expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

MASTERCHEF

TOKEN OVERVIEW

Risk Findings

Severity	Found
High	3
Medium	1
Low	1
Informational	1

Centralization Risks

Owner Privileges	Description
Can Owner Set Taxes >25% ?	Not Detected
Owner needs to enable trading ?	Not Detected
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	**FAILED**
KYC Verification	**No**
Audit Date	**8 June 2023**

CONTRACT DETAILS

Token Name: MasterChef

Symbol: MasterChef

Network: Binance smart chain

Language: Solidity

Contract Address:
0xf3EDAE148Eb0BDc9FE92dF9EE87c6DE846C85B14

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

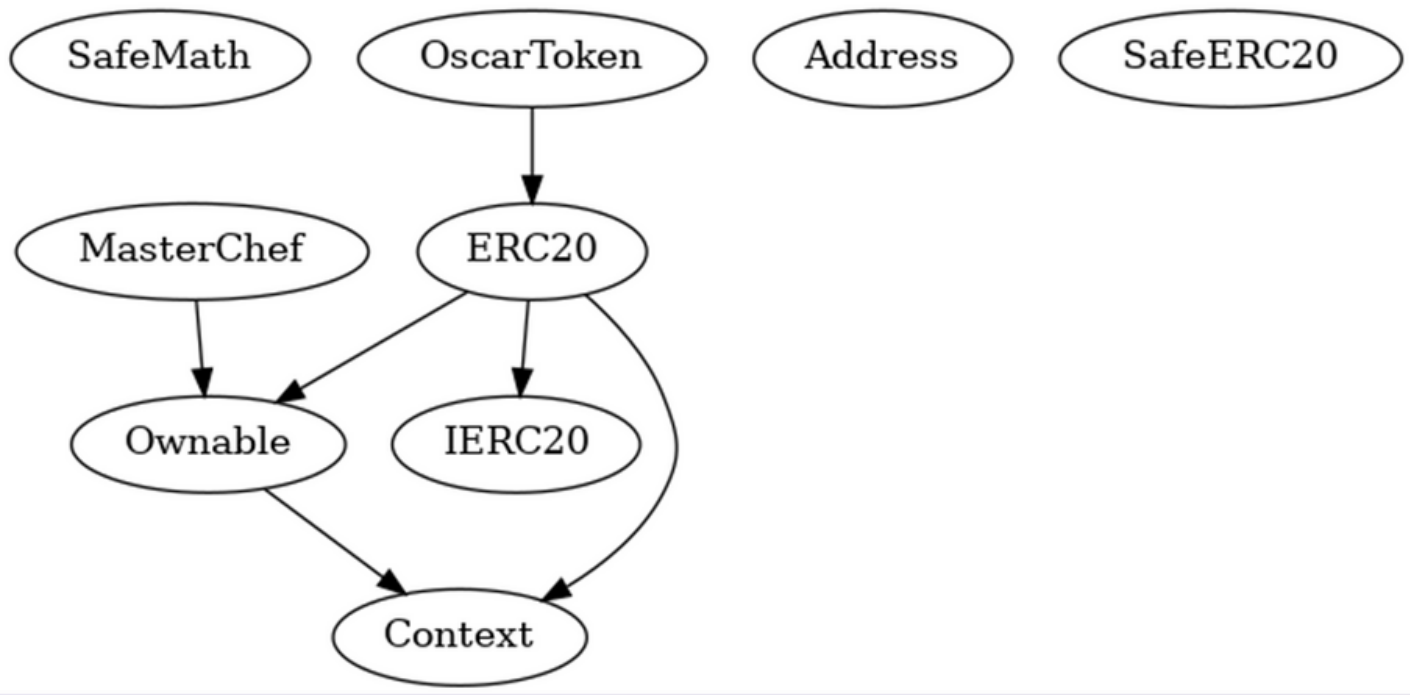
Issues on this level are minor details and warning that can remain unfixed.

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 🔒		
BaseToken	Implementation			
StandardToken	Implementation	IERC20, Ownable, BaseToken		
└	<Constructor>	Public ❗	💵	NO ❗
└	name	Public ❗	NO ❗	
└	symbol	Public ❗	NO ❗	
└	decimals	Public ❗	NO ❗	
└	totalSupply	Public ❗	NO ❗	

FUNCTION DETAILS

└	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

Adding New Pools: Pass (☑)

1. **Rewards Update:** The contract correctly updated the total allocations and adds a new pool
2. **Contract State Update:** The overall state of the contract, including allocation points, and pools array were correctly updated post adding a new pool.

