Untangled Finance Vault Code Audit

Review And Security Analysis Report

Prepared by: Ausec

Untangled.Finance

December 13, 2024

Version: 1.0

Table of Contents

1 Project Overview

Untangled Vault’s Core is built on:

- **ERC4626 Vault**: A tokenized vault standard enabling seamless deposit, withdrawal, and share management.
- **Treasury (Safe)**: A multisig contract powered by Safe, responsible for:
  - Holding and safeguarding the vault's assets.
  - Acting as the vault's manager with full authority over enabling, disabling, or upgrading modules.

Untangled Vault supports **5** customizable module types:

- **Withdraw Module**: Implements withdrawal logic, such as asynchronous withdrawals, to manage liquidity and asset conversions.
- **Valuation Module**: Calculates asset values under treasury management.
- **Authentication Module**: Provides access control for deposits, withdrawals, and share transfers.
- **Fee Module**: Offers flexible fee structures, including AUM-based and performance-based calculations.
- **Crosschain Module**: Enables treasury and vault operations across multiple blockchains, supporting crosschain deposits.

2 Executive Summary

2.1 Approach

Auditors conducted code auditing by analyzing the smart contract’s functionality and performing all necessary checks with the goal of identifying whether the smart contract is vulnerable to known attacks. We performed the following approaches to identify vulnerabilities:

1. Static analysis by using Slither

- Auditors used the Slither tool to conduct a static analysis. Slither examines the code structure, syntax, and specific patterns that may indicate potential vulnerabilities. This step provides an overview of possible weak points in the code.
- After running Slither, auditors manually reviewed its findings to gain deeper insights into the codebase. This step allowed auditors to validate or refute the tool's automated findings, leveraging their knowledge to detect additional vulnerabilities or security issues that automated tools may overlook.

2. Manual codebase assessment

- Auditors performed a detailed, line-by-line review of the source code to gain a comprehensive understanding of the security model and discover potential issues.
- Auditors also reviewed the project's technical whitepaper and supplementary documentation, comparing the functional requirements with the code implementation to detect any inconsistencies or deviations from the intended functionality.

2.2 Scope

- Our audit of the Untangled Protocol Vault only covers the contract in this repository at commit hash `c97836d82030c18c19b5538097882882667dff98`.

- **8** files audited:

ID	Repo	File
VAU	untangled.finance/untangled-vault	`contracts/Vault.sol`
VAF	untangled.finance/untangled-vault	`contracts/VaultFactory.sol`
CRH	untangled.finance/untangled-vault	`contracts/CrosschainHook.sol`
ASW	untangled.finance/untangled-vault	`contracts/modules/AsyncWithdraw.sol`
AUT	untangled.finance/untangled-vault	`contracts/modules/Auth.sol`
CRC	untangled.finance/untangled-vault	`contracts/modules/Crosschain.sol`
FEE	untangled.finance/untangled-vault	`contracts/modules/Fee.sol`
VAL	untangled.finance/untangled-vault	`contracts/modules/Valuation.sol`

- The changes implemented by the Untangle team after acknowledging this finding are reflected in the commit: `7a354e739f453ae37b096134d834bcb914ea4f3d`

2.3 Summary of Findings

In the course of this audit **1 Low** and **4 Info** vulnerabilities were identified:

5
Total Findings

Figure 1 - Distribution of identified vulnerabilities

Below is a high-level overview of each finding identified during auditing. These findings are covered in depth in the Technical Findings Details section of this report.

#	Severity Level	Finding Name	Status	Page
1	Low	[ASW-01] Insufficient Validation of Shares During Withdrawal Request Fulfillment	Resolved	5
2	Info	[ASW-02] Inefficient Loop Implementation	Resolved	6
3	Info	[CRC-01] Potential Inaccuracy in Oracle Update Due to Asynchronous Hook Transactions	Acknowledged	7
4	Info	[VAL-01] Multichain can be Incompatibility Due to AssetInfo Storage Key	Acknowledged	8
5	Info	[VAL-02] Redundant Asset Update Function	Acknowledged	9

3 Technical Findings Details

1. [ASW-01] Insufficient Validation of Shares During Withdrawal Request Fulfillment

Severity	Location	Status
● Low	contracts/modules/AsyncWithdraw.sol: 188	Resolved

Finding Details

```
    uint256 constant ONE_HUNDRED_PERCENT = 10000;
```

```
    epochInfo[epochId].fulfillment =
        (shares * ONE_HUNDRED_PERCENT) / totalSharesWithdraw;
```

It is necessary to verify if the share fulfillment exceeds `0.01%`; otherwise, the fulfillment rate will be zero.

