LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For OortSwap

15 March 2022

lunaray.co

LUNARAY BLOCKCHAINSECURITY

Table of Contents

1. Overview

On Mar 5, 2022, the security team of Lunaray Technology received the security audit request of the **Oortswap project**. The team completed the audit of the **Oortswap smart contract** on Mar 15, 2022. During the audit process, the security audit experts of Lunaray Technology and the Oortswap project interface Personnel communicate and maintain symmetry of information, conduct security audits under controllable operational risks, and avoid risks to project generation and operations during the testing process.

Through communicat and feedback with Oortswap 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 Oortswap smart contract security audit: **Passed**

Audit Report Hash:

5B4C2CFD569AEB90A4D9C1A3186F97770C2AA8D88173EAB16C748C17C1431005

2. Background

2.1 Project Description

Project name	Oortswap
Contract type	Token , DeFi
Code language	Solidity
Public chain	REI NETWORK
Project address	https://oortswap.org/
Contract file	MasterChef.sol,OortswapERC20.sol,OortswapFactory.sol, OortswapPair.sol, OortswapRouter.sol,OortToken.sol, SwapMining.sol, TeamTimeLock.sol
Project Description	Oortswap is an Automated Market Maker (AMM), and the Exchange is at the heart of Oortswap , Oortswap is also a AMM-based decentralized margin trading platform on multi-chains, where users can easily earn interest through lending and perform leveraged trading.

2.2 Audit Range

The smart contract file provided by Oortswap and the corresponding MD5：

Name	address
MasterChef.sol	3D4924EA90118EB2E4517FFE2123335E
OortswapERC20.sol	802DB63FEDAEBE4807E12E46F86A2B3E
OortswapFactory.sol	F0F862F280F8597788266E2D5FBB4EBE
OortswapPair.sol	BBD7B359A63649C1368D57A673E66A21
OortswapRouter.sol	B6BD4031A9E475549AE55EB2710A6E96
OortToken.sol	B8371E395FD496C2842F685A06E41A27
SwapMining.sol	648DF1F3BCD9549CEDE3477976784B6B
TeamTimeLock.sol	13B58D6DE10D6DAB75AE87047192127E

3. Project contract details

3.1 Contract Overview

MasterChef Contract

Farms lets users that are providing liquidity earn OORT rewards by locking their LP tokens into a smart contract. The incentive is to balance out the risk of impermanent loss that comes along with locking in your liquidity.

OortswapERC20 Contract

Mainly implements ERC20 token functionality and adds support for authorizing offline signed messages.

OortswapFactory Contract

Core logic, the main implementation of the creation of trading pairs, development team fees charged switch.

OortswapPair Contract

Provide automated market making and track the balance of tokens in the trading pool, exposing relevant data to external parties as a decentralized price prognosticator.

OortRouter Contract

Mainly liquidity supply-related functions and transaction-to-asset exchange-related functions.

OortToken Contract

Mainly implement multiple contracts to call the mint method to mint coins and have a limit on funds.

SwapMining Contract

The main implementation of SwapMining, through the OortRswapouter contract to call swap function for mining.

TeamTimeLock Contract

The main implementation of the team's restricted funds are locked and cannot be withdrawn for a specific period of time.

3.2 Contract details

SwapMining Contract

Name	Parameter	Attributes
token0	none	external
token1	none	external
feeTo	none	external
feeToSetter	none	external
getPair	address tokenA address tokenB	external
allPairsLength	none	external
createPair	address tokenA address tokenB	external
pairFor	address tokenA address tokenB	external
getMarginPoolStaking	none	external
getMarginPool	none	external
getPriceOracle	none	external
mint	address _addr uint256 _amount	external
transfer	address recipient uint256 amount	external
balanceOf	address account	external
poolLength	none	public
addPair	uint256 _allocPoint address _pair bool _withUpdate	onlyOwner
setPair	uint256 _pid uint256 _allocPoint bool _withUpdate	onlyOwner
setoortPerBlock	uint256 _newPerBlock	onlyOwner