Staking Tokens in pool: Pass (☑)

1. **Rewards Update:** After staking, users got their pending rewards and rewardsDebt updated correctly.
2. **Staker Profile Update:** The staker's profile was accurately updated post-staking action (user.amount and user.rewardsDebt)
3. **Contract State Update:** The overall state of the contract, including pool total deposits and accumulated rewards rate, were correctly updated post-staking.

Withdrawing Staked Tokens: Pass (☑)

1. **Rewards Update:** After withdrawing, users got their pending rewards, withdrawed LP tokens, rewardsDebt updated correctly.
2. **Contract State Update:** The overall state of the contract, including pool total deposits and accumulated rewrds rate updated post-unstaking.
3. **Staker Profile Update:** The staker's profile and staking balance were updated correctly (user.amount and user.rewardsDebt)

Emergency withdraw: Failed (✗) :

Total pool deposits were not updated after emergency withdraw

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			

HIGH RISK FINDING

Configuration / DOS / Data validation – Ability to arbitrary set reward per second settings

Severity : High

Status: Not Resolved

Overview

Owner is able to set an arbitrary value as reward per second and also BONUS_MULTIPLIER, if this reward rate or BONUS_MULTIPLIER is set to max uint256 by a malicious actor, all functions of the contract (except emergency withdraw) would be disabled.

Code:

```
  function updateOscarPerSec(uint256 _oscarPerSec) public
onlyOwner {
    oscarPerSec = _oscarPerSec;
  }

  function updateMultiplier(uint256 multiplierNumber) public
onlyOwner {
    BONUS_MULTIPLIER = multiplierNumber;
  }
```

Suggestion:

Implement a limitation for max amount of oscarPerSec and BONUS_MULTIPLIER or create a governance model to only update this values based on community votes.

HIGH RISK FINDING

Missing logic – Pool states are not updated correctly

Severity : High

Status: Not Resolved

Overview

at emergencyWithdraw function, total deposit of the pool is not updated correctly, exiting the contract throught this function can result in unexpected behaviour

Code:

```
  function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.oscarRewardDebt = 0;
  }
```

Suggestion:

update pool.totalDeposit:

```
  function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    pool.totalDeposit -= user.amount;
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.oscarRewardDebt = 0;
  }
```

HIGH RISK FINDING

Centralization -Ability to add pool for any arbitrary token

Severity : High

Status: Not Resolved

Overview

Owner is able to add any pool to the contract, with an arbitrary amount of allocation point and an arbitrary ERC20 token. A malicious actor can add a new pool with a very large number of allocation points and receive majority of the rewards per second

Code:

```
  function add(uint256 _oscarAllocPoint, IERC20 _lpToken, bool
_withUpdate) public onlyOwner {
    if (_withUpdate) {
      massUpdatePools();
    }
    uint256 lastRewardTime = block.timestamp > startTime ?
block.timestamp : startTime;
    oscarTotalAllocPoint =
oscarTotalAllocPoint.add(_oscarAllocPoint);

 poolInfo.push(
      PoolInfo({
        lpToken: _lpToken,
        oscarAllocPoint: _oscarAllocPoint,
        lastRewardTime: lastRewardTime,
        accOscarPerShare: 0,
        totalDeposit: 0
      })
    );
  }
```

HIGH RISK FINDING

Suggestion:

Implement a more decentralized method for adding new pools or changing states of an existing pool

MEDIUM RISK FINDING

Configuration / DOS / Data validation – Setting treasury wallet to any arbitrary address

Severity : Medium

Status: Not Resolved

Overview

treasury address can be set to any arbitrary address. If treasury address is set to address(0), depending on impelementation of the reward token claiming rewards could be disabled.
This is because in majority of ERC20 tokens, transferring to this address is forbidden

Code:

```
function setTreasury(address _treasury) public onlyOwner {
  treasury = _treasury;
}
```

Suggestion:
Ensure that new treasury wallet is not address(0).

LOW RISK FINDING

Numerics – teamRewardPerSec seems to be invalid according to the comment

Severity : Low

Status: Not Resolved

Overview

According to the comment written in the contract team rewards per second is expected to be >= 0.3, while 3170979198376460O is approximatly 0.0317 (18 decimals)

Code:

```
  // The amount that the team gets per second (a little over 0.3 a second)
  uint256 public teamRewardPerSec = 3170979198376460O;
```

Suggestion:

teamRewardPerSec should be 31709791983764600O according to the documentation

INFORMATIONAL RISK FINDING

Logical – transferOscarOwnership may be redundant

Severity : Informational

Status: Not Resolved

Overview:

transferOscarOwnership function trnasfers ownership of oscar token to a new address, if owner of oscar token is not masterchef conract, this function will be reverted on calls

Code:

```
  function transferOscarOwnership(address _to) external onlyOwner
{
    oscar.transferOwnership(_to);
  }
```

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.