Recommendations

Implement Validation for Shares Exceeding `0.01%` of Total Requests Per Epoch

Results

The team heeded the advice and resolved this issue in commit `7a354e739f453ae37b096134d834bcb914ea4f3d`

2. [ASW-02] Inefficient Loop Implementation

Severity	Location	Status
● Info	contracts/modules/AsyncWithdraw.sol: 152-162	Resolved

Finding Details

```
    while (lastRequestId < epochId) {
        (uint256 assets, uint256 shares) = _claimableRedeemRequest(
            receiver,
            lastRequestId
        );

        totalAssets += assets;
        totalShares += shares;
        redeemRequests[receiver].shares -= shares;

        lastRequestId++;
    }
```

This loop can be terminated when the `shares` value is `0`, as the logic dictates that users can only submit new requests after fully claiming their previous ones, and each request within an epoch must have a non-zero value.

Recommendations

Implement a break statement when the `shares` value equals `0`.

Results

The team heeded the advice and resolved this issue in commit `7a354e739f453ae37b096134d834bcb914ea4f3d`

3. [CRC-01] Potential Inaccuracy in Oracle Update Due to Asynchronous Hook Transactions

Severity	Location	Status
● Info	contracts/modules/Crosschain.sol: 94-136	Acknowledged

Finding Details

The Oracle update process in the Hook may produce incorrect values due to asynchronous execution, specifically when updates are not synchronized with the Oracle. The issue arises because Axelar does not guarantee transaction order preservation on the target chain. When sequential transactions are sent from one chain to another, Axelar processes each independently, meaning the transaction order on the source chain does not necessarily match the order on the target chain.

1. **At time `n`**: User transfers funds to the Hook contract, increasing the SafeWallet balance on source-chain and initiating a call through Axelar to the Vault contract on target-chain.
2. **At time `n+1`**: The Admin triggers `updateAsset()`, calling Axelar to update the Oracle.
3. **At time `n+2`**: The Oracle updates the price first. At this point, the `assetInfo.balance` is still accurate.
4. **At time `n+3`**: The Hook updates the balance by adding the asset. However, due to the lack of guaranteed transaction order on Axelar, this sequence may be incorrect, resulting in an inaccurate `assetInfo.balance` value.

This discrepancy is caused by Axelar's independent processing of each transaction, leading to a potential mismatch between price updates and balance adjustments.

Results

The Untangled team acknowledged the issue and decided not to implement any changes, as it can be mitigated through proper control and management from the off-chain system.

4. [VAL-01] Multichain can be Incompatibility Due to AssetInfo Storage Key

Severity	Location	Status
● Info	contracts/modules/Valuation.sol: 16	Acknowledged

Finding Details

The current implementation of `assetInfo` stores token information using the token's address as the mapping key. While this approach works on a single chain, it creates issues for multichain compatibility when the same token address exists across multiple chains. For instance, tokens like **UNI** (from Uniswap) have identical addresses on different chains. This design limitation prevents the Vault from functioning correctly in a multichain environment.

```
    struct AssetInfo {
        uint256 balance;
        uint256 price;
        uint chainId;
        uint8 decimals;
        address oracle;
    }

    mapping(address => AssetInfo) public assetInfo;
```

Using only the token address as the key makes it impossible to differentiate between tokens with the same address on different chains. This constraint undermines the protocol's ability to operate across multiple chains, as each asset's information cannot be uniquely identified by address alone.

Recommendations

Implement key as combine of asset address and chain id.

Results

The Untangled team acknowledged the issue but determined that the likelihood of tokens with identical addresses across multiple chains being used for investment is low, and therefore decided not to implement any changes at this time.

5. [VAL-02] Redundant Asset Update Function

Severity	Location	Status
● Info	contracts/modules/Valuation.sol: 94	Acknowledged

Finding Details

The `updateAsset()` function is responsible for updating the price and balance of invested tokens within the system. Currently, this function is being called after several actions, which introduces redundancy without contributing to the accuracy of the Vault's value. Specifically, `updateAsset()` is triggered after the following actions:

- `deposit()`
- `withdraw()`
- `open()`
- `addAsset()`

Updating asset values after these actions is unnecessary because the Vault's value needs to be accurately calculated before executing these actions. Performing the update post-action only impacts the display values in the user interface and does not serve any purpose for core Vault operations.

