5f588
- **Files in Scope:**
  - onchain/contracts/borrow/pool/liquidation-manager-v1-2.clar
  - onchain/contracts/borrow/pool/pool-borrow-v1-2.clar
  - onchain/contracts/borrow/vaults/pool-0-reserve
- **Files Out of Scope:**
  - onchain/contracts/borrow/deployment/*
  - onchain/contracts/borrow/math/*
  - onchain/contracts/borrow/mocks/*
  - onchain/contracts/borrow/oracle/*
  - onchain/contracts/borrow/pool/fees-calculator.clar
  - onchain/contracts/borrow/pool/liquidation-manager.clar
  - onchain/contracts/borrow/pool/oracle.clar
  - onchain/contracts/borrow/pool/pool-borrow.clar
  - onchain/contracts/borrow/reads/*
  - onchain/contracts/borrow/token/*
  - onchain/contracts/borrow/trait/*
  - onchain/contracts/borrow/vaults/pool-reserve-data.clar
  - onchain/contracts/borrow/vaults/pool-vault.clar
  - onchain/contracts/borrow/vaults/borrow-helper.clar
  - onchain/contracts/borrow/vaults/supply-wrapped.clar

Documentation

- Developer documentation: https://docs.zestprotocol.com/dev-docs

Section_2.0

Executive Summary

Schedule

This security audit was conducted from April 18, 2024 to May 1, 2024 by 2 senior security auditors for a total of 4 person-weeks.

Overview

This security audit report outlines our approach and details the findings and outcomes of our security audit of the Zest Protocol smart contracts. Zest Protocol is a Stacks blockchain-based lending protocol that leverages Clarity smart contracts. The main protocol feature that we targeted in this security audit was functionality handling, including:

- Deposits into and withdrawals from the liquidity pools where liquidity providers can deposit their assets to generate returns;
- Borrowing from and repaying into the borrowing pools;
- The management of collateral and liquidation for cases where a borrowing position becomes undercollateralized; and
- Flashloans.

Overall, we found that the Zest Protocol team has strongly and demonstrably considered security in their due diligence approach to the security of their protocol. Specifically, they solicited the services of several auditors, including individual white hats and security auditing teams. In addition, they shared information and findings across all participants to promote transparency and optimize coverage of the smart contracts. This approach creates both the redundancy and comprehensive coverage necessary to minimize the potential for security vulnerabilities.

Objectives

We identified the following objectives and areas of concern for our security audit:

- Claim ownership validations and the ability of suppliers (i.e. liquidity providers) to always withdraw all of their assets, unless they are being used as collateral or being borrowed against;
- Caller request validation flows;
- Manipulating the health factor, a critical metric in lending protocols which is determined by the level of collateralization of a borrow position, to exploit the protocol's liquidation mechanisms;
- Manipulating the liquidation-bonus (liquidation spread). Liquidation spreads are used to incentivize liquidations and if manipulated by an attacker, they may be able to disrupt the protocol's liquidation mechanism;
- A borrower using an unapproved assets as collateral to make a borrow position;
- Clarity smart contracts bugs and code smells;
- Gas cost reduction and optimizations; and
- Protocol configuration actions authentication.

Methodology

- We conducted a threat modeling exercise to determine an appropriate threat and trust model for the smart contracts;
- We performed a line-by-line manual code review of the smart contracts in scope, in addition to manual review of some out of scope code where necessary, specifically to verify that the core public functions of the protocol behave as expected;
- We created tests and performed testing on the attack surfaces set out in the objectives section of this audit report; and
- We performed a close investigation and analysis of the known issues provided to us by the Zest Protocol team during the audit, which included issues and proofs of concept (POCs) from security audits performed by other security auditors on recent versions of the smart contracts in scope, the smart contracts that are the target of our current security audit, in addition to the protocol functionality that is outside of the scope of our audit (e.g math.clar).

Threat Model

We conducted a threat model that included malicious users and observers of the protocol as the main threat actors. In doing so, we considered the governance of the protocol (out of scope) to be sufficiently decentralized and not malicious. We assumed that all out of scope components are not trusted, but functioned as intended.

Other security auditing parties were auditing the out of scope components in parallel to our review. All findings were shared with us, including PoCs, suggestions, and fixes to any issues identified. The issues shared with us during the audit relating to our scope of work have been verified and addressed in the audit commit hash.

Security of the Implementation

In our manual code review of the smart contracts, we found that the implementation demonstrates good overall code quality and adherence to smart contract implementation best practices with a modular design, appropriate error handling, and an intuitive function and variable naming convention.

During our review of the implementation, we found that some functions are prone to complexity and should be simplified Issue 3. Increased complexity can result in potential oversights and errors that may lead to vulnerabilities and, as a result, breaking these functions down into smaller, more focused components would improve code readability and maintainability. This also facilitates easier understanding and upgradeability of the code.

In addition, we found that the system implements appropriate error handling, with case-specific errors allowing for robust and reliable operation.

Testing

We found that the Zest Protocol codebase implements a test suite that provides sufficient coverage of the implementation. However, some in scope system components lacked proper tests due to recent changes in the codebase. Specifically, the `pool-borrow-v1-2.clar` smart contract had no tests implemented. As a core smart contract and an in scope component for our security audit, we wrote tests in order to sufficiently test the correctness of its functionality.

