Cyberscope

Audit Report

Opulence Management

April 2022

Type	ERC20
Network	AVAX
SHA256	178a995c82d9c510646c4fd040baaa59111231bd6a9240da3eb08314033ff13c
Audited by	© cyberscope

Table of Contents

Source Files

Filename	SHA256
contract.sol	178a995c82d9c510646c4fd040baaa59111231bd6a9240da3eb08314033ff13c

Audit Updates

Initial Audit	16th April 2022
Corrected	

Contract Analysis

The Management Contract is responsible for creating nodes and delivering rewards. The role OnlyManager can interact with the contract and has access to the following functionality:

- Create New Node for a user and set his rewards per day
- Calculate available rewards for all the Nodes of a specific user
- Get Nodes of specific user
- Get the Node names of a specific user
- Get the Node created time of a specific user
- Get the last time a specific user claimed rewards
- Get the daily Node rewards of a specific user
- Get all the Node rewards of a specific user
- Cashout the rewards for a specific user
- Compound the rewards of a specific user

Contract Diagnostics

● Critical ● Medium ● Minor

Severity	Code	Description
●	MC	Missing Check
●	CR	Code Repetition
●	DN	Duplicate Namings
●	L01	Public Function could be Declared External
●	L04	Conformance to Solidity Naming Conventions
●	L07	Missing Events Arithmetic
●	L11	Unnecessary Boolean equality
●	L13	Divide before Multiply Operation

MC - Missing Check

Criticality	minor
Location	contract.sol#L275

Description

The claimInterval is used as a divisor in the reward calculation. Since the contract owner can arbitrary change it, there are 2 potential issues:

Zero division

If the contract owner sets the claimInterval to zero, then the reward calculation will revert.

Rewards Calculation Underflow

If the contract owner sets a small number to claimInterval, the reward will yield a relatively big number.
Later, if the contract owner sets a big number to claimInterval the expression `(block.timestamp - node.createTime).div(claimInterval).mul(node.rewardPerDay)` may yield a smaller number than the previous `rewardedAmount` value. As a result the entire expression will underflow.

```
function setClaimInterval(uint256 _interval) public onlyOwner {
    claimInterval = _interval;
}
```

```
reward = (block.timestamp - node.createTime)
    .div(claimInterval)
    .mul(node.rewardPerDay)
    .sub(node.rewardedAmount);

totalRewards += reward;
node.lastClaimTime = block.timestamp;
node.rewardedAmount = node.rewardedAmount + reward;
```

Recommendation

The contract should guarantee that the claimInterval will not revert the rewards calculation.

CR - Code Repetition (1/2)

Criticality	minor
Location	contract.sol#L297,323

Description

There are code segments that are repetitive in the contract. Those segments increase the code size of the contract unnecessarily. The functions airdropNode and createNode have the exact same logic, one of them can be removed.

```
function createNode(
    address account,
    string memory _name,
    uint256 _rewardPerDay
  ) external onlyManager {
    require(
      nodeCountOfUser[account] < nodeLimit,
      "MANAGEMENT: CREATE NODE LIMIT ERROR"
    );

    nodesOfUser[account].push(
      NodeInfo({
        name: _name,
        createTime: block.timestamp,
        lastClaimTime: block.timestamp,
        rewardPerDay: _rewardPerDay,
        rewardedAmount: 0
      })
    );

    nodeCountOfUser[account] += 1;
    totalCount += 1;

    emit CREATE_NODE(account, _name, _rewardPerDay, block.timestamp);
  }
```

Recommendation

Remove one of these duplicate functions.

CR - Code Repetition (2/2)

Criticality	minor
Location	contract.sol#L399,404

Description

There are code segments that are repetitive in the contract. Those segments increase the code size of the contract unnecessarily. The line of code that sets the node's last claim to block timestamp can be moved before the if statement as it will execute under both circumstances.

```
if (returnValue + reward < amount) {
          node.lastClaimTime = block.timestamp;
          node.rewardedAmount = node.rewardedAmount + reward;

          returnValue += reward;
        } else {
          node.lastClaimTime = block.timestamp;
          node.rewardedAmount = node.rewardedAmount + amount - returnValue;

          returnValue += amount - returnValue;
        }
```

Recommendation

Move the logic of setting node's lastClaimTime before the if statement.

