QuillAudits

Audit Report
August, 2022

For

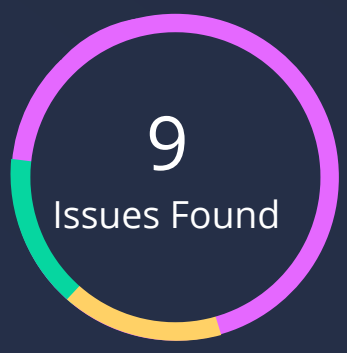
Voltage

Table of Content

Table of Content

Executive Summary

Project Name	Voltage Finance swap
Timeline	May 16, 2022 - July 11, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Voltage Finance swap codebase for quality, security, and correctness. *https://github.com/voltfinance/saddle-contract* Commit hash: 0cc39633cb5ca60406b67ecf0664a8e92095ed4b



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	1	7
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	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 - LPToken

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A1. Unnecessary import statement

```
3   pragma solidity 0.6.12;
4
5   import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20BurnableUpgradeable.sol";
6   import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
7   import "./interfaces/ISwap.sol";
8
```

Description

LPToken imports ISwap interface, this interface is not getting used in LPToken contract.

Remediation

Consider removing unnecessary import statement for ISwap. Incase its needed in future care needs to be taken about what ISwap inherits/implements/imports.

Status

Acknowledged

Informational Issues

A2. Centralization risk

```
42      function mint(address recipient, uint256 amount) external onlyOwner {
43          require(amount != 0, "LPToken: cannot mint 0");
44          _mint(recipient, amount);
45      }
```

Description

Owner can mint any amount of tokens to any address and here minting has no maximum limit.

Remediation

Care needs to be taken for owner address controls, where compromised or malicious owner address can use this functionality for their own profit.

Status

Acknowledged

A3. Incorrect compiler version

```
1   // SPDX-License-Identifier: MIT
2
3   pragma solidity 0.6.12;
4
```

Description

Contract is using old solidity version 0.6.12, Using an old version prevents access to new Solidity security checks.

Remediation

Consider using the latest solidity version.

Status

Acknowledged

B. Contract - SwapFlashLoan

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

B1. Incorrect compiler version

Description

Contract is using old solidity version 0.6.12, Using an old version prevents access to new Solidity security checks.

Remediation

Consider using the latest solidity version.

Status

Acknowledged

C. Contract - Swap.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

C1. Timestamp Dependence

Description

The part of code shows timestamp dependence, malicious miners can use this dependency for their profit by manipulating block timestamp. For eg: deadlineCheck modifier. In some cases its necessary to use timestamps and in these cases care needs to be taken that the manipulation of this timestamp by miners should not affect the mechanism.

Status

Acknowledged

C2. Incorrect compiler version

Description

Contract is using old solidity version 0.6.12, Using an old version prevents access to new Solidity security checks.

Remediation

Consider using the latest solidity version.

Status

Acknowledged

D. Contract - SwapUtils.sol

High Severity Issues

No issues found

Medium Severity Issues

D1. Loss Making updates to A

Description

Since volt finance contracts are derived from Curve finance they are also vulnerable to an economic attack identified by Peter Zeitz.

Remediation

It is recommended to follow the article's recommendation and reduce the step-size in A to not more than 0.1% per block.

References

https://medium.com/@peter_4205/curve-vulnerability-report-a1d7630140ec

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

D1. Timestamp Dependence

Description

The part of code shows timestamp dependence, malicious miners can use this dependency for their profit by manipulating block timestamp upto certain levels. These are some functions using timestamp: _getAPrecise(), AmplificationUtils.rampA(), AmplificationUtils.stopRampA()

Status

Acknowledged

C2. Incorrect compiler version

Description

Contract is using old solidity version 0.6.12, Using an old version prevents access to new Solidity security checks.

Remediation

Consider using the latest solidity version.

Status

Acknowledged

Functional Testing

LPToken

- ✓ Should be able to mint tokens
- ✓ Should revert if unauthorized address tries to mint tokens
- ✓ Should be able to transfer tokens
- ✓ Should be able to approve tokens
- ✓ Should be able to spend approved tokens
- ✓ Should revert if allowance is not enough
- ✓ Should revert if address don't holds enough tokens to transfer
- ✓ Token minting should change token supply

SwapFlashLoan

- ✓ Should be able to take flashlaon without reverting when fee is paid
- ✓ Reverts when the borrower does not have enough to pay back
- ✓ Reverts when flashloan debt is not paid
- ✓ Reverts when calling re-entering swap contract via addLiquidity
- ✓ Reverts when calling re-entering swap contract via swap
- ✓ Reverts when calling re-entering swap contract via removeLiquidity
- ✓ Reverts when calling re-entering swap contract via removeLiquidityOneToken
- ✓ Owner can set valid fees with setFlashLoanFees
- ✓ Should be able to set protocol fee bps to 0
- ✓ Reverts when fees are not in the range

Swap

getToken

- ✓ Returns correct addresses of pooled tokens
- ✓ Reverts when index is out of range

getTokenIndex

- ✓ Returns correct token indexes
- ✓ Reverts when token address is not found

getTokenBalance

- ✓ Returns correct balances of pooled tokens
- ✓ Reverts when index is out of range

getA

- ✓ Returns correct value

addLiquidity

- ✓ Reverts when contract is paused
- ✓ Reverts with 'Amounts must match pooled tokens'
- ✓ Reverts with 'Cannot withdraw more than available'
- ✓ Reverts with 'Must supply all tokens in pool'
- ✓ Succeeds with expected output amount of pool tokens
- ✓ Succeeds with actual pool token amount being within ±0.1% range of calculated pool token
- ✓ Succeeds with correctly updated tokenBalance after imbalanced deposit
- ✓ Returns correct minted lpToken amount
- ✓ Reverts when minToMint is not reached due to front running
- ✓ Reverts when block is mined after deadline
- ✓ Emits addLiquidity event

removeLiquidity

- ✓ Reverts with 'Cannot exceed total supply'
- ✓ Reverts with 'minAmounts must match poolTokens'
- ✓ Succeeds even when contract is paused
- ✓ Succeeds with expected return amounts of underlying tokens
- ✓ Returns correct amounts of received tokens
- ✓ Reverts when user tries to burn more LP tokens than they own
- ✓ Reverts when minAmounts of underlying tokens are not reached due to front running
- ✓ Reverts when block is mined after deadline
- ✓ Emits removeLiquidity event

removeLiquidityImbalance

- ✓ Reverts when contract is paused
- ✓ Reverts with 'Amounts should match pool tokens'
- ✓ Reverts with 'Cannot withdraw more than available'
- ✓ Succeeds with calculated max amount of pool token to be burned (±0.1%)
- ✓ Returns correct amount of burned lpToken
- ✓ Reverts when user tries to burn more LP tokens than they own
- ✓ Reverts when minAmounts of underlying tokens are not reached due to front running
- ✓ Reverts when block is mined after deadline
- ✓ Emits RemoveLiquidityImbalance event

removeLiquidityOneToken

- ✓ Reverts when contract is paused.
- ✓ Reverts with 'Token index out of range'
- ✓ Reverts with 'Withdraw exceeds available'
- ✓ Reverts with 'Token not found'
- ✓ Succeeds with calculated token amount as minAmount
- ✓ Returns correct amount of received token
- ✓ Reverts when user tries to burn more LP tokens than they own
- ✓ Reverts when minAmount of underlying token is not reached due to front running
- ✓ Reverts when block is mined after deadline
- ✓ Emits RemoveLiquidityOne event

swap

- ✓ Reverts when contract is paused
- ✓ Reverts with 'Token index out of range'
- ✓ Reverts with 'Cannot swap more than you own'
- ✓ Succeeds with expected swap amounts
- ✓ Reverts when minDy (minimum amount token to receive) is not reached due to front running
- ✓ Succeeds when using lower minDy even when transaction is front-ran
- ✓ Returns correct amount of received token
- ✓ Reverts when block is mined after deadline
- ✓ Emits TokenSwap event

getVirtualPrice

- ✓ Returns expected value after initial deposit
- ✓ Returns expected values after swaps
- ✓ Returns expected values after imbalanced withdrawal
- ✓ Value is unchanged after balanced deposits
- ✓ Value is unchanged after balanced withdrawals

setSwapFee

- ✓ Emits NewSwapFee event
- ✓ Reverts when called by non-owners
- ✓ Reverts when fee is higher than the limit
- ✓ Succeeds when fee is within the limit

setAdminFee

- ✓ Emits NewAdminFee event
- ✓ Reverts when called by non-owners
- ✓ Reverts when adminFee is higher than the limit
- ✓ Succeeds when adminFee is within the limit

getAdminBalance

- ✓ Reverts with 'Token index out of range'
- ✓ Is always 0 when adminFee is set to 0
- ✓ Returns expected amounts after swaps when adminFee is higher than 0

withdrawAdminFees

- ✓ Reverts when called by non-owners
- ✓ Succeeds when there are no fees withdrawn
- ✓ Succeeds with expected amount of fees withdrawn
- ✓ Withdrawing admin fees has no impact on users' withdrawal

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 Voltage Finance. 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.In the End, Voltfinance team Acknowledged all Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Voltage Finance Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Voltage Finance Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2022

For

Voltage

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com