Security Audit Report

Hourglass Bridge

PREPARED FOR:

Hourglass Foundation

ARCADIA CONTACT INFO
Email: audits@arcadiamgroup.com
Telegram: https://t.me/thearcadiagroup

Table of Contents

Executive Summary

Introduction

Hourglass Foundation engaged Arcadia to perform a security audit of their smart contracts within the hourglass-platform repository within the Pitch Foundation organization. The audit has been made by reviewing the repository at multiple commit hashes as the code changed multiple times during the audit.

Review Team

- Tuan “Anhnt” Nguyen - Security Researcher and Engineer
- Joel Farris - Project Manager

Project Background

Hourglass Foundation is a protocol that allows for the trading of semi-fungible time-locked digital assets. Hourglass builds off of gauges of various incentivized locked stake systems like Convex’s gauges.

Hourglass has a centralized bridge to transfer claims between L1s and L2s which is a pivotal part of their platform.

Coverage

For this audit, we performed research, test coverage, investigation, and review of Hourglass Vesting and Hourglass CustodianV2 followed by issue reporting, along with mitigation and remediation instructions as outlined in this report. The following code repositories, files, and/or libraries are considered in scope for the review.

Contracts
pitch-vesting/src/TimelockVesting.sol
pitch-vesting/src/TwoStepOwnable.sol
pitch-vesting/src/TwoStepOwnableinterface.sol
hourglass-platform/src/ethereum/HourglassCustodianV2.sol

Methodology

Arcadia completed this security review using various methods, primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

The followings are the steps we have performed while auditing the smart contracts:

- Investigating the project and its technical architecture overview through its documentation
- Understanding the overview of the smart contracts, the functions of the contracts, the inheritance, and how the contracts interface with each others thanks to the graph created by Solidity Visual Developer
- Manual smart contract audit:
    - Review the code to find any issue that could be exploited by known attacks listed by Consensys
    - Identifying which existing projects the smart contracts are built upon and what are the known vulnerabilities and remediations to the existing projects
    - Line-by-line manual review of the code to find any algorithmic and arithmetic related vulnerabilities compared to what should be done based on the project's documentation
    - Find any potential code that could be refactored to save gas
    - Run through the unit-tests and test-coverage if exists
- Static Analysis:
    - Scanning for vulnerabilities in the smart contracts using Static Code Analysis Software
    - Making a static analysis of the smart contracts using Slither
- Fuzzing
    - Arcadia assisted in writing and ensuring full coverage of fuzzing implementations
    - Additional review: a follow-up review is done when the smart contracts have any new update. The follow-up is done by reviewing all changes compared to the audited commit revision and its impact to the existing source code and found issue

Summary

There were 4 issues found, 0 of which were deemed to be ‘critical’, and 0 of which were rated as ‘high’. At the end of these issues were found throughout the review of a rapidly changing codebase and not a final static point in time.

Severity Rating	Number of Original Occurrences	Number of Remaining Occurrences
CRITICAL	0	0
HIGH	0	0
MEDIUM	1	0
LOW	1	0
INFORMATIONAL	2	1

Findings in Manual Audit

(HG-1) `requestRedemption` - Absence of Input Validation Compromises Security.

Status

Resolved - The team eliminated the cumbersome two-step redemption process and implemented thorough validation in the new code..

Risk Level

Severity: Medium

Code Segment

```
function requestRedemption(
      uint256[] calldata _assetIds,
      uint256[] calldata _maturities,
      uint256[] calldata _amounts,
      address _redeemTo
  ) public {
```

Description

When performing the *`requestRedemption`* call, it is crucial to validate the lengths of *`_assetIds`*, *`_maturities`*, and *`_amounts`*. Failure to do so can result in unintended consequences, including transaction reversion in *`redeemReceipts.`*

Code Location

```
hourglass-platform/src/ethereum/HourglassCustodianV2.sol
```

Proof of Concept

–

Recommendation

Implement the input validation in *`requestRedemption`*

Findings in Manual Audit

(HG-2) `cancelRedemption` - should remove mapping instead of marking used.

Status

Resolved - This function has been removed in the new code.

Risk Level

Severity: Low

Code Segment

```
function cancelRedemption() external {
    …
    // set the withdrawal status to used to prevent re-cancellation & redemption
    userToWithdrawTS[msg.sender].isUsed = true;

}
```

Description

Using the same flag *`isUsed`* to mark both canceled redemptions and used redemptions could lead to ambiguity in status tracking

Code Location

```
hourglass-platform/src/ethereum/HourglassCustodianV2.sol
```

Proof of Concept

–

Recommendation

To avoid ambiguous status tracking, either delete the canceled mapping or utilize an alternative flag.

(HG-3) `redeemReceipts` - *nonReentrant* is redundant

Status

Resolved

Risk Level

Severity: Informational

Code Segment

```
function redeemReceipts(
       address redeemFor
   ) external nonReentrant {
       // check that withdrawals are not paused
       if (isPaused) revert IsPaused();
       // pull out the withdrawal details
       Withdrawal memory thisWithdrawal = userToWithdrawTS[redeemFor];
       // check that the user is allowed to withdraw
       if (
           thisWithdrawal.allowedTimestamp < block.timestamp
           ||
           thisWithdrawal.allowedTimestamp == 0
       ) revert WithdrawNotYetAuthorized();
       if (thisWithdrawal.isUsed) revert WithdrawalUsed();
       // use up their withdrawal request
       userToWithdrawTS[redeemFor].isUsed = true;
```

Description

Flag *`userToWithdrawTS[redeemFor].isUsed`* is enough to avoid reentrancy.