setHalvingPeriod	uint256 _block	onlyOwner
setRouter	address newRouter	onlyOwner
phase	none	public
reward	none	public
getExtraReward	uint256 _from uint256 _to uint256 _lastRewardBlock	internal
getOortReward	uint256 _lastRewardBlock	public
massMintPools	none	public
swap	address account address input address output uint256 amount	onlyRouter
takerWithdraw	address _addr	onlyStaking
updatePool	uint256 _pid	public
getUserReward	uint256 _pid address _user	external
safeCakeTransfer	address _to uint256 _amount	internal

MultiSigPeriodicTimeLock Contract

Name	Parameter	Attributes
setTokenAddr	IERC20 _token	signatoryOnly
lock	none	signatoryOnly
approve	Action action	signatoryOnly
setBeneficiary	address addr	signatoryOnly
release	none	signatoryOnly

OortRouter Contract

Name	Parameter	Attributes
deposit	none	external
transfer	address to uint value	external
swap	address account address input address output uint256 amount	external
setSwapMining	address _swapMininng	onlyOwner
_addLiquidity	address tokenA address tokenB uint amountADesired uint amountBDesired uint amountAMin uint amountBMin	internal
addLiquidity	address tokenA address tokenB uint amountADesired uint amountBDesired uint amountAMin uint amountBMin address to uint deadline	external
addLiquidityETH	address token uint amountTokenDesired uint amountTokenMin uint amountETHMin address to uint deadline	external
removeLiquidity	address tokenA address tokenB uint liquidity uint amountAMin uint amountBMin address to uint deadline	public
removeLiquidityETH	address token uint liquidity uint amountTokenMin uint amountETHMin address to uint deadline	public

removeLiquidityWithPermit	address tokenA address tokenB uint liquidity uint amountAMin uint amountBMin address to uint deadline bool approveMax uint8 v bytes32 r bytes32 s	external
removeLiquidityETHWith Permit	address token uint liquidity uint amountTokenMin uint amountETHMin address to uint deadline bool approveMax uint8 v bytes32 r bytes32 s	external
removeLiquidityETH SupportingFeeOnTransfer Tokens	address token uint liquidity uint amountTokenMin uint amountETHMin address to uint deadline	public
removeLiquidityETHWith PermitSupportingFeeOn TransferTokens	address token uint liquidity uint amountTokenMin uint amountETHMin address to uint deadline bool approveMax uint8 v bytes32 r bytes32 s	external
quote	uint amountA uint reserveA uint reserveB	public
getAmountOut	uint amountIn uint reserveIn uint reserveOut	public
getAmountIn	uint amountOut uint reserveIn uint reserveOut	public

OortToken Contract

Name	Parameter	Attributes
_mint	address account uint256 amount	internal
_transfer	address sender address recipient uint256 amount	internal
delegate	address delegatee	external
delegateBySig	address delegatee uint nonce uint expiry uint8 v bytes32 r bytes32 s	external
getCurrentVotes	address account	external
getPriorVotes	address account uint blockNumber	external
_delegate	address delegator address delegatee	internal
_moveDelegates	address srcRep address dstRep uint256 amount	internal
_writeCheckpoint	address delegatee uint32 nCheckpoints uint256 oldVotes uint256 newVotes	internal
safe32	uint n string errorMessage	internal
getChainId	none	internal
approve	Action action	signatoryOnly
modifySignatories	address _oldAddress address _newAddress	onlyOwner
addSignatories	address _newAddress	onlyOwner
setChefAddr	address _chef	onlyOwner
setSwapAddr	address _swapPool	onlyOwner
setlendAddr	address _lendPool	onlyOwner
ChefMint	uint256 _amount	onlyChef

