Cyberscope

Audit Report

Nut2Earn

July 2022

Github Nut2Earn/-NUT-Token/blob/main/Nut2Earn.sol

Commit 2e42554cb83ade74a7e0832f67d9aad0b432a87f

Audited by © cyberscope

Table of Contents

Contract Review

Github	Nut2Earn/-NUT-Token/blob/main/Nut2Earn.sol
Commit	2e42554cb83ade74a7e0832f67d9aad0b432a87f

Source Files

Filename	**SHA256**
contract.sol	b85151802f8b382b8438587b955626435e0dd7a6c9c5 b8d677c9a56c8d621561

Audit Updates

Initial Audit	31st July 2022
Corrected	1st August 2022

Contract Analysis

Critical Medium Minor Pass

Severity	Code	Description
●	ST	Contract Owner is not able to stop or pause transactions
●	OCTD	Contract Owner is not able to transfer tokens from specific address
●	OTUT	Owner Transfer User's Tokens
●	ELFM	Contract Owner is not able to increase fees more than a reasonable percent (25%)
●	ULTW	Contract Owner is not able to increase the amount of liquidity taken by dev wallet more than a reasonable percent
●	MT	Contract Owner is not able to mint new tokens
●	BT	Contract Owner is not able to burn tokens from specific wallet
●	BC	Contract Owner is not able to blacklist wallets from selling

ST - Stop Transactions

Criticality	critical
Location	contract.sol#L708

Description

The contract owner has the authority to stop transactions for all users excluding the owner.

The owner may take advantage of it by setting the `selltreasuryFee` `lower than the` `totalReferralFee.` Then the variable will underflow and the function will revert. As a result the contract will turn into a honeypot.

```
 function takeFee(address sender, address recipient, uint256 gonAmount)
internal returns (uint256){
        uint256 setrealFee = totalBuyFee;
        uint256 setbuytreasuryFee = buytreasuryFee;
        uint256 setselltreasuryFee = selltreasuryFee;
        if(automatedMarketMakerPairs[recipient]) setrealFee = totalSellFee;
        uint256 feeAmount = gonAmount.mul(setrealFee).div(feeDenominator);
        // referrals
        if (automatedMarketMakerPairs[sender]) {
            address UplineAddressBuyer = getUpline(recipient);
            if (UplineAddressBuyer != address(0)){
                setbuytreasuryFee -= totalReferralFee;
                uint256 _uplineBuyerReward =
gonAmount.mul(referrer).div(feeDenominator);
                feeAmount = gonAmount.mul(setrealFee -
referee).div(feeDenominator);
                _gonBalances[UplineAddressBuyer] =
_gonBalances[UplineAddressBuyer].add(_uplineBuyerReward);
                addReferralFee(UplineAddressBuyer,
_uplineBuyerReward.div(_gonsPerFragment) );
            }
        }else if (automatedMarketMakerPairs[recipient]) {
            address UplineAddress = getUpline(sender);
            if (UplineAddress != address(0))\{
                setselltreasuryFee -= totalReferralFee;
```

A possible flow could be

```
1. setBuyFees(25, 0, 0, 0)
2. setReferralSettings(25, 25)
3. setBuyFees(1, 0, 0, 0)
```

Recommendation

The contract could embody a check for not allowing setting `selltreasuryFee` lower than the `totalReferralFee`. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply.

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

OCTD - Owner Contract Tokens Drain

Criticality	minor
Location	contract.sol#L921

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `claimTokens` function.

```
function rescueToken(address tokenAddress, uint256 tokens) external onlyOwner
returns (bool success){
             if(tokens == 0){
            tokens = ERC20Detailed(tokenAddress).balanceOf(address(this));
        }
        return ERC20Detailed(tokenAddress).transfer(msg.sender, tokens);
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Contract Diagnostics

Critical Medium Minor

Severity	Code	Description
	MTS	Manipulate Total Supply
	CO	Code Optimization
	L02	State Variables could be Declared Constant
	L04	Conformance to Solidity Naming Conventions
	L13	Divide before Multiply Operation

MTS - Manipulate Total Supply

Criticality	minor
Location	contract.sol#L807

Description

Owner is able to manipulate total supply. This change will have a direct impact on the token price and Market Cap.

```
function coreRebase(int256 supplyDelta) private returns (uint256) {
        uint256 epoch = block.timestamp;
        if (supplyDelta == 0) {
            emit LogRebase(epoch, _totalSupply);
            return _totalSupply;
        } else {
            if ((_totalSupply.add(uint256(supplyDelta))) >= MAX_SUPPLY) {
            // in case the rebase will cause the supply to pass MAX_SUPPLY,
autorebase will be turned off & rebase will not happen.
            autoRebase = false;
            emit LogRebase(epoch, _totalSupply);
            return _totalSupply;
        } else {
            _totalSupply = _totalSupply.add(uint256(supplyDelta));
         }
        }
        _gonsPerFragment = TOTAL_GONS.div(_totalSupply);
        nextRebase = epoch + rebaseFrequency;
        emit LogRebase(epoch, _totalSupply);
        return _totalSupply;
    }
```

Recommendation

The contract owner should carefully manage the adjustment of the circulating supply (increases or decreases), according to the token’s price fluctuations.

CO - Code Optimization

Criticality	medium
Location	contract.sol#L708,884,902

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations

The variable `setbuytreasuryFee` is redundant. The variariable is defined in the function's local scope and is not used in the function implementation.

