QuillAudits

Audit Report
December, 2024

For

iPaza Labs

Table of Content

Executive Summary 02

Number of Security Issues per Severity 03

Checked Vulnerabilities 04

Techniques and Methods 06

Types of Severity 07

Types of Issues 07

Medium Severity Issues 08

1. Abuse of permit functionality to block legitimate transactions 08

Informational Issues 09

2. Missing Check for Calldata length 09

Automated Tests 10

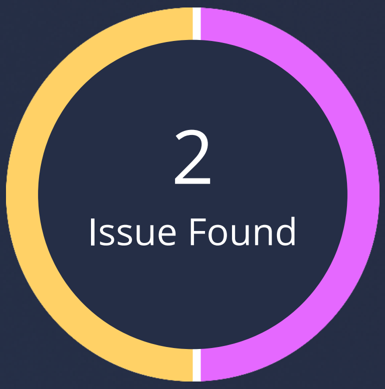
Closing Summary 10

Disclaimer 10

Executive Summary

Project Name	iPazaLabs
Project URL	https://pazalabs.io/
Overview	The Paza (ISPZ) project implements a sophisticated token trading system using a bonding curve mechanism. The core BondingCurve contract allows users to buy and sell PAZA tokens using USDC, with prices dynamically adjusted based on the token supply. It incorporates a custom BondingCurveCalculations library for complex mathematical operations, and includes features like tax accumulation and claiming.
Audit Scope	The scope of this Audit was to analyze the iPaza Smart Contracts for quality, security, and correctness.
Contracts In-Scope	BondingCurve.sol EthereumPaza.sol PolygonPaza.sol
Language	Solidity
Blockchain	Polygon,Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	16th December 2024 - 18th December 2024
Second Review	18th December 2024
Third Review	22nd January 2025 **https://github.com/rahul-ray30/PazaBondingCurve/commit/16d9dc6b6a6b2a0c878694003105e2d5b2893434**
Fourth Review	30th January 2025 **https://github.com/rahul-ray30/PazaBondingCurve/commit/745b2c462f1991a0e3d3c280aa458d7a1bf10f87**

Number of Security Issues per Severity



🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	**1**	0	**1**

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

Checked Vulnerabilities

- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Medium Severity Issues

1. Abuse of permit functionality to block legitimate transactions

Description

The buyWithExactTokenWithPermit and buyWithExactUsdcWithPermit leverages the permit function to approve the transfer of liquidity tokens.

However, this mechanism introduces a potential front-running vulnerability. An attacker can observe the transaction containing the permit call in the mempool and issue their own transaction with the same permit parameters but a different function call before the legitimate transaction is mined. This could result in the legitimate transaction failing due to the altered state.

Recommendation

It is recommended to handle the failure in the execution of permit as advised by OpenZeppelin.

https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#security_considerations

Status

Resolved

Fixed In

https://github.com/rahul-ray30/PazaBondingCurve/commit/6629c0de91dcc9ff772213d4b27a729f6a08df3b

Informational Issues

2. Missing Check for call data length

Path

contracts/BondingCurve.sol

Function Name

_msgSender() and _msgData()

Description

Best to have a check that the call data length is >= contextSuffixLength along with the isTrusted- Forwarder check in both _msgSender() and _msgData() functions. Also should cache the data length in a local variable and then use it to compare and perfrom operations.

Status

Resolved

Fixed In

https://github.com/rahul-ray30/PazaBondingCurve/commit/1e6d896a439e76fc781121da49c4a61efa4553dc

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of iPaza. We performed our audit according to the procedure described above.

One issue of informational and Medium Severity was found. Some suggestions, gas optimizations and best practices were also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in iPaza Labs smart contract. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of iPaza Labs smart contract. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the iPaza Labs team to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2024

For

iPaza Labs

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com