Hypersea Security Analysis

by Pessimistic

This report is public

January 19, 2024

Abstract

In this report, we consider the security of smart contracts of Hypersea project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Hypersea smart contracts. We described the audit process in the section below.

The initial audit showed one issue of medium severity: Price age is not checked. Also, several low-severity issues were found.

After the initial audit, the developers provided us with a new version of the code, in which they either fixed or commented on all issues.

The overall code quality is good. The project is well-structured and well-maintained. However, due to the complexity of the code, its deployment and usage on the L1 network could be expensive if the developers decide to do it. Note that the target network is Arbitrum.

General recommendations

We do not have any further recommendations.

Project overview

Project description

For the audit, we were provided with Hypersea project on a private GitHub repository, commit 34dd54a5bed06aa4d98e6f2186fef952f33cae41.

The scope of the audit included the whole repository except the following files and directories:

- **src/modules/pool** directory;
- **src/modules/governance/PoolGovernor.sol**;
- **src/modules/governance/PoolTimelock.sol**;
- **src/modules/vault/Vault.sol**;
- **src/types/PoolReserves.sol**;
- **src/types/CurvePoints.sol**.

The documentation for the project included whitepaper and governance doc.

All 454 tests pass successfully. The code coverage is 63.2%.

The total LOC of audited sources is 2297.

Codebase update

After the initial audit, the developers provided us with a new version of the code, commit 447e9a82b1b2f8f4873bea185db3d647e1f78809. This update includes fixes for almost all issues. Additionally, the developers commented on the issues that had not been fixed.

473 tests out of 473 pass successfully. The code coverage is 75.34%.

Audit process

We started the audit on December 20, 2023, and finished on December 29, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Possible price manipulation attacks;
- Underflow/overflow of the mathematical expressions;
- Potential problems with the price oracles;
- Arbitrum-specific differences in the EVM do not affect contract logic.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts.

We ran tests and calculated the code coverage.

We combined in a private MD file all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On January 10, 2024, the developers provided us with an updated version of the code. In this update, they fixed almost all issues and commented on the remaining. Moreover, they implemented additional tests.

We conducted a review of the updated codebase and re-run automated tools.

After the review, we prepared the public report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Price age is not checked (fixed)

In **CCO.sol**, the price age is not verified during the valuation update. This oversight could potentially lead to the use of outdated pricing, resulting in valuations being assessed with old prices. While minor discrepancies might not significantly affect outcomes, specific scenarios like Chainlink's transition from an old to a new feed contract for a particular pair could easily go unnoticed by the **CCO**. For unpredictable reasons, these changes might not be detected in time. We strongly advise implementing a price age verification process, including a grace period (even a generous one, e.g., 1-2 days), to ensure that notably outdated data is not used.

The issue has been fixed and is not present in the latest version of the code.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Code quality (fixed)

In **Asset.sol**, consider marking assembly blocks as `memory-safe`.

The issue has been fixed and is not present in the latest version of the code.

L02. Duplicated code (fixed)

In several instances, the codebase contains duplicate code, with two parts that are slightly different yet share common logic. By consolidating these parts, there is a possibility of reducing code size and enhancing readability. We recommend reviewing the duplicate checks for the price path in **CCO.sol**, particularly at lines 757 and 759-761. Similarly, the price calculation logic in **PriceSources.sol** at lines 178-196 and 242-264.

The issue has been fixed and is not present in the latest version of the code.

L03. Gas optimization (fixed)

In **Treasury.sol**, there is no risk of underflow at lines 39 and 41. To optimize gas usage, we recommend wrapping these lines with an `unchecked` block.

The issue has been fixed and is not present in the latest version of the code.

L04. Pools are always in allowlist (fixed)

In **PriceSourceAllowlist.sol**, pools are automatically whitelisted, which helps reduce gas costs and eliminates the need to manually add pools to the whitelist. However, this approach could pose a risk if a token in the pool becomes compromised. To mitigate this risk, we recommend implementing a blacklist mechanism for the pools.

The issue has been fixed and is not present in the latest version of the code.

L05. Possible DoS (commented)

In **CCO.sol**, the `updateValuation` function utilizes the prices of all whitelisted treasury assets. If the number of these assets is high, the execution cost of the function could significantly increase, potentially leading to a situation where it exceeds the maximum block size.

> *Comment from the developers: We have a list for treasury valuation assets which is controlled by government, it is government responsibility not to expand valuation asset list too much.*

Notes

N01. Fee on transfer and rebalance tokens are not supported (commented)

The **Treasury** contract, in its current implementation, does not support fee-on-transfer tokens and rebalancing tokens. If this limitation is intentional, we recommend explicitly clarifying it in the documentation.

> *Comment from the developers: Currently, we do not have plans to support exclusive fee on transfer tokens. This may change in the future since the Treasury is upgradeable.*

N02. Possible UX improvement (fixed)

In the **VestingWallet** contract, consider adding a delegation option for the owner of the vesting. This addition would be beneficial since the owner retains voting power even when not all tokens are vested. Enabling delegation can provide more flexibility in how voting power is exercised, reflecting the vested ownership status more accurately.

The issue has been fixed and is not present in the latest version of the code.

N03. Possible UX improvement (commented)

In the **CCO** contract, the user's purchase transaction currently reverts if the price is stale. To enhance user experience and functionality, consider adding an option for an automatic valuation update within the purchase transaction.

> *Comment from the developers: We will call `updateValuation()` on some regular basis (for example, every 4 hours). So it should not be the case that the price is stale.*

N04. Price Impact in CCO (commented)

In the **CCO** contract, there are several scenarios where the selling price could change, beyond the common scenario of swaps. These scenarios are possible due to the fact that the floor price is not fixed:

- The price may change if someone calls `updateValuation` function after a swap. This occurs as the `yMin` value increases;
- Price volatility of the asset in the treasury can lead to changes;
- Low liquidity in the pool used for the asset can affect the price;
- Adding or removing an asset and its price source can also impact the selling price.

> *Comment from the developers: We suppose that the effect of this price impact is not in favor of any party. When the valuation of the Treasury decreases the price impact is in favor of the user (floor price will go down). When the valuation of the Treasury increases the price impact is in favor of the Treasury (floor price will go up). That's why* `updateValuation()` *is* `public`*. The user can update the valuation before performing their operation to get a fair price.*

N05. Test price calculation with weird tokens (fixed)

We recommend testing scenarios where tokens have unconventional decimal places (fewer than 6 or more than 18) to ensure that the precision is maintained for the price calculation.

The issue has been fixed and is not present in the latest version of the code.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Pavel Kondratenkov, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Irina Vikhareva, Project Manager
Konstantin Zherebtsov, Business Development Lead
Alexander Seleznev, Founder

January 19, 2024