QuillAudits

Audit Report
January, 2022

For

SWAP

Contents

Overview

Oboswap is the leading decentralized exchange on Binance Smart Chain, Ethereum and Polygon with the highest trading volumes in the market.

Scope of the Audit

The scope of this audit was to analyze the Oboswap smart contract's codebase for quality, security, and correctness.

Commit: 4f358a6eefc80c394df953fdcca0d4cc4fbc53de

Fixed In: f58706f0f8d9bb9f359f0b8173566142893d27df

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- 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 the 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/BEP20 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
Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

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.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	1	0	0	0
Closed	1	1	1	0

Functional Testing Results

Complete functional testing report has been attached below: Visit Link

Some of the tests performed are mentioned below:

- should be able to mint obosyrup & obotoken by the MasterChef contract only. PASS
- should be able to burn the obosyrup funds. PASS
- should be able to correctly delegate the funds during burning. PASS
- should be able to transfer the obosyrup funds from MasterChef to so someone else. PASS
- should be able to retrieve correct delegates for a given delegatee. PASS
- should be able to delegate the voting power. PASS
- should be able to delegate the voting power using offchain signature. PASS
- should be able to retrieve votes using getCurrentVotes & getPriorVotes. PASS
- should be able to set delay. PASS
- should be able to accept admin. PASS
- should be able to set pending admin. PASS
- should be able to queue transaction. PASS
- should be able to cancel the transaction. PASS
- should be able to execute transaction. PASS

- should be able to deposit the liquidity tokens. PASS
- should be able to withdraw reward for a given pool id. PASS
- should be able to enter and leave staking. PASS
- should be able to emergency withdraw. PASS
- should be able to retrieve the correct amount of pending Obo tokens PASS

Issues Found

High severity issues

- **Burning of tokens increases the voting power instead of decreasing by the same amount.**

  In [#L1106] line movement of delegate is happening when the funds get burned but instead of moving the voting power to the address(0), It is moving the voting power from the address(0) to _from account event that _from doesn't even have the ownership of those funds.

  Recommendation
  Consider switching the srcRep and dstRep the parameter of the _moveDelegates() function at [#L1106].

  Status: Fixed

- **Voting power is not changing during the transfer of tokens.**

  transfer()/transferFrom() function is not calling the _movingDelegates() function so even after the transfer of funds doesn't lead to the transfer of voting power from sender to the receiver for both ObosToken and OboSyrup.

  ex - Alice transferred funds 100 tokens to Bob leading to having the same voting power as remain so if before transfer Alice has 100 voting power and even after the transfer of 100 tokens the voting power of Alice remains the same.

  Recommendation
  It is recommended to use the _movingDelegates() function within the transfer and transferFrom() function to move the delegated power from sender to receiver.

  Status: Acknowledged

Medium severity issues

- **Re-entrancy leads to loss of funds**

  [L#1755][MasterChef.sol] Allows re-entrancy as an attacker could re-enter into the emergencyWithdraw() function using the safeTransfer function call on lp token. Because of that attacker would get more tokens then it entitled to.

Recommendation
We recommend to update state variables before the safeTransfer call then even if the attacker tries to re-enter into the emergencyWithdraw function it ends up burning gas instead of reaping the funds from the system.

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        user.amount = 0;
        user.rewardDebt = 0;
        pool.lpToken.safeTransfer(address(msg.sender), user.amount);
        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    }
```

Or can use OZ's Reentrancy guard.

Status: Fixed

Low severity issues

- **Inefficiency in the code that leads to more gas consumption.**

  [#L1587] In the current implementation there are two sloads is happening to read the poolInfo[_pid].allocPoint which could be reduced to one.

Recommendation
Rewrite the function like below.

```
function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 prevAllocPoint = poolInfo[_pid].allocPoint;
    if (prevAllocPoint == _allocPoint) {
        return;
    }
    poolInfo[_pid].allocPoint = _allocPoint;
    totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint);
    updateStakingPool();
}
```

Status: Fixed

Informational issues

No issues found

Closing Summary

Some issues were found during the Initial Audit, Almost all of them are fixed now.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Oboswap. 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 Oboswap team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
January, 2022

For

SWAP

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com