DN - Duplicate Namings

Criticality	minor
Location	contract.sol#L344,369

Description

Duplicate named functions can be problematic and should be avoided according to solidity naming conventions. Use declarative naming patterns.

```
function calculateAvailableReward(address account)
   external
   view
   onlyManager
   returns (uint256)
 {
   uint256 totalRewards = 0;

   for (uint256 i = 0; i < nodeCountOfUser[account]; i++) {
     totalRewards += _calculateAvailableReward(account, i);
   }
   return totalRewards;
 }

 function calculateAvailableReward(address account, uint256 _index)
   external
   view
   onlyManager
   returns (uint256)
 {
   uint256 reward;
   reward = _calculateAvailableReward(account, _index);
   return reward;
 }
```

Recommendation

Rename one of the functions above to a unique name.

L01 - Public Function could be Declared External

Criticality	minor
Location	contract.sol#L179,198,207,260,264,268,272

Description

Public functions that are never called by the contract should be declared external to save gas.

```
setClaimInterval
setNodeLimit
removeManager
addManager
transferOwnership
renounceOwnership
owner
```

Recommendation

Use the external attribute for functions never called from the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	minor
Location	contract.sol#L237,243,260,264,268,272,299,300,325,326,364,443,644

Description

Solidity defines a naming convention that should be followed. Rule exceptions:

- Allow constant variable name/symbol/decimals to be lowercase.
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
_i
_index
_rewardPerDay
_name
_interval
_limit
_manager
AIREDROP_NODE
CREATE_NODE
...
```

Recommendation

Follow the Solidity naming convention.
https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions

L07 - Missing Events Arithmetic

Criticality	minor
Location	contract.sol#L268,272

Description

Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
claimInterval = _interval
nodeLimit = _limit
```

Recommendation

Emit an event for critical parameter changes.

L11 - Unnecessary Boolean equality

Criticality	minor
Location	contract.sol#L251

Description

The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(bool,string)(managers[msg.sender] == true,MANAGEMENT: NOT MANAGER)
```

Recommendation

Remove the equality to the boolean constant.

L13 - Divide before Multiply Operation

Criticality	minor
Location	contract.sol#L377,414,443,616,644

Description

Performing divisions before multiplications may cause lose of prediction.

