Jorge Rodriguez

CYBERSEC CONTRACT AUDIT REPORT

Typhoon Cash - CTDSEC.com

Introduction

During January of 2021, Typhoon Cash engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. Typhoon Cash provided CTDSec with access to their code repository and whitepaper.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bugfree status. The audit documentation is for discussion purposes only.

I always recommend having a bug bounty program opened to detect future bugs.

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Typhoon Cash contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:
Github:
https://gist.github.com/TyphoonCash/cb6df51a4ef3d6233b003f00f5a8ec34

- ERC20Tornado.sol
- ERC20YFIRewards.sol
- MerkleTreeWithHistory.sol
- Tornado.sol
- UniswapLPReward.sol
- Verifier.sol

Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

Correctness of the protocol implementation [Result OK]

User funds are secure on the blockchain and cannot be transferred without user permission [Result OK]

Vulnerabilities within each component as well as secure interaction between the network components [Result OK]

Correctly passing requests to the network core [Result OK]

Data privacy, data leaking, and information integrity [Result OK]

Susceptible to reentrancy attack [Result OK]

Key management implementation: secure private key storage and proper management of encryption and signing keys [Result OK]

Handling large volumes of network traffic [Result OK]

Resistance to DDoS and similar attacks [Result OK]

Aligning incentives with the rest of the network [Result OK]

Any attack that impacts funds, such as draining or manipulating of funds [Result OK]

Mismanagement of funds via transactions [Result OK]

Inappropriate permissions and excess authority [Result OK]

Special token issuance model [Result OK]

Vulnerabilities

ISSUES

LOW

Old versions used in contracts/libraries.

Contract use old versions of openzeppelin/solidity, we recommend that always developer uses the latest versions to avoid security issues that can appear in old versions.

Examples:

```
pragma solidity 0.5.17;

library Hasher {
  function MiMCSponge(uint256 in_xL, uint256 in_xR) public pure returns (uint256 xL, uint256 xR);
}
```

LOW

Compiler versions are not fixed.

Solidity versions in source files are not fixed; we recommend fixing them to avoid unwanted behaviors of other versions which have not been developed for it.

LOW

Contract can be changed to library.

Contract MerkleTreeWithHistory could be turned into library. There is common pattern for this. The library defines a structure, calling contract defines storage variables of the structure type, and then passes these variables to the library by reference, so the library may read and update them.

Architecture

Since the images are too large and cannot be attached to the document due to loss of quality, we attach hyperlinks to be able to view them correctly.

Tornado: https://pasteboard.co/JKy1Q0t.png

ERC20YFIRewards: https://pasteboard.co/JKy26oo.png

MerkleTreeWithHistory: https://pasteboard.co/JKy2zVr.png

Summary of the Audit

The contracts are safe as there are no administration or upgrade options, so no one can change or eliminate any parameter.
The protocol parameters are governed by governance and zkSnarks have been established as layers of security.
After reviewing the contract we came to the conclusion that is safe to deploy.