Recommendations

Instead of calling `updateAsset()` redundantly, consider the following optimizations:

- **Pre-Action Update**: Calculate asset values before each action where an accurate Vault value is required.
- **On-Demand Calculation in View Functions**: For UI display purposes, integrate asset value calculations directly within view functions, avoiding redundant updates in the main contract flow.

This change would streamline the process, reducing unnecessary calls to `updateAsset()` and enhancing system efficiency without compromising the accuracy or functionality of the Vault's value calculations.

Results

The Untangled team acknowledged that they want to control the timing of price updates from the off-chain server due to certain special cases.

A Appendix

A.1 Finding Severities

Each finding has been assigned a severity rating of **Critical**, **High**, **Medium**, **Low**, or **Info**. The rating is based on an assessment of the priority with which each finding should be addressed and the potential impact on the contract.

Severity	Description
Critical	Critical vulnerabilities are flaws that could be easily exploited and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities often require specific conditions to exploit or have a narrower impact but can still result in user fund loss or manipulation.
Medium	Medium vulnerabilities are generally confined to state manipulation and typically do not result in asset loss. They may involve contradictions, requirement violations, or significant deviations from best practices.
Low	This rating applies to all other issues that might have a security impact. These vulnerabilities are considered to require unlikely conditions for exploitation or would result in minimal consequences if exploited.
Info	Major deviations from best practices or significant gas inefficiencies do not notably impact code execution or the security score but can affect the code quality score.

A.2 Static Analysis

missing-zero-address-validation

- `CrosschainHook.constructor(address,address,address,address,address,address,bool,string)._safeWallet (contracts/CrosschainHook.sol#32)` lacks a zero-check on:
  - `safeWallet = _safeWallet (contracts/CrosschainHook.sol#41)`
- `CrosschainHook.constructor(address,address,address,address,address,address,bool,string)._asset (contracts/CrosschainHook.sol#33)` lacks a zero-check on:
  - `asset = _asset (contracts/CrosschainHook.sol#42)`
- `CrosschainHook.constructor(address,address,address,address,address,address,bool,string)._vault (contracts/CrosschainHook.sol#34)` lacks a zero-check on:
  - `vault = _vault (contracts/CrosschainHook.sol#43)`
- `CrosschainHook.constructor(address,address,address,address,address,address,bool,string)._authModule (contracts/CrosschainHook.sol#35)` lacks a zero-check on:
  - `authModule = _authModule (contracts/CrosschainHook.sol#44)`
- `Vault.constructor(address,string,string,address)._safeWalet (contracts/Vault.sol#37)` lacks a zero-check on:
  - `treasury = _safeWalet (contracts/Vault.sol#39)`
- `AsyncWithdraw.constructor(address)._vault (contracts/modules/AsyncWithdraw.sol#30)` lacks a zero-check on:
  - `vault = _vault (contracts/modules/AsyncWithdraw.sol#31)`
- `Auth.constructor(address)._go (contracts/modules/Auth.sol#12)` lacks a zero-check on:
  - `go = _go (contracts/modules/Auth.sol#13)`
- `Crosschain.constructor(address,address,address)._vault (contracts/modules/Crosschain.sol#24)` lacks a zero-check on:
  - `vault = _vault (contracts/modules/Crosschain.sol#25)`
- `Fee.constructor(address)._vault (contracts/modules/Fee.sol#21)` lacks a zero-check on:
  - `vault = _vault (contracts/modules/Fee.sol#22)`
- `Valuation.constructor(address)._vault (contracts/modules/Valuation.sol#20)` lacks a zero-check on:
  - `vault = _vault (contracts/modules/Valuation.sol#21)`

boolean-equality

- `AsyncWithdraw.whenNotClosed()` (contracts/modules/AsyncWithdraw.sol#44-47) compares to a boolean constant:
  - `require(bool,string)(isVaultOpen == true,AsyncWithdraw: Vault close)` (contracts/modules/AsyncWithdraw.sol#45)

state-variables-that-could-be-declared-immutable

- `Vault.treasury` (contracts/Vault.sol#20) should be immutable

cache-array-length

- Loop condition `i < assets.length` (contracts/modules/Valuation.sol#35) should use cached array length instead of referencing `length` member of the storage array.