#Hashlock.

JuicyFi

SMART CONTRACT

Security Audit

Performed on Contracts:

JuicyFiShares.sol

Platform

ETH / ARBITRUM

hashlock.com.au

MARCH 2024

Table of Contents

Executive Summary	4
Project Context	4
Audit scope	6
Security Rating	7
Intended Smart Contract Behaviour	8
Code Quality	9
Audit Resources	9
Dependencies	9
Severity Definitions	10
Audit Findings	11
Centralisation	14
Conclusion	15
Our Methodology	16
Disclaimers	18
About Hashlock	19

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

The JuicyFi team partnered with Hashlock to conduct a security audit of their JuicyFiShares.sol smart contract. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts were secure.

Project Context

JuicyFi is a DeFi dApp that allows users to buy and sell shares.

Project Name: Template
Compiler Version: ^0.8.2 <0.9.0
Website: https://www.juicyfi.com
Logo:

Visualised Context:

Project Name
JuicyFi

Launch Date
21-05-2024

Compiler Version
>=0.8.2 <0.9.0;

Language
Solidity

Network
ETHEREUM / ARBITRUM

Token Ticker
TEMP20

Audit scope

We at Hashlock audited the solidity code within the JuicyFi project, the scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	**Template Protocol Smart Contracts**
Platform	**Ethereum / Solidity / Arbitrum**
Audit Date	**March, 2024**
Contract 1	JuicyFiShares.sol

Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Secure"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces, and the protocol uses a list of Open Zeppelin contracts. We initially identified some significant vulnerabilities that have since been addressed.

The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on chain monitoring technology.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit Findings section.

Hashlock found:

2 High severity vulnerabilities

1 Medium severity vulnerabilities

Caution: *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

Intended Smart Contract Behaviour

Claimed Behaviour	Actual Behaviour
JuicyFiShares.sol - Allow users to buy and sell shares	**Contract achieves this functionality.**

Code Quality

This Audit scope involves the smart contracts of the JuicyFi project, as outlined in the Audit Scope section. All contracts, libraries and interfaces mostly follow standard best practices and to help avoid unnecessary complexity that increases the likelihood of exploitation, however some refactoring is required.

The code is very well commented and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the JuicyFi projects smart contract code in the form of Github access.

As mentioned above, code parts are well commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in understanding the overall architecture of the protocol.

Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well known industry standard open source projects.
Apart from libraries, its functions are used in external smart contract calls.

Severity Definitions

Significance	Description
High	High severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community.
Medium	Medium level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low level vulnerabilities are areas that lack best practices that may cause small complications in the future.
Gas	Gas Optimisations, issues and inefficiencies

Audit Findings

High

[H-01] JuicyFiShares.sol#buyShares and sellShares - Anyone can steal the fees

Description

There are certain fees that should be payed for each user when buying shares in buyShares() or in sellShares(). These fees are transferred to the fee receiver addresses but as they are parameters of the function any user can manipulate the receiver.

Vulnerability Details

The buyShares() and sellShares() function receive as parameters:

```
address sharesSubject, address referralAddress, address referral2LevelAddress
```

This parameters can be set to the same user address and he will be receiving the fees.

```
function buyShares(address sharesSubject, address referralAddress, address referral2LevelAddress, uint256 amount, uint256 orderId) public payable
```

```
function sellShares(address sharesSubject, address referralAddress, address referral2LevelAddress, uint256 amount, uint256 orderId) public payable
```

Impact

Any user can steal the fees.

Recommendation

Set these addresses in the constructor and not as function parameters.

Status

Fixed

[H-02] JuicyFiShares.sol - Fees can be updated up to 100% and without grace period

Description

There are several fees like protocolFeePercent, subjectFeePercent, referralFeePercert, referral2LevelFeePercent. These fees implement methods for updating their values without any cap and grace period.

Vulnerability Details

The owner can call any of the methods for updating the fees and set them for example to 100% without a grace period, an scenario where the owner frontruns a large transaction to get 100% of the fee can also take place.

```
function setProtocolFeePercent(uint256 _feePercent) public onlyOwner {
        protocolFeePercent = _feePercent;
    }
```

```
function setSubjectFeePercent(uint256 _feePercent) public onlyOwner {
        subjectFeePercent = _feePercent;
    }
```

```
function setReferralFeePercent(uint256 _feePercent) public onlyOwner {
        referralFeePercent = _feePercent;
    }
```

```
function setReferral2LevelFeePercent(uint256 _feePercent) public onlyOwner {
        referral2LevelFeePercent = _feePercent;
    }
```

Impact

Owner can change fees to 100% without the users knowing it.

Recommendation

Introduce a maximum value for the fees and it is also recommendable to introduce a grace period.

Status

Fixed

Medium

[M-01] JuicyFiShares.sol#sellShares - Unneeded require

Description

The sellShares function implements a Require function to revert if the caller address is the same as sharesSubject and if it is trying to sell the last share. This is not needed, as there is no risk of selling the last share.

Vulnerability Details

```
if(msg.sender==sharesSubject)

require(sharesBalance[sharesSubject][msg.sender] > 1, "E06 : Creator cannot sell the last share");
```

Impact

The code will revert in the mentioned scenario.

Recommendation

Remove the above statement.

Status

Acknowledged

Centralisation

The project values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on internal team responsibility.

Conclusion

After Hashlocks analysis, the JuicyFi project seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved and acknowledged. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits are to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:

In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our methodologies include manual code analysis, user interface interaction, and whitebox penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contracts details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Hashlock is not responsible for the safety of any funds, and is not in any way liable for the security of the project.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

About Hashlock

Hashlock is an Australian based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3 oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au
Contact: info@hashlock.com.au


#Hashlock.