```
function takeFee(address sender, address recipient, uint256 gonAmount)
internal returns (uint256){
        uint256 setrealFee = totalBuyFee;
        uint256 setbuytreasuryFee = buytreasuryFee;
        uint256 setselltreasuryFee = selltreasuryFee;
        if(automatedMarketMakerPairs[recipient]) setrealFee = totalSellFee;
        uint256 feeAmount = gonAmount.mul(setrealFee).div(feeDenominator);
        // referrals
        if (automatedMarketMakerPairs[sender]) {
            address UplineAddressBuyer = getUpline(recipient);
            if (UplineAddressBuyer != address(0)){
                setbuytreasuryFee -= totalReferralFee;
                uint256 _uplineBuyerReward =
gonAmount.mul(referrer).div(feeDenominator);
                feeAmount = gonAmount.mul(setrealFee -
referee).div(feeDenominator);
                _gonBalances[UplineAddressBuyer] =
_gonBalances[UplineAddressBuyer].add(_uplineBuyerReward);
                addReferralFee(UplineAddressBuyer,
_uplineBuyerReward.div(_gonsPerFragment) );
            }
        }
```

The first `require` statement is redundant.

```
function setBuyFees(uint256 setbuyliquidityFee, uint256 setbuyyieldFee,
uint256 setbuytreasuryFee, uint256 setbuyfirepitFee) external onlyOwner {
        require(
            setbuyliquidityFee <= MAX_FEE_RATE && // $NUT Liquidity Pool
Growth
            setbuyfirepitFee <= MAX_FEE_RATE && // $NUT Liquidity Pool Growth
            setbuyyieldFee <= MAX_FEE_RATE && // Yield Treasury Fee
            setbuytreasuryFee <= MAX_FEE_RATE, // $NUT Funding Marketing &
Development
            "wrong"
        );
        buyliquidityFee = setbuyliquidityFee;
        buyyieldFee = setbuyyieldFee;
        buytreasuryFee = setbuytreasuryFee;
        buyfirepitfee = setbuyfirepitFee;
        totalBuyFee =
buyliquidityFee.add(buytreasuryFee).add(buyyieldFee).add(buyfirepitfee);
        require(totalBuyFee <= feeDenominator / 4);
        emit SetBuyFees(buyliquidityFee, buyyieldFee, buytreasuryFee,
buyfirepitfee, totalBuyFee);
    }
```

Recommendation

Rewrite some code segments so the runtime will be more performant.

L02 - State Variables could be Declared Constant

Criticality	minor
Location	contract.sol#L324

Description

Constant state variables should be declared constant to save gas.

```
feeDenominator
```

Recommendation

Add the constant attribute to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	minor
Location	contract.sol#L178,404,419,865,333,860,331,332

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.

```
deadWalletAddress
busdToken
setSwapBackSettings_liquifyAll
zeroWalletAddress
_nutfirepit
gonSwapThreshold
user
WETH
```

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
Location	contract.sol#L675

Description

Performing divisions before multiplications may cause lose of prediction.

```
contractTokenBalance = _gonBalances[address(this)].div(_gonsPerFragment)
```

Recommendation

The multiplications should be prior to the divisions.

Contract Functions

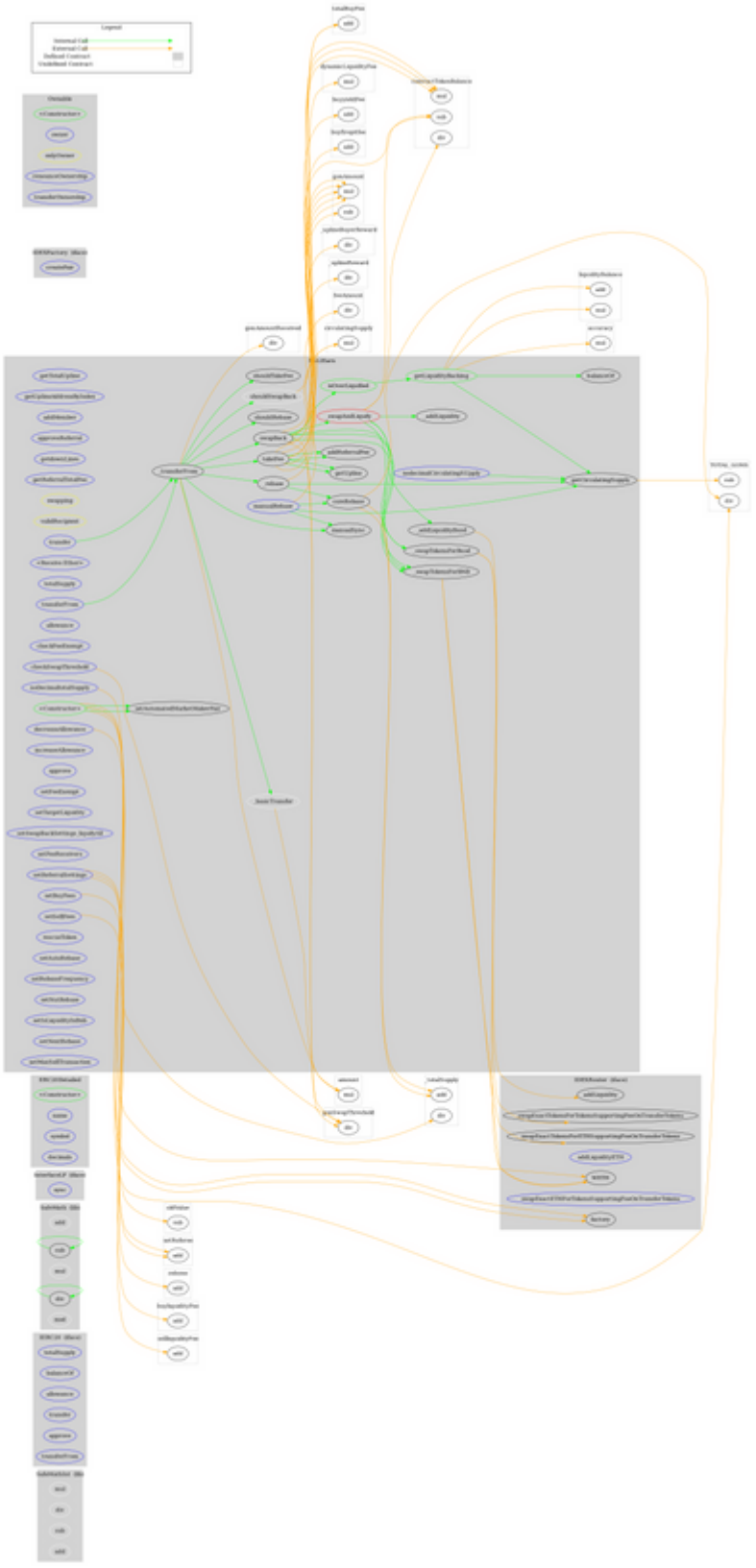
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
InterfaceLP	Interface			
	sync	External	✓	-
ERC20Detailed	Implementation	IERC20		
	<Constructor>	Public	✓	-

	name	External		-
	symbol	External		-
	decimals	External		-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IDEXFactory	Interface			
	createPair	External	✓	-
Ownable	Implementation			
	<Constructor>	Public	✓	-
	owner	External		-
	renounceOwnership	External	✓	onlyOwner
	transferOwnership	External	✓	onlyOwner
Nut2Earn	Implementation	ERC20Detailed, Ownable		
	getTotalUpline	External		-
	getUplineAddressByIndex	External		-
	addMember	External	✓	onlyOwner
	approveReferral	External	✓	-
	getUpline	Public		-
	getdownLines	External		-
	addReferralFee	Public	✓	-
	getReferralTotalFee	External		-
	<Constructor>	Public	✓	ERC20Detaile

				d
	<Receive Ether>	External	Payable	-
	totalSupply	External		-
	noDecimaltotalSupply	External		-
	nodecimalCirculatingSUpply	External		-
	allowance	External		-
	balanceOf	Public		-
	checkFeeExempt	External		-
	checkSwapThreshold	External		-
	shouldRebase	Internal		
	shouldTakeFee	Internal		
	shouldSwapBack	Internal		
	getCirculatingSupply	Public		-
	getLiquidityBacking	Public		-
	isOverLiquified	Public		-
	manualSync	Public	✓	-
	transfer	External	✓	validRecipient
	_basicTransfer	Internal	✓	
	_transferFrom	Internal	✓	
	transferFrom	External	✓	validRecipient
	_swapAndLiquify	Private	✓	
	_addLiquidity	Private	✓	
	_addLiquidityBusd	Private	✓	
	_swapTokensForBNB	Private	✓	
	_swapTokensForBusd	Private	✓	
	swapBack	Internal	✓	swapping
	takeFee	Internal	✓	
	decreaseAllowance	External	✓	-
	increaseAllowance	External	✓	-
	approve	External	✓	-
	_rebase	Private	✓	
	coreRebase	Private	✓	
	manualRebase	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	setFeeExempt	External	✓	onlyOwner

	setTargetLiquidity	External	✓	onlyOwner
	setSwapBackSettings_liquifyAll	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	setReferralSettings	External	✓	onlyOwner
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	rescueToken	External	✓	onlyOwner
	setAutoRebase	External	✓	onlyOwner
	setRebaseFrequency	External	✓	onlyOwner
	setNutRebase	External	✓	onlyOwner
	setIsLiquidityInBnb	External	✓	onlyOwner
	setNextRebase	External	✓	onlyOwner
	setMaxSellTransaction	External	✓	onlyOwner

Contract Flow



Summary

Nut2Earn is an interesting project that has a friendly and growing community. The Smart Contract uses a total supply manipulation business model to provide value for the holders.

There are some functions that can be abused by the owner like stopping transactions and transferring tokens to the team's wallet. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions.

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

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