OpenZeppelin

Coinbase Liquid Staking Token Audit

OPENZEPPELIN SECURITY | AUGUST 24, 2022

Security Audits

The Coinbase team engaged us to review and audit their `Wrapped tokens` project. We looked at the code and now publish our results.

Scope

We audited the contracts that were shared by the Coinbase team privately in a compressed folder named as `wrapped-tokens-master`, with two auditors over two weeks. The scope includes the following files:

- contracts/wrapped-tokens/MintForwarder.sol
- contracts/wrapped-tokens/MintUtil.sol
- contracts/wrapped-tokens/RateLimit.sol
- contracts/wrapped-tokens/staking/ExchangeRateUpdater.sol
- contracts/wrapped-tokens/staking/ExchangeRateUtil.sol
- contracts/wrapped-tokens/staking/StakedTokenV1.sol

All other project files and directories, along with external dependencies and projects, were excluded from the scope of this audit. External code and contract dependencies were assumed to work as documented.

Summary

OpenZeppelin

System Overview

The goal of this project is to introduce wrapped assets for Coinbase whether they are staked or not. The `StakedTokenV1` staked token, which inherits and extends from Centre's `FiatTokenV2_1`, is issued to represent the corresponding staked wrapped asset, while a plain `FiatTokenV2_1` is used for non-staked wrapped tokens. The only difference between the two is that staked wrapped tokens implement an `exchangeRate` parameter to improve composability with the ecosystem.

The minting of tokens and update of exchange rate of staked tokens is done using a rate limiting functionality. The callers executing these functionalities are configured in the `RateLimit` contract by the owner. The rate limiting parameters, such as, maximum allowance of the caller, current allowance, interval, and last time of setting allowance, determine how many times a caller can exercise these functionalities and how much value can be minted. Given the rate limit over time, the current allowance of the caller is increased by calling the public `allowanceCurrent` function in `RateLimit` contract or during the minting and exchange rate updating processes. Once the allowance has been used for these operations, the amount used is reduced from the caller's allowance balance. At the time of this audit, any changes to a caller's maximum allowance or interval would require reconfiguration of the caller.

The `MintForwarder` contract is used to direct the mint call to the respective token address for minting wrapped tokens or to the `FiatTokenV2_1` contract address for minting the staked tokens.

The `ExchangeRateUpdater` contract acts as an oracle to provide the exchange rate between the staked token and the corresponding backing asset.

Privileged roles

There are a number of privileged roles that exercise wide-ranging powers over the wrapped and staked tokens:

- Owner of `RateLimit` contract
  - can configure a caller and define their rate limiting parameters

    - can initialize the contract by setting the token contract address where the mint calls are forwarded and transferring ownership to a new owner
- Owner of `ExchangeRateUpdater` contract
    - can initialize the contract by setting the token contract address for which the exchange rate is needed and transferring ownership to a new owner
- Owner of `StakedTokenV1` contract
    - inherits all the owner privileges from Centre's `FiatTokenV2_1`
    - can update the oracle that provides exchange rates
- Caller
    - can mint tokens
    - update exchange rates

***Update:** During the fix review process, Coinbase team has asked us to review PR #6. A `FiatTokenProxy` contract was added to the audited repo. This contract interacts with the Centre's `FiatTokenProxy`, which is out of the scope of this audited and is assumed to function as intended.*

Critical severity

None.

High severity

None.

Medium severity

[M01] Insufficient allowance of the callers can prevent the update of exchange rate

The owner of the `RateLimit` contract configures the allowance of the `caller` which decreases as when the caller mints new tokens or updates the exchange rate and it increases up to a `maxAllowance` parameter with a time schedule dictated by the rate limit functionality.

the crypto market, if the exchange rate is not updated timely, systems and protocols depending on it can be dramatically affected and the trading of the staked tokens could be impacted.

The severity of this issue depends on the number of callers or which schedules and parameters are meant to be setup for callers. At the time of this audit, there is no documentation stating the number of callers and their setup parameters which are intended to interact with the system. The lesser number of callers or larger intervals with low allowances, the more likelihood of this situation can occur. We recommend to ensure that, at all times, there are sufficient number of callers interacting with the system and they have enough allowance to maintain the health of the system.

Update: *Acknowledged.*

> Coinbase acknowledges this risk and will have a cold key that has sufficient allowance to account for all exchange rate updates at all times. Coinbase's on-chain exchange rate may become out of date in the event of a loss of funds due to slashing of staked ETH, as the staked ETH slashing period happens over the course of several weeks, and we cannot know the extent of loss until the end of that period.

