Security assessment and code review

Initial Delivery: October 11, 2021
Most recent version: October 22, 2021

Prepared for:
Ted Shao | Shadows Network
Lin Bruce | Shadows Network

Prepared by:
Er-Cheng Tang | HashCloak Inc
Peter Lai | HashCloak Inc

Table Of Contents

Executive Summary

Shadows Network engaged HashCloak Inc for an audit of their Shadows Network smart contracts, written in Solidity. The audit was done with two auditors over a 2 week period, from September 27, 2021 to October 11, 2021. The Shadows Network codebase was assessed at commit 431aa3f00a1f554f77233b4e20dca4f57c633786.

The files within scope were all files ending in **.sol**.

During the first week we familiarized ourselves with the Shadows Network smart contracts and started our manual analysis of the smart contracts. In the second week we further investigated the code base and an automated analysis was performed with off-the-shelf automated tools.

We found a variety of issues ranging from Medium to Informational and provide some general guidance to improve the code quality.

The Shadows Network team responded with a series of fixes, with a final commit 3b90e8623dedb4133b71118ca3474373d2b79561 on October 17. From October 18 to October 22, we reviewed the new changes and have verified that no new issues were added.

Severity	Number of Findings
Critical	0
High	0
Medium	1
Low	2
Informational	4

Overview

The Shadows Network is a decentralized trading platform for financial derivatives. Under collateration of the native token DOWS, participants can obtain xUSD, and invest in synthesized assets whose prices are updated by oracles. As the overall value on the network could change with the prices of synthesized assets, the investors are indeed competing against each other over the network to earn their profits.

The smart contract codebase can be separated into several logical units:

Oracle:	Report the prices of synthesized assets
Shadows:	Act as the native token for collateration
Synth:	Act as a possible target of investment
Synthesizer:	Responsible for collateralizing and redeeming
Exchanger:	Responsible for asset exchanges
Liquidation:	Responsible for deciding liquidation conditions
FeePool:	Responsible for calculating fees
RewardEscrow:	Keep track of reward escrow information
AddressResolver:	Keep track of contract addresses within the network

Findings

Account debt can be modified without permission

Type: Medium
Files affected: Synthesizer.sol

As `issueSynthsFrom` is a public method, it allows anyone to make other accounts issue debt without their permission. Although all of the issuance rules still apply, the accounts could be unwillingly put into risky situations. This is because the chance of being liquidated increases with the account collateration ratio. For example, a malicious user can raise its victim's amount of debt to exactly match the maximum issuance ratio. As soon as the value of the synth pool increases, the malicious player can liquidate its victim and take away DOWS.

Impact: Normal users' liquidation risk could be unwillingly increased.
Suggestion: Add a related allowance method and maintain the allowance data, or remove the method `issueSynthsFrom`.
Status: The Shadows team issued a commit to make `issueSynthsFrom` internally.

Incorrect business logic

Type: Low
Files affected: Synthesizer.sol, Synth.sol

In Synthesizer.sol, the method `_burnSynthsForLiquidation` aims to burn the account's debt. However, in line 368 the liquidator's debt is also burnt, which seems unintended. In Synth.sol, the method `_transfer` should call `_transferToFeeAddress` in case the recipient is the fee address. But in line 60 the fee is transferred back to the sender instead of the recipient.

Impact: There are unintended operations that are applied to liquidators, and there are missing fee collections in some cases.
Suggestion:

1. Remove `ISynth(address(synths[xUSD])).burn(liquidator, amount)` in line 368 of Synthesizer.sol.
2. Change `sender` to `recipient` in line 60 of Synth.sol.

Status:

1. The Shadows team responded that they intended to liquidate the liquidator to make sure the debt is fully repaid.

2. The Shadows team issued a commit to replace sender with recipient.

Missing input parameter validation

Type: Low
Files affected: FeePool.sol

It is a good practice to validate every input parameter. In `initialize`, the range of the parameter `_exchangeFeeRate` was not properly limited.

Impact: The business logic of having a maximum exchange fee rate is not enforced.
Suggestion: Apply the same requirement on `_exchangeFeeRate` in `initialize` as in `setExchangeFeeRate`, or call `setExchangeFeeRate` (change it to a public method) inside `initialize`.
Status: The Shadows team issued a commit to verify `_exchangeFeeRate` in initialize.

Emit events for important change of system parameter

Type: Informational
Files affected: AddressResolverUpgradable.sol, AddressResolverable.sol

The resolver is crucial throughout the system to keep track of all the contracts in interaction. As such, any attempt to change the resolver address should come with emitting an event to signal all users.

Suggestion: Emit an event at the end of `setResolver`.
Status: The Shadows team issued a commit to emit an event at `setResolver`, but only one file out of two has made such a change.

Check for contract address

Type: Informational
Files affected: AddressResolver.sol

The job of the address resolver is to map out the contracts in interaction. As such, it would be better to check that the input addresses to `importAddresses` are indeed contracts and not externally owned addresses.

Suggestion: Check that addresses are contracts whenever it should be the case. An implementation of contract check can be found in OpenZeppelin.
Status: The Shadows team issued a commit to check addresses are smart contracts at `importAddresses`.

Unused variables and methods

Type: Informational
Files affected: Oracle.sol

There are declared but unused variables and methods that could be removed: `DECIMAL` in Synth.sol and `effectiveValueAtRound` in Oracle.sol. Also, there is no need to add the modifier `rateNotStale` for `effectiveValueAtRound` as it is calculating for a particular round.

Suggestion: Remove unused variables and methods.
Status: The Shadows team issued a commit to remove unused variables. Furthermore, after clarification from the Shadows team, we learned that `effectiveValueAtRound` is used by the Shadows Network front-end. As such, `effectiveValueAtRound` does not need to be removed.

Reassignment of variables

Type: Informational
Files affected: Exchanger.sol

In `calculateExchangeAmountMinusFees`, `amountReceived` was assigned in both lines 105 and 110, so the previous assignment is void.

Suggestion: Remove the redundant assignment.
Status: The Shadows team issued a commit to remove the redundant assignment.
Verification: Resolved.

General Recommendations

Add documentation and comments

We consider the level of documentation and comments in the code to be insufficient and think that the Shadows Network team and the code quality would benefit greatly if that would be changed. Especially providing introductory comments for every smart contract file would be a great productivity booster for people new to the codebase and reduce the number of bugs during development.

Remove constructor visibility

Since version 0.7 the concept of constructor visibility (internal/public) has been abandoned. We recommend removing the `public` keyword when using constructors. The affected contract is AddressResolverable.sol.

Remark on the use of initializer instead of constructors

The usage of initializers instead of constructors can be dangerous. Unfortunately the usage of proxy contracts leaves no other option. We want to point out that a lot of care has to be taken during contract deployment when using initializers, especially:

- Every contract has to be correctly initialized. This has to be made sure for every single contract. The main risk involved here is an operational one due to human errors in the deployment procedure.
- Ideally the contract deployment and initialization is an atomic state change, meaning that deployment and initialization take place in the same transaction.

Public vs External Function

Public functions which are solely called from external (other smart contracts or externally owned accounts) should be marked `external` instead of `public`, since it saves gas costs.

Zero address validation

We recommend checking for `!= address(0)` whenever possible, since it prevents errors due to uninitialized addresses. This is done in several but not all places in the codebase.

Event declaration before function declaration

We recommend adhering to the official order of layout when writing smart contracts.
https://docs.soliditylang.org/en/v0.8.9/style-guide.html#order-of-layout