Security Audit Report

Hermetica Labs:

Hermetica Smart Contracts

Final Audit Report: April 4, 2024

thesisdefense
defense@thesis.co

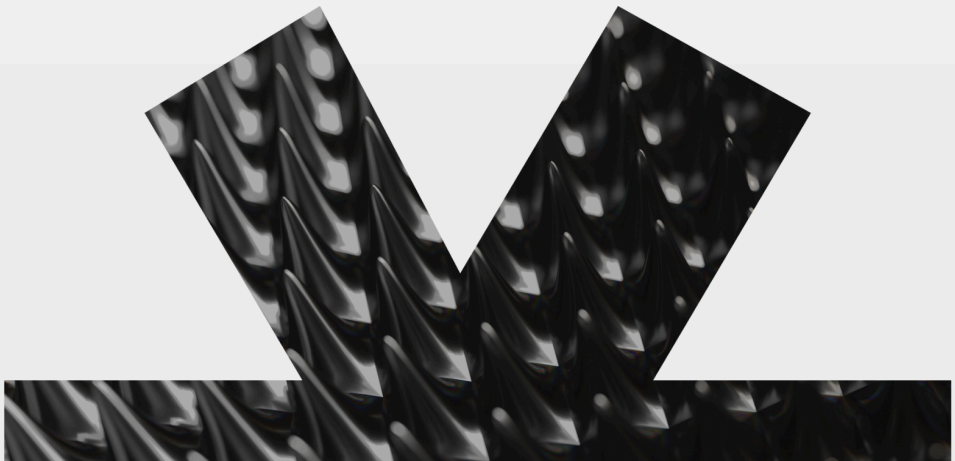

Table of Contents

About Thesis Defense

Thesis Defense serves as the auditing services arm within Thesis, Inc., the venture studio behind tBTC, Fold, Taho, Etcher, and Embody. Our team of senior security and cryptography auditors has extensive security experience in the decentralized technology space. In addition, the Thesis Defense team has a demonstrated track record in a variety of languages and technologies, including, but not limited to, smart contracts, cryptographic protocols including zk-cryptography, dApps including wallets and browser extensions, and bridges. Thesis Defense has extensive experience conducting security audits across a number of ecosystems, including, but not limited to, Ethereum, Zcash, Stacks, Mina, Polygon, Filecoin, and Bitcoin.

Thesis Defense will employ the Thesis Defense Audit Approach and Audit Process to the above in-scope service. In the event that certain processes and methodologies are not applicable to the aforementioned 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.

Scope

Overview

Thesis Defense conducted a manual code review of the Hermetica smart contracts.

Project Team

- **Mukesh Jaiswal,** Security Auditor
- **El-Hassan Wanas,** Senior Security Auditor
- **Bashir Abu-Amr,** Senior Technical Writer

Schedule

- **Code Review:** March 11 - 15, 2024
- **Audit Report Delivery:** March 15, 2024
- **Final Audit Report Delivery:** April 4, 2024

Code

- **Audit**

    - **Repository:** https://github.com/hermetica-fi/hermetica-contracts
    - **Hash:** `bca6b9db31a827a912e6dd14b6facd84d09e0714`
- **Verification**
    - **Repository:** https://github.com/hermetica-fi/hermetica-contracts
    - **Hash:** `a9f12f6c67feaa6dad196c1d240453f7b4af7f96`
- **Reference Repository:** https://github.com/hermetica-fi/hermetica-contracts-dev

Project Documentation

- **Technical Documentation:** https://docs.hermetica.fi/

Findings

The Hermetica smart contracts represent the on-chain component of the Hermetica options vault protocol for BTC and Stacks sip-010 tokens. The smart contracts are implemented in Clarity and are intended to be deployed on the Stacks blockchain.

The Stacks blockchain, which leverages Bitcoin's security and decentralization, is designed with robust security mechanisms to protect its network and users. It integrates Bitcoin's Proof of Work (PoW) consensus mechanism with its own Proof of Transfer (PoX) consensus mechanism, allowing for flexible smart contracts while leveraging the robustness of the bitcoin blockchain.

