Coinbase cbBTC

Security Review

Cantina Managed review by:
Mridul Garg, Security Researcher
Akshay Srivastav, Security Researcher

November 22, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Coinbase is a secure online platform for buying, selling, transferring, and storing cryptocurrency.

From Oct 14th to Oct 15th the Cantina team conducted a review of wrapped-tokens-audit on commit hash 0121d6e0. The team identified a total of **10** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 2
- Low Risk: 4
- Gas Optimizations: 3
- Informational: 1

3 Findings

3.1 Medium Risk

3.1.1 Minters can mint double the max allowance in an interval

Severity: Medium Risk

Context: RateLimit.sol#L137

Description: Callers can mint more tokens than `maxAllownce` in an interval. Let's say interval is 1 hour and `maxAllowance` is 60 cbBTC. At `t = 0`, caller's allowance is 60 cbBTC. Consider this scenario (more scenarios can be crafted):

- At `t = 0`, caller mints 60 cbBTC, their allowance drops to 0.
- At `t = 0.5 hour`, `amountToReplenish` is 30 cbBTC, so the new allowance becomes 30 cbBTC. Caller mints 30 cbBTC.
- Just before the 1 hour intervals finishes, `amountToReplenish` becomes 30 cbBTC and the caller can mint another 30 cbBTC.

So with maxAllowance = 60 cbBTC, caller can mint 120 cbBTC in the same interval. A similar issue happens when a minter doesn't use his allowance in an interval. This allows the minter to mint double the max allowance in the next interval.

Proof of Concept:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.6;

import "forge-std/Test.sol";
import "forge-std/mocks/MockERC20.sol";
import "src/cbbtc/wrapped-tokens/MintForwarder.sol";

contract MockToken is MockERC20 {
    constructor() {
        initialize("MockToken", "TKN", 18);
    }
    function mint(address to, uint256 amount) public {
        _mint(to, amount);
    }
}

contract MintForwarderTest is Test {
    address receiver = makeAddr("receiver");

    MintForwarder mintForwarder;
    MockToken mockToken;

    function setUp() public {
        mockToken = new MockToken();
        mintForwarder = new MintForwarder();
        mintForwarder.initialize(address(this), address(mockToken));
    }

    function test_poc_mintDoubleInOneInterval() public {
        mintForwarder.configureCaller(address(this), 1000, 100 seconds);        // 1000 tokens in 100 seconds
↪   interval

        vm.warp(block.timestamp + 100 seconds);
        mintForwarder.mint(receiver, 1000);
        mintForwarder.mint(receiver, 1000);              // minted 2X tokens in one interval

        vm.warp(block.timestamp + 100 seconds);
        mintForwarder.allowanceCurrent(address(this));

        vm.warp(block.timestamp + 100 seconds);
        mintForwarder.mint(receiver, 1000);
        mintForwarder.mint(receiver, 1000);              // minted 2X tokens in one interval

        vm.warp(block.timestamp + 100 seconds);
        mintForwarder.allowanceCurrent(address(this));