SwapMint	uint256 _amount	onlySwapPool
lendMint	uint256 _amount	onlyLendPool
AirDropMint	address _dao	signatoryOnly
MarketMint	address _dao uint256 _amount	signatoryOnly
OtherMint	address _dao uint256 _amount	signatoryOnly

OortswapERC20 Contract

Name	Parameter	Attributes
_mint	address to uint value	internal
_burn	address from uint value	internal
_approve	address owner address spender uint value	private
_transfer	address from address to uint value	private
approve	address spender uint value	external
transfer	address to uint value	external
transferFrom	address from address to uint value	external
permit	address owner address spender uint value uint deadline uint8 v bytes32 r bytes32 s	external

MasterChef Contract

Name	Parameter	Attributes
migrate	IERC20 token	external
setHalvingPeriod	uint256 _block	onlyOwner
phase	none	public
reward	none	public
getExtraReward	uint256 _from uint256 _to uint256 _lastRewardBlock	internal
getOortReward	uint256 _lastRewardBlock	public
poolLength	none	external
add	uint256 _allocPoint address _lpToken bool _withUpdate	onlyOwner
set	uint256 _pid uint256 _allocPoint bool _withUpdate	onlyOwner
setMigrator	IMigratorChef _migrator	onlyOwner
migrate	uint256 _pid	public
pendingOort	uint256 _pid address _user	external
massUpdatePools	none	public
updatePool	uint256 _pid	public
deposit	uint256 _pid uint256 _amount	public
withdraw	uint256 _pid uint256 _amount	public
emergencyWithdraw	uint256 _pid	public
safeOortTransfer	address _to uint256 _amount	internal

OortswapPair Contract

Name	Parameter	Attributes
getReserves	none	public
_safeTransfer	address token address to uint value	private
initialize	address _token0 address _token1	external
_update	uint balance0 uint balance1 uint112 _reserve0 uint112 _reserve1	private
_mintFee	uint112 _reserve0 uint112 _reserve1	private
mint	address to	external
burn	address to	external
swap	uint amount0Out uint amount1Out address to bytes data	external
skim	address to	external
sync	none	external

OortswapFactory Contract

Name	Parameter	Attributes
allPairsLength	none	external
createPair	address tokenA address tokenB	external
setFeeTo	address _feeTo	external
setFeeToSetter	address _feeToSetter	external

4. Audit details

4.1 Findings Summary

Severity	Found	Resolved	Acknowledged
● High	0	0	0
● Medium	2	2	0
● Low	3	3	0
● Info	2	2	0

4.2 Risk distribution

Name	Risk level	Repair status
Administrator Permissions	Low	Resolved
No events added	Info	Resolved
K value calibration problem	Medium	Resolved
Users can transfer excess tokens	Info	Resolved
LP Token Transfer Risk	Low	Resolved
The swap method of lightning loan risk	Medium	Resolved
Accuracy of onlyStaking modifier	Low	Resolved
Variables are updated	No	normal
Floating Point and Numeric Precision	No	normal
Default visibility	No	normal
tx.origin authentication	No	normal
Faulty constructor	No	normal
Unverified return value	No	normal
Insecure random numbers	No	normal
Timestamp Dependent	No	normal
Transaction order dependency	No	normal
Delegatecall	No	normal
Call	No	normal
Denial of Service	No	normal
Logical Design Flaw	No	normal
Fake recharge vulnerability	No	normal

Short address attack Vulnerability	No	normal
Uninitialized storage pointer	No	normal
Frozen account bypass	No	normal
Uninitialized	No	normal
Reentry attack	No	normal
Integer Overflow	No	normal

4.3 Risk audit details

4.3.1. Administrator Permissions

- **Risk description**

SwapMining, OortToken, OortRouter, MasterChef contracts all have administrator privileges, the administrator can set and transfer sensitive operations, if the private key is lost and controlled by malicious people, it may lead to abnormal flow of funds and shake the stability of the market, part of the code is as follows.

```
function setMigrator(IMigratorChef _migrator) public onlyOwner {
    migrator = _migrator;
}

function setSwapMining(address _swapMininng) public onlyOwner {
    swapMining = _swapMininng;
}
```

- **Safety advice**

It is recommended that the administrator address use a multi-signature wallet to manage the private key, and it is recommended that the administrator address use a lock contract to restrict the administrator's sensitive operations.

- **Repair Status**

Oortswap has officially fixed the risk.

4.3.2. No events added

- **Risk description**

There are sensitive operations in SwapMining, OortToken, OortRouter and MasterChef contracts. In order to facilitate users and administrators to understand the operation of the project, it is recommended to add event logging, part of the code is as follows.

```
    function setMigrator(IMigratorChef _migrator) public onlyOwner {
        migrator = _migrator;
    }

    function setSwapMining(address _swapMininng) public onlyOwner {
        swapMining = _swapMininng;
    }

    function migrate(uint256 _pid) public {
        require(address(migrator) != address(0), "migrate: no migrator
");
        PoolInfo storage pool = poolInfo[_pid];
        IERC20 lpToken = IERC20(pool.lpToken);
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        IERC20 newLpToken = migrator.migrate(lpToken);
        require(bal == newLpToken.balanceOf(address(this)), "migrate: b
ad");
        pool.lpToken = address(newLpToken);
    }
```

- **Safety advice**

It is recommended to add event logging for all functions that contain sensitive operations.

- **Repair Status**

Oortswap has officially fixed the risk.

4.3.3. K value calibration problem

- **Risk description**

OortswapPair contract, Swap method, mainly to achieve the function of asset trading in trading pairs, the final constant product K value for verification, but in the current contract code, we modified the original three thousandths of the fee, updated to 25 thousandths, but in the calculation of the verification of the constant product, the multiplier 1000 * * 2 is not modified, which will lead to the constant product calculation in a certain range, losing the value of verification.

```
    // this low-level function should be called from a contract which p
erforms important safety checks
    function swap(uint amount0Out, uint amount1Out, address to, bytes c
alldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'Oortswap: INSUFFICIE
NT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas
 savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'Oort
swap: INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
        address _token0 = token0;
        address _token1 = token1;
        require(to != _token0 && to != _token1, 'Oortswap: INVALID_TO
');
        if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); //
optimistically transfer tokens
        if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); //
optimistically transfer tokens
        if (data.length > 0) IOortswapCallee(to).OortswapCall(msg.sende
r, amount0Out, amount1Out, data);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 -
 (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 -
 (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'Oortswap: INSUFFICIENT
_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep erro
rs
        uint balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(2
5));
        uint balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(2
5));
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve
0).mul(_reserve1).mul(1000**2), 'Oortswap: K');  // Lunaray1:uint(_rese
rve0).mul(_reserve1).mul(10000**2)
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1
Out, to);
    }
```

- **Safety advice**

It is suggested to modify the constant product calculation to 10000**2, for example:

```
require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_
reserve1).mul(10000**2), 'Oortswap: K');
```

- **Repair Status**

Oortswap has officially fixed the risk.

4.3.4. Users can transfer excess tokens

- **Risk description**

OortswapPair contract, skim method is possible for other users to transfer excess funds out.

```
    function skim(address to) external lock {
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(th
is)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(th
is)).sub(reserve1));
    }
```

- **Safety advice**

You can add this fund transfer as a fixed address.

- **Repair Status**

Oortswap has officially fixed the risk.

