Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.06.20, the SlowMist security team received the Neopin team's security audit application for Neopin, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit
		Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Audit version:

https://github.com/Neopin/neopin-defi-contracts

commit: d765a3efdf9e080b95b2e8a2b3508e7edc3d40ee

-packages/dex-contract/contracts

-packages/stake-contract/contracts

Fixed version:

https://github.com/Neopin/neopin-defi-contracts

commit: 6698f361c67abe6111ab2114a3571538f6d56161

-packages/dex-contract/contracts

-packages/stake-contract/contracts

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N2	Re-initialize issue	Others	Low	Fixed
N3	Risk of replay attack	Replay Vulnerability	Suggestion	Confirmed
N4	Malleable attack risk	Replay Vulnerability	Suggestion	Confirmed
N5	Missing event log	Others	Suggestion	Fixed
N6	Failure to follow the Checks-Effects-Interactions principle	Reentrancy Vulnerability	Suggestion	Fixed

NO	Title	Category	Level	Status
N7	Compatibility issue	Others	Suggestion	Confirmed
N8	Potential external call risk	Unsafe External Call Audit	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

UniswapV2ERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
approve	External	Can Modify State	-
transfer	External	Can Modify State	-

UniswapV2ERC20			
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

UniswapV2Factory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
pairCodeHash	External	-	-
createPair	External	Can Modify State	-
setFeeTo	External	Can Modify State	-
setMigrator	External	Can Modify State	-
setFeeToSetter	External	Can Modify State	-

UniswapV2Pair			
Function Name	Visibility	Mutability	Modifiers
getReserves	Public	-	-
_safeTransfer	Private	Can Modify State	-
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
_update	Private	Can Modify State	-
_mintFee	Private	Can Modify State	-

UniswapV2Pair			
mint	External	Can Modify State	lock
burn	External	Can Modify State	lock
swap	External	Can Modify State	lock
skim	External	Can Modify State	lock
sync	External	Can Modify State	lock

UniswapV2Router02			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
_addLiquidity	Internal	Can Modify State	-
addLiquidity	External	Can Modify State	ensure
addLiquidityETH	External	Payable	ensure
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-
removeLiquidityETHSupportingFeeOnTransferTokens	Public	Can Modify State	ensure

UniswapV2Router02			
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	Can Modify State	-
_swap	Internal	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
_swapSupportingFeeOnTransferTokens	Internal	Can Modify State	-
swapExactTokensForTokensSupportingFeeOnTransferTokens	External	Can Modify State	ensure
swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	ensure
swapExactTokensForETHSupportingFeeOnTransferTokens	External	Can Modify State	ensure
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

WETH			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
deposit	Public	Payable	-
withdraw	Public	Can Modify State	-
totalSupply	Public	-	-
approve	Public	Can Modify State	-
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-

StRewardBar			
Function Name	Visibility	Mutability	Modifiers
mint	Public	Can Modify State	-
burn	Public	Can Modify State	-
<Constructor>	Public	Can Modify State	-
safeRewardTransfer	Public	Can Modify State	-
delegates	External	-	-
delegate	External	Can Modify State	-
delegateBySig	External	Can Modify State	-
getCurrentVotes	External	-	-
getPriorVotes	External	-	-

StRewardBar			
_delegate	Internal	Can Modify State	-
_moveDelegates	Internal	Can Modify State	-
_writeCheckpoint	Internal	Can Modify State	-
safe32	Internal	-	-
getChainId	Internal	-	-

StBonusChef			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	onlyOwner
poolLength	External	-	-
updateRewardProvider	Public	Can Modify State	onlyOwner
addPool	Public	Can Modify State	onlyOwner
getMultiplier	Public	-	-
pendingReward	External	-	-
updatePool	Public	Can Modify State	-
claimByDeposit	External	Can Modify State	nonReentrant
claimByWithdraw	External	Can Modify State	nonReentrant
claimReward	External	Can Modify State	nonReentrant
attachStakeChef	External	Can Modify State	-

StBonusChef			
detachStakeChef	External	Can Modify State	-
emergencyRedeemReward	Public	Can Modify State	onlyOwner
safeRewardTransfer	Internal	Can Modify State	-

