QuillAudits

Audit Report
January 2023

For

Mute.

Table of Content

Executive Summary

Project Name	Mute
Overview	Mute contract is based on Uniswap V2 that allows for swaps, exchanges and reserves of token and operates on the zk-sync. Users have the ability to contribute to and remove from the pool. In MuteAmplifier, users can stake their LP tokens for rewards in mute tokens.
Timeline	15 September, 2022 - 15 November, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze MuteSwitchFactory.sol, MuteSwitchPair.sol,MuteSwitchRouter.sol, MuteSwitchERC20.sol, MuteAmplifier.sol, SwitchLimitOrder.sol, SwitchLimitOrderBot.sol, and MultiCall.sol. Github Link: *https://github.com/muteio/mute-switch-core/* Commit hash: ae8be01db473111b13088e6bef23714b6aed5a06
Fixed In	f0f3d9040b68f82fc9ed8a84710395ce850b0c55

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	2	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	3	3	4

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 - Common Issues

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

1. Missing Events for Significant Actions

Contracts Affected - MuteSwitchFactory.sol

Description

Events emission are relevant to deployed contracts because it allows for filtering of emitted information. When certain state changes are happening, it is expected to emit these significant actions to track these changes.

In MuteSwitchFactory.sol:
- setProtocolFeeFixed
- setProtocolFeeDynamic
- setFeeTo

Remediation

Consider emitting an event whenever certain significant changes are made in the contracts.

Status

Resolved

2. Renounce ownership

Description

The renounceOwnership function can be called accidentally by the admin leading to immediate renouncement of ownership to zero address after which
any onlyOwner functions will not be callable which can be risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Refer this post for additional info - *https://www.linkedin.com/posts/razzor_github-razzorsecrazzorsec-contracts-activity-6873251560864968705-HOS8*

Status

Acknowledged

3. Transfer ownership

Description

The transferOwnership() function in contract allows the current admin
to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored into the storage owner, after validating the newOwner is a non-zero address, and immediately overwrites the current owner. This can lead to cases where the admin has transferred ownership to an incorrect address and wants to revoke the transfer of ownership or in the cases where the current admin comes to know that the new admin has lost access to his account.

Remediation

It is advised to make ownership transfer a two-step process.
Refer - Link1 and Link 2.

Status

Acknowledged

Informational Issues

4. Compiler version

Description

A different compiler version greater than 0.8.0 as opposed to previous compiler versions of Uniswap are used in the codebase. This could lead to unintended and undiscovered bugs as the newer compiler versions have breaking changes. For example the Safemath usage is not required as compiler version greater than 0.8.0 have built-in overflow and underflow checks and revert if this happens.

Recommendation

It is advised to use the same compiler version as the Uniswap. This is because the compiler version has been time tested on Mainnet. Also use a fixed compiler version instead of floating pragma as it could lead to another compiler version

Status

Acknowledged

5. Lack of comment

Description

The code base has insufficient comments to describe the functions, parameters, its business logic and working.

Recommendation

It is advised to add comments for the same.

Status

Acknowledged

B. Contract - MuteSwitchFactory.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

1. Missing zero address check

Description

This function omits the critical check to ensure that the feeTo is not set to address zero, for funds sent into this address is forever lost and irredeemable. This function also fails to check for when the existing FeeTo address calls the function with passing his address unknowingly, for this will cause a waste of gas when making the existing owner, the new owner.

In MuteSwitchFactory.sol:
- setFeeTo

```
50      function setFeeTo(address _feeTo) external {
51          require(msg.sender == feeTo, 'MuteSwitch: FORBIDDEN');
52          feeTo = _feeTo;
53      }
54
```

Remediation

Consider adding a require statement that validates input against zero address to mitigate the same.

Status

Resolved

Informational Issues

No issues found

C. Contract - MuteSwitchPair.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

D. Contract - MuteSwitchRouter.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

E. Contract - MuteSwitchERC20.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

F. Contract - MuteAmplifier.sol

High Severity Issues

No issues found

Medium Severity Issues

1. Missing sanity checks

Description

_stakeDivisor parameter in the constructor has no sanity value checks. The _stakeDivisor variable is used in the calculateMultiplier() function in which
uint256 stakeDifference = _stakeDivisor.sub(10 ** 18);
But if _stakeDivisor is set to a value less than 10 ** 18, the function will revert. So it is advised to add at least a value check so that _stakeDivisor is greater than 10 ** 18.

Also changeDivisor() function lacks sanity value checks for stakeDivisor() function.

```
55      /// @dev Expects a LP token address & the base reward muteToken address
56      constructor (address _lpToken, address _muteToken, address _dToken, uint256 divisor) {
57          lpToken = _lpToken;
58          muteToken = _muteToken;
59          dToken = _dToken;
60          _stakeDivisor = divisor;
61      }
62
63      function changeDivisor(uint256 divisor) public onlyOwner {
64          _stakeDivisor = divisor;
65      }
```

Remediation

Consider adding necessary checks for the same.

Status

Resolved

2. Centralization issue in changeDivisor()

Description

Moreover the _stakeDivisor can be changed by the admin even after the staking has started. A malicious admin could exploit this by manipulating the value of _stakeDivisor which could lead to existing stakers getting lesser rewards.

```
63      function changeDivisor(uint256 divisor) public onlyOwner {
64          _stakeDivisor = divisor;
65      }
```

Remediation

It is advised to let existing stakers time to withdraw rewards if the value of stakeDivisor is changed.

Status

Resolved

3. Possible staking before initializeDeposit()

Description

Staking can be done once even if initializeDeposit() is not called. This could lead to unintended issues. If this happens, firstStakeTime will be lesser than startTime which could affect rewards calculation adversely and introduce bugs.
For example, if someone stakes at epoch timestamp1 and then initializeDeposit() is called, then the startTime will be greater than this timestamp1. But as the first staking was done at timestamp1, firstStakeTime will be timestamp1, instead of a timestamp later than startTime. This will result in inconsistency in the perSecondReward calculation in the update() modifier

```
100     modifier update() {
101         if (_mostRecentValueCalcTime == 0) {
102             _mostRecentValueCalcTime = firstStakeTime;
103         }
104
105         uint256 totalCurrentStake = totalStake();
106
107         if (totalCurrentStake > 0 && _mostRecentValueCalcTime < endTime) {
108             uint256 value = 0;
109             uint256 sinceLastCalc = block.timestamp.sub(_mostRecentValueCalcTime);
110             uint256 perSecondReward = totalRewards.div(endTime.sub(firstStakeTime));
111
```

Remediation

Add a check in stake() function such as this In order to prevent this issue.
require(block.timestamp >= startTime && startTime !=0, "MuteAmplifier::stake: not live yet");

Status

Resolved

4. Centralization of rescueTokens()

Description

Although this function can be used to rescue funds in case of any malicious activity, the contract can be drained of all the LP tokens as well as reward tokens by a malicious admin.

```
263     function rescueTokens(address tokenToRescue, address to, uint256 amount) public virtual onlyOwner nonReentrant {
264         if (tokenToRescue == lpToken) {
265             require(amount <= IERC20(lpToken).balanceOf(address(this)).sub(_totalStakeLpToken),
266                 "MuteAmplifier::rescueTokens: that Token-Eth belongs to stakers"
267             );
268         } else if (tokenToRescue == muteToken) {
269             if (totalStakers > 0) {
270                 require(amount <= IERC20(muteToken).balanceOf(address(this)).sub(totalRewards.sub(totalClaimedRewards)),
271                     "MuteAmplifier::rescueTokens: that muteToken belongs to stakers"
272                 );
273             }
274         }
275
276         IERC20(tokenToRescue).transfer(to, amount);
277     }
```

Remediation

It is advised to use a multisig wallet to increase safety against loss of private keys or preventing bad actors. It is also advised to add a sufficient window period for stakers to withdraw their LP tokens before the admin is able to drain all their LP tokens.

Status

Acknowledged

Low Severity Issues

5. Missing zero address check

Description

Missing zero address for constructor arguments- _lpToken, _muteToken and_dToken.

```
55      /// @dev Expects a LP token address & the base reward muteToken address
56      constructor (address _lpToken, address _muteToken, address _dToken, uint256 divisor) {
57          lpToken = _lpToken;
58          muteToken = _muteToken;
59          dToken = _dToken;
60          _stakeDivisor = divisor;
61      }
62
```

Remediation

Consider adding a require statement that validates input against zero address to mitigate the same.

Status

Resolved

Informational Issues

6. Redundant If statement

Description

The if statement is not required because the require statement on line: 138 already takes care of the fact that lpTokenIn is always greater than zero

```
127     function stake(uint256 lpTokenIn) public virtual update nonReentrant {
128         require(lpTokenIn > 0, "MuteAmplifier::stake: missing stake");
129         require(block.timestamp >= startTime, "MuteAmplifier::stake: not live yet");
130         require(IERC20(muteToken).balanceOf(address(this)) > 0, "MuteAmplifier::stake: no reward balance");
131 
132         if (firstStakeTime == 0) {
133             firstStakeTime = block.timestamp;
134         } else {
135             require(block.timestamp < endTime, "MuteAmplifier::stake: staking is over");
136         }
137 
138         if (lpTokenIn > 0) {
139             lpToken.safeTransferFrom(msg.sender, address(this), lpTokenIn);
140         }
```

Remediation

Consider removing the redundant if statement.

Status

Resolved

7. Comparison with boolean

Description

calculateMultiplier() has an if statement that compares to a boolean constant. But this is not required as this boolean can be used directly instead.

```
282     function calculateMultiplier(address account, bool enforce) public view returns (uint256) {
283         require(account != address(0), "MuteAmplifier::calculateMultiplier: missing account");
284
285         uint256 accountDTokenValue;
286
287         if(_userStakedBlock[account] != 0 && enforce == true)
288           accountDTokenValue = IDMute(dToken).getPriorVotes(account, _userStakedBlock[account]);
289         else
290           accountDTokenValue = IDMute(dToken).getPriorVotes(account, block.number);
291
292         if(accountDTokenValue == 0){
293           return _stakeDivisor;
294         }
```

Remediation

It is advised to make the necessary changes as suggested above.
Refer - Link

Status

Resolved

8. Gas optimization

Description

changeDivisor(), initializeDeposit(), userStakedBlock(), dripInfo(), stake(), withdraw() payout() and rescueTokens() could be declared external instead of public to save gas.

Recommendation

It is advised to make the necessary changes as suggested above.

Status

Resolved

G. Contract - SwitchLimitOrder.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

1. Redundant statements

Description

Unnecessary deposit() and withdraw() to WETH in OrderMatchETHForToken() function

```
135     function OrderMatchETHForToken(uint256 index) external lock payable {
136         LimitOrder memory order = limitOrders[index];
137         uint256 amountOut = msg.value;
138 
139         require(amountOut >= order.amountOutMin, 'SwitchLimitOrder: INVALID_MATCH');
140         require(order.path[1] == WETH, 'SwitchLimitOrder: INVALID_PATH');
141 
142         IWETH(WETH).deposit{value: order.amountOutMin}();
143         IWETH(WETH).withdraw(order.amountOutMin);
144         TransferHelper.safeTransferETH(order.to, order.amountOutMin);
```

Remediation

Consider removing the redundant statements.

Status

Resolved

H. Contract - SwitchLimitOrderBot.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

I. Contract - MultiCall.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

1. Declare Functions As External To Save Gas

Description

When functions are not called internally, or intended to be called internally or by called by other contracts that inherits it, it is recommended that these functions are given the external visibility to save gas. Here are list of functions under discussion:

- aggregate
- blockAndAggregate
- getBlockHash
- getBlockNumber
- getCurrentBlockCoinbase
- getCurrentBlockDifficulty
- getCurrentBlockGasLimit
- getCurrentBlockTimestamp
- getEthBalance
- getLastBlockHash

Remediation

For gas optimization reasons, it is recommended to make these functions have external visibility to save gas since they are not called internally.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below:

- Should successfully add tokens as liquidity to the pool
- Should revert if insufficient eth is provided as liquidity to the pool
- Should successfully add sufficient eth and token as liquidity to the pool
- Should revert if signer has insufficient mute token balance
- Should revert if signer approves less than passed value
- Should revert when insufficient dMute tokens is to be minted after computation
- Should revert when lock time parameter is less than the weekly time
- Should revert when lock time parameter is greater than max lock time
- A user should successfully lock mute token consecutively
- Should revert if a caller of the function does not lock tokens in contract
- Should revert if caller calls redeem within the lock period
- Should revert if user attempts to redeem twice from just one locked
- Should lock two to three times and claim all three locked tokens

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 the Mute. 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 audit is not a security warranty, investment advice, or an endorsement of the Mute 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 Mute 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.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
January, 2023

For

Mute.

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com