4.3.5. LP Token Transfer Risk

- **Risk description**

MasterChef contract, when the administrator set migrator address, migrate method is able to any pool LP Token all authorized to migrator, that is, the address can transfer any pool LP Token, if the address or administrator is malicious operation, may lead to the pool funds all lost.

```
    // Set the migrator contract. Can only be called by the owner.
    function setMigrator(IMigratorChef _migrator) public onlyOwner {
        migrator = _migrator;
    }

    // Migrate lp token to another lp contract. Can be called by anyon
e. We trust that migrator contract is good.
    function migrate(uint256 _pid) public {
        require(address(migrator) != address(0), "migrate: no migrator
");
        PoolInfo storage pool = poolInfo[_pid];
        IERC20 lpToken = IERC20(pool.lpToken);
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        IERC20 newLpToken = migrator.migrate(lpToken);
        require(bal == newLpToken.balanceOf(address(this)), "migrate: b
ad");
        pool.lpToken = address(newLpToken);
    }
```

- **Safety advice**

Suggest setting up event logs for the above two methods.It is recommended that this administrative address be added to the time lock contract to avoid causing panic in the community.

- **Repair Status**

Oortswap has officially fixed the risk.

4.3.6. The swap method of lightning loan risk

- **Risk description**

SwapMining contract swap method, the main function is pair pair is for transaction mining, here swap method pass parameters are obtained by the user controllable parameters in the OortswapRouter contract and passed in, swap method, when the pair address is reasonable and other part of the conditions are met, you can pass in here the amount parameter funds to the user for accrual.

To get the reward, you need to call the takerWithdraw method. The final value of the reward obtained by this method is the userSub variable, which is obtained by the operation of user.amount, and when the value of user.amount is larger, the userSub variable increases accordingly.

Back to the beginning, when the user has a large amount at the swap method, more rewards are obtained, so there is a possibility that the lightning credit makes the amount at the swap method large

swap

```
    function swap(address account, address input, address output, uint2
56 amount) public onlyRouter returns (bool) {
        require(account != address(0), "SwapMining: taker swap account 
is the zero address");
        require(input != address(0), "SwapMining: taker swap input is t
he zero address");
        require(output != address(0), "SwapMining: taker swap output is
 the zero address");
        if (poolLength() <= 0) {
            return false;
        }
        address pair = factory.getPair(input, output);
        if(pair ==address(0x00)) {
            return false;
        }
        PoolInfo storage pool = poolInfo[pairOfPid[pair]];
        // If it does not exist or the allocPoint is 0 then return
        if (pool.pair != pair || pool.allocPoint <= 0) {
            return false;
        }
        uint256 _pid = pairOfPid[pair];
        UserInfo storage user = userInfo[_pid][account];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accCakePerShare).div
(1e12).sub(user.rewardDebt);
```

```
        if(pending > 0) {
             user.unclaimedBuni = user.unclaimedBuni.add(pending);}
        }
        if(amount > 0){
            user.amount = user.amount.add(amount);
            pool.amount = pool.amount.add(amount);
            emit Swap(account, _pid, amount);
        }
     user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12);
        return true;
    }
```

takerWithdraw

```
    function takerWithdraw(address _addr) public onlyStaking {
        uint256 userSub;
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            PoolInfo storage pool = poolInfo[pid];
            UserInfo storage user = userInfo[pid][_addr];
            updatePool(pid);
            if (user.amount > 0) {
                uint256 pending = user.amount.mul(pool.accCakePerShar
e).div(1e12).sub(user.rewardDebt);
                uint256 userReward = pending.add(user.unclaimedBuni);
                user.rewardDebt = user.amount.mul(pool.accCakePerShar
e).div(1e12);
                if(userReward > 0) {
                    user.unclaimedBuni = 0;
                    pool.amount = pool.amount.sub(user.amount);
                    user.amount = 0;
                    userSub = userSub.add(userReward);
                    withdrawAmounts[pid][_addr] = withdrawAmounts[pid]
[_addr].add(userReward);
                }
            }
            user.rewardDebt = user.amount.mul(pool.accCakePerShare).div
(1e12);
        }
        if (userSub <= 0) { return; }
        safeCakeTransfer(_addr, userSub);
    }
```

- **Repair Status**

Oortswap has officially fixed the risk.