In our testing, we investigated the attack surfaces detailed in the objectives section of this audit report. Our tests ruled out the following scenarios:

- Closing a borrow position by making a partial payment during liquidation;
- Closing a borrow position without repaying the entire amount;
- A liquidator liquidating more assets than needed to end under-collateralization during liquidation;
- A severely under-collateralized position that is fully liquidated but does not lead to closing the position;
- A borrower benefits from self-liquidation of a borrow position;
- Closing a borrow position by repaying with a different asset than what was borrowed;
- A supplier (liquidity provider) withdraws more than the amount they supplied, excluding the pool earnings;
- Repaying a closed borrow position;
- Closing a borrow position by making a partial payment during liquidation;
- Creating a flashloan without paying the flashloan fee; and
- Repaying more than a borrow positions value in liquidation;

Through manual code review and testing, we confirmed the following assumptions:

- Repaying in installments yields correct calculations; and
- Repaying a borrow position with a surplus amount to the loan value deducts only the outstanding balance.

The tests created and used in this review have been shared with the Zest Protocol team.

Project Documentation

The documentation provided by the Zest Protocol team was sufficient to inform and facilitate our understanding of the smart contracts. This allowed us to sufficiently reason about the correctness of the implementation and intended functionality. We recommend continuous updates to the documentation and improvements to the code comments as the codebase continues to evolve and the smart contracts are modified for improvements.

We also note that during the security audit, the Zest Protocol team was readily available to answer our questions and to facilitate a comprehensive understanding of the smart contracts.

Recommendations Summary

During this security audit, we identified a low severity issue (Issue 1) that is related to the lack of implementation of input validation, which mitigates against unintended outcomes when administering and configuring the smart contracts.

After rigorous code review and testing of the core public functions of the smart contracts we did not identify any critical, high, or medium severity security issues that can impact the core functionality of the Zest Protocol.

We also noted that little input validation is implemented on key protocol parameters, thus making their configuration prone to human error. We recommend adhering to best practices in implementing sanity checks on security critical parameters where applicable.

Section_3.0

Key Findings Table

Issues	Severity	Status
ISSUE #1 Unvalidated Input in set-health-factor-liquidation-treshold Function	Low	Fixed
ISSUE #2 Overly Complex borrow Function	None	Fixed
ISSUE #3 Unnecessary Gas Consumption	None	Fixed

Severity definitions can be found in Appendix A

Section_4.0

Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE #1

Unvalidated Input in set-health-factor-liquidation-treshold Function

Low Fixed

Location

pool-0-reserve.clar#L53-L56

Description

The `set-health-factor-liquidation-treshold` function in the referenced smart contract does not validate the value range of its input, allowing for the possibility of a Health Factor Liquidation Threshold either too low (less than one) or too high. This vulnerability could lead to increased false positive cases where more accounts would be deemed insolvent and trigger unnecessary liquidations, as well as reduced sensitivity to actual insolvency risks if the threshold is set too high.

Impact

A wrongly set Health Factor Liquidation Threshold value could trigger unnecessary liquidations of borrow positions.

Recommendation

We recommend implementing input validation and range checking for the Health Factor Liquidation Threshold value in the `set-health-factor-liquidation-treshold` function. This can be achieved by adding a simple check to ensure that the input value is within a valid range (e.g., greater than or equal to one).

ISSUE #2

Overly Complex borrow Function

None Fixed

Location

pool-borrow-v1-2.clar#L187

Description

The borrow function in this smart contract contains multiple nested conditions, making it difficult to read and understand. This complexity can lead to errors and bugs that may be hard to detect or fix.

Impact

No security impact.

Recommendation

We recommend splitting the `borrow` function into smaller, more manageable functions (e.g., `borrow-with-isolated` and `borrow`) to improve readability and maintainability.

ISSUE#3

Unnecessary Gas Consumption

None Fixed

Location

pool-borrow-v1-2.clar#L169

pool-0-reserve.clar#L1663

Description

We identified an opportunity to optimize gas consumption by simplifying the comparison logic. Specifically, the following code snippet can be optimized to remove unnecessary arithmetic operations:

```
(if (is-eq (- current-balance amount-to-redeem) u0))
```

This code checks if the difference between `current-balance` and `amount-to-redeem` is equal to 0. However, instead of subtracting these values and then checking for equality, it's more efficient to use the `is-eq` operator directly on the two values.

By removing the unnecessary subtraction, we can reduce gas consumption and improve contract performance.

Impact

If left unchecked, this issue may lead to increased gas consumption and slower contract execution times. This could result in higher transaction costs for users.

Recommendation

We recommend replacing the code snippet with a more efficient implementation using `is-eq: (if (is-eq current-balance, amount-to-redeem))`

Section_5.0

Appendix A

Severity Rating Definitions

At Thesis Defense, we utilize the Immunefi Vulnerability Severity Classification System - v2.3.

Critical

- Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results
- Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield
- Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties
- Permanent freezing of funds
- Permanent freezing of NFTs
- Unauthorized minting of NFTs
- Predictable or manipulable RNG that results in abuse of the principal or NFT
- Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content)
- Protocol insolvency

High

- Theft of unclaimed yield
- Theft of unclaimed royalties
- Permanent freezing of unclaimed yield
- Permanent freezing of unclaimed royalties
- Temporary freezing of funds
- Temporary freezing NFTs

Medium

- Smart contract unable to operate due to lack of token funds
- Block stuffing
- Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)
- Theft of gas
- Unbounded gas consumption

Low

- Contract fails to deliver promised returns, but doesn't lose value

None

- We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.

Section_6.0

Appendix B

Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.