QuillAudits

Audit Report
November, 2024

For

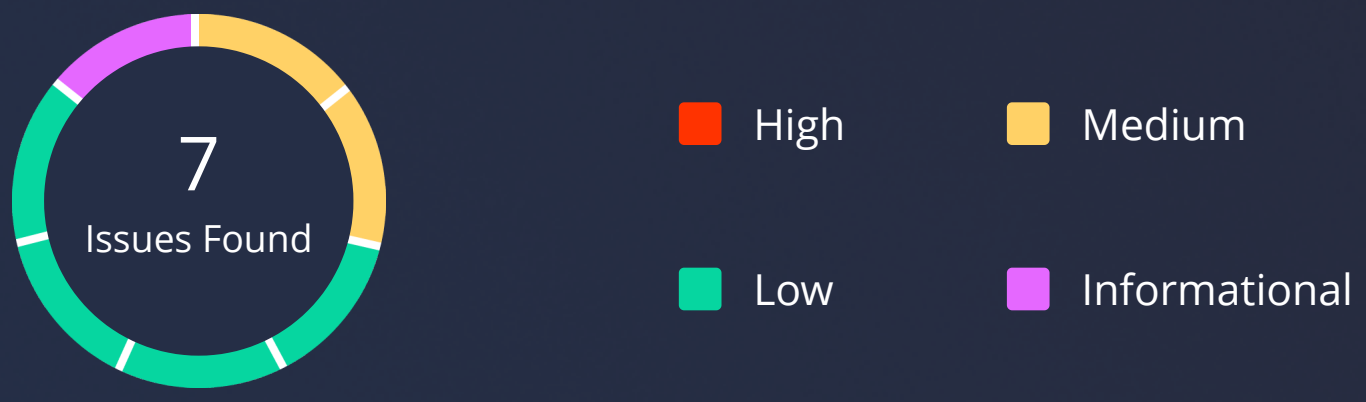
LAXCE

Table of Content

Executive Summary

Project Name	LAXCE
Project URL	https://www.laxce.com/
Overview	The LAXCE project codebase contains two contracts • An ERC20 token contract with a blacklist to restrict selected accounts from interacting with the contract, and • A crowdsale contract with variable rates that allows users buy tokens using an owner-specified token (e.g. USDT). Ownership of these contracts can not be renounced.
Audit Scope	https://github.com/LaxceL2/Laxce-token
Contracts In-Scope	- Laxce.sol - LaxceCrowdSale.sol
Commit Hash	1b05f683bc07afee68f5264215f82f636c9890ea
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	29th October 2024 - 31st October 2024
Updated Code Received	20th November 2024
Second Review	20th November 2024
Fixed In	https://github.com/LaxceL2/Laxce-token/tree/quill-audit-fix-v1

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	4	1

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- 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
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- 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 analyzed 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

A 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 analyzed, 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 behavior 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

Hardhat, Slither, Remix IDE

Types of Severity

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.

High Severity Issues

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 Severity Issues

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 Severity Issues

Low-level severity issues can cause minor impact and 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 four 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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Medium Severity Issues

1. USDT stored can be inaccurate

Path

LaxceCrowdSale.sol

Description

The current contract architecture allows for the USDT contract to be changed without the following variables, struct and mapping being correctly updated:

```
uint256 public rate;
uint256 public usdtRaised;

struct UserInfo {
        uint256 usdtContributed;
        uint256 laxceRecieved;
    }

mapping(address => UserInfo) public users;
```

If the contract owner updates the usdt contract by calling _changeUsdtToken() before the end of the token sale, the value of usdtRaised would become inaccurate because it now holds the value of the old tokens contributed as well as the new tokens. Similarly, the UserInfo struct and users mapping which keeps track of the user's individual contributions would become inaccurate.

Recommendation

- Restrict the contract to a single token used for contributions, or allow multi-token contributions.
- Remove changeUsdtToken() function to avoid this error

Status

Resolved

Fixed in Commit

7fe420b: Function removed

2. Centralization risk

Description

The owner has multiple rights reserved within the contract, they are able to start and stop sales, adjust rates, and withdraw tokens without any restrictions. If a single user's private key is responsible for handling these transactions and this key gets compromised, users would be opened up to griefing attacks and loss of funds.

Recommendation

It is advisable to use a multi-signature wallet for crucially owned accounts, and also implement a timelock/governance system to ensure users are abreast of protocol changes via proposals or a delayed implementation of the changes.

