QuillAudits

Audit Report
April, 2021

Contents

Introduction

This Audit Report mainly focuses on the overall security of PYRToken contract. With this report, we have tried to ensure the reliability and correctness of their smart contract by a complete and rigorous assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed rigorous testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is well structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find any potential issues like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested and this included -

- Analyzing the complexity of the code in-depth and detailed, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

Audit Details

Project Name: PYR
Website/Etherscan Code (Testnet):
PYR Token: 0x93b04b70189ba269d39421c03eb83384883b637c

Languages: Solidity
Platforms and Tools: Remix IDE, Solhint, VScode, Slither, Mythril

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient and working according to the specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity

Issue Categories

Every issue in this report was assigned a severity level from the following:

High severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	1	3
Closed	0	0	0	0

Manual Audit

High level severity issues

No issues found

Medium level severity issues

No issues found

Low level severity issues

1. It is a good practice to lock the solidity version for a live deployment (use **0.5.17** instead of **^0.5.17**). Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

```
1.    pragma solidity ^ 0.5.17;
```

Recommendations

1. In the deposit() function lines 555 to 558 can be replaced with:
   _mint(user, amount);

```
547     function deposit(address user, bytes calldata depositData) external {
548         require(
549             msg.sender == childChainManagerProxy,
550             "You're not allowed to deposit"
551         );
552
553         uint256 amount = abi.decode(depositData, (uint256));
554
555         _totalSupply = _totalSupply.add(amount);
556         _balances[user] = _balances[user].add(amount);
557
558         emit Transfer(address(0), user, amount);
559     }
```

2. You can include the **_burn()** internal function in the **ERC20** contract and use it in the **withdraw()** function and replace the entire function logic with: **_burn(msg.sender, amount);**

```
561     function withdraw(uint256 amount) external {
562         _balances[msg.sender] = _balances[msg.sender].sub(
563             amount,
564             "ERC20: burn amount exceeds balance"
565         );
566         _totalSupply = _totalSupply.sub(amount);
567
568         emit Transfer(msg.sender, address(0), amount);
569     }
```

3. You can make the **deployer** address public to keep track of the address.

Automated Testing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. No violations were detected by Solhint, it is recommended to use **Solhint's npm package** to lint the contract.

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities

```
luv@luv:~/Desktop$ myth a PYRToken.sol
The analysis was completed successfully. No issues were detected.
```

Mythril detected no issues.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

```
INFO:Detectors:
PYRToken.constructor(address,address)._childChainManagerProxy (PYRToken.sol#530) lacks a zero-check on :
        - childChainManagerProxy = _childChainManagerProxy (PYRToken.sol#531)
PYRToken.constructor(address,address)._owner (PYRToken.sol#530) lacks a zero-check on :
        - deployer = _owner (PYRToken.sol#532)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Variable ERC20._balances (PYRToken.sol#312) is not in mixedCase
Variable ERC20._totalSupply (PYRToken.sol#316) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (PYRToken.sol#25)" inContext (PYRToken.sol#13-28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
PYRToken.decimals (PYRToken.sol#526) should be constant
```

```
PYRToken.name (PYRToken.sol#524) should be constant
PYRToken.symbol (PYRToken.sol#525) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
totalSupply() should be declared external:
    - ERC20.totalSupply() (PYRToken.sol#321-323)
balanceOf(address) should be declared external:
    - ERC20.balanceOf(address) (PYRToken.sol#328-330)
transfer(address,uint256) should be declared external:
    - ERC20.transfer(address,uint256) (PYRToken.sol#340-343)
allowance(address,address) should be declared external:
    - ERC20.allowance(address,address) (PYRToken.sol#348-354)
approve(address,uint256) should be declared external:
    - ERC20.approve(address,uint256) (PYRToken.sol#363-366)
transferFrom(address,address,uint256) should be declared external:
    - ERC20.transferFrom(address,address,uint256) (PYRToken.sol#380-395)
increaseAllowance(address,uint256) should be declared external:
    - ERC20.increaseAllowance(address,uint256) (PYRToken.sol#409-419)
decreaseAllowance(address,uint256) should be declared external:
    - ERC20.decreaseAllowance(address,uint256) (PYRToken.sol#435-448)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:PYRToken.sol analyzed (5 contracts with 72 detectors), 16 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the PYRToken contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

The use case of the smart contract is simple and the code is relatively small. Altogether, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. Overall the code is well written and readable with no high and medium level concerns, but can be improved according to Solidity's style guide which is recommended to be fixed before implementing a live version.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com