StStakeChef			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	onlyOwner
poolLength	External	-	-
updateRewardBar	Public	Can Modify State	onlyOwner
addPool	Public	Can Modify State	onlyOwner
updateBonusChef	Public	Can Modify State	onlyOwner
updateNextRewardPerBlock	Public	Can Modify State	onlyOwner
getStakeToken	Public	-	-
getUserAmount	Public	-	-
getMultiplier	Public	-	-
pendingReward	External	-	-
updatePool	Public	Can Modify State	-
deposit	Public	Can Modify State	nonReentrant whenNotPaused
withdraw	Public	Can Modify State	nonReentrant whenNotPaused

StStakeChef			
enterStaking	Public	Can Modify State	nonReentrant whenNotPaused
leaveStaking	Public	Can Modify State	nonReentrant whenNotPaused
emergencyWithdraw	Public	Can Modify State	nonReentrant whenNotPaused
emergencyRedeemReward	Public	Can Modify State	onlyOwner
safeRewardTransfer	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

1.In the StRewardBar contract, the admin role can grant the MINTER_BURNER_ROLE, and the MINTER_BURNER_ROLE can mint tokens arbitrarily through the mint function, and there is no upper limit on the number of minted tokens. And the MINTER_BURNER_ROLE role can also burn users' tokens through the burn function.

Code location:

stake-contract/contracts/stake/StRewardBar.sol#16-26

```
function mint(address _to, uint256 _amount) public {
     require(hasRole(MINTER_BURNER_ROLE, msg.sender), "not minter/burner role");
     _mint(_to, _amount);
     _moveDelegates(address(0), _delegates[_to], _amount);
 }

 function burn(address _from ,uint256 _amount) public {
     require(hasRole(MINTER_BURNER_ROLE, msg.sender), "not minter/burner role");
     _burn(_from, _amount);
```

```
        _moveDelegates(_delegates[_from], address(0), _amount);
    }
```

2.In the StBonusChef and StStakeChef contract, the owner role can withdraw any amount of the rewardToken through the emergencyRedeemReward function.

Code location:

stake-contract/contracts/stake/StBonusChef.sol#230-234

stake-contract/contracts/stake/StStakeChef.sol#291-295

```
    function emergencyRedeemReward(uint256 _pid, uint256 _amount) public onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];
        safeRewardTransfer(pool.rewardToken, msg.sender, _amount);
        emit EmergencyRedeemReward(msg.sender, _pid, _amount);
    }
```

3.The owner role can add a pool arbitrarily through the add function, and there is a risk that the Owner can add apool to mine by itself to obtain rewards. When calling the add function to add a pool, the lastRewardBlock and totalAllocPoint will be updated, and related information about the pool will be stored.

Code location:

stake-contract/contracts/stake/StBonusChef.sol#61-83

```
    function addPool(IERC20 _rewardToken, uint256 _rewardPerBlock, uint256
_startBlock, uint256 _endBlock) public onlyOwner {
        require(address(stakeChef) != address(0), "not initialized stakeChef");
        require(address(rewardProvider) != address(0), "not initialized
rewardProvider");
        require(_endBlock > _startBlock, "invalid endBlock");

        uint256 totalRewardAmount = _rewardPerBlock * (_endBlock - _startBlock);
        _rewardToken.safeTransferFrom(rewardProvider, address(this),
totalRewardAmount);

        uint256 lastRewardBlock = block.number > _startBlock ? block.number :
_startBlock;
        poolInfo.push(PoolInfo({
```

```
            rewardToken: _rewardToken,
            rewardPerBlock: _rewardPerBlock,
            lastRewardBlock: lastRewardBlock,
            accRewardPerShare: 0,
            startBlock: _startBlock,
            endBlock: _endBlock,
            spid: 0,
            stakeSupply: 0,
            isAttached: false
        }));
        uint256 _pid = poolInfo.length - 1;
        emit AddPool(_pid, _rewardToken, _rewardPerBlock, _startBlock, _endBlock);
    }
```