```
temp = (48 + uint8(_i - (_i / 10) * 10))
reward = (block.timestamp -
node.createTime).div(claimInterval).mul(node.rewardPerDay).sub(node.rewardedAmo
unt)
```

Recommendation

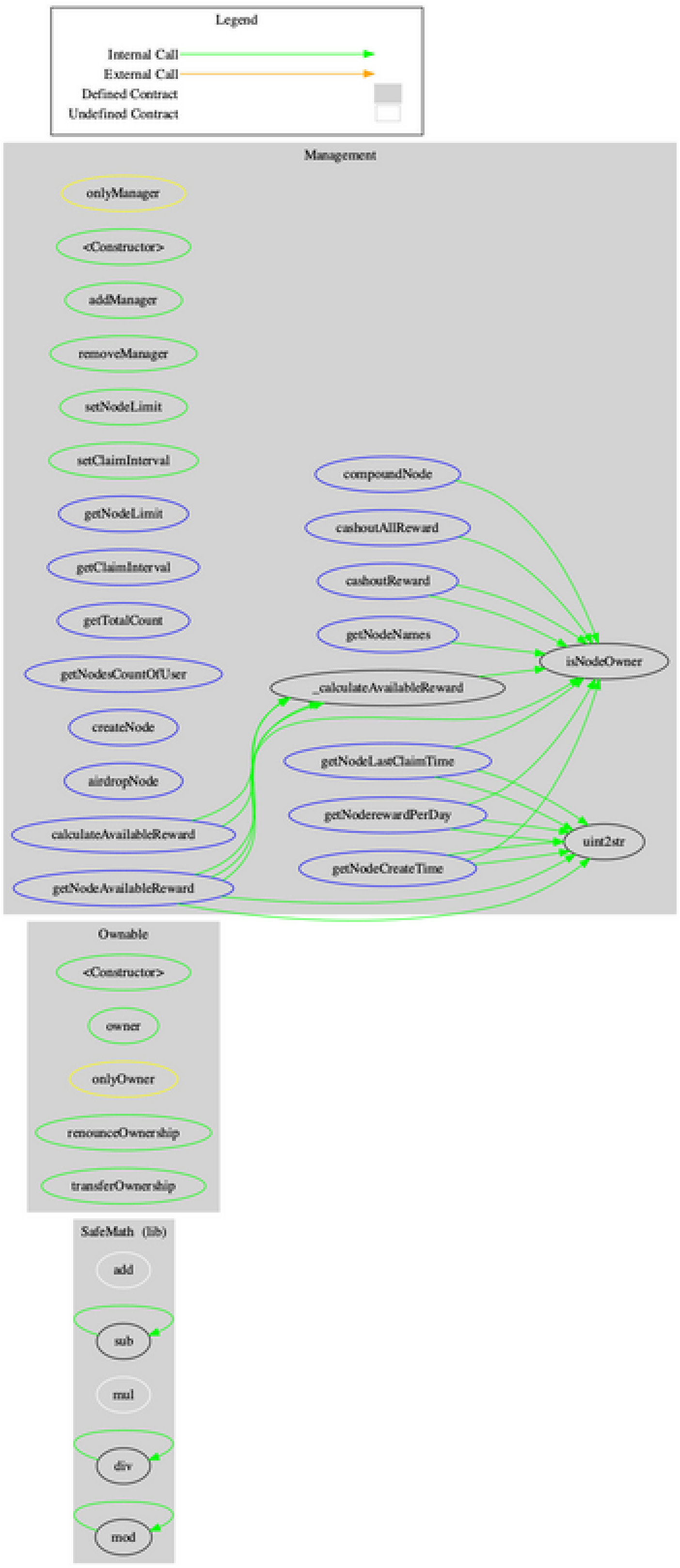
The multiplications should be prior to the divisions.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation			
	<Constructor>	Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
Management	Implementation	Ownable		
	<Constructor>	Public	✓	-
	addManager	Public	✓	onlyOwner
	removeManager	Public	✓	onlyOwner
	setNodeLimit	Public	✓	onlyOwner
	setClaimInterval	Public	✓	onlyOwner
	getNodeLimit	External		onlyManager
	getClaimInterval	External		onlyManager
	getTotalCount	External		onlyManager
	getNodesCountOfUser	External		onlyManager
	createNode	External	✓	onlyManager
	airdropNode	External	✓	onlyManager
	calculateAvailableReward	External		onlyManager

	calculateAvailableReward	External		onlyManager
	compoundNode	External	✓	onlyManager
	cashoutAllReward	External	✓	onlyManager
	cashoutReward	External	✓	onlyManager
	getNodeNames	External		onlyManager
	getNodeCreateTime	External		onlyManager
	getNodeLastClaimTime	External		onlyManager
	getNoderewardPerDay	External		onlyManager
	getNodeAvailableReward	External		onlyManager
	isNodeOwner	Internal		
	_calculateAvailableReward	Internal		
	uint2str	Internal		

Contract Flow



Summary

Opec Management is the reward mechanism contract for the Opec protocol. This audit focuses on the correct functionality, the security concerns and some performance improvements.

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.

Cyberscope team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed.

The Cyberscope 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 Cyberscope receive a payment to manipulate those results or change the awarding badge that we will be adding in our website.

Always 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 Cyberscope team disclaims any liability for the resulting losses.

About Cyberscope

Coinscope audit and K.Y.C. service has been rebranded to Cyberscope.

Coinscope is the leading early coin listing, voting and auditing authority firm. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.

Cyberscope and Coinscope are aiming to make crypto discoverable and efficient globally. They provides all the essential tools to assist users draw their own conclusions.

The Cyberscope team

https://www.cyberscope.io