Smart contract security audit report

Cherryswap smart contract security audit report

Audit Team：Noneage security team

Audit Date：March 21, 2021

Cherryswap Smart Contract Security Audit Report

1. Overview

On March 13, 2021, the security team of LS Technology received the security audit request of the **Cherryswap project**. The team will conduct a report on the **Cherryswap smart contract** from March 15, 2021 to March 21, 2021. During the audit process, the security audit experts of Zero Hour Technology communicate with the relevant interface people of the Cherryswap 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 Cherryswap 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 Cherryswap smart contract security audit: **passed.**

Audit Report MD5：9553FF2929514DC947706BBE721B041B

2. Background

2.1 Project Description

> Project name: Cherryswap
>
> official website: https://www.cherryswap.net/
>
> Contract type: DeFi Token contract
>
> Code language: Solidity
>
> Official GitHub repository address: https://github.com/cherryswapnet/
>
> Contract documents: CherryToken.sol，MasterChef.sol，SmartChef.sol，SyrupBar.sol，CherryERC20.sol，CherryFactory.sol，CherryPair.sol，CherryMigrator.sol，CherryRouter.sol，CherryRouter01.sol

2.2 Audit Range

Cherryswap officially provides contract and MD5：

CherryToken.sol	6C17EA78785083E8DD596E76A96C305C
MasterChef.sol	2250B0CBDBF7398DAD2B0E91B3E389F4
SmartChef.sol	1258A607835B667811DE211FC9C3EA59
SyrupBar.sol	9EB0C60BAFB31FAE2AAB941446BC0E5A
CherryERC20.sol	6B497701DB11C5D36A9634CF6D0B9C29

CherryFactory.sol	B7452F76A1960BDB3DBE6323D3CDAE14
CherryPair.sol	FA8A3309A7BB36742DEC4714E867E982
CherryMigrator.sol	E0A4E2E8C39B1BBF229689F123143664
CherryRouter.sol	7B24CB0694FA99B6E5232D45F810F512
CherryRouter01.sol	ADC124088EAB7E125C450F5D15636174

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

```
└─Cherryswap
  ├─swap-core-main
  │  └─contracts
  │        CherryERC20.sol
  │        CherryFactory.sol
```

```
│       CherryPair.sol
│
├─swap-farm-main
│  └─contracts
│       CherryToken.sol
│       MasterChef.sol
│       SmartChef.sol
│       SyrupBar.sol
│
└─swap-periphery-main
   └─contracts
      CherryMigrator.sol
      CherryRouter.sol
      CherryRouter01.sol
```

3.2 CORE branch

Contract

CherryERC20

- _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)

CherryFactory

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

CherryPair

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

3.3 FARM branch

Contract

CherryToken

- delegates(address delegator)
- delegate(address delegatee)
- elegateBySig(address delegatee,uint nonce,uint expiry,uint8 v,bytes32 r,bytes32 s)
- getCurrentVotes(address account)
- getPriorVotes(address account, uint blockNumber)
- _delegate(address delegator, address delegatee)
- _moveDelegates(address srcRep, address dstRep, uint256 amount)
- _writeCheckpoint(address delegatee,uint32 nCheckpoints,uint256 oldVotes,uint256 newVotes)
- safe32(uint n, string memory errorMessage)
- getChainId()

MasterChef

- updateMultiplier(uint256 multiplierNumber)
- poolLength()
- add(uint256 _allocPoint, IBEP20 _lpToken, bool _withUpdate)
- set(uint256 _pid, uint256 _allocPoint, bool _withUpdate)
- updateStakingPool()
- setMigrator(IMigratorChef _migrator)
- migrate(uint256 _pid)
- getMultiplier(uint256 _from, uint256 _to)
- pendingCherry(uint256 _pid, address _user)
- massUpdatePools()
- updatePool(uint256 _pid)
- deposit(uint256 _pid, uint256 _amount)
- withdraw(uint256 _pid, uint256 _amount)
- enterStaking(uint256 _amount)
- leaveStaking(uint256 _amount)
- emergencyWithdraw(uint256 _pid)
- safeCherryTransfer(address _to, uint256 _amount)
- dev(address _devaddr)

SmartChef

- stopReward()
- getMultiplier(uint256 _from, uint256 _to)
- updateMultiplier(uint256 multiplierNumber)
- pendingReward(address _user)
- updatePool(uint256 _pid)
- massUpdatePools()
- deposit(uint256 _amount)
- withdraw(uint256 _amount)
- emergencyWithdraw()
- emergencyRewardWithdraw(uint256 _amount)

SyrupBar

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

- elegateBySig(address delegatee,uint nonce,uint expiry,uint8 v,bytes32 r,bytes32 s)
- getCurrentVotes(address account)
- getPriorVotes(address account, uint blockNumber)
- _delegate(address delegator, address delegatee)
- _moveDelegates(address srcRep, address dstRep, uint256 amount)
- _writeCheckpoint(address delegatee,uint32 nCheckpoints,uint256 oldVotes,uint256 newVotes)
- safe32(uint n, string memory errorMessage)
- getChainId()

3.4 PERIPHERY branch

Contract

CherryMigrator

- migrate(address token, uint amountTokenMin, uint amountETHMin, address to, uint deadline)

CherryRouter

- _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)