stake-contract/contracts/stake/StStakeChef.sol#61-80

```
    function addPool(IERC20 _stakeToken, IERC20 _rewardToken, uint256 _startBlock,
uint256 _rewardPerBlock, uint256 _lockupDuration, IStBonusChef _bonusChef, uint256
_bpid) public onlyOwner {
        require(address(rewardBar) != address(0), 'invalid RewardBar');
        uint256 lastRewardBlock = block.number > _startBlock ? block.number :
_startBlock;
        poolInfo.push(PoolInfo({
            stakeToken: _stakeToken,
            rewardToken: _rewardToken,
            rewardPerBlock: _rewardPerBlock,
            lastRewardBlock: lastRewardBlock,
            accRewardPerShare: 0,
            lockupDuration: _lockupDuration,
            nextRewardPerBlock: 0,
            nextBlockNumber: 0,
            bonusChef: _bonusChef,
            bpid: _bpid
        }));
        uint256 _pid = poolInfo.length - 1;
        if (address(_bonusChef) != address(0)) {
            _bonusChef.attachStakeChef(_pid, _bpid);
        }
    }
```

Solution

It is recommended to use a time lock mechanism or community governance to restrict.

Status

Confirmed

[N2] [Low] Re-initialize issue

Category: Others

Content

In the StStakeChef and StBonusChef contracts, the owner role can initialize the stakeChef and rewardProvider address through the initialize function, but it can be initialized repeatedly and there is no event logging performed or the 0 address check.

Code location:

stake-contract/contracts/stake/StBonusChef.sol#48-51

```
    function initialize(StStakeChef _stakeChef, address _rewardProvider) public
onlyOwner {
        stakeChef = _stakeChef;
        rewardProvider = _rewardProvider;
    }
```

stake-contract/contracts/stake/StStakeChef.sol#49-51

```
    function initialize(StRewardBar _rewardBar) public onlyOwner {
        rewardBar = _rewardBar;
    }
```

Solution

It is recommended to prohibit repeated initialization operations and use a separate interface to set related parameters and record events when modifying sensitive parameter.

Status

Fixed

[N3] [Suggestion] Risk of replay attack

Category: Replay Vulnerability

Content

1.DOMAIN_SEPARATOR is defined when the contract is initialized, but it is not reimplemented when DOMAIN_SEPARATOR is used in the permit function. So the DOMAIN_SEPARATOR contains the chainId and is defined at contract deployment instead of reconstructed for every signature, there is a risk of possible replay attacks between chains in the event of a future chain split.

Code location:

dex-contract/contracts/UniswapV2/UniswapV2ERC20.sol#25-39

```
    constructor() {
     uint256 chainId;
     assembly {
       chainId := chainid()
     }
     DOMAIN_SEPARATOR = keccak256(
       abi.encode(
         keccak256('EIP712Domain(string name,string version,uint256 chainId,address
verifyingContract)'),
         keccak256(bytes(name)),
         keccak256(bytes('1')),
         chainId,
         address(this)
       )
     );
   }
```

2.The delegateBySig function uses the ecrecover of the contract to verify the signature. The signature data and v, r, s are passed in from the chain. If the random k value under the chain is repeated, the private key of the signature will be calculated.

Code location:

stake-contract/contracts/stake/StRewardBar.sol#104-136

```
    function delegateBySig(address delegatee, uint nonce, uint expiry, uint8 v,
bytes32 r, bytes32 s) external {
        bytes32 domainSeparator = keccak256(
            abi.encode(
                DOMAIN_TYPEHASH,
                keccak256(bytes(name())),
                getChainId(),
                address(this)
            )
        );

        bytes32 structHash = keccak256(
            abi.encode(
                DELEGATION_TYPEHASH,
                delegatee,
                nonce,
                expiry
            )
        );

        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                domainSeparator,
                structHash
            )
        );

        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "vNPT::delegateBySig: invalid sig");
        require(nonce == nonces[signatory]++, "vNPT::delegateBySig: invalid nonce");
        require(block.timestamp <= expiry, "vNPT::delegateBySig: sig expired");
        return _delegate(signatory, delegatee);
    }
```

Solution

1. It is recommended to redefine when using DOMAIN_SEPARATOR.

2. It is recommended that when generating signature data off-chain, avoid repetition of random k values, resulting in corresponding repeated r values being submitted to the chain.

   Referance: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2612.md

Status

Confirmed

[N4] [Suggestion] Malleable attack risk

Category: Replay Vulnerability

Content

In the permit function, it restores the address of the signer through the ecrecover function but does not check the value of v and s. Since EIP2 still allows the malleability for ecrecover, this will lead to the risk of transaction malleability attacks.

