QuillAudits

AUDIT REPORT

December, 2024

For

Table of Content

Executive Summary 03

Checked Vulnerabilities 05

Number of Issues per Severity 07

Techniques & Methods 08

Types of Severity 10

Types of Issues 11

Low Severity Issues 12

1. Farm Deployment Allows Creation of Multiple Farms, Potentially Violating Business Logic 12

2. Farm Deployment Allows Zero Reward Rate, Leading to Unrewarded Staking and Gas Costs 14

3. Risk of Permanent Contract Lockout due to renounce Ownership 15

Informational Issues 16

1. Lack of Minimum Buffer Restriction Leads to Price Manipulation 16

2. Optimizing Gas Usage with Prefix Incrementation in Loops 17

3. Optimizing Smart Contract Efficiency by Avoiding Redundant Initializations 18

4. Missing Check-Effects-Interactions Pattern 19

Closing Summary & Disclaimer 20

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	28 November 2024 - 05 December 2024
Updated code received	07 December 2024
Second Review	10 Dec 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 New-Contracts & Updates for quality, security, and correctness.
Blockchain	EVM
Source Code	https://github.com/tkzo/memeswap-contracts/tree/core-revision Commit: 140405780640e20efac8d2f891b832359dc1fc95
Contracts in Scope	New functions to audit on the old scope: MemeswapToken.sol MemeswapTokenFactory.sol MemeswapVault.sol MemeswapPair.sol New Contract which required full audit: MemeswapFarm.sol MememeswapFarmFactory.sol MemeswapFarmToken.sol Library/FixedPoint contracts MemeswapLock.sol

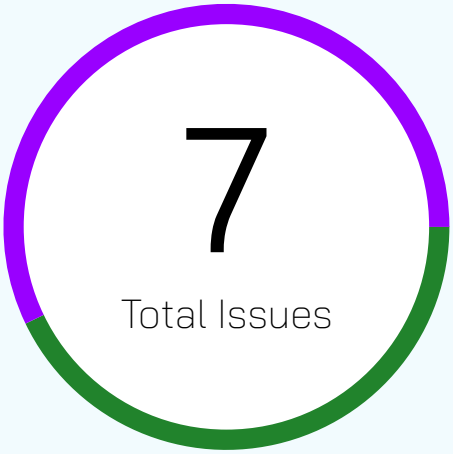
Branch	Main
Fixed In	140405780640e20efac8d2f891b832359dc1fc95

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Balance Equality
- ✓ Byte Array
- ✓ Transfer Forwards All Gas
- ✓ Compiler Version Not Fixed
- ✓ Redundant Fallback Function
- ✓ Send Instead of Transfer
- ✓ Style Guide Violation
- ✓ Unchecked External Call
- ✓ Unchecked Math
- ✓ Unsafe Type Inference
- ✓ Implicit Visibility Level
- ✓ Access Management
- ✓ Arbitrary Write to Storage
- ✓ Centralization of Control
- ✓ Ether Theft
- ✓ Improper or Missing Events
- ✓ Logical Issues and Flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race Conditions/Front Running
- ✓ SWC Registry
- ✓ Malicious Libraries
- ✓ 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
- Using Inline Assembly

Number of Issues per Severity



High	0 (0.00%)
Medium	0 (0.00%)
Low	3 (42.86%)
Informational	4 (57.14%)

Severity

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

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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

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 Issues

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

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

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.

Low Severity Issues

Farm Deployment Allows Creation of Multiple Farms, Potentially Violating Business Logic

Acknowledged

Path

src/MemeswapFarmFactory.sol

Description

The MemeswapFarmFactory contract allows token deployers to create farms using the deployFarm function. Based on the business logic dictates that a farm should consist of one reward token and up to three stake tokens. However, the current implementation of the deployFarm function does not enforce a limit on the number of farms that can be created for a single token. This oversight can allow deployers to create multiple farms with the same reward token, effectively spamming the system and competing for user attention.

Recommendation

Restrict Multiple Farm Deployments
Add a mechanism in the deployFarm function to ensure that only one farm can exist per reward token.
Use a mapping(address => bool) to track deployed farms for each reward token.
Revert if a farm already exists for the reward token.

