Cyberscope

Audit Report

NUTGAIN Staking

April 2022

Type	BEP20
Network	BSC
Address	0xF1477207249C4429a0B872dFE91E276176DCedff
Audited by	© cyberscope

Table of Contents

Contract Review

Contract Name	NutGainSingleStaking
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Licence	MIT
Explorer	https://bscscan.com/token/0xf1477207249c4429a0b872dfe91e276176dcedff

Source Files

Filename	**SHA256**
contract.sol	1d7015d97935ed3fc35fbd55165d192b66105ecde365fbd4642989491e0dcca8

Audit Updates

Initial Audit	21st April 2022
Corrected	

Contract Diagnostics

Critical Medium Minor

Severity	Code	Description
	DSO	Data Structure Optimization
	CO	Code Optimization
	STC	Succeeded Transfer Check
	CR	Code Repetition
	MC	Missing Check
	L01	Public Function could be Declared External
	L02	State Variables could be Declared Constant
	L04	Conformance to Solidity Naming Conventions
	L07	Missing Events Arithmetic
	L09	Dead Code Elimination
	L13	Divide before Multiply Operation
	L14	Uninitialized Variables in Local Scope

Notes

- if tokenSupply * apy is less than 1051200000, then no rewards will be distributed since the division will be less than 1.
- A user can withdraw the deposited amount with two ways. The emergencyWithdraw() that receives the entire amount, and the withdraw() that receives the amount redacted by an early withdrawal tax. The users have no reason to withdraw tokens using the withdraw() method since they have the option to call the emergencyWithdraw() method

Data Structure Optimization

Criticality	minor
Location	contract.sol#L778

Description

The poolInfo contains the array of the pools. The userInfo contains a map from the pool index to a map from the user's address to the user info. The first mapping of userInfo is mirroring the poolinfo array, as a result the code should keep in sync these to structures.

```
PoolInfo[] public poolInfo;
// Info of each user that stakes LP tokens.
mapping(uint256 => mapping(address => UserInfo)) public userInfo;
```

Recommendation

Since the userInfo is a subset of the poolInfo structure, the PoolInfo could embody the mapping(address => UserInfo) structure. Hence, the source will not have to keep in sync these to indexes.

```
struct PoolInfo {
    uint256 apy;
    uint256 lastRewardBlock;
    uint256 accTokenPerShare;
    uint256 withdrawLockPeriod;
    uint256 balance;
    mapping(address => UserInfo) userInfo;
}
```

CO - Code Optimization

Criticality	minor
Location	contract.sol#L856,876

Description

The contract is declaring that there are properties that will be used for storage even if they are used for temporary calculations.

```
function pendingToken(uint256 _pid, address _user) public view returns (uint256)
{
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];

...
function canHarvest(uint256 _pid, address _user) public view returns (bool) {
    UserInfo storage user = userInfo[_pid][_user];
    PoolInfo storage pool = poolInfo[_pid];
    return block.timestamp >= user.lastDepositTime.add(pool.withdrawLockPeriod);
}
```

Recommendation

The class properties that are not affected should be declared as *memory*.

STC - Succeeded Transfer Check

Criticality	minor
Location	contract.sol#L1045

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.

```
function safeTokenTransfer(address _to, uint256 _amount) internal {
    uint256 tokenBal = token.balanceOf(address(this));
    if (_amount > tokenBal) {
        token.transfer(_to, tokenBal);
    } else {
        token.transfer(_to, _amount);
    }
}
```

Recommendation

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

CR - Code Repetition

Criticality	minor
Location	contract.sol#L973,943,985,1001

Description

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

```
pool.balance = pool.balance.sub(amount);
user.amount = 0;
user.rewardDebt = 0;
user.rewardLockedUp = 0;

///

uint256 totalRewards = pending.add(user.rewardLockedUp);

// reset lockup
totalLockedUpRewards = totalLockedUpRewards.sub(user.rewardLockedUp);
user.rewardLockedUp = 0;
// send rewards
safeTokenTransfer(msg.sender, totalRewards);
```

Recommendation

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

MC - Missing Check

Criticality	minor
Location	contract.sol#L891,914,935,955,981,994

Description

The following functions require the pid or the sender’s address but are not checking if the pool id and sender’s address exist in the current structures.

- updatePool
- Deposit
- Harvest
- Withdraw
- emergencyWithdraw
- payOrLockupPendingToken

```
function deposit(uint256 _pid, uint256 _amount) external nonReentrant {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    updatePool(_pid);
    payOrLockupPendingToken(_pid);
...
```

Recommendation

The contract should properly check if the indexes exist in the current structures.

L01 - Public Function could be Declared External

Criticality	minor
Location	contract.sol#L671,680,856,1054

Description

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

```
getBlock
pendingToken
transferOwnership
renounceOwnership
```

Recommendation

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

L02 - State Variables could be Declared Constant

Criticality	minor
Location	contract.sol#L786

Description

Constant state variables should be declared constant to save gas.

```
DEAD
```

Recommendation

Add the constant attribute to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	minor
Location	contract.sol#L819,820,836,837,847,856,876,891,914,935,955,981,994,1016,1024,1045,786

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.

```
DEAD
_amount
_to
_earlyFeePercent
_pid
_user
_from
_withdrawLockPeriod
_withUpdate
...
```

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#L1058

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.

```
treasure = treasure.sub(amount)
```

Recommendation

Emit an event for critical parameter changes.