Status

Acknowledged

Client's Comments

Owner will be using a multi-signature wallet or timelock later on.

Low Severity Issues

3. buyTokens() reverts when buy limit is active

Path

LaxceCrowdSale.sol#L215-216

Function

initialize()

Description

The contract implements a limit to the amount of USDT a user can use in a single purchase described in the initialize function:

```
minimumBuyLimit = 5000;
maximumBuyLimit = 500000;
```

These values do not take into account the decimals of the USDT token (6 decimals on ETH) thereby making the equivalent amounts to be less than viable purchase amounts.

```
minimumBuyLimit = 5000;      // 0.005 USDT
maximumBuyLimit = 500000;  // 0.5 USDT
```

Recommendation

Consider updating the buy limits to factor in the decimals as necessary.

Status

Resolved

Fixed in Commit

7fe420b: Limits updated to 5,000 and 500,000 USDT respectively

4. Incorrect initialization setup

Path

LaxceCrowdSale.sol#L204

Function

initialize()

Description

The contract does not call UUPSUpgradeable_init() in its initialize function.

Recommendation

Call UUPSUpgradeable_init() to maintain consistency and prevent non-initialization of storage variables while maintaining the proper storage layout.

Status

Resolved

Fixed in Commit

7fe420b: initialize() now calls UUPSUpgradeable_init()

5. Using transfer instead of safeTransfer

Path

LaxceCrowdsale.sol#L125

Function

_deliverTokens()

Description

The _deliverTokens function uses the vanilla erc20 transfer and doesn't check the return values of the transfer function or perform any address checks which could potentially revert execution. To handle potential reverts or failures during token transfers, the SafeERC20 library is recommended to be used - in this case, the safeTransfer function within this library would be advised to implement to take care of any potential errors unhandled by the regular transfer function. This is implemented in some parts of the codebase but is missing on the line specified.

Recommendation

Use safeTransfer for token transfers for uniformity.

Status

Resolved

Fixed in Commit

7fe420b: deliverTokens now uses safeTransfer()

6. Redundant code

Path

LaxceCrowdSale.sol#L81

Function

buyTokens()

Description

There are a number of redundancies within the codebase described below:

1. **Unused struct:** Currently there is no functionality to provide users with any benefits for contributing to the private sale. The usdtContributed and laxceReceived fields are unused after they are set in the buyTokens function call.

   user.usdtContributed += usdtAmount; // @audit-issue unused
   user.laxceRecieved += tokens; // @audit-issue unused

2. **Inaccurate userInfo details**: When a user calls buyTokens , the _beneficiary address passed in is redundant. Payment is made from the msg.sender wallet and tokens are transferred to the msg.sender, not the beneficiary wallet. Within the UserInfo struct the user key is the _beneficiary and not the msg.sender who makes all transactions and should accrue all benefits.

Recommendation

1. Remove redundant struct.
2. Remove the beneficiary as a function call argument as it appears to be redundant, or update the code to reflect the use of this variable.

Status

Resolved

Fixed in Commit

7fe420b: beneficiary variable and UserInfo struct have been removed

Informational Issues

7. Non-adherence to Solidity style guide

Path

Laxce.sol, LaxceCrowdSale.sol

Function

buyTokens() internal, buyTokens() external

Description

The contract could follow the conventions of the Solidity style guide which provide that non-external (and public) functions should be prefixed by an underscore. This would improve contract readability. **Reference**.

Recommendation

Rename the internal buyTokens() function to _buyTokens() to improve code readability.

Status

Resolved

Fixed in Commit

7fe420b: Function has been renamed.

Functional Tests Cases

Some of the tests performed are mentioned below:

- blacklist addresses
- remove addresses from blacklist
- buyTokens() transfers USDT from msg.sender to the crowdsale contract
- buyTokens() transfers LAXCE from contract to the crowdsale contract when tokens are available
- withdrawToken() allows only contract owner to withdraw tokens (including LAXCE)
- withdrawETH() allows only contract owner to withdraw contributed native tokens
- allow owner set minimum and maximum buy limit

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of LAXCE. We performed our audit according to the procedure described above.

Some issues of Medium, Low and Informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture. In The End, the Laxce team resolved almost all the Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in LAXCE. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of LAXCE. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of LAXCE to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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

Audit Report
November, 2024

For

LAXCE

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com