Smart Contract Security Audit Report

Contents

1. Executive Summary

On September 9, 2020, the SlowMist security team received the MoonSwap team's security audit application for MoonSwap, 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.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module 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.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk	Medium vulnerability will affect the operation of DeFi project. It is recommended

vulnerablities	to fix medium-risk vulnerabilities.
Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house 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:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack

- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

MoonSwap is a High speed, 0 GAS AMM DEX with Ethereum and Conflux.

Project website:

https://moonswap.fi/

Audit version code:

https://github.com/moon-migration/moonswap-eth-contract/tree/d6a833e1d5494ae09ee791c93b4aa9ba10510172

Fixed version code:

https://github.com/moon-migration/moonswap-eth-contract/tree/ef19afe4b6bfd624dd79903c36ea335be6a7b283

4. Code Overview

4.1 Contracts Description

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

GovernorAlpha			
Function Name	Visibility	Mutability	Modifiers
quorumVotes	Public	-	-
proposalThreshold	Public	-	-
proposalMaxOperations	Public	-	-
votingDelay	Public	-	-
votingPeriod	Public	-	-
propose	Public	Can modify state	-
queue	Public	Can modify state	-
_queueOrRevert	Internal	Can modify state	-
execute	Public	Payable	-
cancel	Public	Can modify state	-
getActions	Public	-	-
getReceipt	Public	-	-
state	Public	-	-
castVote	Public	Can modify state	-
castVoteBySig	Public	Can modify state	-
_castVote	Internal	Can modify state	-
__acceptAdmin	Public	Can modify state	-
__abdicate	Public	Can modify state	-
__queueSetTimelockPendingAdmin	Public	Can modify state	-
__executeSetTimelockPendingAdmin	Public	Can modify state	-
add256	Internal	-	-
sub256	Internal	-	-
getChainId	Internal	-	-

MoonToken			
Function Name	Visibility	Mutability	Modifiers
mint	Public	Can modify state	-
_transfer	Internal	Can modify state	-
delegates	External	-	-
delegate	External	Can modify state	-
delegateBySig	External	Can modify state	-
getCurrentVotes	External	-	-
getPriorVotes	External	-	-
_delegate	Internal	Can modify state	-
_moveDelegates	Internal	Can modify state	-
_writeCheckpoint	Internal	Can modify state	-
safe32	Internal	-	-
getChainId	Internal	-	-

Ownable			
Function Name	Visibility	Mutability	Modifiers
owner	Public	-	-
renounceOwnership	Public	Can modify state	onlyOwner
transferOwnership	Public	Can modify state	onlyOwner

MasterStar			
Function Name	Visibility	Mutability	Modifiers
poolLength	External	-	-
mintEarlybirdToken	Public	Can modify state	onlyOwner
add	Public	Can modify state	onlyOwner
set	Public	Can modify state	onlyOwner
setMigrator	Public	Can modify state	onlyOwner
migrate	Public	Can modify state	-
setCrosschain	Public	Can modify state	onlyOwner
pendingToken	External	-	-

massUpdatePools	Public	Can modify state	-
updatePool	Public	Can modify state	-
deposit	Public	Can modify state	-
withdraw	Public	Can modify state	-
emergencyWithdraw	Public	Can modify state	-
safeTokenTransfer	Internal	Can modify state	-
tokenConvert	Public	Can modify state	-
dev	Public	Can modify state	-
_depositMigratePoolAddr	Internal	Can modify state	-
_transferMigratePoolAddr	Internal	Can modify state	-
_getPoolReward	Internal	-	-
_getPhaseBlocknum	Internal	-	-
_assignPoolReward	Internal	-	-

Migrator			
Function Name	Visibility	Mutability	Modifiers
migrate	Public	Can modify state	-

Timelock			
Function Name	Visibility	Mutability	Modifiers
setDelay	Public	Can modify state	-
acceptAdmin	Public	Can modify state	-
setPendingAdmin	Public	Can modify state	-
queueTransaction	Public	Can modify state	-
cancelTransaction	Public	Can modify state	-
executeTransaction	Public	Payable	-
getBlockTimestamp	Internal	-	-
receive()	External	Payable	-

4.2 Code Audit

4.2.1 Low-risk vulnerabilities

4.2.1.1 Excessive auditing authority

The owner in MasterStar contract can add a new lpToken through the add function, but if there is a black swan event, such as the addition of a malicious lpToken, there will be useless lpToken to recharge to get rewards. It is suggested that the owner can be handed over to the governance contract or time lock contract for management.

```
function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    require(poolIndexs[address(_lpToken)] < 1, "LpToken exists");
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        tokenPerBlock: currentTokenPerBlock,
        accTokenPerShare: 0,
        finishMigrate: false,
        lockCrosschainAmount:0,
        crosschain_enable: false
    }));
    poolIndexs[address(_lpToken)] = poolInfo.length;
}
```

Owner can set migrator , It is suggested that the owner can be handed over to the governance contract or time lock contract for management.

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

```
    }

    // Migrate lp token to another lp contract. Can be called by anyone. 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 = 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: bad");
        pool.lpToken = newLpToken;
        pool.finishMigrate = true;
    }
```

Fixed: The owner authority has been transferred to the timelock contract.

Reference:

https://etherscan.io/tx/0x3e8be2489c824906c7fe1abe376ccea198e3cd28cb225dee91d4f9c3e962a889

4.2.2 Enhancement Suggestions

4.2.2.1 Compiler version is inconsistent

The compiler version used by the imported contract is inconsistent. It is recommended to use a unified fixed compiler version when deploying.

```
pragma solidity ^0.6.0;
pragma solidity ^0.6.2;
pragma solidity 0.6.12;
```

4.2.2.2 Better handling of ownership transfers

When using the transferOwnership function to change the owner, it is recommended to add a

confirmation method that newOwner accepts the owner. The real authority transfer is performed after the new address is signed and confirmed to avoid the loss of authority.

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

4.2.2.3 Enhancement point of delegateBySig function

The nonce in the delegateBySig function is input by the user. When the user input a larger nonce, the current transaction cannot be success but the relevant signature data will still remain on the chain, causing this signature to be available for some time in the future. It is recommended to fix it according to EIP-2612.

Reference: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2612.md#implementation.

```
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), "MOON::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "MOON::delegateBySig: invalid nonce");
        require(now <= expiry, "MOON::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }
```

4.2.2.4 Mint issue

The owner can mint tokens unlimitedly through mint function, but the owner's authority of the token contract is changed to MasterStar contract for the first time.

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

Fixed: The owner authority has actually been transferred to the MasterStar contract.

Reference:

https://etherscan.io/tx/0x0303672ee5045cd01102fdb50787541d11bddc3e1bfc446d4f6b46db85e65bff

4.2.2.5 Using now globally available variables that will be deprecated

The now globally available variables is used, which has been deprecated in compiler solidity 0.7.0.

```
        require(signatory != address(0), "MOON::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "MOON::delegateBySig: invalid nonce");
        require(now <= expiry, "MOON::delegateBySig: signature expired");
```

4.2.2.6 0 value is not checked

The withdraw function suggests adding a check of _amount> 0, which can optimize the gas consumption when _amount is 0.

```
    function withdraw(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];
        require(!pool.finishMigrate, "migrate not withdraw");
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);
        uint256 pending = user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardDebt);
        safeTokenTransfer(msg.sender, pending);
        user.amount = user.amount.sub(_amount);
        user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12);
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
        emit Withdraw(msg.sender, _pid, _amount);
    }
```

4.2.2.7 Prompt Error

The prompt of setCrosschain function require has an error "migrate not deposit", it is recommended to modify the prompt to "migrate not setCrosschain".

```
function setCrosschain(uint256 _pid, bool isOk, address cmoonAddr) public onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];
        require(pool.finishMigrate, "migrate not deposit");
        pool.crosschain_enable = isOk;
        require(cmoonAddr != address(0), "address invalid");
        migratePoolAddrs[_pid] = cmoonAddr;
    }
```

Fixed: The issue has been fixed by this commit: ef19afe4b6bfd624dd79903c36ea335be6a7b283.

4.2.2.8 Better handling of devaddr transfers

When changing devaddr in the dev function, it is recommended to add newDevaddr to accept the replacement confirmation method. After the new address is signed and confirmed, the real change to devaddr can be made to avoid setting errors and the income cannot be normally obtained.

```
    function dev(address _devaddr) public {
        require(msg.sender == devaddr, "dev: wut?");
        devaddr = _devaddr;
    }
```

4.2.2.9 Coding Standards

The coding style of emergencyWithdraw function is to make an external call first, and then change the value of the contract variable. This way of writing, because lpToken is considered safe, there is no reentrancy problem, but it is recommended to use the correct coding standard: The variable is changed, and then an external call is made. A lock modifier for reentrancy prevention can also be added.

```
    function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        require(!pool.finishMigrate, "migrate not withdraw");
        UserInfo storage user = userInfo[_pid][msg.sender];
        pool.lpToken.safeTransfer(address(msg.sender), user.amount);
```

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

Fixed: The issue has been fixed by this commit: ef19afe4b6bfd624dd79903c36ea335be6a7b283

5. Audit Result

5.1 Conclusion

Audit Result : Passed(Low risks)

Audit Number : 0X002009090001

Audit Date : September 09, 2020

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist Team in-house analysis tool audit of the codes for security issues, no critical, high-risk, medium-risk were found during the audit. There has a low-risk vulnerabilitie and 9 enhancement suggestions. MoonSwap project has an early bird plan, owner can mint early bird Lp token only once. The minimum delay time of timelock is 1 day.

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 base 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 this 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

WeChat Official Account