4.3.7. Accuracy of onlyStaking modifier

- **Risk description**

SwapMining contract takerWithdraw method, the main function is the user to extract money rewards, the method uses onlyStaking modifier, the modifier in the onlyStaking address can be called, that is, the method does not allow the user to call their own rewards.

```
function takerWithdraw(address _addr) public onlyStaking {
    uint256 userSub;
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        PoolInfo storage pool = poolInfo[pid];
        UserInfo storage user = userInfo[pid][_addr];
        updatePool(pid);
        if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accCakePerShare).div
(1e12).sub(user.rewardDebt);
            uint256 userReward = pending.add(user.unclaimedBuni);
            user.rewardDebt = user.amount.mul(pool.accCakePerShare).div
(1e12);
            if(userReward > 0) {
                user.unclaimedBuni = 0;
                pool.amount = pool.amount.sub(user.amount);
                user.amount = 0;
                userSub = userSub.add(userReward);
                withdrawAmounts[pid][_addr] = withdrawAmounts[pid][_add
r].add(userReward);
            }
        }
     user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12);
        }
        if (userSub <= 0) {
            return;
        }
        safeCakeTransfer(_addr, userSub);
    }
    modifier onlyStaking {
        require(_msgSender() == staking, "onlyStaking getMarginPoolStak
ing");
        _;
   }
```

- **Repair Status**

Oortswap has officially fixed the risk.

4.3.8 Variables are updated

- **Risk description**

When there is a contract logic to obtain rewards or transfer funds, the coder mistakenly updates the value of the variable that sends the funds, so that the user can use the value of the variable that is not updated to obtain funds, thus affecting the normal operation of the project.

- **Audit Results : Passed**

4.3.9 Floating Point and Numeric Precision

- **Risk Description**

In Solidity, the floating-point type is not supported, and the fixed-length floating-point type is not fully supported. The result of the division operation will be rounded off, and if there is a decimal number, the part after the decimal point will be discarded and only the integer part will be taken, for example, dividing 5 pass 2 directly will result in 2. If the result of the operation is less than 1 in the token operation, for example, 4.9 tokens will be approximately equal to 4, bringing a certain degree of The tokens are not only the tokens of the same size, but also the tokens of the same size. Due to the economic properties of tokens, the loss of precision is equivalent to the loss of assets, so this is a cumulative problem in tokens that are frequently traded.

- **Audit Results : Passed**

4.3.10 Default Visibility

- **Risk description**

In Solidity, the visibility of contract functions is public pass default. therefore, functions that do not specify any visibility can be called externally pass the user. This can lead to serious vulnerabilities when developers incorrectly ignore visibility specifiers for functions that should be private, or visibility specifiers that can only be called from within the contract itself. One of the first hacks on Parity's multi-signature wallet was the failure to set the visibility of a function, which defaults to public, leading to the theft of a large amount of money.

- **Audit Results : Passed**

4.3.11 tx.origin authentication

- **Risk Description**

tx.origin is a global variable in Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract can make the contract vulnerable to phishing-like attacks.

- **Audit Results : Passed**

4.3.12 Faulty constructor

- **Risk description**

Prior to version 0.4.22 in solidity smart contracts, all contracts and constructors had the same name. When writing a contract, if the constructor name and the contract name are not the same, the contract will add a default constructor and the constructor you set up will be treated as a normal function, resulting in your original contract settings not being executed as expected, which can lead to terrible consequences, especially if the constructor is performing a privileged operation.

- **Audit Results : Passed**

4.3.13 Unverified return value

- **Risk description**

Three methods exist in Solidity for sending tokens to an address: transfer(), send(), call.value(). The difference between them is that the transfer function throws an exception throw when sending fails, rolls back the transaction state, and costs 2300gas; the send function returns false when sending fails and costs 2300gas; the call.value method returns false when sending fails and costs all gas to call, which will lead to the risk of reentrant attacks. If the send or call.value method is used in the contract code to send tokens without checking the return value of the method, if an error occurs, the contract will continue to execute the code later, which will lead to the thought result.