L09 - Dead Code Elimination

Criticality	minor
Location	contract.sol#L392,417,466,476,442,452,366,542,567,558

Description

Functions that are not used in the contract, and make the code's size bigger.

```
safeIncreaseAllowance
safeDecreaseAllowance
safeApprove
sendValue
functionStaticCall
functionDelegateCall
functionCallWithValue
functionCall
...
```

Recommendation

Remove unused functions.

L13 - Divide before Multiply Operation

Criticality	minor
Location	contract.sol#L856,891

Description

Performing divisions before multiplications may cause lose of prediction.

```
tokenPerBlock = tokenSupply.mul(pool.apy).div(100).div(10512000)
```

Recommendation

The multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	minor
Location	contract.sol#L962

Description

The are variables that are defined in the local scope and are not initialized.

```
burnAmount
```

Recommendation

All the local scoped variables should be initialized.

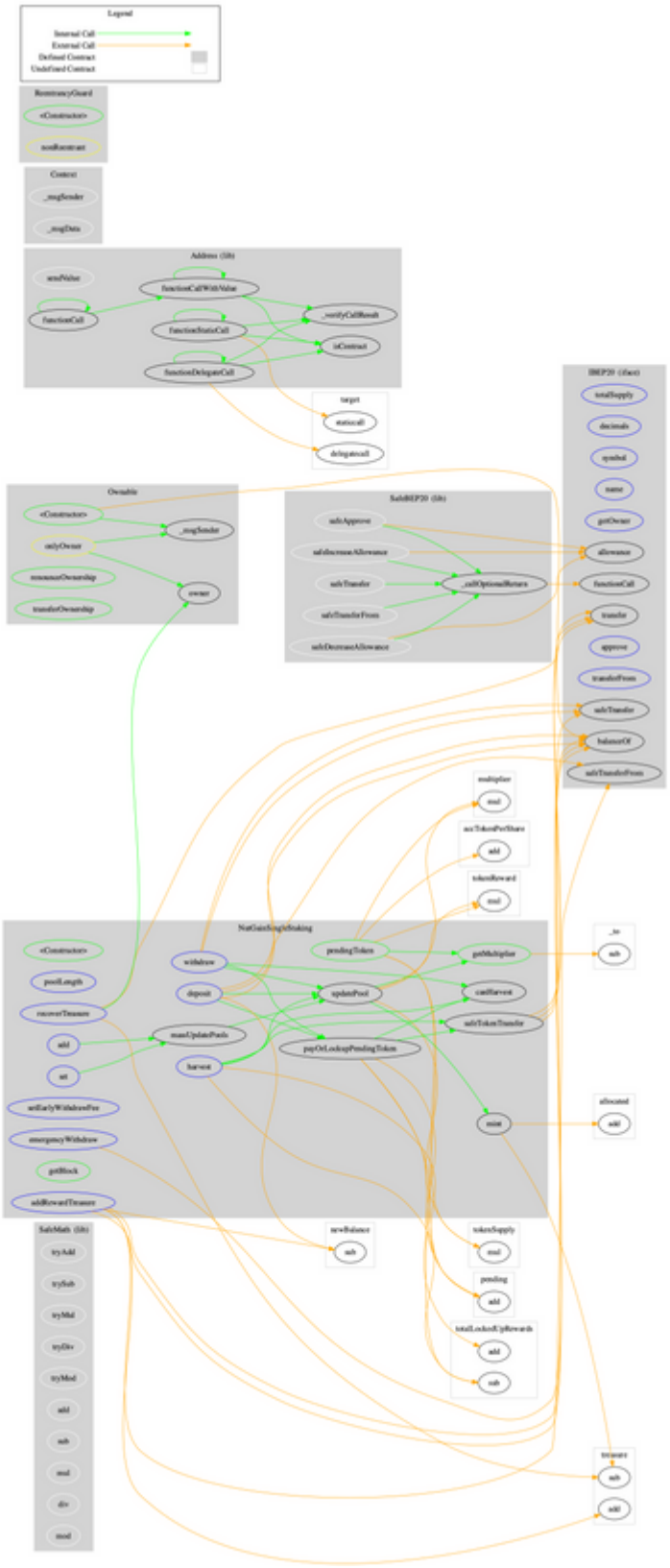
Contract Functions

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
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		
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Address	Library			
	isContract	Internal		

	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
SafeBEP20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
	<Constructor>	Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
ReentrancyGuard	Implementation			
	<Constructor>	Public	✓	-
NutGainSingleStaking	Implementation	Ownable, ReentrancyGuard		

	<Constructor>	Public	✓	-
	poolLength	External		-
	add	External	✓	onlyOwner
	set	External	✓	onlyOwner
	getMultiplier	Public		-
	pendingToken	Public		-
	canHarvest	Public		-
	massUpdatePools	Public	✓	-
	updatePool	Public	✓	-
	deposit	External	✓	nonReentrant
	harvest	External	✓	nonReentrant
	withdraw	External	✓	nonReentrant
	emergencyWithdraw	External	✓	nonReentrant
	payOrLockupPendingToken	Internal	✓	
	setEarlyWithdrawFee	External	✓	onlyOwner
	addRewardTreasure	External	✓	-
	mint	Internal	✓	
	safeTokenTransfer	Internal	✓	
	getBlock	Public		-
	recoverTreasure	External	✓	onlyOwner

Contract Flow



Summary

NUTGAIN staking contract implements a staking related functionality. There are pools that combine different APYs and Locking periods. The users have the ability to deposit funds in one of the pools. The users can either redeem the rewards or withdraw the entire amount. This audit focuses on some security concerns, performance optimizations and business logic 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