QuillAudits

Audit Report
August, 2023

For

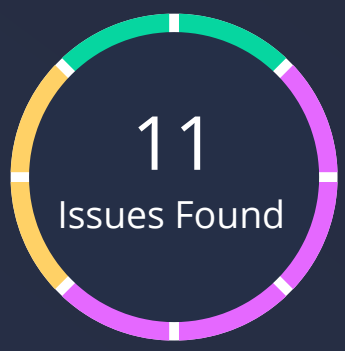
voltage

Table of Content

Executive Summary

Project Name	Volt finance launchpad
Project URL	*https://voltage.finance/*
Overview	LaunchpadFactory deploys new launchpad contracts. The launchpad contract allows users to buy project tokens, withdraw saletokens and claim bought project tokens. users can buy the amount of tokens based on the calculated allocation. Additionally it allows to send sale tokens gained from selling project tokens to the owner and treasury and withdraw unsold project tokens.
Audit Scope	*https://github.com/voltfinance/voltage-core-v2/tree/main/contracts/launchpad*
Contracts in Scope	Launchpad.sol LaunchpadFactory.sol
Commit Hash	*92f93168357afd5e2f617955800ed1aed49ee3ca*
Language	Solidity
Blockchain	Fuse
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	12 July 2023 - 19 July 2023
Updated Code Received	7 August 2023
Review 2	8 August 2023
Fixed In	*a23b24e44ed72527b5fa8af70027c44bfbdc0650*

Executive Summary



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	2	5
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	2	1	0

Types of Severities

High

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

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

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

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token 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 analysed 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

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 analysed, 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 behaviour 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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Launchpad

High Severity Issues

No issues found

Medium Severity Issues

A.1 Possible issues because lack of support for non reverting tokens

Description

User can buy project tokens without spending sale tokens incase of non reverting token is getting used.
E.g buy() uses token.transferFrom() this token can be non reverting token which won't revert on failure (e.g in case of not having enough token to transfer). In this case it will still increase user.balance and saleTokenReserve as the transferFrom won't revert for this type of token and the status is also not getting checked (in case token transfer returns boolean status)

User can then use this increased amount of user.balance for withdrawing tokens or for claiming project tokens even though no saletokens were transferred to the contract while buying.

Remediation

Use *SafeERC20* for ERC20 transfer and transferFrom. Here safeTransfer and safeTransferFrom can be used, so in the case of non reverting tokens the library will take care of returned values.

Status

Resolved

A.2 Contract lacks support for Fee-on-transfer tokens

Description

The contract lacks support for fee on transfer tokens.

E.g lets say saleToken is fee-on-transfer token for some project, when user use buy() to buy tokens it use transferFrom to transfer tokens from msg.sender to the contract. lets say user entered _amount as 100 and there is some tax applied while transferring tokens so contract won't receive exact _amount but will receive _amount-tax value, But while adding user.balance on L280 it adds _amount which wasn't exactly transferred to the contract (in the case of fee-on-transfer token).

So while claiming user can claim the amount according to _amount entered. but there are less number of sale tokens present to withdraw for project team.

This issue can also affect withdraw() upto some extent.

Remediation

In buy() while adding tokens to the user.balance on L280 , first while transferring it should be checked that how much tokens are getting received by contract by using local variables to check balance before the tranferFrom from msg.sender to contract and balance after the transferFrom. and this amount can be added to user.balance.

This remediation mitigates the risk for buy() in this case but some points needs to be noted as follows:
It should be noted that in withdrawSaleTokens() when it will transfer tokens to launchpadFactory.owner() and projectTreasury, these addresses will also receive less number of tokens because of the tax/fees that is getting taken on the transfers.
Also, in withdraw() user will be msg.sender and launchpadFactory.owner() will receive less number of tokens because of the tax/fees.

Status

Acknowledged

Volt Finance Team comment: we don’t support fee on transfer tokens on the launchpad.

Low Severity Issues

A.3 daysClaimed addition can be incorrect in some cases.

Description

claim() increaments userInfo.daysClaimed by the daysVested returned by calculateUserClaim() but this function returns daysVested as vestingDays when elapsedDays >= vestingDay

Example:

1. Lets say it's 5 days vesting. user claims allocation every day for 4 days.
2. Then at the 5 th day (or greater than 5th day) while claiming, calculateUserClaim() will return vestingDays as 5 and it will get added to the the userInfo.daysClaimed which is 4, so total would be 9 in this case. but in reality user is getting amount for allocation/vestingDays I.e for 5 days

It can create problem if daysClaimed are getting used for any reason going ahead.