Low severity

[L01] Allowance of a removed caller can be replenished

The `RateLimit` contract allows its owner to configure and remove callers. Caller can be removed by calling the `removeCaller` function which sets the corresponding value in `callers` mapping as false. However, it fails to update the `intervals`, `allowancesLastSet`, `maxAllowances` and `allowances` mappings for the removed caller.

Furthermore, the `allowanceCurrent` function in the `RateLimit` contract is used to get the current allowance of a caller. This function in-turn calls the `_replenishAllowance` function which replenishes the allowances of the input `caller`. Neither of the functions implement a check to see if this `caller` address has been removed from the `callers` mapping.

however, it can result in wastage of gas.

Consider resetting the value of all the mappings to make sure the caller is removed completely and implementing proper checks in the `allowanceCurrent` function.

***Update:** Fixed as of commit `3ed1be6` in PR #5.*

[L02] Duplicate getters

In the `RateLimit` contract, the `callers` and `allowances` mapping are declared as public, implying that automatic getters are generated and exposed to the public contract's API.

However, explicit getter functions, such as the `isCaller` and `allowanceStored` functions, are implemented which duplicate the automatically generated getters.

To improve code's readability and consistency, and to improve gas consumption, consider either declaring the mappings as private or removing the duplicated functions.

***Update:** Fixed as of commit `7f2d6f0` in PR #4.*

[L03] Lack of input validation

In the `initialize` functions of the `ExchangeRateUpdater` and `MintForwarder` contracts, the `newTokenContract` parameter is not checked to be either a valid contract or an ERC20 token. Moreover, the `newOwner` can be either an EOA or a contract but this is not described in the docstrings.

Consider improving the docstrings to reflect the exact intended behaviour, and using `Address.isContract` function from OpenZeppelin's library to detect if an address is effectively a contract. Moreover, consider adopting the `Initializable` to implement the current functionality.

***Update:** Partially fixed as of commit `273a2b8` in PR #3. While docstrings have been added, there are no checks for the input address parameters.*

[L04] Missing, incomplete or incorrect docstrings

There are some places in the codebase, where docstrings are either missing, incomplete or incorrect. Some examples are:

- Public state variables and event definitions in the `RateLimit` contract are missing comments or docstrings.
- Some docstrings, like the ones in the `StakedTokenV1.oracle()` function, have `@return` tag, while others, like the `RateLimit.estimatedAllowance()` function don't have it.
- The `ExchangeRateUtil` title is missing a `@dev` tag.
- The docstring above the `ExchangeRateUpdater.initialize()` function on line 43 states that the input parameter `newTokenContract` is the address of the token contract for which the contract mints, whereas, it is the address of the token contract for which the exchange rate is updated.

These examples hinder reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Fixed as of commit* `be71b75` *in PR #2.*

[L05] Possible truncation

In the `RateLimit` contract, the `amountToReplenish` is calculated as a division between `secondsSinceAllowanceSet * maxAllowances[caller]` and `intervals[caller]`.

Since the result of the division is returned by the `_getReplenishAmount` function, and is used in `_replenishAllowance` function to update caller's allowance, this truncation can lead to a failure in updating caller's allowance.

To avoid such scenarios, when configuring a new caller, consider setting the `intervals`, `maxAllowances` and `allowances` to compatible values.

Update: *Acknowledged.*

> Coinbase acknowledges that truncation can occur though has not fixed it as the maximum impact of the truncation would be a failed on-chain transaction and needing to reconfigure the caller.

Notes & Additional Information

[N01] Multiple Solidity versions in use

Throughout the code base there are different versions of Solidity being used. For example, the `StakedTokenV1` contract is specifically using version 0.6.12 while other contracts allow compiling with version 0.8.6.

To avoid unexpected behaviors, all contracts in the code base should allow being compiled with the same Solidity version.

Update: *Acknowledged.*

> Coinbase acknowledges that multiple Solidity versions are being used. This was an intentional decision made to allow battle tested Solidity 0.6.12 USDC code reuse for both the `FiatTokenProxy` and `StakedTokenV1` smart contracts. We've used Solidity 0.8.6 for our new contracts.

[N02] Naming issue

In order to improve the readability, consider changing the name of the `allowanceCurrent` function on line 123 of `RateLimit` contract

Update: *Fixed as of commit* `e676460` *in PR #1.*

Conclusions

No critical or high severity issues were found in the codebase. Several minor vulnerabilities have been found and recommendations and fixes have been suggested.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits