QuillAudits

Lite Technical Assessment

For

FIRE LAUNCH

Table of Content

Project Information

Project Name	Firelaunch
Project Website	*https://launchpad-web-six.vercel.app/landing*
Date	18th April 2024 - 24th April 2024
Executive Summary	FireLaunch is Liquidity Bootstraped Protocol, it is a fork on Jford.
Source Code	https://github.com/firelaunchio/LBP/blob/master/contracts/LiquidityPool.sol

Project Maturity Assessment

1. Code Completion

Are all smart contracts for the core functionality written and finalized?

- While the code has inline comments, a crucial comprehensive document explaining the purpose and functionalities of each smart contract is missing. Additionally, a separate document detailing the mathematical aspects is required for auditors to fully understand the logic.

Are unit tests covering all critical functionalities implemented?

- Unit tests were implemented properly.

2. Documentation

Is there comprehensive documentation explaining the purpose and functionality of each smart contract? Does the documentation include detailed comments within the code itself? Is the documentation clear, concise, and easy to understand for auditors?

- While the code has inline comments, a crucial comprehensive document explaining the purpose and functionalities of each smart contract is missing. Additionally, a separate document detailing the mathematical aspects is required for auditors to fully understand the logic.

3. External Dependencies

Are all external libraries and dependencies used in the smart contracts well-established and secure?

- The code utilizes well-regarded libraries like OpenZeppelin, Sablier, and Solady, known for their security focus. While this is positive, it's essential to remember that relying solely on secure libraries isn't enough. Thorough testing of the entire system, particularly how these libraries interact within your specific smart contracts, remains crucial.

Are the versions of these dependencies up-to-date and free of known vulnerabilities?

- Use of the latest version of OpenZeppelin Contracts (v 5.0) is recommended.

4. Best Coding Practices

Does the code adhere to secure coding practices such as access control best practices (e.g., least privilege), proper error handling, and random number generation best practices?

- Yes.

Are reentrancy vulnerabilities mitigated?

- Yes, the Check Effect Interaction pattern is implemented.

Is the code written in a gas-efficient manner? (See Gas Optimization section).

- Yes, but it can be further improved.

5. Gas Optimization

Have techniques been employed to optimize gas consumption (e.g., using libraries for common functions, avoiding unnecessary storage writes)?

- Needs to be improved. Here’s a list of potential gas optimizations:

- Mark Constructor as payable
- Cache the length of the array outside of the loop
- Make addresses (if any) as immutable if they are not intended to be changed.

Code Assessment Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after **v0.5.X** (Currently using solidity **v >= 0.8.6**).
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before **v0.8.X**
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma.
SWC-104	Unchecked Call Return Value	Not Vulnerable	**call()** is not used.
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and required validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	**selfdestruct()** is not used anywhere.
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable.

SWC ID	SWC Checklist	Test Result	Notes
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, **v0.5.0**
SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like **block.blockhash()**, **msg.gas**, **throw**, **sha3()**, **callcode()**, **suicide()** are in use.
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	**tx.origin** is not used anywhere in the code.
SWC-116	Block values as a proxy for time	Not Vulnerable	**Block.timestamp** is not used.
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere.
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the **constructor** keyword rather than functions.

SWC ID	SWC Checklist	Test Result	Notes
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version **0.6.0**
SWC-120	Weak Sources of Randomness from Chain Attributes from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found.

Top Level Smart Contract Issues Identified

- The **updateRecipients()** function first deletes the entire recipient array and implements a new array with some logic implementation. This method of array updation is prone to errors because a single mistake in the new implementation might have unintended consequences later on.
- The for loop iterates over the recipient array for fee percentage, which also increases the possibility of Denial-Of-Service, provided that the array becomes exceedingly large and outstretched the block gas limit.
- It is recommended that a mapping approach be used instead of an array for recipient management or that the number of recipients accommodating the array be limited.
- If the contract intends to do so, the recipient array percentage should also be checked against its sum, i.e., 100 percent.

Top Level dApp Pentest Issues Identified

- **Clickjacking POC**

  https://clickjacker.io/test?url=https://launchpad-web-six.vercel.app/landing#google_vignette

  Clickjacking is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website.

- **Missing Security Headers POC**

  https://www.serpworx.com/check-security-headers/?url=https%3A%2F%2Flaunchpad-web-six.vercel.app%2Flanding

Note: The Firelaunch team has resolved Both Pentest Issues

Social Engineering Checks

Testing Parameters
Discord Phishing - Not Applicable.
Checking web page source code in browser developer mode - Not Applicable.
Suspicious links on Linktree - Not Applicable.
Web Spoofing / Phishing - Not Applicable.
Domain Name and Subdomains - Not Applicable.
Domain Certificate - Not Applicable.
Telegram Phishing - Not Applicable.
Checking the remotely loaded JavaScript script in the web page source code - Not Applicable.
Server - Not Applicable.
Login Panel Disclosed - Not Applicable.

Readiness Assessment

Based on the evaluation of the aforementioned criteria, this report concludes that the FireLaunch smart contracts are:

- [x] Ready for a full-fledged security audit.
- [ ] ~~Partially ready for a security audit. Further work is needed on [Specify areas for improvement].~~
  - [ ] ~~The protocol needs to undergo a rigorous testing.~~
  - [ ] ~~A thorough documentation stating the purpose of contracts is needed.~~
- [ ] ~~Not yet ready for a security audit. Significant development and testing are required.~~

Disclaimer

This report presents the findings of a light technical due diligence conducted on the provided codebase. It is intended to identify potential areas for improvement in coding practices and security but does not constitute a comprehensive security audit.

While this review has focused on best practices and secure coding principles, it is important to acknowledge that vulnerabilities may still exist within the codebase. A more thorough security audit is strongly recommended before deploying the code to a production environment (mainnet).

Quillaudits shall not be held liable for any security breaches or vulnerabilities that may be discovered after the completion of this light technical due diligence.

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

Lite Technical Assessment

For

FIRE LAUNCH

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com