Code Location

`hourglass-platform/src/ethereum/HourglassCustodianV2.sol`

Recommendation

Remove *nonReentrant* modifier.

(HG-4) addBeneficiary - Remove obvious comments.

Status

Unresolved

Risk Level

Severity: Informational

Code Segment

```
function addBeneficiary(
      address _beneficiary,
      uint256 _amount,
      uint256 _startTimestamp,
      uint256 _cliffDate,
      uint256 _durationInSeconds
) external onlyOwner {
      // cannot vest zero tokens
      require(_amount != 0, "!amount");
      // cannot vest to zero address
      require(_beneficiary != address(0), "!_beneficiary");
      // cannot be zero (but can be past)
      require(_startTimestamp != 0, "!startTimestamp");
      // cannot be zero
      require(_durationInSeconds != 0, "!duration");
      // cliff must be after start time
      require(
           _cliffDate > _startTimestamp,
          "!cliffStart"
      );
      // cliff must be before or the end of vesting (no vest, only lockup)
      require(
          _cliffDate <= _startTimestamp + _durationInSeconds,
```

```
            "!cliffEnd"
        );
```

Description

The code speaks themself, we don't need obvious comments.

Code Location

```
pitch-vesting/src/TimelockVesting.sol
```

Proof of Concept

-

Recommendation

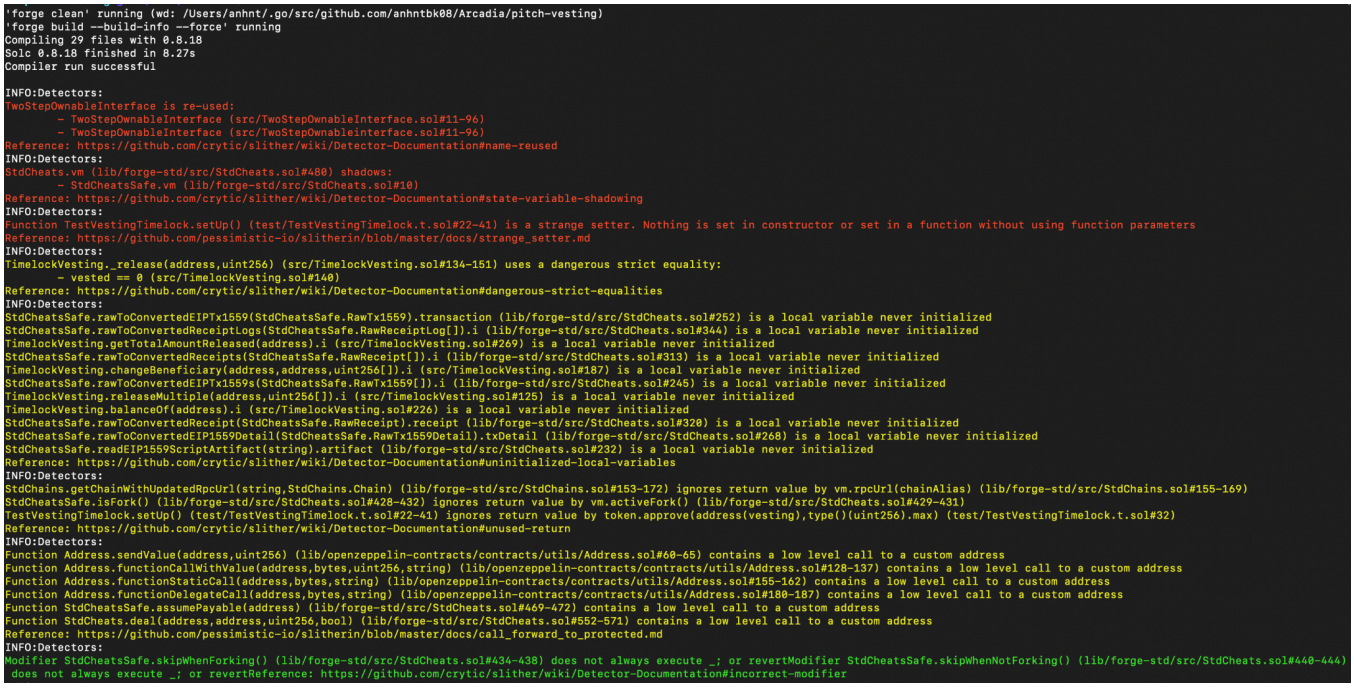
Remove the redundant comments.

Automated Tests and Tooling

Static Analysis with Slither

As a part of our engagement with Hourglass, we ran a static analysis against the source code using Slither, which is a Solidity static analysis framework written in Python. Slither runs a suite of vulnerability detectors and prints visual information about contract details. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

While Slither is not the primary element of Arcadia's offering, in some cases, it can be useful. The following shows the results found by the static analysis by Slither. We reviewed the results, and all of the issues found by Slither were at that point in time false positives.



Conclusion

After a comprehensive review, it has been determined that no Critical and High issues were identified during the audit. The contract has been meticulously crafted, exhibiting attention to detail, extensive comments, and thorough testing integrated into the code

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.


THE
ARCADIA
GROUP