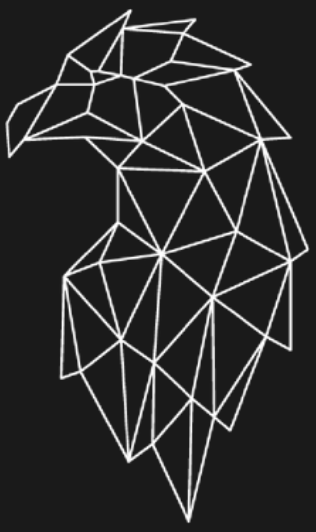
Tetragaze

Smart contract audit



YAAN

the polygon launchpad

Tetragaze

October 16, 2021 | v 2.0

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.

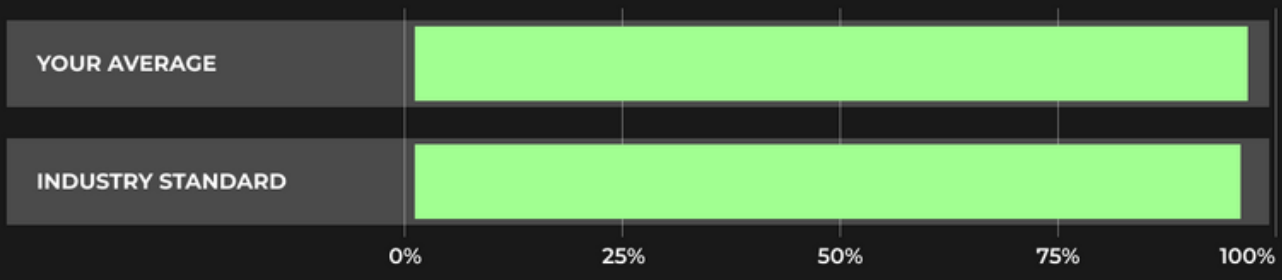
Technical summary

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

Contract Status

There were no critical issues found during the audit.

Testable Code



Testable code is 96%, 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 Polygon network's fast-paced and rapidly changing environment, we at Tetragaze recommend that the YAAN Launchpad 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

Smart contracts were found to be well-written and in compliance with guidelines. However, there were no instances of Integer Overflow, Underflow vulnerabilities, or Back-Door Entry found. However, the use of other contracts could potentially lead to a Reentrancy Vulnerability.

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

Range check while updating percent

Low

The setPercent() function does not contain a check to ensure that the _percent remains within the range of 0 to 100. Allowing an invalid percent value may result in unexpected behavior in the contract.

Recomendation:

We recommend adding a require statement that checks if (0<= _percent && _percent <= 100).

Complete Analysis

No check for address(0) before assignment:

Low

In the following functions:
- setAddressToChange()
- setAddressToSend()

There is no check to ensure that the incoming address in the parameters is not address(0).

Recomendation:

We recommend adding a require statement that checks if (incoming address != address(0)).

Complete Analysis

Lack of event emissions

Informational

In conclusion, the smart contracts reviewed were written with care and adhered to guidelines. No vulnerabilities such as Integer Overflow and Underflow or Back-Door Entry were identified in the contracts. However, it is important to consider that using other contracts may introduce the risk of Reentrancy Vulnerability.

Recomendation:

We recommend emitting an event to log the updated variables.

We are thankful for the opportunity to collaborate with the YAAN Launchpad 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 YAAN Launchpad team implement a bug bounty program in order to encourage external parties to scrutinize the smart contract further.