POC

```
function test_DeployFarmm() public {
 address[] memory pairs = new address[](3);
 pairs[0] = address(pair);
 pairs[1] = address(pair1);
 pairs[2] = address(pair2);
 IERC20(address(rewardToken)).approve(address(factory), 100 ether);

 address farm = factory.deployFarm(address(rewardToken), pairs, 0 , 30 days);
 address farm1 = factory.deployFarm(address(rewardToken), pairs, 0 , 30 days);
 address farm2 = factory.deployFarm(address(rewardToken), pairs, 0 , 30 days);
 address farm3 = factory.deployFarm(address(rewardToken), pairs, 0 , 30 days);
}
```

Impact

System Exploitation: Deployers could spam the platform with multiple farms, leading to inefficiency and reduced user engagement.
User Confusion: Multiple farms for the same reward token may need to be clarified for users, reducing trust and usability.
Unfair Competition: Genuine token deployers adhering to the intended logic might face competition from spam farms, disrupting the ecosystem balance

Farm Deployment Allows Zero Reward Rate, Leading to Unrewarded Staking and Gas Costs

Resolved

Path

src/MemeswapFarmFactory.sol

Description

The deployFarm function in the MemeswapFarmFactory contract does not enforce a minimum reward rate for farms during deployment. This allows deployers to create farms with a reward rate of zero. Users who stake in such farms will bear gas costs for staking and removing tokens without receiving any rewards, leading to a poor user experience.

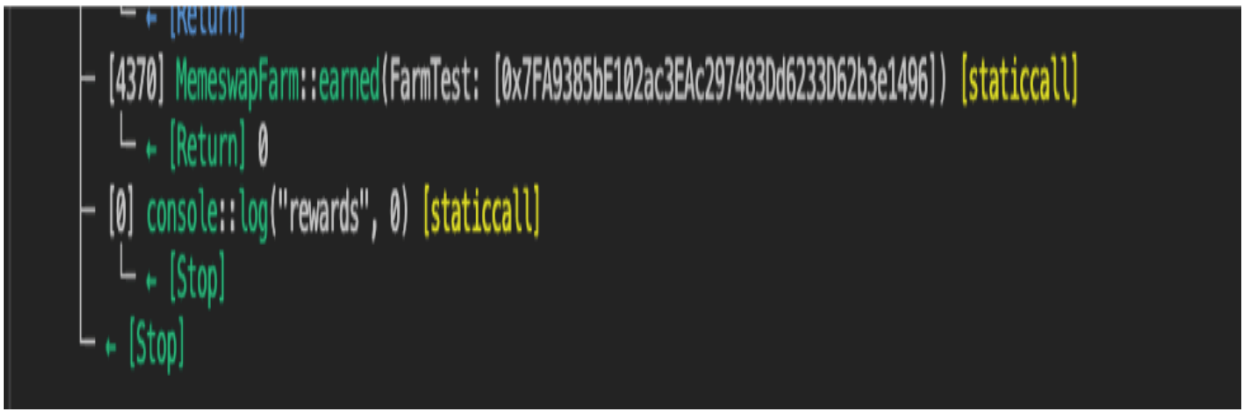
Recommendation

Add a validation in the deployFarm function to ensure that the _amount parameter (reward amount) is greater than zero.

POC

```
function setUp() public {
// .... ////
 farm = new MemeswapFarm(address(rewardToken),address(this), pairs, 30 days);
 rewardToken.transfer(address(farm), 0 ether );
 farm.notifyRewardAmount(0 ether);

 function test_Stakee() public {
 vm.warp(block.timestamp + farm.buffer());
 pair.setReserves(4 ether, 2 ether);
 pair.setCumulativePrices();
 token.approve(address(farm), 1000 ether);
 farm.stake(address(pair), 100 ether  );
 uint256 farmTokenBalance = IERC20(ft).balanceOf(address(this));
 vm.warp(block.timestamp + 1 days);
 uint256 rewards = farm.earned(address(this));
 console2.log("rewards", rewards);
}
```



Risk of Permanent Contract Lockout due to renounce Ownership

Resolved

Path

src/MemeswapFarm.sol, src/MemeswapFactory.sol, src/MemeswapFarmFactory.sol, src/MemeswapFarmToken.sol, src/MemeswapToken.sol, src/MemeswapTokenFactory.sol, src/MemeswapVault.sol

Description

The Memeswap contracts import Ownable.sol from OpenZeppelin, which includes a renounceOwnership function. This function transfers ownership to the zero address (ad- dress(0)), effectively removing the contract owner. As a result, any functions restricted by the on- lyOwner modifier will become inaccessible if renounceOwnership is called, either intentionally, by mistake, or due to malicious activity. This can lead to a scenario where the contract is permanently locked, making it impossible to manage or upgrade the contract, or to access critical functionalities.

Recommendation

Override the renounceOwnership function function in the Memeswap contracts to disable or restrict it.

Impact

This vulnerability could lead to a complete loss of control over the contract's critical administrative functions. Once the ownership is renounced, functions restricted to the owner will no longer be accessible, potentially locking the contract indefinitely.

Informational Severity Issues

Lack of Minimum Buffer Restriction Leads to Price Manipulation

Resolved

Path

src/MemeswapFarmFactory.sol

Description

The buffer parameter in the MemeswapFarm contract determines the minimum time interval for calculating the time-weighted average price (TWAP) from a pair. However, the buffer value is configurable via the setBuffer function in the MemeswapFarmFactory contract and lacks a minimum value restriction. This oversight allows the owner to set an extremely low buffer value, leading to short TWAP windows. Short TWAP intervals are highly susceptible to price manipulation, as the average price can be influenced by a single trade or a series of rapid trades within the short timeframe.

```
function setBuffer(uint256 _buffer) external onlyOwner {
 buffer = _buffer;
}
```

Recommendation

Restrict the buffer value to a reasonable minimum to ensure that TWAP calculations reflect stable price movements.

Impact

Price Manipulation Risk: A small buffer window allows attackers to manipulate prices easily, leading to inaccurate TWAP calculations.
Unfair Reward Distribution: Manipulated TWAP values could result in unfair rewards for attackers, disadvantaging honest participants.

Optimizing Gas Usage with Prefix Incrementation in Loops

Resolved

Path

src/MemeswapFarm.sol, src/MemeswapFactory.sol, src/MemeswapVault.sol

Description

Within the smart contracts, the current use of postfix incrementation (i++) in loops is identified as a less gas-efficient practice. Switching to prefix incrementation (++i) to update uint variables inside loops can yield gas savings. This optimization opportunity applies to loop iterators and any increment operations within the loop's body. The efficiency gain stems from how Solidity handles these two operations at the bytecode level, with prefix incrementation slightly more gas-efficient due to its direct modification of the variable's state without storing an intermediate value.

Recommendation

identify instances where i++ is used within loops. Refactor these to use ++i for direct incrementation.

Optimizing Smart Contract Efficiency by Avoiding Redundant Initializations

Resolved

Description

In smart contracts, there is a recurring pattern of initializing loop variables and accu- mulators to their default values. While common and clear for understanding, this practice introduces unnecessary operations, as Solidity automatically initializes variables to their default values. Specifically, instances of loop counters being set to 0 (uint256 i = 0) and accumulators being initialized to 0 can be optimized by leveraging Solidity's default initialization to reduce bytecode size and potentially gas costs during contract deployment.

Missing Check-Effects-Interactions Pattern

Resolved

Path

src/MemeswapFarm.sol

Description

The Check-Effects-Interactions (CFI) pattern ensures that internal state updates occur before any external calls. This minimizes the risk of reentrancy attacks and unintended side effects.

In the MemeswapFarm contract, the stake and exit functions violate this pattern. External calls to untrusted contracts or tokens are made before updating internal state variables, creating potential risks. While the contract implements ReentrancyGuard, following the Check-Effects-Interactions pattern is still considered best practice to enhance security and robustness.

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 MemeSwap contracts. We performed our audit according to the procedure described above.

Some issues of 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 contract. 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 Team 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.

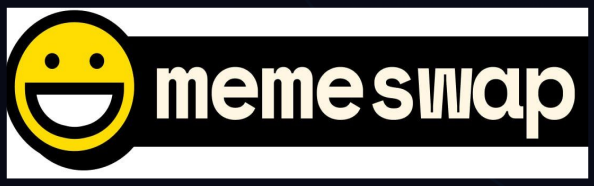
6+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1K+** Projects Secured

Follow Our Journey

AUDIT REPORT

December, 2024

For



QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com