Code location：

dex-contract/contracts/UniswapV2/UniswapV2ERC20.sol#94-110

```
  function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
  ) external {
    require(deadline >= block.timestamp, 'NeopinswapERC20: EXPIRED');
    bytes32 digest = keccak256(
      abi.encodePacked('\x19\x01', DOMAIN_SEPARATOR,
keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++,
deadline)))
    );
    address recoveredAddress = ecrecover(digest, v, r, s);
    require(recoveredAddress != address(0) && recoveredAddress == owner,
'NeopinswapERC20: INVALID_SIGNATURE');
```

```
    _approve(owner, spender, value);
  }
```

Solution

It is recommended to use the ECDSA library of openzeppelin to check the signature.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol

Status

Confirmed

[N5] [Suggestion] Missing event log

Category: Others

Content

In the StStakeChef and StBonusChef contract, the Owner role can set stakeChef, rewardProvider, rewardBar, nextRewardPerBlock, nextBlockNumber and addPool through the initialize, updateRewardProvider, updateRewardBar and addPool function, but no event logging is preformed.

Codel ocation:

stake-contract/contracts/stake/StBonusChef.sol#48-51,57-60

```
    function initialize(StStakeChef _stakeChef, address _rewardProvider) public
onlyOwner {
        stakeChef = _stakeChef;
        rewardProvider = _rewardProvider;
    }

    function updateRewardProvider(address _rewardProvider) public onlyOwner {
        rewardProvider = _rewardProvider;
    }
```

stake-contract/contracts/stake/StStakeChef.sol#49-51, 57-59, 61-80, 96-103

```
    function initialize(StRewardBar _rewardBar) public onlyOwner {
        rewardBar = _rewardBar;
```

```
    }

    function updateRewardBar(StRewardBar _rewardBar) public onlyOwner {
        rewardBar = _rewardBar;
    }

    function addPool(IERC20 _stakeToken, IERC20 _rewardToken, uint256 _startBlock,
uint256 _rewardPerBlock, uint256 _lockupDuration, IStBonusChef _bonusChef, uint256
_bpid) public onlyOwner {
        require(address(rewardBar) != address(0), 'invalid RewardBar');
        uint256 lastRewardBlock = block.number > _startBlock ? block.number :
_startBlock;
        poolInfo.push(PoolInfo({
            stakeToken: _stakeToken,
            rewardToken: _rewardToken,
            rewardPerBlock: _rewardPerBlock,
            lastRewardBlock: lastRewardBlock,
            accRewardPerShare: 0,
            lockupDuration: _lockupDuration,
            nextRewardPerBlock: 0,
            nextBlockNumber: 0,
            bonusChef: _bonusChef,
            bpid: _bpid
        }));
        uint256 _pid = poolInfo.length - 1;
        if (address(_bonusChef) != address(0)) {
            _bonusChef.attachStakeChef(_pid, _bpid);
        }
    }

    function updateNextRewardPerBlock(uint256 _pid, uint256 _nextRewardPerBlock,
uint256 _nextBlockNumber) public onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];
        require(block.number < _nextBlockNumber, "invalid nextBlockNumber");
        require (address(pool.stakeToken) != address(0), "invalid pid");

        pool.nextRewardPerBlock = _nextRewardPerBlock;
        pool.nextBlockNumber = _nextBlockNumber;
    }
```

Solution

It is recommended to record events when modifying sensitive parameters.

Status

Fixed

[N6] [Suggestion] Failure to follow the Checks-Effects-Interactions principle

Category: Reentrancy Vulnerability

Content

In the event of an emergency, users can withdraw their staked assets through the emergencyWithdraw function. However, during the withdrawal process, it first transfers the staked funds to the user through the safeTransfer function, and then sets user.amount and user.rewardDebt to 0, which does not comply with the Checks-Effects-Interactions principle.

Code location：

stake-contract/contracts/stake/StStakeChef.sol#278-288

```
    function emergencyWithdraw(uint256 _pid) public nonReentrant whenNotPaused {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        require(block.number > user.unlockBlockNumber, "withdraw locked");

        pool.stakeToken.safeTransfer(address(msg.sender), user.amount);
        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
        user.amount = 0;
        user.rewardDebt = 0;
    }
```

Solution

It is recommended to follow the Checks-Effects-Interactions principle and change the status first before performing the transfer operation.

Status

Fixed

[N7] [Suggestion] Compatibility issue

Category: Others

Content

In the StBonusChef contract, users can stake/withdraw their tokens through the deposit function and withdraw function. It will directly record the amount parameter passed by the user into user.amount, and transfer the tokens to the contract through the safeTransferFrom function. If the contract receives deflationary tokens, the actual number of tokens received by the contract will not match the number of tokens recorded in the contract.

Code location:

stake-contract/contracts/stake/StBonusChef.sol#162-217

```
    function deposit(uint256 _pid, uint256 _amount) public nonReentrant whenNotPaused
{
        require (_pid != 0, "deposit NPT by staking");

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        updatePool(_pid);

        if (user.amount > 0) {
            uint256 pending = user.amount * pool.accRewardPerShare / 1e12 -
user.rewardDebt;
            if(pending > 0) {
                safeRewardTransfer(pool.rewardToken, msg.sender, pending);
                emit ClaimReward(msg.sender, _pid, pending);
            }
            user.claimedReward += pending;
        }
        if (address(pool.bonusChef) != address(0)) {
            pool.bonusChef.claimByDeposit(_pid, pool.bpid, msg.sender, _amount);
        }
        if (_amount > 0) {
            pool.stakeToken.safeTransferFrom(address(msg.sender), address(this),
_amount);
            user.amount = user.amount + _amount;
            user.unlockBlockNumber = block.number + pool.lockupDuration;
```

```
        }
        user.rewardDebt = user.amount * pool.accRewardPerShare / 1e12;

        emit Deposit(msg.sender, _pid, _amount);
    }

    // Withdraw tokens from StakeChef.
    function withdraw(uint256 _pid, uint256 _amount) public nonReentrant
whenNotPaused {
        require (_pid != 0, "withdraw NPT by unstaking");
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        require(block.number > user.unlockBlockNumber, "withdraw locked");
        require(user.amount >= _amount, "withdraw not good");
        updatePool(_pid);

        uint256 pending = user.amount * pool.accRewardPerShare / 1e12 -
user.rewardDebt;
        if(pending > 0) {
            safeRewardTransfer(pool.rewardToken, msg.sender, pending);
            user.claimedReward += pending;
            emit ClaimReward(msg.sender, _pid, pending);
        }
        if (address(pool.bonusChef) != address(0)) {
            pool.bonusChef.claimByWithdraw(_pid, pool.bpid, msg.sender, _amount);
        }
        if(_amount > 0) {
            user.amount = user.amount - _amount;
            pool.stakeToken.safeTransfer(address(msg.sender), _amount);
        }
        user.rewardDebt = user.amount * pool.accRewardPerShare / 1e12;

        emit Withdraw(msg.sender, _pid, _amount);
    }
```

Solution

It is recommended to use the difference between the contract balance before and after the transfer to record the user's actual recharge amount.

Status

Confirmed

[N8] [Suggestion] Potential external call risk

Category: Unsafe External Call Audit

Content

In the UniswapV2Pair contract, the liquidty value will get from the call of the desiredLiquidity by the migrator in the mint function. But the IMigrator is an external part and do not included in the audit scope, if the external part has security issues, the project itself may cause unknown risks.

Code location:

dex-contract/contracts/UniswapV2/UniswapV2Pair.sol#138

```
if (_totalSupply == 0) {
      address migrator = IUniswapV2Factory(factory).migrator();
      if (msg.sender == migrator) {
        liquidity = IMigrator(migrator).desiredLiquidity();
        require(liquidity > 0 && liquidity != type(uint256).max, 'Bad desired
liquidity');
      }
```

Solution

It is recommended to clarify whether this external call contract is a credible contract.

Status

Confirmed; After communication with the project team, they expressed that they do not support migrators and this part is not being used.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002206270002	SlowMist Security Team	2022.06.20 - 2022.06.27	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 1 low risk, 6 suggestion vulnerabilities. And 1 medium risk, 4 suggestion vulnerabilities were confirmed; All other findings were fixed. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist