SECURITY AUDIT REPORT FOR

QUAIL FINANCE

June 24, 2024

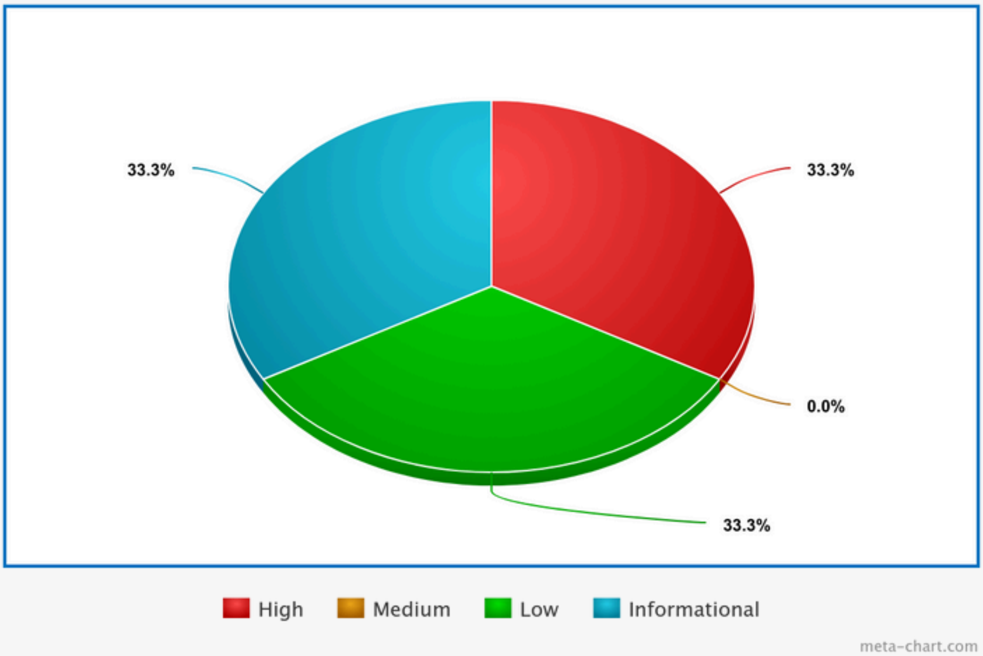
Executive Summary

This smart contract audit report is prepared for *QuailFinance.sol, IBlast.sol, IBlastPoints.sol, IERC20Rebasing.sol, proxy.sol*, smart contracts of *Quail Finance*, after a successful functionality testing, source code review and black box application security penetration exercise.

Type Smart Contract	Languages Solidity
Timeline June 19, 2024 to June 24, 2024	Methodology White Box Testing

Change log

June 07, 2024 - Requirement received via github - commit history - ec7a5986c94801e7912791dd064312d33f77cc66
June 19, 2024 - Beginning of first phase of audit.
June 24, 2024 - Completion of first phase of audit, report submission.



High Risk Vulnerabilities that can be exploited publicly, workaround or fix/ patch available by the vendor.	1	**Low Risk** Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Vulnerability observed may not have a high rate of occurrence. Patch/ workaround released by the vendor.	1
Medium Risk Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Patch/ workaround not yet released.	0	**Informational Risk** Vulnerabilities that have the minimum impact on the system.	1
Undetermined Risk Vulnerabilities that has uncertain impact.	0		

Summary of Findings

Vulnerability	Risk	Status
1. Pot participant can remove other participants from the pot	High	Open
2. Use of floating pragma	Low	Open
3. Missing Natspec and Incorrect comments uses	Informational	Open

Objective

The objective of the security audit of smart contract of *Quail Finance* was to assess the state of security and uncover vulnerabilities in the smart contract and provide with a detail report comprising remediation strategy and recommendations to help mitigate the identified vulnerabilities and risks during the activity.

The assessment was conducted using two methods; source code review and black box testing.

Scope & Requirements

The technical scope for the conducted application security testing activity was restricted to:

- QuailFinance.sol
- IBlast.sol
- IBlastpoints.sol
- IERC20Rebasing.sol
- QuailFinance.sol
- proxy.sol

Files url

https://github.com/Quail-Finance/QuailContracts/tree/master/contracts
Git commit hash - ec7a5986c94801e7912791dd064312d33f77cc66

Provided Date

June 07, 2024; Email Access given via Github.

Methodology

White Box Testing/Source Code Review

Understand Program Specification

Understanding programming language used and coding practices followed is a key in performing efficient source code review solution. Our security researchers begin source code review by first understanding the program specification.

Obtain and Review Source Code

Our security researchers then co-ordinate with the development team to obtain the source code and start reviewing the codes line by line to identify flaws.

Indentify Flaws

We apply a rigorous approach to identify flaws. That is we adhere to strict standard practices and retest major functions again to make sure we don't miss a single loophole.

Reporting

A documentation of where the flaws has been found and how the patches can be done will be submitted to the development team for a fix.

Findings from the assessment

We found the following findings after performing the security audit of the smart contracts of *Quail Finance* through source code review and black box testing.

Findings from Manual Testing

1. Pot participant can remove other participants from the pot

Description

In QuailFinance.withdrawFromPot(), there is no access checks implemented to verify whether a given participant (represented by the index in ether pot.winners or pot.participants array) can be removed by other participants without their consent.

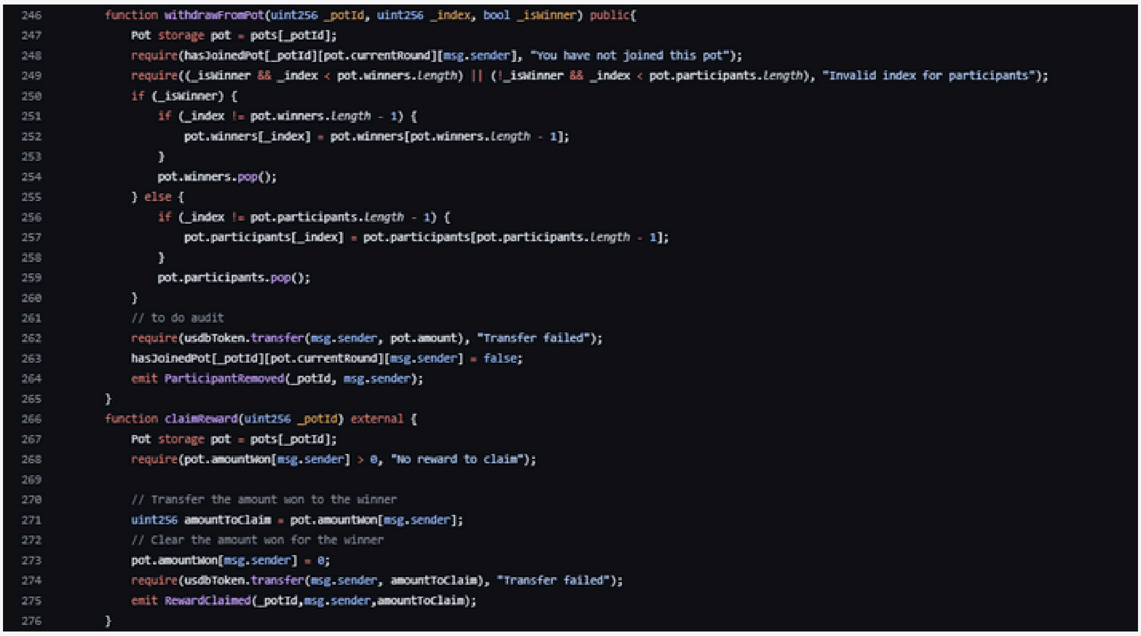
Impact

High

Risk	Status
High	Open

Proof of Concept

contracts/QuailFinance.sol:L#238-L#255



```
    function withdrawFromPot(uint256 _potId, uint256 _index, bool _isWinner) public{
        Pot storage pot = pots[_potId];
        require(hasJoinedPot[_potId][pot.currentRound][msg.sender], "You have not joined this pot");
        require((_isWinner && _index < pot.winners.length) || (!_isWinner && _index < pot.participants.length), "Invalid index for participants");
        if (_isWinner) {
            if (_index != pot.winners.length - 1) {
                pot.winners[_index] = pot.winners[pot.winners.length - 1];
            }
            pot.winners.pop();
        } else {
            if (_index != pot.participants.length - 1) {
                pot.participants[_index] = pot.participants[pot.participants.length - 1];
            }
            pot.participants.pop();
        }
        // to do audit
        require(usdbToken.transfer(msg.sender, pot.amount), "Transfer failed");
        hasJoinedPot[_potId][pot.currentRound][msg.sender] = false;
        emit ParticipantRemoved(_potId, msg.sender);
    }
    function claimReward(uint256 _potId) external {
        Pot storage pot = pots[_potId];
        require(pot.amountWon[msg.sender] > 0, "No reward to claim");

        // Transfer the amount won to the winner
        uint256 amountToClaim = pot.amountWon[msg.sender];
        // Clear the amount won for the winner
        pot.amountWon[msg.sender] = 0;
        require(usdbToken.transfer(msg.sender, amountToClaim), "Transfer failed");
        emit RewardClaimed(_potId,msg.sender,amountToClaim);
    }
```

Remediation

Adding a check to verify if the Index of the participant to be removed belongs to msg.sender can be a fix of this issue.

For Example:

```
require (msg.sender == pot.participants[_index] || msg.sender == pot.winners[_index], "Not authorized")
```

2. Use of floating pragma

Description

All the contracts in scope use floating pragma. The use of floating pragma implies that the compiler will not use a specified version which brings inconsistency and unexpected behaviour in the future.

Impact

Brings inconsistency

Risk	Status
Low	Fixed

Proof of Concept

Line#2 of the following:

- QuailFinance.sol
- IBlast.sol
- IBlastpoints.sol
- IERC20Rebasing.sol
- QuailFinance.sol
- proxy.sol

Remediation

Remove Caret (^) from pragma and specify solidity version.

```
pragma solidity 0.8.23;
```

3. Missing NatSpec and Incorrect comments uses

Description

It is identified that NatSpec format is not followed for comments in the contract. Furthermore, it is identified that the comments are incomplete and not present for all the functions in the contract.

Impact

Reduces code readability and maintanability. Lack of proper documentation via code comments also reduces user trust in the contract.

Risk	Status
Informational	Fixed

Remediation

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI).

Conclusion

We found a total of 4 vulnerabilities in *QuailFinance.sol, IBlast.sol, IBlastPoints.sol, IERC20Rebasing.sol, proxy.sol,* out of which 1 possessed high risk, 1 possessed low risk and 1 possessed informational risk.

In addition to this audit, we encourage the team to organize a public vulnerability disclosure program and adopt other necessary measures to minimize cyber risks associated with smart contracts.

Disclaimer

Considering the evolving nature of risks and vulnerabilities associated with solidity language, smart contracts and Etherium network, ReconwithMe cannot wholly guarantee that the contract will be free of vulnerabilities after the audit. We encourage the team to organize a public vulnerability disclosure program and adopt other necessary measures to minimize cyber risks associated with smart contracts.

About ReconwithMe

ReconwithMe is a cyber security firm dedicated to providing vulnerability management solutions. ReconwithMe has been trusted by industries ranging from fintech to blockchain and SAAS to Ecommerce. With cyber attacks rising rapidly, businesses require cyber security solutions more than ever. Our tailor-made vulnerability management solutions help prevent cyber attacks. Our team consists of self-taught professional cyber security practitioners who follow international standard security protocols and possess strong work ethics. Our team has been recognized by tech giants such as Facebook, Microsoft, Sony, Etsy and others for contributing to make them more secure.

ReconwithMe Pvt. Ltd.
Address - Kupondole, Lalitpur, Nepal
Email - support@ReconwithMe.io