CherryRouter01

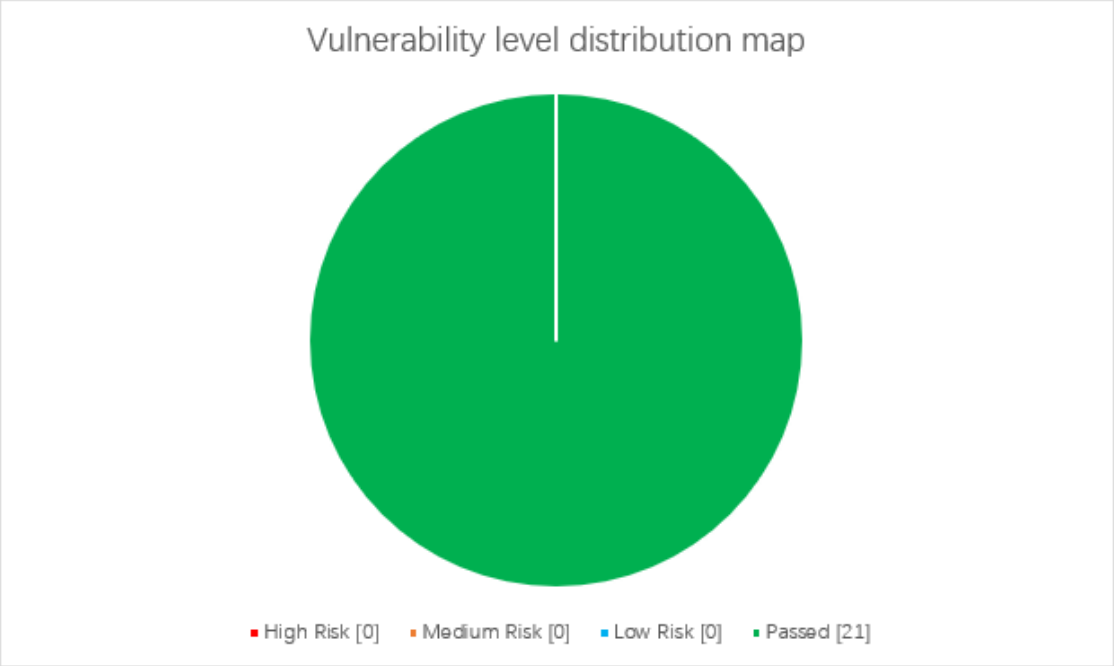
- _addLiquidity(address tokenA,address tokenB,uint amountADesired,uint amountBDesired,uint amountAMin,uint amountBMin)
- addLiquidity(address tokenA,address tokenB,uint amountADesired,uint amountBDesired,uint amountAMin,uint amountBMin,address to,uint deadline)
- addLiquidityETH(address token,uint256 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)
- _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)
- 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 Administrator rights security

- **Question detail**

By auditing the FARM branch contract, it is found that both the CherryToken contract and the SyrupBar contract have mint() issuance function and burn() destruction function, and there are multiple administrator operations in the FARM branch contract. When the administrator key is lost or controlled by malicious personnel Malicious operations will affect the normal operation of the project, as shown in the following figure:

```
function mint(address _to, uint256 _amount) public onlyOwner {
    _mint(_to, _amount);
    _moveDelegates(address(0), _delegates[_to], _amount);
}

function burn(address _from ,uint256 _amount) public onlyOwner {
    _burn(_from, _amount);
    _moveDelegates(_delegates[_from], address(0), _amount);
}
```

- **Safety advice**

It is necessary to store the private key of the deployer's address safely and effectively to avoid loss or acquisition by malicious persons; transfer the administrator authority to Timelock.

- **Update status**

Through communication with the Cherryswap team, the above-mentioned administrator functions are normal operations, and the owner's private key will be properly managed.

4.3.2 The owner of the contract should be on Timelock

- **Question detail**

By auditing the FARM branch, it is found that there is an administrator's emergency transfer function in the SmartChef contract code, as shown in the following code:

```
function emergencyRewardWithdraw(uint256 _amount) public onlyOwner {
    require(_amount < rewardToken.balanceOf(address(this)), 'not enough token');
    rewardToken.safeTransfer(address(msg.sender), _amount);
}
```

- **Safety advice**

It is recommended that the owner of the contract should be added to the Timelock contract.

- **Update status**

Through communication with the Cherryswap team, the team has added the transferOwnership() function interface and verified the validity of the address.

4.3.3 Numerical calculation defects

- **Question detail**

By auditing the CORE branch, it is found that in the CherryPair contract code, the result calculated by 1000**3 in the require judgment statement, this line of code can never be satisfied, as shown in the following code:

```
require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**3), 'Cherry: K')
```

- **Safety advice**

It is recommended to modify the calculated value in the require judgment statement to 1000**2, as shown in the following code:

```
require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'Cherry: K')
```

- **Update status**

Through communication with the Cherryswap team, the team has revised the above calculated value to 1000**2.

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.

Disclaimer:

Noneage Technology only issues a report and assumes corresponding responsibilities for the facts that occurred or existed before the issuance of this report, Since the facts that occurred after the issuance of the report cannot determine the security status of the smart contract, it is not responsible for this.
Noneage Technology conducts security audits on the security audit items in the project agreement, and is not responsible for the project background and other circumstances, The subsequent on-chain deployment and operation methods of the project party are beyond the scope of this audit.
This report only conducts a security audit based on the information provided by the information provider to Noneage at the time the report is issued, If the information of this project is concealed or the situation reflected is inconsistent with the actual situation, Noneage Technology shall not be liable for any losses and adverse effects caused thereby.

NONEAGE

Telephone：86-17391945345 18511993344

Email ：support@noneage.com

Site ：www.noneage.com

Weibo ：weibo.com/noneage