Tetragaze

Smart contract audit

Tetragaze

July 8, 2021 | v 1.0

Pass

Tetragaze Security Team has concluded that there are no issues that can have an impact on contract security. The contract is well written and is production-ready.

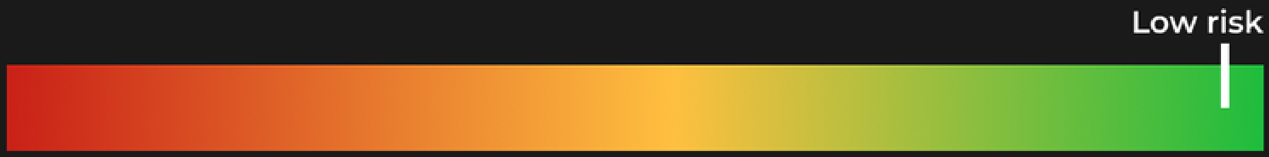
Score

94

Technical summary

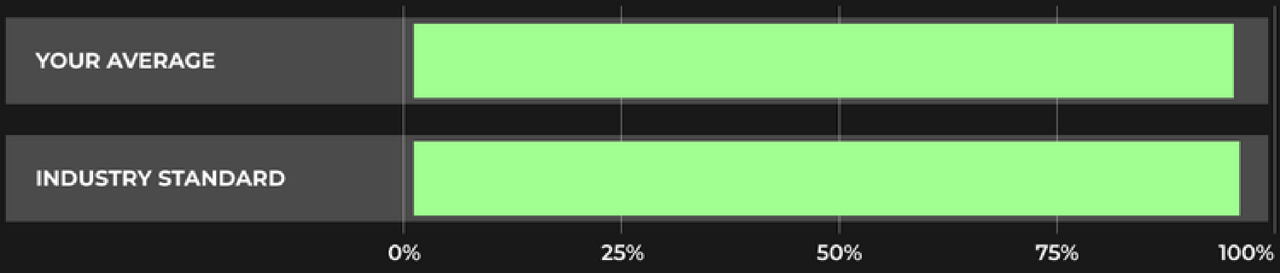
This document outlines the overall security of the Catharsis Stacking smart contracts, evaluated by Tetragaze Blockchain Security team.
The scope of this audit was to analyze and document the Catharsis Stacking smart contract codebase for quality, security, and correctness

Contract Status



There were no critical issues found during the audit.

Testable Code



Testable code is 94%, which is close to the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Tetragaze recommend that the Catharsis team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of content

Auditing Strategy and Techniques Applied

Requirements: FIP8 spec

During the review process, we made sure to carefully check that the token contract:

- Adheres to established Token standards and ensures effective implementation;
- Ensures that documentation and code comments accurately reflect the logic and behavior of the code;
- Ensures that token distribution is in line with calculations;
- Utilizes best practices to efficiently use gas, avoiding unnecessary waste;
- Employs methods that are safe from reentrance attacks and immune to the latest vulnerabilities;
- Follows best practices for code readability

Tetragaze team of expert pentesters and smart contract developers carefully evaluated the repository for security issues, code quality, and compliance with specifications and best practices. They conducted a line-by-line review, documenting any issues that were identified during the process. Special care was taken to ensure that the review was thorough and comprehensive.

1	Due diligence in assessing the overall code quality of the codebase.	2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.	4	Thorough, manual review of the codebase, line-by-line.

Summary

In summary, the smart contracts were well written and followed established guidelines. No vulnerabilities such as Integer Overflow or Underflow or Back-Door Entry were identified in the contract, however the use of other contracts could potentially lead to Reentrancy Vulnerability. During the initial audit, several issues were discovered, but all have now been resolved and thoroughly checked for accuracy.

Structure and Organization of Document

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged "Resolved" or "Unresolved" depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

Low

The issue has minimal impact on the contract's ability to operate.

Informational

The issue has no impact on the contract's ability to operate.

Complete Analysis

Wrong inputs can be passed to locks

Low

If the lockBatch() function is passed an empty array as input for unlockAt[] in the lock, the nested for loop will not be executed and the requirements within it will not be checked. This allows the Owner role to input address(0) as the account, an empty array [] as unlockAt[], and an array with any number of elements (including 0) as amounts[]. If this input is provided, the pendingReward() and claim() functions will revert.

```
 for (i; i < inputsLen; i++) {
   lockLen = _input[i].unlockAt.length;
   for (ii; ii < lockLen; ii++) {
     if (_input[i].account == address(0)) {
        require(false, "Zero address");
     } else if (
       _input[i].unlockAt.length != _input[i].amounts.length  ||
       _input[i].unlockAt.length > MAX_LOCK_LENGTH
     ) {
        require(false, "Wrong array length");
     } else if (_input[i].unlockAt.length == 0) {
require(false, "Zero array length");
     }
```

Recomendation:

The function is only accessible to the Owner, so it is important to ensure that the input values are correct before the Owner calls it. It is also advisable to verify that, in the case of a lock, the input array is not empty before adding it to the _balance[account] mapping.

Complete Analysis

Wrong inputs can be passed to locks

Low

In the lockBatch() function, the TokenVested event is emitted with the amount and address as parameters for each new lock. However, the event is being emitted before the for loop that calculates the amount has finished running, resulting in incorrect values for the amount. The issue is caused by a mistake in the statement that marks the end of the loop.

```
if (ii == l - ii) {
```

Recomendation:

Replace l - ii with l - 1 to determine the end of the loop.

Complete Analysis

Events Not Recorded for Significant Transactions

Low

It is challenging to monitor changes in off-chain liquidity fees because there is a missing event. To address this issue, an event should be emitted for significant transactions that utilize the lock() and lockBatch() functions.

Recomendation:

It is suggested to emit the event TokensVested when the lock() and lockBatch() functions are utilized and not being used.

Complete Analysis

Redundant Code

Low

If the unlockAt[] array has a length of 0, the for loop will not be executed and the condition inside the loop will not be checked.

```
} else if (_input[i].unlockAt.length == 0) {
require(false, "Zero array length");
}
```

Recomendation:

We suggest deleting this code.

Complete Analysis

Floating Pragma

Low

pragma solidity ^0.8.4;

It is important to deploy contracts using the same compiler version and flags that were used during thorough testing. Using the "pragma" function helps to prevent the accidental use of outdated compiler versions that may introduce bugs and negatively impact the contract system. By locking the pragma, it helps to ensure that the contract is not deployed with an outdated compiler version.

Recomendation:

Ensure that the chosen compiler version is locked in with the pragma version and take into account any known bugs for that version.

Complete Analysis

State variables that could be declared immutable

Low

uint256 public startAt
IERC20 public token

To save gas, the above constant state variable should be declared immutable.

Recomendation:

Include state variables that are immutable and do not alter after contact creation in the state variables.

Complete Analysis

Public function that could be declared external

Low

The public functions listed below are never used by the contract and should be marked as external to conserve gas:

getNextUnlock()
getNextUnlockByIndex()
getLocks()
getLocksLength()

Recomendation:

Functions that are never called from within the contract should use the external attribute.

Complete Analysis

Public function that could be declared external

Low

The public functions listed below are never used by the contract and should be marked as external to conserve gas:

getNextUnlock()
getNextUnlockByIndex()
getLocks()
getLocksLength()

Recomendation:

Functions that are never called from within the contract should use the external attribute.

We are thankful for the opportunity to collaborate with the Catharsis team.

This document's statements should not be taken as investment or legal advice, and the authors cannot be held responsible for any decisions made based on them.

It is suggested by the Tetragaze Security Team that the Catharsis team implement a bug bounty program in order to encourage external parties to scrutinize the smart contract further.