Threat Model

For this review, our team considered a threat model whereby external components to the smart contracts are untrusted but function as intended. These components include any user interface that enables interaction with the protocol, any off chain components that are an integral part of the system, and any third party dependencies or services that are necessary for the protocol to function as intended. Furthermore, we considered the governance of the protocol not malicious.

In our investigation and threat model, we considered threats from malicious traders, sellers and buyers, and observers. This facilitated our ability to consider and explore various attack vectors during our review of the system design and coded implementation.

Security by Design

Hermetica's system design is robust and the security of the protocol has been considered and prioritized. This is demonstrated by the implementation of emergency mechanisms to mitigate the effects of a vulnerability. Specifically, there is a set proportion of assets controlled by the

vault that can be transferred within a one week time window, safeguarding deposits in the vault.

Secure Implementation

The Hermetica smart contracts implementation is well organized and adheres to best practices. We performed a manual code review of the implementation and, although we did not identify any critical security vulnerabilities in the implementation, we did identify an implementation error that would lead to unintended behavior (Issue B). We also identified a public function that is callable by any user which would allow a deposit to be made without a corresponding claim leading to a potential loss of funds.. However this issue would not affect the security of the protocol itself since the function is intended to be used internally by the `core` smart contract (Issue A). Additionally, we identified incorrect validation when registering a trade, which would block registrations if the previous epoch was settled longer than 72 blocks in the past. This would require admin intervention with the `force-settle` function to resolve (Issue C).

Use of Dependencies

Hermetica utilizes the Pyth oracle as a timestamp source as well as a price oracle for assets. Specifically, the Hermetica smart contracts receive data from the Pyth oracle through the use of the Wormhole bridge. The Pyth oracle is sufficiently decentralized, and we did not identify any issues in the use of this dependency. Although there are general challenges associated with cross-chain bridges such as the Wormhole bridge, we did not identify specific weaknesses in the Wormhole bridge that are relevant to the context of the Hermetica smart contracts. As a result, we found that Hermetica's use of dependencies adheres to security best practices.

Tests

The in-scope repository for this security audit did not contain any tests. However, the Hermetica team provided us with their `dev` repository (referenced above), which contains tests that provide 100% coverage of the implementation covering every possible branch of execution, in adherence with security best practice and industry standards for implementing a test suite with sufficient coverage.

Project Documentation

The Hermetica smart contracts are comprehensively documented in accordance with best practices. The documentation included comprehensive technical documentation that detailed critical functionality of the smart contracts, and was supplemented by an architectural diagram. We commend the Hermetica team for providing sufficient documentation, aiding both developers and auditors familiarizing themselves with Hermetica protocol.

Similar to the tests, the codebase is well commented in the `dev` repository. We recommend publishing the code comments (Suggestion 2).

Issues and Suggestions

Issues	Status
Issue A: Lack of Access Control	Fixed
Issue B: Incorrectly Set Equality Check	Fixed
Issue C: Incorrect Validation in Trade Registration	Fixed
Issue D: Incorrect Validation in `start-new-epoch`	Fixed
Issue E: A Zero Value of total-tokens-active Prevents Calculation of get-underlying-per-token	Fixed

Suggestions	Status
Suggestion 1: Implement Input Validation for Setter Functions	Fixed
Suggestion 2: Publish Code Comments	Fixed

Issues

Issue A: Lack of Access Control **[Fixed]**

Location

vault-v1.clar#L277

Description

The `deposit-funds` function is designed to move the underlying SIP-010 token from the depositor to the `vault` smart contract. Ideally, the function should be accessed through the `core` smart contract for payment processing by counterparties and delinquent counterparties after calculating profits and losses. However, the function lacks access control.

Impact

If any depositor directly calls this function to deposit the underlying SIP-010 tokens, the contract won't record any related data nor will it update the counterparty's status for the sender by settling the corresponding debt. This implies that any transferred amount will end up locked in the contract without being properly processed or tracked and the user would consequently have to resend the deposit via `make-payment` in order for their payment to be counted towards a trade.

Remediation

We recommend implementing an appropriate access control mechanism for the function.

Verification Status

Fixed. The Hermetica team implemented validation confirming the caller of the function is the `core` smart contract.

Issue B: Incorrectly Set Equality Check **[Fixed]**

Location

pnl-calculator-v1.clar#L160

Description

A knock-in option becomes active when the price reaches either the barrier up or barrier down price levels. However, in the current implementation, when calculating the profit and loss (PNL), the knock-in option is considered active only when the price exceeds the barrier up or barrier down levels.

Impact

If the option type is a knock-in option (u3), the implication is that the profit and loss (PNL) won't be computed even if the option becomes active.

Remediation

We recommend that the barrier price is included when checking the price by also checking for equality.

Verification Status

Fixed. The Hermetica team has added an equality check on barrier-up and barrier-down in the case of a knock-in option type.

Issue C: Incorrect Validation in Trade Registration **[Fixed]**

Location

core-v1.clar#L421-L423

Description

In `register-trade` there is an assertion that the current block height is at most the previous epoch's settled block plus the registration window (72 blocks). If the previous block was settled longer than that time window, all calls to `register-trade` would fail for the current epoch forcing the use of `force-settle` in order to unblock the contract.

Impact

No trades can be registered for the current epoch due to the settlement block height of the previous block being too far in the past.

Remediation

We recommend revising this assertion so that late settling of an epoch does not affect the `register-trade` function.

Verification Status

Fixed. The Hermetica team has changed the assertion to check that trades are being registered within 72 blocks (12 hours on average) of the current epoch start date.

Issue D: Incorrect Validation in `start-new-epoch` **[Fixed]**

Location

core-v1.clar#L362

Description

The Hermetica team identified an issue whereby an assert in `start-new-epoch` blocks starting a new epoch if trades have been made in the current epoch.

Impact

This would block trading after having traded in the first epoch. This issue would not affect assets in the vault.

Remediation

The assertion must only be made in the case of overwriting an unsettled epoch, which ensures that an unsettled epoch with confirmed trades needs to be settled before a new epoch is started.

Verification Status

Fixed. The Hermetica team has updated the function implementation according to the remediation.

Issue E: A Zero Value of total-tokens-active Prevents Calculation of get-underlying-per-token [Fixed]

Location

vault-v1.clar#L276

Description

The Hermetica team identified an issue where `total-token-active` becomes zero in the edge case where all funds are withdrawn from the vault, which leads to `get-underlying-per-token` to fail due to a `DivisionByZero` error.

Impact

Since `get-underlying-per-token` is used in the claim activation functions this error prevents future deposits and withdrawals to be activated.

Remediation

While calculating the value for `underlying-per-token`, the function must confirm that neither `current-total-token-active` nor `current-total-underlying-active` is zero.

Verification Status

Fixed. The Hermetica team has updated the function implementation according to the remediation.

Suggestions

Suggestion 1: Implement Input Validation for Setter Functions [Fixed]

Location

hq-v1.clar#L327
hq-v1.clar#L332
hq-v1.clar#L285
hq-v1.clar#L296
hq-v1.clar#L306

Description

The `HQ` smart contract defines all protocol settings with its corresponding getter and setter functions as well as defines all of the protocol's roles. Setter functions which set the values for security critical parameters including `unit-size`, `registration-window`, `confirmation-window`, `payment-window`, `vault-capacity`, `min-deposit-amount`, and others, lack input validation which can lead to unintended behavior.

For example, if the value of `vault-capacity` is set 0, then the user will not be able to deposit their underlying token.

Remediation

We recommend implementing checks that validate the input values and ensure that they are in the intended range.

Verification Status

Fixed. The Hermetica team has implemented the check in one instance(`unit-size`), but stated that there are no security implications resulting from a lack of validation on the rest of the referenced parameters. Our team agrees with this assessment.

Suggestion 2: Publish Code Comments **[Fixed]**

Description

The repository in scope does not include any code comments. The Hermetica team has written comprehensive code comments in their dev repository, which were made available to us for this security audit. We found the code comments to be comprehensive, accurate, and helpful in understanding the intended functionality of security critical components.

Remediation

We recommend publishing the code comments to enable those who interact with the system to build a deeper understanding of the functionality of the system, which improves security.

Verification Status

Fixed. The Hermetica team has published code comments for all of the public functions in the implementation.