expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

OSCARS TAKING NO LOCKING

TOKEN OVERVIEW

Risk Findings

Severity	Found
Critical	1
High	4
Medium	2
Low	0
Informational	0

Centralization Risks

Owner Privileges	Description
Can Owner Set Taxes >25% ?	Not Detected
Owner needs to enable trading ?	Yes, owner needs to enable trades
Can Owner Disable Trades ?	Not Detected
Can Owner Mint ?	Not Detected
Can Owner Blacklist ?	Not Detected
Can Owner set Max Wallet amount ?	Not Detected
Can Owner Set Max TX amount ?	Not Detected

TABLE OF CONTENTS

OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Audit Result	**Passed**
KYC Verification	**No**
Audit Date	**8 June 2023**

expelee

CONTRACT DETAILS

Token Name: OscarStakingNoLocking

Symbol: OscarStakingNoLocking

Network: Binance smart chain

Language: Solidity

Contract Address:
0x96De4B2a85A436A6fEBF68524825d7AD3E458074

Total Supply: ---

Owner's Wallet:
0x3166Dfd7cFb2F66e9Fc6188955b29D9F1c35A679

Deployer's Wallet:
0x3166Dfd7cFb2F66e9Fc6188955b29D9F1c35A679

AUDIT METHODOLOGY

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch , that lead to scams and rugpulls.

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- DE
- Open Zeppelin
- Code Analyzer
- Solidity Code
- Compiler
- Hardhat

VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zepplin module	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and acces control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Low Risk

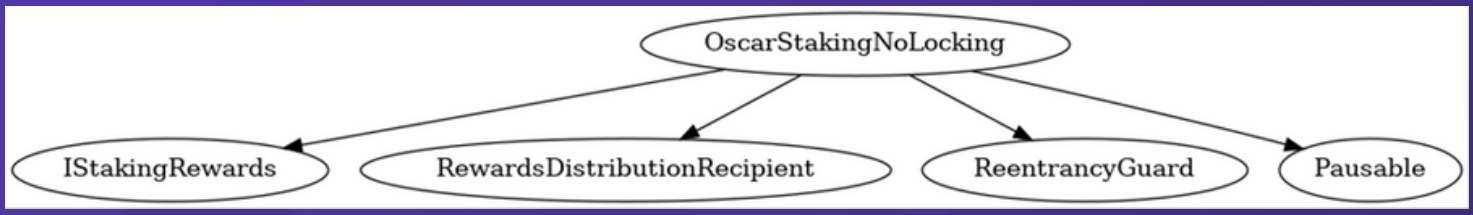
Issues on this level are minor details and warning that can remain unfixed.

Informational

Issues on this level are minor details and warning that can remain unfixed.

expelee

INHERITANCE TREES



expelee

FUNCTION DETAILS

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	transfer	External ❗	🔴	NO ❗
└	allowance	External ❗		NO ❗
└	approve	External ❗	🔴	NO ❗
└	transferFrom	External ❗	🔴	NO ❗
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🔴	NO ❗
└	owner	Public ❗		NO ❗
└	renounceOwnership	Public ❗	🔴	onlyOwner
└	transferOwnership	Public ❗	🔴	onlyOwner
└	_setOwner	Private 🔐	🔴	
SafeMath	Library			
└	tryAdd	Internal 🔒		
└	trySub	Internal 🔒		
└	tryMul	Internal 🔒		
└	tryDiv	Internal 🔒		
└	tryMod	Internal 🔒		
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	sub	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		

expelee

FUNCTION DETAILS

BaseToken	Implementation			
StandardToken	Implementation	IERC20, Ownable, BaseToken		
└	<Constructor>	Public ❗	💵	NO ❗
└	name	Public ❗		NO ❗
└	symbol	Public ❗		NO ❗
└	decimals	Public ❗		NO ❗
└	totalSupply	Public ❗		NO ❗
└	balanceOf	Public ❗		NO ❗
└	transfer	Public ❗	🛑	NO ❗
└	allowance	Public ❗		NO ❗
└	approve	Public ❗	🛑	NO ❗
└	transferFrom	Public ❗	🛑	NO ❗
└	increaseAllowance	Public ❗	🛑	NO ❗
└	decreaseAllowance	Public ❗	🛑	NO ❗
└	_transfer	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_setupDecimals	Internal 🔒	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

UNIT TESTS

Unit Tests:

Staking Tokens: Fail (❌)

1. **Rewards Update:** The contract correctly updated the rewards allocation for the staker.
2. **Staker Profile Update:** The staker's profile was accurately updated post-staking action.
3. **Contract State Update:** The overall state of the contract, including total supply and balances, were correctly updated post-staking.
4. **Locked Tokens Update:** The contract correctly handled the process of locking staker's tokens post-staking.
5. **Security Concerns:** A High vulnerability was detected where an attacker could use the function to be able to have unlocked tokens more than what is intended

Withdrawing Staked Tokens: Pass (☑)

1. **Rewards Update:** The contract correctly updated the rewards allocation for the staker post-withdrawal.
2. **Contract State Update:** The overall state of the contract, including total supply and balances, were correctly updated post-withdrawal.
3. **Staker Profile Update:** The staker's profile and token balance were accurately updated post-withdrawal. The contract correctly transferred tokens back to the staker.
4. **Lock Mechanism Validation:** The contract correctly enforced the lock mechanism, allowing only the withdrawal of unlocked tokens.

