QuillAudits

Audit Report
March, 2022

For

PAXO

Contents

Overview

Paxo by Paxo Finance

PAXO is a decentralized money market protocol which opens up investment loan options in the DeFi space that allows users to invest in digital assets through multi-pool borrowing.

Scope of Audit

The scope of this audit was to analyze the Paxo Finance Smart Contract's codebase for quality, security, and correctness.

Commit: 5a95229bb2c5fc1729ba68a8bee0eb4cefcb0246

Fixed In: 7e5a4182b762c8edec9e49cf21d2b9ee6ed2960e

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- 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 analysed 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
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 analysed, 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 behaviour 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
Mythril, Slither, C4udit , Solhint

Issue Categories

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.

Risk-level	Description
High	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	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	Low-level severity issues can cause minor impact and or 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 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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	1
Closed	0	2	0	0

Issues Found – Code Review / Manual Testing

High severity issues

No issues found

Medium severity issues

1. **Re-entrancy is allowed in the borrowAndBuy function.**

   In PToken.sol [#L2719] borrowAndBuy() function allows the attacker to re-enter into the function again and that gives the attacker an opportunity to behave maliciously.

   Recommendation
   It is always a best practice to use OZ's re-entrancy guard in functions which lead the movement of any type of funds and has interaction with the third party contracts

   Status: Closed

2. **Incorrect estimation of the borrowed amount would lead to failure of borrow and buy functionality.**

   In PToken.sol[#L2742]

```
if (getCashPrior() < borrowAmount) {
        return fail(Error.TOKEN_INSUFFICIENT_CASH,
FailureInfo.BORROW_CASH_NOT_AVAILABLE);
    }
```

   Above if statement checks whether the position has the enough cash to cover the borrowAmount or not but this check happen after the partialAmount get transferred to the position that means it also considered that amount as borrowAmount while the contract needs partialAmount + borrowAmount balance of underlying token to successfully execute the swap. This would lead to failure of the swap transaction if the contract has not enough prior cash to fill the borrowAmount.

Recommendation
It is recommended to move the if statement before transferring the partialAmount (i.e before [#L2736]) to get correct value of prior cash and fail with correct error code if available cash wouldn't be able to cover the borrowAmount.

Status: Closed

Low severity issues

No issues found

Informational issues

3. Unnecessary function

In PToken.sol[#L2707] increaseUniswapAllowance() is a non-essential function as uniswap router already has type(uint256).max allowance that get set at [#L2476], Which would be very unrealistic to burn out unless there is a bug in the contract.

Recommendation
It is recommended to completely remove the function from the contract.

Status: Acknowledged

Functional Testing Results

The complete functional testing report has been attached below:
Paxo test case

Some of the tests performed are mentioned below:
- ☑ should be able to enter into the markets.
- ☑ should be able to exit from the market.
- ☑ should be able to get the list of markets an account is in.
- ☑ should be able to mint and redeem pToken.
- ☑ should be able to borrow and liquidate pToken.
- ☑ should be able to seize assets.
- ☑ should be able to set a close factor.
- ☑ should be able to set the price oracle.
- ☑ should be able to set collateral factors.
- ☑ should be able to set pending admin.
- ☑ should be able to pause mint, borrow, redeem, transfer & seize.
- ☑ should be able to claim comp.
- ☑ should be able to get the exchange rate.
- ☑ should be able to borrow and buy.
- ☑ should be able to set the buy factor
- ☑ should be able to accrue interest.
- ☑ should be able to get a borrowRate per block.
- ☑ should be able to get a supplyRate per block

Closing Summary

Some issues of Medium & Informational severity were found,Almost most of the issues are now fixed. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Paxo Finance platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Paxo Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

PAXO

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com