- **Audit Results : Passed**

4.3.14 Insecure random numbers

- **Risk Description**

All transactions on the blockchain are deterministic state transition operations with no uncertainty, which ultimately means that there is no source of entropy or randomness within the blockchain ecosystem. Therefore, there is no random number function like rand() in Solidity. Many developers use future block variables such as block hashes, timestamps, block highs and lows or Gas caps to generate random numbers. These quantities are controlled pass the miners who mine them and are therefore not truly random, so using past or present block variables to generate random numbers could lead to a destructive vulnerability.

- **Audit Results : Passed**

4.3.15 Timestamp Dependency

- **Risk description**

In blockchains, data block timestamps (block.timestamp) are used in a variety of applications, such as functions for random numbers, locking funds for a period of time, and conditional statements for various time-related state changes. Miners have the ability to adjust the timestamp as needed, for example block.timestamp or the alias now can be manipulated pass the miner. This can lead to serious vulnerabilities if the wrong block timestamp is used in a smart contract. This may not be necessary if the contract is not particularly concerned with miner manipulation of block timestamps, but care should be taken when developing the contract.

- **Audit Results : Passed**

4.3.16 Transaction order dependency

- **Risk description**

In a blockchain, the miner chooses which transactions from that pool will be included in the block, which is usually determined pass the gasPrice transaction, and the miner will choose the transaction with the highest transaction fee to pack into the block. Since the information about the transactions in the block is publicly available, an attacker can watch the transaction pool for transactions that may contain problematic solutions, modify or revoke the attacker’s privileges or change the state of the contract to the attacker’s detriment. The attacker can then take data from this transaction and create a higher-level transaction gasPrice and include its transactions in a block before the original, which will preempt the original transaction solution.

- **Audit Results : Passed**

4.3.17 Delegatecall

- **Risk Description**

In Solidity, the delegatecall function is the standard message call method, but the code in the target address runs in the context of the calling contract, i.e., keeping msg.sender and msg.value unchanged. This feature supports implementation libraries, where developers can create reusable code for future contracts. The code in the library itself can be secure and bug-free, but when run in another application’s environment, new vulnerabilities may arise, so using the delegatecall function may lead to unexpected code execution.

- **Audit Results : Passed**

4.3.18 Call

- **Risk Description**

The call function is similar to the delegatecall function in that it is an underlying function provided pass Solidity, a smart contract writing language, to interact with external contracts or libraries, but when the call function method is used to handle an external Standard Message Call to a contract, the code runs in the environment of the external contract/function The call function is used to interact with an external contract or library. The use of such functions requires a determination of the security of the call parameters, and caution is recommended. An attacker could easily borrow the identity of the current contract to perform other malicious operations, leading to serious vulnerabilities.

- **Audit Results : Passed**

4.3.19 Denial of Service

- **Risk Description**

Denial of service attacks have a broad category of causes and are designed to keep the user from making the contract work properly for a period of time or permanently in certain situations, including malicious behavior while acting as the recipient of a transaction, artificially increasing the gas required to compute a function causing gas exhaustion (such as controlling the size of variables in a for loop), misuse of access control to access the private component of the contract, in which the Owners with privileges are modified, progress state based on external calls, use of obfuscation and oversight, etc. can lead to denial of service attacks.

- **Audit Results : Passed**

4.3.20 Logic Design Flaw

- **Risk Description**

In smart contracts, developers design special features for their contracts intended to stabilize the market value of tokens or the life of the project and increase the highlight of the project, however, the more complex the system, the more likely it is to have the possibility of errors. It is in these logic and functions that a minor mistake can lead to serious depasstions from the whole logic and expectations, leaving fatal hidden dangers, such as errors in logic judgment, functional implementation and design and so on.