Claiming Rewards: Pass (☑)

1. **Owner Rewards Deposit:** The contract owner was able to successfully deposit rewards into the contract.
2. **Reward Claims:** The stakers were able to correctly claim their respective rewards, with the contract distributing the right amounts to each staker.

MANUAL REVIEW

Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standarts.

Vulnerabilities are dividend into three primary risk categroies:
High
Medium
Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

CRITICAL RISK FINDING

Centralization - Ability to Withdraw Staked Tokens

Severity : Critical

Status: Not Resolved

Overview

The owner has the ability to withdraw any ERC20 token from the contract using the recoverERC20 function, potentially including staked tokens.

Code:

```
solidity
function recoverERC20(address _token, uint256 _amount) public
onlyOwner {
  IERC20(_token).transfer(owner(), _amount);
}
```

Suggestion:

Limit the recoverERC20 function to only allow the recovery of unintended tokens, not the tokens being staked.

HIGH RISK FINDING

Logical - Stake Function can be used to escape lock time

Severity : High

Status: Not Resolved

Overview

An attacker can exploit the stake function by making multiple stakes with a single block and same timestamp, then overwriting the last stake with a large number of tokens. This allows them to be able to have a large amount of withdrawable tokens.

Example:

Staking 1 tokens 1000 times within one single transaction (using a contract), then staking 1000 tokens last time, this means staker has 1000 stakeTimes all refering to 1000 tokens. This causes withdrawable function to return 1, 000, 000 tokens as withdrawable balance.

Its expected that attacker only be able to withdraw as much as they have staked in the contract (not more than that) because withdraw function is subtracting the unstake amount from staker's balance.

Code:

```
function stake(uint256 _amount) public {
    TimedStake storage _timedStake = timeStakeInfo[msg.sender];
    _timedStake.stakes[now] = amount;
    _timedStake.stakeTimes.push(now);
    timeStakeInfo[msg.sender] = _timedStake;
}
```

Suggestion:

Implement a mechanism that prevents a single address from overwriting stakes multiple times within the same timestamp or prevents a contract from calling stake function

HIGH RISK FINDING

Centralization - Ability to Arbitrarily Set Lock Period

Severity : High

Status: Not Resolved

Overview

The contract owner possesses the ability to set the lock period to any arbitrary value. This level of control significantly centralizes the contract, potentially leading to misuse.

Code:

```
solidity
function setLockTime(uint256 _lockTime) public onlyOwner {
  lockTime = _lockTime;
}
```

Suggestion:

Implement a cap on the maximum lock period that can be set, or allow the community of stakers to vote on changes to the lock period.

HIGH RISK FINDING

Logical - Staker Staked More than 256 Times

Severity : High

Status: Not Resolved

Overview

The contract uses a uint8 to track stakes, limiting the number of stakes a user can make to 256. If a staker staked more than 256 times, the whole array can't be iterated, potentially causing issues in functionality.

Code:

```
    for (uint8 index = 0; index < _timedStake.stakeTimes.length;
index++) {
        uint256 key = _timedStake.stakeTimes[index];
        if (now.sub(key) > lockDownDuration) {
          amount = amount.add(_timedStake.stakes[key]);
        }
    }
```

Suggestion:

Use a larger integer type like uint256 instead of uint8 for tracking stakes to avoid this issue.

HIGH RISK FINDING

Logical - Not Deleting Unlocked Tokens

Severity : High

Status: Not Resolved

Overview

The dealWithLockdown function is not deleting unlocked tokens from the array; instead, it sets all of them to zero. This may cause unnecessary usage of storage which may also lead to extensive gas usage and could lead to unexpected behaviors.

Code:

```
function dealWithLockdown(address _staker) internal {
  ...
}
```

Suggestion:

Consider deleting unlocked stakes from the array instead of setting them to zero.

MEDIUM RISK FINDING

Centralization -Ability to Charge Up to 10% Unstaking Fee

Severity : Medium

Status: Not Resolved

Overview:

The contract owner can set an unstaking fee (withdraw rate) up to 10%. This creates potential for misuse and adds an extra layer of centralization.

Code:

```
solidity
function setWithdrawFee(uint256 _withdrawFee) public onlyOwner
{
  require(_withdrawFee <= 1000, "invalid fee"); // Max 10%
  withdrawFee = _withdrawFee;
}
```

Suggestion:

Consider reducing the maximum withdraw fee or implement a mechanism where fee changes are subject to community approval.

MEDIUM RISK FINDING

Centralization -Fee Collector Set to Address 0

Severity : Medium

Status: Not Resolved

Overview:

: If the fee collector is set to address 0, any transfers to that address can potentially revert withdrawals, causing unexpected behaviors and disruptions to the contract's operation.

Code:

```
  function setFeeCollector(address _feeCollector) external
onlyOwner {
      feeCollector = _feeCollector;
  }
```

Suggestion:

Implement a condition check to prevent setting the fee collector address to address 0.

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

www.expelee.com

expeleeofficial
expelee
Expelee
expelee
expelee_official
expelee-co

expelee

Building the Futuristic Blockchain Ecosystem

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.