QuillAudits

Audit Report, August, 2024

For

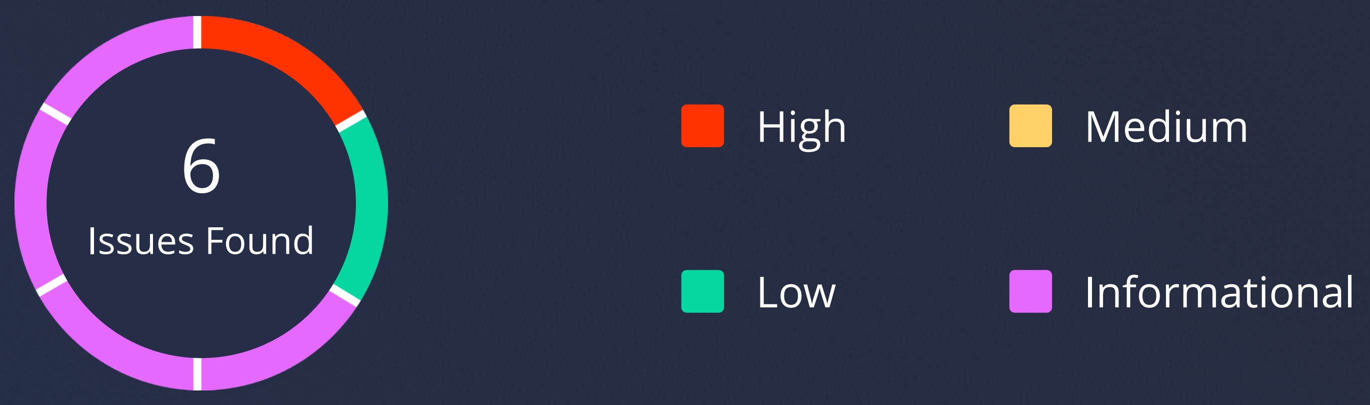
istakapaza

Table of Content

Executive Summary

Project Name	xPAZA Token
Project URL	https://github.com/rahul-ray30/Polygon-Bridge-Contracts/commit/a9ec1cbb327e963ff5880ccb1a40f7e3ba4b3d1d
Overview	XPAZA Token
Audit Scope	The Scope of the Audit was to check, security of ISPZ Codebase for vulnerabilities and code quality.
Contracts In-Scope	Branch: Main Contracts:- -FxERC20.sol -FxERC20ChildTunnel.sol
Language	Solidity
Blockchain	Polygon
Method	Manual Review, Automated tools,Functional testing
Review 1	25th July 2024 - 30th July 2024
Updated Code Received	2nd August 2024
Review 2	5th August 2024
Fixed In	9c86d9917b2a548cf33f9a61bea26627ebf1e043

Number of Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	0	4

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.
- Staking contract
  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, Hardhat, Solhint, Slither, Mythril, Solidity statistical 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.

High Severity Issues

1. Stuck Tax in BondingCurve Contract

Line **Function -** _transfer

92-107

```
function _transfer(address sender, address recipient, uint256 amount) internal
override {
    if (!isAMM[recipient]) {
        taxPercentage =
IBondingCurve(bondingCurve).calculateTaxPercent(amount);\
        uint256 tax = (amount * taxPercentage) / 100;
        uint256 taxedAmount = amount - tax;
        require(taxedAmount > 0, "Taxed amount must be greater than 0");

        super._transfer(sender, recipient, taxedAmount);

        super._transfer(sender, bondingCurve, tax); // @audit they should check
that the bonding curve address is set.

        emit TaxDeducted(sender, recipient, amount, tax);
    } else {
        super._transfer(sender, recipient, amount);
    }
    }
```

Description

The _transfer function deducts a tax when tokens are transferred to a recipient not listed as an Automated Market Maker (AMM). This tax is sent to the bondingCurve contract. However, the bondingCurve contract lacks a mechanism for withdrawing these collected taxes, resulting in tokens being permanently stuck. Additionally, the logic appears flawed as it taxes addresses that are whitelisted, which is likely unintended behavior.

Remediation

Correct Tax Logic: Review and correct the condition that applies the tax. Ensure that only non-whitelisted addresses are taxed, or implement logic that reflects the intended behavior.
Check BondingCurve Address: Before transferring tax to the bondingCurve, ensure that the address is set and valid to avoid errors during transfer.

require(bondingCurve != address(0), “Bonding curve address is not set”);

Status

Resolved

Low Severity Issues

2. Missing FX Manager Change Function

Description

The current implementation allows for changing the admin address through the changeAdminAddress function. However, there is no similar function to change the _fxManager address, which could be necessary for contract flexibility and maintenance.

Recommendation

To enhance the flexibility and security of the contract, implement a function that allows for changing the _fxManager address. This function should have appropriate access controls to ensure that only authorized accounts can make this change.

Status

Acknowledged

(May lead to many changes in contracts, which can lead to errors in existing contracts).

Informational Issues

3. Duplicated require()/revert() Checks Should Be Refactored To A Modifier Or Function

Line	**Contract -** FxERC20.sol
45	require(_msgSender() == admin, “Invalid Admin”); // @note can be changed to an modifier.
51	require(_msgSender() == admin, “Invalid Admin”);
57	require(_msgSender() == admin, “Invalid Admin”);
62	require(_msgSender() == admin, “Invalid Admin”);
78	require(_msgSender() == _fxManager, “Invalid sender”); // @audit need to change becuase it won’t allow contract to be deployed.
83	require(_msgSender() == _fxManager, “Invalid sender”);
88	require(_msgSender() == _fxManager, “Invalid sender”);

Status

Resolved

4. Missing checks for address(0) when assigning values to address state variables

Line	Contract - FxERC20.sol
35	_fxManager = fxManager_;
36	_connectedToken = connectedToken_;
37	admin = _admin;
46	bondingCurve = _bondingCurveContract;
63	admin = _newAdmin;

Status

Resolved

5. Contract Is Not Modified According To Documentation

Line	Contract - FxERC20ChildTunnel.sol
15	string public constant SUFFIX_NAME = “ TOKEN”;
16	string public constant PREFIX_SYMBOL = “X”;

Description

The contract is not modified according to what changes are mentioned in the documentation here.

Status

Resolved

6. Unused array

Line	Contract - BasicMetaTransaction.sol
73	uint256[50] private __gap;

Status

Resolved

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 the xPAZA. We performed our audit according to the procedure described above. One high , one low and few Informational Severity Issue found,which the ISPZ Team Resolved and Acknowledged few issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in xPAZA smart contracts. 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 xPAZA smart contracts. 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 xPAZA 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
August, 2024

For

Sistakapaza

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com