HashCloak

Code Review and Security Assessment
For
Term Structure Labs

Delivery: September 25, 2023
Updated: October 4, 2023

Prepared For:
Term Structure Labs

Prepared by:
Onur Inanc Dogruyol | *HashCloak Inc.*
Soumen Jana | *HashCloak Inc.*

Table Of Contents

Executive Summary

Term Structure Labs engaged *HashCloak Inc.* for an audit of its *Term Structure* product, which is a decentralized bond protocol that enables peer-to-peer lending and borrowing with fixed interest rates. The audit was done from *August 28th, 2023* to *September 25th*, *2023*. The relevant codebases were:

- Smart contract repository: ts-contract-diamond, assessed at commit: 483d2cf65124bd3bc55142c61e488ba2e2d20cf8
- Circom repository: ts-circom-dd, assessed at commit: 956ef6b2383cc883e9e7797c2cff49f421561018

The scope of the audit was all files in the following folders:

- `decentral-portal/ts-circom-dd`
- `decentral-portal/ts-contract-diamond`

In response to Term Structure Labs' request, this Audit Report is organized into two distinct sections. In this section, we will comprehensively assess the Solidity component, examining its codebase, functions, and security considerations. We will outline our findings, including potential vulnerabilities and recommendations for enhancements or mitigations.

Throughout the audit, we familiarized ourselves with the contracts in scope and sought to understand how the overall *Term Structure* protocol works. For the first 2 weeks of the audit, we familiarized ourselves with the smart contracts and circuits. During this time, we focused on finding simple bugs and issues within the codebases. In the subsequent two weeks of the audit, we focused on finding more complex bugs and issues. For the smart contract portions, we combined our manual analysis skills alongside automated, off-the-shelf tooling such as Slither. For the circuits, we mainly relied on manual analysis but used circumspect and circumscribe to help identify simpler issues within the circuits.

Overall, we found the issues range from high to informational:

Severity	Number of Findings	Severity	Number of Findings
Critical	0	Low	1
High	1	Informational	0
Medium	2		

Findings

TKS-S-1: Reentrancy in `flashLoan()` function for ERC777 tokens

Type: High
Files affected: FlashLoanFacet.sol#L48 , FlashLoanFacet.sol#L54

Description: In the `flashLoan()` function if any ERC777 token (extension of ERC20) is whitelisted then it is possible to steal tokens from the FlashLoanFacet.sol contract as ERC777 tokens allow the sender/receiver to hook before/after sending the token. In this case, the malicious user can create a hook that will re-enter into the `flashLoan()` function once the `flashLoan()` function transfers the token to the malicious user. Then, the user continues doing this for maximum profit and now the malicious user finally finishes the transaction by repaying the first initiated `flashLoaned` amount and he will take the profit of the rest of the re-entrance calls token amount.

PoC:

1. Suppose imBTC (ERC777 Token) is whitelisted in the FlashLoanFacet.sol contract and the contract has $10,000 worth of tokens.
2. Now a malicious user can call the `flashLoan()` function with $1000 worth of tokens, and this user already created a hook in the ERC777 token which will re-enter into the `flashLoan()` function 9 times.
3. The malicious user now has $10,000 worth of tokens and the rest of the `flashLoan()` function code will be executed and the user needs to repay only $1000 worth of tokens + $0.3 worth of tokens as a fee (as a fee percentage 0.03%).
4. The malicious user profited ~$9000 worth of tokens from this attack.

Impact: Any malicious user can drain all funds from the FlashLoanFacet.sol contract.

Suggestion: Use `non-Reentrant` modifiers to safeguard from reentrancy attack in the `flashLoan()` function, or use a lock variable in the function that will block the re-entrancy call made to the `flashLoan()` function and once the function code is executed revert back the lock variable to in its previous step.

Status: We reviewed the commit hash f8b7cf9aecea87bc7cc9f8149e3ee37687cc5682 and the team fixed the issue by adding `ReentrancyGuard` in the `flashLoan` function as suggested in the report.

TKS-S-2: Single Point of Failure in 'adminAddr' and 'operatorAddr'

Type: Medium
Files affected: ZkTrueUpInit.sol#L52-L57

Description: The '`adminAddr`' and '`operatorAddr`' both exhibit a single point of failure in the '`onlyRole()`' modifier with '`OPERATOR_ROLE`, `COMMITTER_ROLE`, `VERIFIER_ROLE`, `EXECUTER_ROLE`' role can access many critical functions.

Impact: This design poses a significant operational risk as the failure or compromise of either '`adminAddr`' or '`operatorAddr`' can lead to a disruption in critical system functions.

Suggestion: It is recommended to use multisig addresses for critical functions.

Status: We reviewed the commit hash f8b7cf9aecea87bc7cc9f8149e3ee37687cc5682 and the team acknowledged the issue by stating that: "*At the beginning, the `admin` and the `operator` address are managed by us and the `admin` will be a multisig wallet. Eventually, the `admin` role will evolve to be managed by DAO, and multiple committer, verifier, and executor roles will also be expanded to achieve decentralization.*"

TKS-S-3: Absence of zero address check in wETH, poseidonUnit2, and aaveV3Pool initialization

Type: Medium
Files affected: ZkTrueUpInit.sol#L66-L70

Description: The audit has identified a vulnerability in the protocol as there is no 0 address check present for the initialization of '`wETH`', '`poseidonUnit2`', and '`aaveV3Pool`'. This means that if, for any reason, these components are initialized with a 0 address, it could potentially disrupt the protocol's logic.

Impact: This vulnerability poses a significant risk to the integrity and stability of the protocol. Initialization with a 0 address to these critical components can lead to unpredictable behavior, financial risks, etc.

Suggestion: To mitigate this risk, it is recommended to implement a check for 0 addresses during the initialization of '`wETH`', '`poseidonUnit2`', and '`aaveV3Pool`'.

Status: We reviewed the commit hash [f8b7cf9aecea87bc7cc9f8149e3ee37687cc5682] and the team fixed the issue and added a `*not zero address*` checker as suggested in the report. In addition, they also added this checker to all address parameters in the initialized contract.

TKS-S-4: Single Address Usage for Multiple Roles

Type: Low
Files affected: ZkTrueUpInit.sol#L54-L57

Description: The audit identified a practice where a single address is being utilized for various roles within the system.

Impact: This approach could potentially result in a lack of clear accountability and control over each specific role, making it challenging to monitor and manage permissions and responsibilities effectively. Additionally, it may pose security and operational risks if unauthorized access or actions occur.

Suggestion: It is recommended to either use a single role for all addresses or use different addresses for different roles. This segregation of roles and addresses will enhance clarity, security, and management of responsibilities.

Status: We reviewed the commit hash [f8b7cf9aecea87bc7cc9f8149e3ee37687cc5682] and the team acknowledged the issue by stating that: "*At the beginning, the roles are managed by the protocol team and they use the multisig wallet to manage the highest `admin` role which can set protocol roles and parameters, and the other roles that need to frequently send transactions, such as `committer`, `verifier` and `executer`, are managed by the `operator` address first. In the future, these roles will be decentralized, like `admin` will be managed by DAO, as well as opening external operators for `committer`, `verifier` and `executer`, etc.*"