        vm.warp(block.timestamp + 100 seconds);
```

```
        mintForwarder.mint(receiver, 1000);
        mintForwarder.mint(receiver, 1000);             // minted 2X tokens in one interval
    }
}
```

Recommendation: Consider a replenish only after the last update to allowance is `interval` seconds apart, or setting the initial `allowance` to `0`.

Coinbase: Fixed in commit 576792cf.

Cantina Managed: Fixed.

3.1.2 Every `RateLimit.configureCaller` call grants instant allowance to callers which leads to instant token minting

Severity: Medium Risk

Context: RateLimit.sol#L58-L67

Description: The `RateLimit.configureCaller` function sets the `allowances` map value for caller to `amount`. This gives the caller an instant allowance of `amount` amount. After the `configureCaller` call the caller can instantly mint `amount` amount of tokens without any delay. Consider the following scenario:

- At time `t1`, a caller is granted an allowance of `1000` tokens per interval.
- At time `t1` the caller mints `1000` tokens.
- After that the caller can only mint `1000` tokens in every interval.
- At time `t2` the caller's allowance is changed to `800` tokens per interval.
- At time `t2` the caller can mint `800` tokens instantly without any interval delay.

This ability of callers to utilize their new allowance instantly could be unintended and can lead to tokens getting minted faster than expected.

Recommendation: Consider initializing `allowances` value to `0` in `configureCaller` function.

Coinbase: This behavior is intentional, especially when increasing the limits due to a spike in demand, we need the new allowance to take place immediately.

Cantina Managed: Acknowledged.

3.2 Low Risk

3.2.1 Returned values of low-level external calls are ignored

Severity: Low Risk

Context: BatchBlacklister.sol#L47-L54, MintUtil.sol#L41-L44

Description: The `MintUtil.safeMint` and `BatchBlacklister.blacklistAddresses` functions use `Address.functionCall` helper function which returns `bytes` values returned by the low level function calls. Currently these returned values are completely ignored.

The `FiatTokenV1.mint` returns a `boolean` value & the `FiatTokenV1.blacklist` does not return any data.

In case the `MintUtil` and `BatchBlacklister` interact with any other token implementation which returns a logical value without reverting on calls then that can lead to unintended scenarios (like unexpected `token.mint` outcome).

Recommendation: Consider explicitly checking the returned values of low level external calls.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.2.2 Exact duplicate `FiatToken` and `MintForwarder` contract can be deployed

Severity: Low Risk

Context: MintForwarderFactory.sol#L40-L53, TokenFactory.sol#L87-L111

Description: The `TokenFactory.deployToken` and `MintForwarderFactory.deployMintForwarder` functions use `msg.data` to generate the create2 `salt`for `FiatTokenProxy` and `MintForwarder` deployments.

The intention of doing this is to have deterministic addresses across chains for contracts having pre-determined construction parameters. However the `msg.data` of a function call can contain more `bytes` data than what a solidity function expects.

In case a caller appends random data to `deployToken` call but with same input parameters as a previous `deployToken` call, it will lead to a different `create2 salt` value. Hence a different `FiatTokenProxy` will be deployed for the same input parameters of an existing `FiatTokenProxy` contract. The same issue exist for `MintForwarderFactory.deployMintForwarder` function.

Consider the following scenario:

- Create a `FiatTokenProxy` for some input parameters `X` by calling `TokenFactory.deployToken(X)`.
- A `FiatTokenProxy` at address `M` will get deployed.
- Call `TokenFactory.deployToken` again with parameters `X + extra_bytes`.
- A duplicate `FiatTokenProxy` contract with same parameters as of the `FiatTokenProxy` at address `M` will get deployed at address `N`.

Recommendation: Consider explicitly validating the `msg.data.length` of `TokenFactory.deployToken` and `MintForwarderFactory.deployMintForwarder` functions.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.2.3 Callers can frontrun the `configureCaller` call to gain additional allowance

Severity: Low Risk

Context: RateLimit.sol#L58-L67

Description: The `RateLimit.configureCaller` function overwrites the existing allowance state of a caller. This can be misused by existing callers in case their allowance is about to be changed by `owner`. Consider the following scenario:

- Suppose there is a caller whose current allowance is `X`.
- Owner decides to change that caller's allowance to `Y`.
- The caller frontruns the `configureCaller` call and exhausts its allowance `X` by minting `X` amount of tokens.
- After the `configureCaller` call gets executed the caller will now get `Y` allowance.
- Overall the caller gained `X + Y` allowance.

Recommendation: Consider always resetting an existing caller's allowance to `0` before updating it to any new value.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.2.4 `RateLimit.removeCaller` does not reset the allowance state of caller

Severity: Low Risk

Context: RateLimit.sol#L73-L76

Description: The `removeCaller` function only resets the `callers` mapping value for the removed caller. It does not changes the `maxAllowances, allowances, allowancesLastSet & intervals` states of the caller.

This leads to some unintended scenarios:

1. `allowanceCurrent` function still works for the removed caller.
2. `estimatedAllowance` & `allowanceStored` functions return non-zero allowances for the removed caller.

Any external agent relying on these functions will observe incorrect properties for the caller.

Recommendation: Consider resetting all allowance states for the caller who is getting removed.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 Cache storage variables to avoid multiple reads

Severity: Gas Optimization

Context: MintForwarder.sol#L65-L67, RateLimit.sol#L117-L127, RateLimit.sol#L135-L142

Description: `allowances` and `maxAllowances` mappings are called for the same address multiple times leading to reading the same value from storage. You can save gas by reading these values just once.

Recommendation: Cache these values in stack to avoid multiple `SLOAD`s.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 `TokenFactory::implementation` variable can be marked as `immutable`

Severity: Gas Optimization

Context: TokenFactory.sol#L36

Description: The `implementation` state variable in `TokenFactory` can be made `immutable`. Immutables are available in Solidity v0.6.12.

Recommendation:

```
- address public implementation;
+ address public immutable implementation;
```

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 Avoid redundant ownership checks in `MintForwarder.initialize`

Severity: Gas Optimization

Context: MintForwarder.sol#L44-L49

Description: The `MintForwarder.initialize` function has the `onlyOwner` modifier which validates the ownership of `msg.sender`. It then performs `transferOwnership` which again validates the ownership of `msg.sender`.

Recommendation: Consider using `_setOwner()` function instead of `transferOwnership()` in the `initialize` function.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Anyone can deploy `FiatTokenProxy` and `MintForwarder` contracts

Severity: Informational

Context: MintForwarderFactory.sol#L40, TokenFactory.sol#L87

Description: The `TokenFactory.deployToken` and `MintForwarderFactory.deployMintForwarder` functions are publicly accessible functions which means any user can deploy new `FiatTokenProxy` and `MintForwarder` contracts at their wish.

Since cbBTC protocol is a permissioned system, ideally only protocol team should be allowed to deploy new `FiatTokenProxy` and `MintForwarder` contracts.

Combined with the issue titled as `Exact duplicate FiatToken and MintForwarder contract can be deployed`, any user can deploy exact copies of `FiatTokenProxy` and `MintForwarder` contracts.

Recommendation: Consider inheriting `Ownable` and adding the `onlyOwner` modifier to `TokenFactory.deployToken` and `MintForwarderFactory.deployMintForwarder` functions.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.