- **Audit Results : Passed**

4.3.21 Fake recharge vulnerability

- **Risk Description**

The success or failure (true or false) status of a token transaction depends on whether an exception is thrown during the execution of the transaction (e.g., using mechanisms such as require/assert/revert/throw). When a user calls the transfer function of a token contract to transfer funds, if the transfer function runs normally without throwing an exception, the transaction will be successful or not, and the status of the transaction will be true. When balances[msg.sender] < _value goes to the else logic and returns false, no exception is thrown, but the transaction acknowledgement is successful, then we believe that a mild if/else judgment is an undisciplined way of coding in sensitive function scenarios like transfer, which will lead to Fake top-up vulnerability in centralized exchanges, centralized wallets, and token contracts.

- **Audit Results : Passed**

4.3.22 Short Address Attack Vulnerability

- **Risk Description**

In Solidity smart contracts, when passing parameters to a smart contract, the parameters are encoded according to the ABI specification. the EVM runs the attacker to send encoded parameters that are shorter than the expected parameter length. For example, when transferring money on an exchange or wallet, you need to send the transfer address address and the transfer amount value. The attacker could send a 19-passte address instead of the standard 20-passte address, in which case the EVM would fill in the 0 at the end of the encoded parameter to make up the expected length, which would result in an overflow of the final transfer amount parameter value, thus changing the original transfer amount.

- **Audit Results : Passed**

4.3.23 Uninitialized storage pointer

- **Risk description**

EVM uses both storage and memory to store variables. Local variables within functions are stored in storage or memory pass default, depending on their type. uninitialized local storage variables could point to other unexpected storage variables in the contract, leading to intentional or unintentional vulnerabilities.

- **Audit Results : Passed**

4.3.24 Frozen Account bypass

- **Risk Description**

In the transfer operation code in the contract, detect the risk that the logical functionality to check the freeze status of the transfer account exists in the contract code and can be passpassed if the transfer account has been frozen.

- **Audit Results : Passed**

4.3.25 Uninitialized

- **Risk description**

The initialize function in the contract can be called pass another attacker before the owner, thus initializing the administrator address.

- **Audit Results : Passed**

4.3.26 Reentry Attack

- **Risk Description**

An attacker constructs a contract containing malicious code at an external address in the Fallback function When the contract sends tokens to this address, it will call the malicious code. The call.value() function in Solidity will consume all the gas he receives when it is used to send tokens, so a re-entry attack will occur when the call to the call.value() function to send tokens occurs before the actual reduction of the sender's account balance. The re-entry vulnerability led to the famous The DAO attack.

- **Audit Results : Passed**

4.3.27 Integer Overflow

- **Risk Description**

Integer overflows are generally classified as overflows and underflows. The types of integer overflows that occur in smart contracts include three types: multiplicative overflows, additive overflows, and subtractive overflows. In Solidity language, variables support integer types in steps of 8, from uint8 to uint256, and int8 to int256, integers specify fixed size data types and are unsigned, for example, a uint8 type , can only be stored in the range 0 to 2^8-1, that is, [0,255] numbers, a uint256 type can only store numbers in the range 0 to 2^256-1. This means that an integer variable can only have a certain range of numbers represented, and cannot exceed this formulated range. Exceeding the range of values expressed pass the variable type will result in an integer overflow vulnerability.

- **Audit Results : Passed**

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
Lunaray Toolkit	self-developed toolkit

Disclaimer：

Lunaray 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.

Lunaray 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 Lunaray 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, Lunaray Technology shall not be liable for any losses and adverse effects caused thereby.

There are risks in the market, and investment needs to be cautious. This report only conducts security audits and results announcements on smart contract codes, and does not make investment recommendations and basis.

LUNARAY
BLOCKCHAINSECURITY

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec