Cyberscope

Audit Report

Hubin Network Staking

October 2022

Type	BEP20
Network	BSC
Address	0xFC22a69e367F8c58cE65BcD3Db3cCDcE1734f05a
Audited by	© cyberscope

Table of Contents

Contract Review

Contract Name	HubinNetworkStaking
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Licence	Unlicense
Explorer	https://bscscan.com/token/0xFC22a69e367F8c58cE65BcD3Db3cCDcE1734f05a

Audit Updates

Initial Audit	27th October 2022
Corrected	

Source Files

Filename	SHA256
Address.sol	d547cab49a97d7f8fd633db312b8a074ef816dd4544af72e4208382e76391647
Context.sol	6de5302543723d32c8eaf17becc4525936e16d9c4551455c93d306b9b72c0799
HubinNetworkStaking.sol	0d46a574c1843c6a4854eaa9f5d310e3b3ea304914c2d21cd5529b220375928d
IERC20.sol	6654ca211d7ed22937fae539bcf24e0bda89ba7489d4a2f439cc52f53db6ec4d
Ownable.sol	c53bedd328735571fdf8d130387e2ac3c12a56ea27978e4694b22457b8ab821f
ReentrancyGuard.sol	d403c9c184c27e1320a5bc543a8efbdc54079110043c827ec513b785c2db20a3
SafeERC20.sol	71c37232113f52433042d788efd366ceaecf78d412f009b8a88d776cb6934646
SafeMath.sol	8213cd58437a8a6b5acb2a85358cd245f5ae0e44674af84c60a312b8b86049d7

Introduction

The contract Hubin Network Staking implements a staking mechanism. The contract has three preconfigured staking plans. Each plan has a different duration and APRs. The durations are 15, 30, and 45 days and the APRs are 30, 60, and 90. Each staking plan has its corresponding deposit, withdrawal fee, and early withdrawal fee.

When the staking plan from a user is finished, the user can keep withdrawing the rewards. The rewards are calculated proportionally to the time that has elapsed since the last claim.

The contract uses the staking plan finish date to apply early unstake penalty fees.

Plan	Duration	APR
1	15 days	30%
2	30 days	60%
3	45 days	90%

Roles

The contract has an admin role. The admin role has the authority to configure:

- Change the APR of a plan.
- Change the deduction amount of a plan.
- Change the early penalty of a plan.
- Stop the staking of a plan.

The users has the authority to:

- Stake multiple times in a Staking plan. On every staking, a deposit fee is applied.
- Monitor how much tokes they have invested in a Staking plan.
- Unstake the invested amount. If a user unstakes before the staking period elapsed penalty is applied. Additionally, there is a withdrawal fee.
- Earn the staking reward after the staking period has elapsed.

Contract Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	STC	Succeeded Transfer Check	Unresolved
●	BLC	Business Logic Concern	Unresolved
●	SKM	Storage Keywork Misuse	Unresolved
●	CR	Code Repetition	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

STC - Succeeded Transfer Check

Criticality	minor / informative
Location	contract.sol#L89,95,219,223,247
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(stakingToken).transferFrom(msg.sender, address(this), _amount);

IERC20(stakingToken).transfer(stakingToken, deductionAmount);

IERC20(stakingToken).transfer(stakingToken, deductionAmount);

IERC20(stakingToken).transfer(msg.sender, tamount - _penalty + _earned);

IERC20(stakingToken).transfer(msg.sender, _earned);
```

Recommendation

The contract should check if the result of the transfer methods is successful.

BLC - Business Logic Concern

Criticality	minor / informative
Location	contract.sol#L116
Status	Unresolved

Description

The method ‘canWithdrawAmount’ calculates the same result twice. The variables ‘_stakedAmount’ and ‘_canWithdraw’ always return exactly the same number.

```
function canWithdrawAmount(uint256 _stakingId, address account) public
override view returns (uint256, uint256) {
        uint256 _stakedAmount = 0;
        uint256 _canWithdraw = 0;
        for (uint256 i = 0; i < stakes[_stakingId][account].length; i++) {
            Staking storage _staking = stakes[_stakingId][account][i];
            _stakedAmount = _stakedAmount.add(_staking.amount);
            _canWithdraw = _canWithdraw.add(_staking.amount);
        }
        return (_stakedAmount, _canWithdraw);
}
```

Recommendation

The contract could safely remove one of the two variables since they produce the same result.

SKM - Storage Keywork Misuse

Criticality	minor / informative
Location	contract.sol#L120,130
Status	Unresolved

Description

The methods ‘canWithdrawAmount’ and ‘earnedToken’ are using the storage keyword but they do not mutate the state.

```
function canWithdrawAmount(uint256 _stakingId, address account) public
override view returns (uint256, uint256) {
...
}

function earnedToken(uint256 _stakingId, address account) public override view
returns (uint256, uint256) {
...
}
```

Recommendation

The contract should use the storage keyword when it is mutating the state variables.

CR - Code Repetition

Criticality	minor / informative
Location	contract.sol
Status	Unresolved

Description

There are code segments that are repetitive in the contract. Those segments increase the code size of the contract unnecessarily.

- The `amount.sub(deductionAmount);` expression is calculated twince in the same method.
- The calculation of the early penalty is repetitive in the source code.
- The calculation of the earnings is repetitive in the source code.
- The method ‘claimEarned()’ uses the same calculations with the ‘earnedToken()’.

```
_staking.amount = amount.sub(deductionAmount);
_staking.stakeAt = block.timestamp;
_staking.endstakeAt = block.timestamp + plan.stakeDuration;

plan.overallStaked = plan.overallStaked.add(
    amount.sub(deductionAmount)
);

//

x = x.add(
    y
    .mul(plan.earlyPenalty)
    .div(100)
);

//
```

```
x = x.add(
    y
    .mul(block.timestamp - _staking.stakeAt)
    .div(periodicTime)
    .mul(plan.apr)
    .div(100)
);
```

Recommendation

Create an internal function that contains the code segment and remove it from all the sections.

L02 - State Variables could be Declared Constant

Criticality	minor / informative
Location	HubinNetworkStaking.sol#L51,52,55,54,53,50
Status	Unresolved

Description

Constant state variables should be declared constant to save gas.

```
planLimit
minAPR
maxEarlyPenalty
maxWithdrawDeduction
maxDepositDeduction
periodicTime
```

Recommendation

Add the constant attribute to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	minor / informative
Location	HubinNetworkStaking.sol#L229,263,268,116,258,77,250,152,278,273,127
Status	Unresolved

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.

```
_stakingId
_deduction
_percent
_amount
_account
_penalty
_conclude
...
```

Recommendation

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

L13 - Divide before Multiply Operation

Criticality	minor / informative
Location	HubinNetworkStaking.sol#L127,152
Status	Unresolved

Description

Performing divisions before multiplications may cause lose of prediction.

```
_canClaim = _canClaim.add(_staking.amount.mul(block.timestamp -
_staking.stakeAt).div(periodicTime).mul(plan.apr).div(100))
_earned = _earned.add(_staking.amount.mul(block.timestamp -
_staking.stakeAt).div(periodicTime).mul(plan.apr).div(100))
_earned = _earned.add(amount.mul(block.timestamp -
_staking.stakeAt).div(periodicTime).mul(plan.apr).div(100))
```

Recommendation

The multiplications should be prior to the divisions.

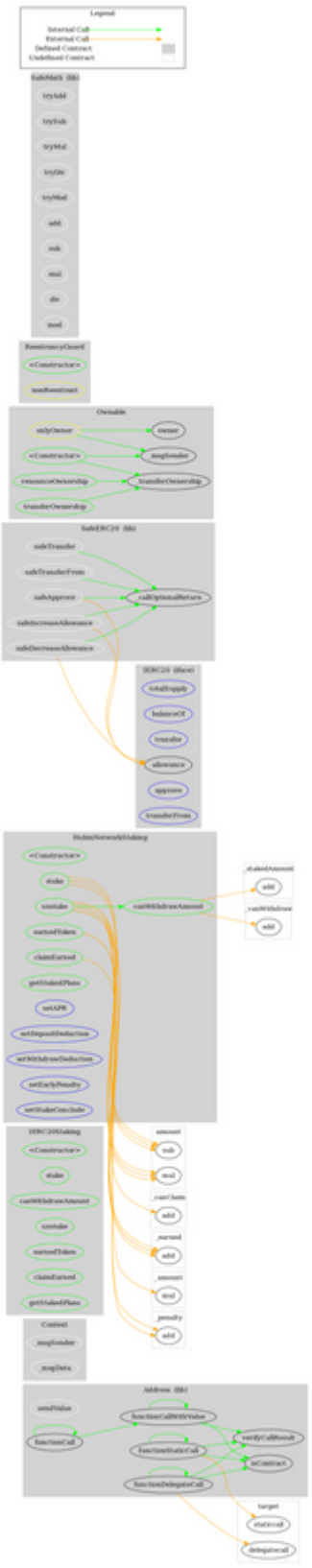
Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20Staking	Implementation	ReentrancyGuard, Ownable		
	<Constructor>	Public	✓	-
	stake	Public	✓	-
	canWithdrawAmount	Public		-
	unstake	Public	✓	-
	earnedToken	Public		-
	claimEarned	Public	✓	-
	getStakedPlans	Public		-
HubinNetwork	Implementation	IERC20Stak		

Staking		ing		
	<Constructor>	Public	✓	IERC20Staking
	stake	Public	✓	-
	canWithdrawAmount	Public		-
	earnedToken	Public		-
	unstake	Public	✓	-
	claimEarned	Public	✓	-
	getStakedPlans	Public		-
	setAPR	External	✓	onlyOwner
	setDepositDeduction	External	✓	onlyOwner
	setWithdrawDeduction	External	✓	onlyOwner
	setEarlyPenalty	External	✓	onlyOwner
	setStakeConclude	External	✓	onlyOwner
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation	Context		
	<Constructor>	Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ReentrancyGuard	Implementation			
	<Constructor>	Public	✓	-
SafeERC20	Library			

	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	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		

Contract Flow



Summary

The Hubin Network Staking contract implements a staking mechanism. This audit investigates security issues and mentions business logic concerns and potential 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 provide all the essential tools to assist users draw their own conclusions.

The Cyberscope team

https://www.cyberscope.io