Smart contract security audit report

NONEAGE

NUT smart contract security audit report

Audit Team : Noneage security team

Audit Date : May 13 , 2021

NUT Smart Contract Security Audit Report

1. Overview

On May 10, 2021, the security team of Noneage Technology received the security audit request of the **NUT project**. The team completed the **NUT smart contract** security audit on May 13. the security audit experts of Noneage Technology communicate with the relevant interface people of the NUT project, maintain information symmetry, conduct security audits under controllable operational risks, and try to avoid project generation and operation during the test process. Cause risks.

Through communicat and feedback with NUT project party, it is confirmed that the loopholes and risks found in the audit process have been repaired or within the acceptable range. The result of this NUT smart contract security audit: **passed.**

Audit Report MD5： F6286920F3F0ADBB7F2650657A3273B6

2. Background

2.1 Project Description

> Project name: NUT
>
> Contract type: DeFi Token contract
>
> Code language: Solidity
>
> Project official Github: https://github.com/nutmoneydefi/launching/tree/main/contracts
>
> Contract documents: NutToken.sol，StakingRewards.sol，UniswapV2Factory.sol，UniswapV2Router02.sol

2.2 Audit Range

The contract file and corresponding MD5 provided by NUT:

NutToken.sol	A070EE16011C0861377FB3965425B9C3
StakingRewards.sol	71D2BF7C88EAF84131E7ADC908BF3254
UniswapV2Factory.sol	D6CDE918C62CDD7CCC80A60E034B3BC4
UniswapV2Router02.sol	75869E2A1631FA8963C09C56BC39E2BD

2.3 Security Audit List

The security experts of Noneage Technology conduct security audits on the security audit list within the agreement, The scope of this smart contract security audit does not include new attack methods that may appear in the future, does not include the code after contract upgrades or tampering, and is not included in the subsequent cross-country, does not include cross-chain deployment, does not include project front-end code security and project platform server security.

This smart contract security audit list includes the following:

- Integer overflow
- Reentry attack
- Floating point numbers and numerical precision
- Default visibility
- Tx.origin authentication
- Wrong constructor
- Return value not verified
- Insecure random numbers
- Timestamp dependency
- Transaction order is dependent
- Delegatecall
- Call
- Denial of service
- Logic design flaws
- Fake recharge vulnerability
- Short address attack
- Uninitialized storage pointer
- Additional token issuance
- Frozen account bypass
- Access control
- Gas usage

3. Contract Structure Analysis

3.1 Directory Structure

```
|
└─NUT
  │ NutToken.sol
  │ StakingRewards.sol
  │ UniswapV2Factory.sol
  │ UniswapV2Router02.sol
```

3.2 NUT contract

Contract

NutToken

- mint(address _to, uint256 _amount)
- delegates(address delegator)
- delegate(address delegatee)

- getCurrentVotes(address account)
- getPriorVotes(address account, uint blockNumber)
- _delegate(address delegator, address delegatee)
- _moveDelegates(address srcRep, address dstRep, uint256 amount)
- safe32(uint n, string memory errorMessage)
- getChainId()

StakingRewards

- totalSupply()
- balanceOf(address account)
- lastTimeRewardApplicable()
- rewardPerToken()
- earned(address account)
- getRewardForDuration()
- stakeWithPermit(uint256 amount, uint deadline, uint8 v, bytes32 r, bytes32 s)
- stake(uint256 amount)
- withdraw(uint256 amount)
- getReward()
- exit()
- notifyRewardAmount(uint256 reward)

UniswapV2ERC20

- _mint(address to, uint value)
- _burn(address from, uint value)
- _approve(address owner, address spender, uint value)
- _transfer(address from, address to, uint value)
- approve(address spender, uint value)
- transfer(address to, uint value)
- transferFrom(address from, address to, uint value)
- permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s)

UniswapV2Pair

- function getReserves()
- function _safeTransfer(address token, address to, uint value)
- function initialize(address _token0, address _token1)
- function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1)
- function _mintFee(uint112 _reserve0, uint112 _reserve1)
- function mint(address to)
- function burn(address to)
- function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data)
- function skim(address to)
- function sync()

UniswapV2Factory

- function allPairsLength()
- function createPair(address tokenA, address tokenB)
- function setFeeTo(address _feeTo)
- function setFeeToSetter(address _feeToSetter)

UniswapV2Router02

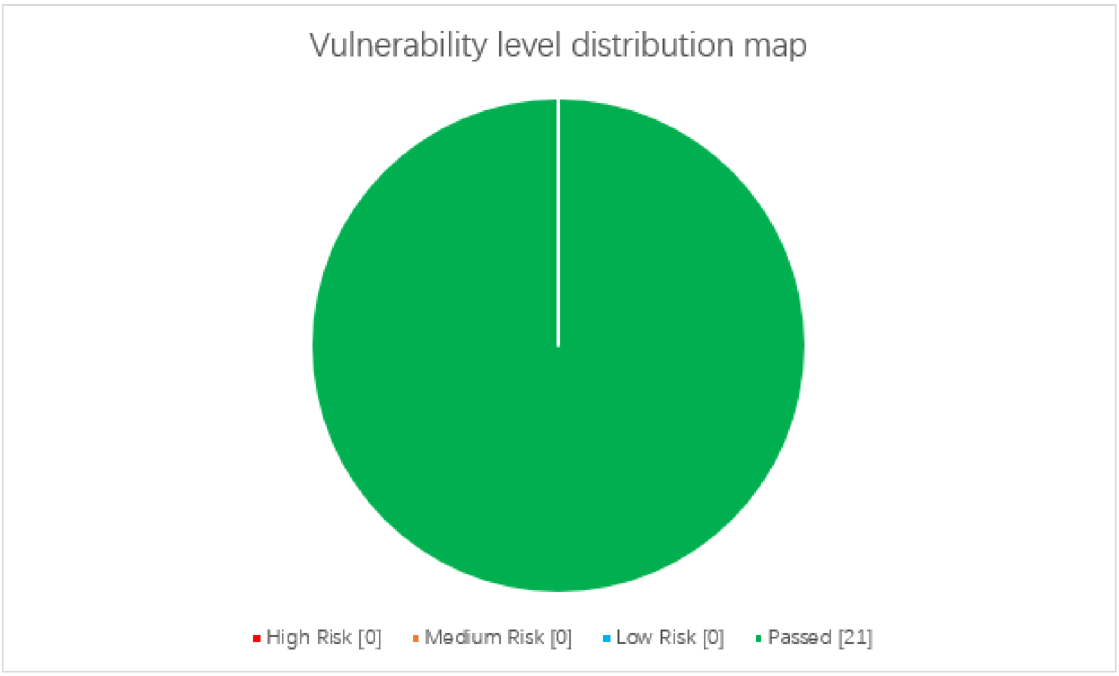
- _addLiquidity(address tokenA,address tokenB,uint amountADesired,uint amountBDesired,uint amountAMin,uint amountBMin)
- addLiquidity(address tokenA,address tokenB,uint amountADesired,uint amountBDesired,uint amountAMin,uint amountBMin,address to,uint deadline)
- addLiquidityETH(address token,uint256 amountTokenDesired,uint256 amountTokenMin,uint256 amountETHMin,address to,uint256 deadline)
- removeLiquidity(address tokenA,address tokenB,uint256 liquidity,uint256 amountAMin,uint256 amountBMin,address to,uint256 deadline)
- removeLiquidityETH(address token,uint256 liquidity,uint256 amountTokenMin,uint256 amountETHMin,address to,uint256 deadline)
- removeLiquidityWithPermit(address tokenA,address tokenB,uint256 liquidity,uint256 amountAMin,uint256 amountBMin,address to,uint256 deadline,bool approveMax,uint8 v,bytes32 r,bytes32 s)
- removeLiquidityETHWithPermit(address token,uint256 liquidity,uint256 amountTokenMin,uint256 amountETHMin,address to,uint256 deadline,bool approveMax,uint8 v,bytes32 r,bytes32 s)
- removeLiquidityETHSupportingFeeOnTransferTokens(address token,uint liquidity,uint amountTokenMin,uint amountETHMin,address to,uint deadline)
- removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(address token,uint liquidity,uint amountTokenMin,uint amountETHMin,address to,uint deadline,bool approveMax, uint8 v, bytes32 r, bytes32 s)
- _swap(uint[] memory amounts, address[] memory path, address _to)
- swapExactTokensForTokens(uint256 amountIn,uint256 amountOutMin,address[] calldata path,address to,uint256 deadline)
- swapTokensForExactTokens(uint256 amountOut,uint256 amountInMax,address[] calldata path,address to,uint256 deadline)
- swapExactETHForTokens(uint256 amountOutMin,address[] calldata path,address to,uint256 deadline)
- swapTokensForExactETH(uint256 amountOut,uint256 amountInMax,address[] calldata path,address to,uint256 deadline)
- swapExactTokensForETH(uint256 amountIn,uint256 amountOutMin,address[] calldata path,address to,uint256 deadline)
- swapETHForExactTokens(uint256 amountOut,address[] calldata path,address to,uint256 deadline)
- _swapSupportingFeeOnTransferTokens(address[] memory path, address _to)
- swapExactTokensForTokensSupportingFeeOnTransferTokens(uint amountIn,uint amountOutMin,address[] calldata path,address to,uint deadline)
- swapExactETHForTokensSupportingFeeOnTransferTokens(uint amountOutMin,address[] calldata path,address to,uint deadline)
- swapExactTokensForETHSupportingFeeOnTransferTokens(uint amountIn,uint amountOutMin,address[] calldata path,address to,uint deadline)
- quote(uint256 amountA,uint256 reserveA,uint256 reserveB)
- getAmountOut(uint256 amountIn,uint256 reserveIn,uint256 reserveOut)
- getAmountIn(uint256 amountOut,uint256 reserveIn,uint256 reserveOut)
- getAmountsOut(uint256 amountIn, address[] calldata path)
- getAmountsIn(uint256 amountOut, address[] calldata path)

4. Audit Details

4.1 Vulnerabilities Distribution

Vulnerabilities in this security audit are distributed by risk level, as follows:

Vulnerability level distribution			
High risk	Medium risk	Low risk	Passed
0	0	0	21



This smart contract security audit has 0 high-risk vulnerabilities, 0 medium-risk vulnerabilities, 0 low-risk vulnerabilities, and 21 passed, with a high security level.

4.2 Vulnerabilities Details

A security audit was conducted on the smart contract within the agreement, and no security vulnerabilities that could be directly exploited and generated security problems were found, and the security audit was passed.

4.3 Other Risks

Other risks refer to the code that contract security auditors consider to be risky, which may affect the stability of the project under certain circumstances, but cannot constitute a security issue that directly endangers the security.

4.3.1 Judgement of transfer amount

- **Question detail**

The main function of the StakingRewards contract withdraw() method is to perform withdrawal operations. In this method, only whether the withdrawal value is greater than zero is judged, and it is not judged whether the total amount of the account and funds is greater than the withdrawal limit.

```
    function withdraw(uint256 amount) public nonReentrant
updateReward(msg.sender) {
        require(amount > 0, "Cannot withdraw 0");
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        stakingToken.safeTransfer(msg.sender, amount);
        emit Withdrawn(msg.sender, amount);
    }
```

- **Safety advice**

It is recommended to add the function of judging the amount of account funds in this method.

- **Update status**

Through communication with the NUT team, safety recommendations have been adopted.

4.3.2 Redundant code

- **Question detail**

The UniswapV2Factory contract defines the INIT_CODE_PAIR_HASH variable, but this variable is not used in the project contract code.

```
bytes32 public constant INIT_CODE_PAIR_HASH =
keccak256(abi.encodePacked(type(UniswapV2Pair).creationCode));
```

- **Safety advice**

It is recommended to remove the INIT_CODE_PAIR_HASH variable definition.

- **Update status**

Through communication with the NUT team, safety recommendations have been adopted.

5. Security Audit Tool

Tool name	Tool Features
Oyente	Can be used to detect common bugs in smart contracts
securify	Common types of smart contracts that can be verified
MAIAN	Multiple smart contract vulnerabilities can be found and classified
Noneage Internal Toolkit	Noneage(hawkeye system) self-developed toolkit + https://audit.noneage.com

6. Vulnerability assessment criteria

Vulnerability level	Vulnerability description
High risk	Vulnerabilities that can directly lead to the loss of contracts or users' digital assets, such as integer overflow vulnerabilities, false recharge vulnerabilities, re-entry vulnerabilities, illegal token issuance, etc. Vulnerabilities that can directly cause the ownership change of the token contract or verification bypass, such as: permission verification bypass, call code injection, variable coverage, unverified return value, etc. Vulnerabilities that can directly cause the token to work normally, such as denial of service vulnerabilities, insecure random numbers, etc.
Medium risk	Vulnerabilities that require certain conditions to trigger, such as vulnerabilities triggered by the token owner's high authority, and transaction sequence dependent vulnerabilities. Vulnerabilities that cannot directly cause asset loss, such as function default visibility errors, logic design flaws, etc.
Low risk	Vulnerabilities that are difficult to trigger, or vulnerabilities that cannot lead to asset loss, such as vulnerabilities that need to be triggered at a cost higher than the benefit of the attack, cannot lead to incorrect coding of security vulnerabilities.

NONEAGE

Telephone	:	86-17391945345 18511993344
Email	:	support@noneage.com
Site	:	www.noneage.com
Weibo	:	weibo.com/noneage