Remediation

Here while returning vestingDays on L210 , (vestingDays - userInfo.totalClaimed) can be returned, so in the case where user claimed for 4 days and user is now going to claim where elapsedDays >= vestingDays (5) then while returning it will return 5 - 4 = 1. So in this way daysClaimed will get incremented by 1 in claim() L327. So userInfo.daysClaimed would be 5.

Status

Resolved

Volt Finance Team comment: we only use the daysClaimed for the claim no where else so we should be good.

A.4 ERC20 contracts with no optional method like decimal() can break the functionality

Description

In getUserBuyAmount() on L228 it calls decimal() on saleTokens. Ensure that saleToken has optional decimal() method which is not part of ERC20 standard ref: *https://eips.ethereum.org/EIPS/eip-20#decimals*
So it may happen that any tokens with no decimal() function will revert the transaction.

Remediation

Ensure that saleToken has optional decimal() method.

Status

Acknowledged

Volt Finance Team comment: we will check if the token has decimals, for A.2 and A.4 we will do a check on the tokens and tell partners that we don't support that.

Informational Issues

A.5 Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts are using floating pragma (pragma solidity ^0.8.0), Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version gets selected while deploying a contract which has higher chances of having bugs in it.

Remediation

Remove floating pragma and use a specific compiler version with which contracts have been tested.

Status

Acknowledged

A.6 Misleading error messages

Description

In constructor L90 reverts with "constructor: _projectTokenReserve > 0" where it is reverting when _projectTokenReserve is equal to 0, it can be changed to "constructor: _projectTokenReserve == 0". Similarly on L94 , L99, L103, L107, L111, L115 same types of changes can be made.

L325 reverts with error message "claim: amount vested > 0" , it should be "amount vested == 0" which say that the transaction reverted when amount was equal to 0.

Remediation

Change the error messages as suggested.

Status

Acknowledged

A.7 Unused import

Description

Ownable is unused but getting imported.

Remediation

Import statement for unused contract can be removed

Status

Acknowledged

A.8 Fix the allowed maximum vesting period

Description

error in contract states that "vesting maximum 90 days" But contract only allows 89 as maximum days of vesting period. Because of check in constructor checks for “_params.vestingDays < 90” which will limit vestingDays to be less than 90.

Remediation

Verify and change the require check on L117 to support 90 days By replacing < with <=.

Status

Acknowledged

A.9 Care needs to be taken for different decimals

Description

projectTokenReserve, minSaleTokenReserve, maxSaleTokenReserve Should be according to the decimals that project token uses.

stakedUserMaxBuyAmount, unstakedUserMaxBuyAmount should be according to decimal value that saletoken uses.

veVoltPerProjectToken should be specified with the same decimals which veVoltBalance uses.

Remediation

Care should be taken while setting constructor arguments.

Status

Acknowledged

B. Contract - LaunchpadFactory

High Severity Issues

No issues found

Medium Severity Issues

B.1 lack of support for non reverting tokens

Description

When owner creates launchpad using createLaunchpad it uses transferFrom to transfer tokens from msg.sender to launchpad. incase owner doesnt own the project tokens to transfer, it may continue without reverting in case of non reverting tokens as the value of transfer status is also not getting checked.

Remediation

Use *SafeERC20* for ERC20 transferFrom.

Status

Resolved

Low Severity Issues

B.2 Unused event

Description

LaunchCreated is declared but not getting used.

Remediation

Use the event in the createLaunchpad() function or remove the declared event.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below:

Launchpad

- ✓ User should be able to buy with sale tokens when sale is active
- ✓ User should be able to claim bought project tokens after sale ends
- ✓ User should be able to to withdraw sale tokens when sale is active
- ✓ Should be able to withraw sale tokens with withdrawSaleTokens
- ✓ Should be able to withraw unsold project tokens with withdrawUnsoldProjectTokens
- ✓ Should be able to get whole amount of bought tokens directly once claimed for non
- ✓ vested sale.
- ✓ Should be able to get amount of bought tokens for vested days for vested sale.
- ✓ Should revert if amount vested is zero while claiming
- ✓ Should revert if user tries to withdraw after sale ends
- ✓ Should revert if user tries buy more than allowed hardcap

LaunchpadFactory

- ✓ Owner should be able to create new launchpad contract
- ✓ Owner should be able to set withdrawFee
- ✓ Owner should be able to set launchpadFee

Automated Tests

Static analysis covered some issues like unchecked return values. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Summary

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

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Volt finance launchpad smart contracts. 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 Volt finance launchpad smart contracts. 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 as a result of using our audit services. It is the responsibility of the Volt finance launchpad to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2023

For

voltage

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com