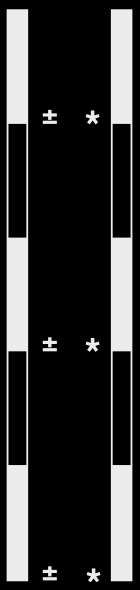
VERILOG

2024

REDACTED HIDDEN HAND MARIONETTE (THENA): SECURITY REVIEW REPORT



Jan 22, 2024

Contents

1 | Introduction

Figure 1.1: Hidden Hand Marionette Thena Report Cover

This report presents our engineering engagement with the Redacted-Cartel team on their inclusion of the Thena adapter and incentive distributor to their veNFT wrapper protocol.

Project Name	**Hidden Hand Marionette Thena**
Repository Link	https://github.com/redacted-cartel/hidden-hand-marionette
First Commit Hash	First: 2360625;
Final Commit Hash	Final: af7643e;
Language	Solidity
Chain	BNB Smart Chain

2 | About Verilog Solutions

Founded by a group of cryptography researchers and smart contract engineers in North America, Verilog Solutions elevates the security standards for Web3 ecosystems by being a full-stack Web3 security firm covering smart contract security, consensus security, and operational security for Web3 projects.
Verilog Solutions team works closely with major ecosystems and Web3 projects and applies a quality above quantity approach with a continuous security model. Verilog Solutions onboards the best and most innovative projects and provides the best-in-class advisory services on security needs, including on-chain and off-chain components.

3 | Service Scope

3.1 | Service Stages

Our auditing service includes the following two stages:

- Smart Contract Auditing Service

3.1.1 | Smart Contract Auditing Service

The Verilog Solutions team analyzed the entire project using a detailed-oriented approach to capture the fundamental logic and suggested improvements to the existing code. Details can be found under Findings And Improvement Suggestions.

3.2 | Methodology

- **Code Assessment**
  - We evaluate the overall quality of the code and comments as well as the architecture of the repository.
  - We help the project dev team improve the overall quality of the repository by providing suggestions on refactorization to follow the best practices of Web3 software engineering.
- **Code Logic Analysis**
  - We dive into the data structures and algorithms in the repository and provide suggestions to improve the data structures and algorithms for the lower time and space complexities.
  - We analyze the hierarchy among multiple modules and the relations among the source code files in the repository and provide suggestions to improve the code architecture with better readability, reusability, and extensibility.

3.3 | Audit Scope

Our auditing for Redacted-Cartel covered the Solidity smart contracts under the folder 'solidity/contracts' in the repository (https://github.com/redacted-cartel/hidden-hand-marionette) with commit hash **2360625**.

4 | Findings and Improvement Suggestions

Severity	Total	Acknowledged	Resolved
High	1	1	1
Medium	1	1	1
Low	3	3	3
Informational	0	0	0

4.1 | High

4.1.1 | Issue with if-else condition in the `_swappableAmount()` function

Severity	High
Source	contracts/adapters/thena/ThenaAdapter.sol#L847-L849;
Commit	2360625;
Status	Resolved in commit c4b41c2;

- **Description**
  The `_swappableAmount()` function calculates the cumulative amount to be swapped from the last epoch claimed or swapped to the current epoch. It is calculated as the difference between the amount collected and the amount claimed for each epoch:

```
if (amount > totalRewardModeEpochTokenAmountClaimed[rm][epoch][_token]) {
    amount -= totalRewardModeEpochTokenAmountClaimed[rm][epoch][_token];
} else {
    amount = 0;
}
```

  When an epoch doesn't satisfy the condition that the amount collected is greater than the amount claimed, the variable is restarted to zero. This means that the final amount won't represent the cumulative amount, which is what the function is intended to return.

- **Exploit Scenario**

  - ☐ The rewards haven't been claimed for at least one epoch;
  - ☐ The `_swappableAmount()` gets called to calculate the amount to be swapped;
  - ☐ The given amount will consider only the current epoch and not the cumulative sum of previous epochs.

- **Recommendations**
  Change the `else` condition to `amount+=0`.

- **Results**
  Resolved in commit c4b41c2.

  The `_swappableAmount()` was refactored to consider this change.

4.2 | Medium

4.2.1 | Suggestion of voting and collecting rewards every epoch

Severity	Medium
Source	solidity/contracts/adapters/thena/ThenaAdapter.sol;
Commit	2360625;
Status	Acknowledged;

- **Description**
  Based on a discussion with the redacted team, the `claimedEpochInitialVoteEpoch` variable is used when claiming after several epochs. However, considering that the distribution of rewards might not be consistent every epoch and users can leave and join the protocol at any time, we recommend that the collection of rewards is carried out every epoch.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Collect rewards every epoch.
- **Results**
  Acknowledged.

4.3 | Low

4.3.1 | Lack of event emission for critical operations

Severity	Low
Source	solidity/contracts/adapters/hidden-hand/HiddenHandAdapter.sol#L66-L72;
Commit	2360625;
Status	Resolved in commit be686c7;

- **Description**
  Events are vital aids in monitoring contracts and detecting suspicious behavior. The `setDistributor()` and `setHarvester()` functions perform critical state changes and therefore, they should consider the emission of an event.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Add events to functions that perform state changes.
- **Results**
  Resolved in commit be686c7.

  The events were added.

4.3.2 | Lack of zero address checks

Severity	Low
Source	contracts/adapters/hidden-hand/HiddenHandAdapter.sol#L70;
Commit	2360625;
Status	Resolved in commit 3c17fa3;

- **Description**
  Both the `setDistributor()` and `setHarvester()` functions should perform zero address checks when setting the new input address. Additionally, the `constructor()` should perform the same check on initialization.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Add zero address checks to the specified functions.
- **Results**
  Resolved in commit 3c17fa3.

  The checks were added.

4.3.3 | Inconsistency in the use of the `_currentEpoch()` function

Severity	Low
Source	solidity/contracts/adapters/thena/ThenaAdapter.sol#L828;
Commit	2360625;
Status	Resolved in commit 3d67675;

- **Description**
  The last epoch considered for claiming rewards is `_currentEpoch() - 1` and the swappable amount is calculated as the difference between the rewards collected and the rewards claimed. Therefore the `_swappableAmount()` function should also consider the same last epoch.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Change `uint256 lastEpoch = _currentEpoch();` to `uint256 lastEpoch = _currentEpoch() - 1;` in the `_swappableAmount()` function.
- **Results**
  Resolved in commit 3d67675.

  The suggestion was implemented.

5 | Further Review

After the first round of the security review which covered until the commit hash **cccb5d8**, a new update that considered adding a two-step process for the withdrawal of Marionettes and the migration from the previous to the new system was implemented by the Redacted team. The following section corresponds to the findings that covered this update, which specifically covers until the commit hash **af7643e**.

Severity	Total	Acknowledged	Resolved
High	1	1	1
Medium	2	2	2
Low	0	0	0
Informational	0	0	0

5.1 | High

5.1.1 | Incorrect check in the `_canBurn()` function

Severity	High
Source	contracts/adapters/thena/ThenaAdapter.sol#L1169-L1182;
Commit	7d18a55;
Status	Resolved in commit db04b9d;

- **Description**
  The `_canBurn()` function checks whether or not the Marionette can be burned based on the vote power of each user. As the code stands, this check is performed on the total vote power, i.e. the sum of the vote powers of all the Marionettess:

```
uint256 totalVotesLastEpoch =
rewardModeEpochTotalVotePower[Marionette.RewardMode.Compound][currEpoch - 1]
  + rewardModeEpochTotalVotePower[Marionette.RewardMode.Default][currEpoch - 1];
uint256 totalVotesEpochMinus2 =
rewardModeEpochTotalVotePower[Marionette.RewardMode.Compound][currEpoch -2]
  + rewardModeEpochTotalVotePower[Marionette.RewardMode.Default][currEpoch - 2];

bool isVoteOnly = veTokenInfo[_tokenId].rewardMode == Marionette.RewardMode.VoteOnly;
bool hasVotedLastEpoch = totalVotesLastEpoch > 0;
bool hasNotVotedEpochMinus2OrClaimed =
totalVotesEpochMinus2 == 0 || lastClaimedEpoch == currEpoch - 1;
```

  Instead, it should check the vote power of each `tokenId` that is about to be burned.

- **Exploit Scenario**

  - Alice hasn't voted in the last round and wants to withdraw her `veThena` from the system;
  - The `_canBurn()` function doesn't perform the correct check for the vote power of her specific `tokenId`;
  - Alice is able to burn her Marionette and get the `veThena`

- **Recommendations**
  Change the checks that use the `rewardModeEpochTotalVotePower` mapping to ones that consider the vote power of each `tokenId`.

- **Results**
  Resolved in commit c4b41c2.

  The suggestion was implemented.

5.2 | Medium

5.2.1 | Issue with the `increaseMany()` function

Severity	Medium
Source	solidity/contracts/adapters/thena/ThenaAdapter.sol#L402;
Commit	7d18a55;
Status	Resolved in commit 9b00a64;

- **Description**
  The `increaseMany()` function allows keepers to increase the position in the voting escrow contract by calling the `VE.increase _amount()` function. Since there is no way for the Thena Adapter to calculate the rewards that are being received, a malicious keeper could increase the position using the rewards from other users.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Add a `sender` input parameter to the `increaseMany()` function, so that the amount to be increased is transferred from this sender to the contract and the `VE.increase _amount()` function uses that amount to increase the position in the voting escrow contract.

- **Results**
  Resolved in commit 9b00a64.

  The suggestion was implemented.

5.3 | Low

5.3.1 | Incorrect event emission

Severity	Low
Source	solidity/contracts/MarionetteZapper.sol#L56;
Commit	7d18a55;
Status	Resolved in commit be40433;

- **Description**
  The `SetVeOperator` event is emitted when the operator of the Marionette Zapper is changed. However, the new address of the operator is not considered in this event.

```
emit SetVeOperator(address(votingEscrow), _status);
```

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Consider adding the new operator address to the `SetVeOperator` event.

- **Results**
  Resolved in commit be686c7.

  The suggestion was implemented. The event now considers the following inputs:

```
emit SetVeOperator(address(votingEscrow), operator, _status);
```

6 | Appendix

6.1 | Appendix I: Severity Categories

Severity	Description
High	Issues that are highly exploitable security vulnerabilities. It may cause direct loss of funds / permanent freezing of funds. All high severity issues should be resolved.
Medium	Issues that are only exploitable under some conditions or with some privileged access to the system. Users' yields/rewards/information is at risk. All medium severity issues should be resolved unless there is a clear reason not to.
Low	Issues that are low risk. Not fixing those issues will not result in the failure of the system. A fix on low severity issues is recommended but subject to the clients' decisions.
Information	Issues that pose no risk to the system and are related to the security best practices. Not fixing those issues will not result in the failure of the system. A fix on informational issues or adoption of those security best practices-related suggestions is recommended but subject to clients' decision.

6.2 | Appendix II: Status Categories

Severity	Description
Unresolved	The issue is not acknowledged and not resolved.
Partially Resolved	The issue has been partially resolved
Acknowledged	The Finding / Suggestion is acknowledged but not fixed / not implemented.
Resolved	The issue has been sufficiently resolved

7 | Disclaimer

Verilog Solutions receives compensation from one or more clients for performing the smart contract and auditing analysis contained in these reports. The report created is solely for Clients and published with their consent. As such, the scope of our audit is limited to a review of code, and only the code we note as being within the scope of our audit is detailed in this report. It is important to note that the Solidity code itself presents unique and unquantifiable risks since the Solidity language itself remains under current development and is subject to unknown risks and flaws. Our sole goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies. Thus, Verilog Solutions in no way claims any guarantee of security or functionality of the technology we agree to analyze.
In addition, Verilog Solutions reports do not provide any indication of the technology's proprietors, business, business model, or legal compliance. As such, reports do not provide investment advice and should not be used to make decisions about investment or involvement with any particular project. Verilog Solutions has the right to distribute the Report through other means, including via Verilog Solutions publications and other distributions. Verilog Solutions makes the reports available to parties other than the Clients (i.e., "third parties") – on its website in hopes that it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.