Smart Contract Security Audit Report

Contents

1. Executive Summary

On Feb. 22, 2021, the SlowMist security team received the XDeFi team's security audit application for xdefi-governance-token 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 vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended 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

Many best protocols are introduced in xDeFi ecosystem, including but re- stricted to - xDEX, xHalfLife, xOption, xPerp, xSTA, xNFTEX, and any other DeFi building blocks which are essential for a well-ordered market.

Project website:

https://xdefi.com

Audit version code:

https://github.com/xdefilab/xdefi-governance-token/tree/2edfbc3097732e8d49ee80ce799132b4

4. Code Overview

4.1 Contracts Description

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

XDEX			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20Detailed
setCore	Public	Can Modify State	onlyCore
mint	Public	Can Modify State	onlyCore
burnForSelf	External	Can Modify State	-

XdexStream			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
hasStream	Public	-	-
getStreamId	Public	-	lockStreamExists
createStream	External	Can Modify State	nonReentrant validStreamType
fundsToStream	Public	Can Modify State	-

FarmMaster			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
poolLength	External	-	-
setVotingPool	External	Can Modify State	onlyCore
setStream	External	Can Modify State	onlyCore
setCore	External	Can Modify State	onlyCore
addPool	External	Can Modify State	onlyCore
addLpTokenToPool	Public	Can Modify State	onlyCore poolExists
getLpTokenInfosByPoolId	External	-	poolExists
setLpFactor	Public	Can Modify State	onlyCore poolExists
massUpdatePools	Public	Can Modify State	-
updatePool	Public	Can Modify State	poolExists
pendingXDEX	External	-	poolExists
deposit	External	Can Modify State	poolExists
withdraw	Public	Can Modify State	poolExists
emergencyWithdraw	External	Can Modify State	nonReentrant poolExists
batchCollectReward	External	Can Modify State	poolExists
getUserLpAmounts	External	-	poolExists
getXCountToReward	Public	-	-

getCurRewardPerBlock	External	-	-
_getLpIndexInPool	Internal	-	-

XHalfLife			
Function Name	Visibility	Mutability	Modifiers
createStream	External	Can Modify State	createStreamPreflight
createEtherStream	External	Payable	createStreamPreflight
hasStream	External	-	-
getStream	External	-	streamExists
fundStream	External	Payable	nonReentrant streamExists
balanceOf	Public	-	streamExists
withdrawFromStream	External	Can Modify State	nonReentrant streamExists
cancelStream	External	Can Modify State	nonReentrant streamExists
getVersion	External	-	-

Timelock			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Fallback>	External	Payable	-
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	-	-

4.2 Code Audit

4.2.1 Low-risk vulnerabilities

4.2.1.1 Coding specification error

lpInfo uses memory when it is declared, which will not modify the global storage variables, there are cases where `lpInfo.lpAccPerShare = lpInfo.lpAccPerShare.add(lpReward.mul(LpRewardFixDec) .div(lpSupply));` is assigned during function execution; In the case of assignment, and the visibility of the pendingXDEX function is view, it will not change the global storage variables. Developers are asked to confirm the specific business logic to avoid lpInfo from being unable to update the value of the variable due to the visibility of the function and the memory type , Resulting in data errors.

- contracts/FarmMaster.sol   Line：374-427

```
function pendingXDEX(uint256 _pid, address _user)
        external
        view
        poolExists(_pid)
        returns (uint256)
    {
        PoolInfo memory pool = poolInfo[_pid];

        uint256 totalPending = 0;
        if (totalXFactor == 0 || pool.poolFactor == 0) {
            for (uint8 i = 0; i < pool.LpTokenInfos.length; i++) {
                UserInfo memory user =
                    poolInfo[_pid].LpTokenInfos[i].userInfo[_user];
                totalPending = totalPending.add(
                    user
                        .amount
                        .mul(pool.LpTokenInfos[i].lpAccPerShare)
                        .div(LpRewardFixDec)
                        .sub(user.rewardDebt)
                );
            }

            return totalPending;
        }

        (uint256 xdexReward, , ) =
            getXCountToReward(pool.lastRewardBlock, block.number);

        uint256 poolReward = xdexReward.mul(pool.poolFactor).div(totalXFactor);

        for (uint8 i = 0; i < pool.LpTokenInfos.length; i++) {
            LpTokenInfo memory lpInfo = pool.LpTokenInfos[i];
            uint256 lpSupply = lpInfo.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                continue;
            }
            if (block.number > pool.lastRewardBlock) {
                uint256 lpReward =
                    poolReward.mul(lpInfo.lpFactor).div(pool.poolFactor);
                lpInfo.lpAccPerShare = lpInfo.lpAccPerShare.add(
                    lpReward.mul(LpRewardFixDec).div(lpSupply)
                );
            }
            UserInfo memory user =
                poolInfo[_pid].LpTokenInfos[i].userInfo[_user];
            totalPending = totalPending.add(
                user.amount.mul(lpInfo.lpAccPerShare).div(LpRewardFixDec).sub(
                    user.rewardDebt
                )
            );
        }

        return totalPending;
    }
```

Fix Status: This issues has not been fixed yet. After communicating with the project party, pendingXDEX is only used to read data and does not modify the data on the chain, so it is an error in the coding specification. The following code will be changed to temporary variables in the future.

`lpInfo.lpAccPerShare = lpInfo.lpAccPerShare.add(lpReward.mul(LpRewardFixDec).div(lpSupply));`

4.2.1.2 Excessive authority issues

The core role can arbitrarily mint coins, change the address of the Core role, change the Stream address, change the VotingPool information, add Pool and change the information of the Pool, there is a issues of excessive authority, it is recommended to set the Core as a timelock contract. After communication and feedback, because the project is online in the initial stage, a trial operation is required. Immediately transferring the permissions to the timelock contract will affect the time plan of the trial operation. After the project trial operation ends, the project party will transfer the permissions of the Core role to the timelock contract.

- contracts/XDEX.sol

```
function setCore(address _core) public onlyCore {
        require(_core != address(0), "ERR_ZERO_ADDRESS");
        emit SET_CORE(core, _core);
        core = _core;
    }

    function mint(address account, uint256 amount) public onlyCore {
        _mint(account, amount);
    }
```

- contracts/FarmMaster.sol

```
    function setVotingPool(uint256 _pid) external onlyCore {
        votingPoolId = _pid;
        emit SetVotingPool(_pid);
    }
    // Set the xdex stream proxy.
    function setStream(address _stream) external onlyCore {
        require(_stream != address(0), "ERR_ZERO_ADDRESS");
        emit SetStream(address(stream), _stream);
        stream = XdexStream(_stream);
```

```
}
// Set new core
function setCore(address _core) external onlyCore {
    require(_core != address(0), "ERR_ZERO_ADDRESS");
    emit SetCore(core, _core);
    core = _core;
}
// Add a new lp to the pool. Can only be called by the core.
// DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function addPool(
    IERC20 _lpToken,
    uint256 _lpTokenType,
    uint256 _lpFactor,
    bool _withUpdate
) external onlyCore {
    require(poolInfo.length < PoolMaxCount, "MAX Pool Count Error");
    require(_lpFactor > 0, "Lp Token Factor is zero");

    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 _lastRewardBlock =
        block.number > startBlock ? block.number : startBlock;

    totalXFactor = totalXFactor.add(_lpFactor);

    uint256 poolinfos_id = poolInfo.length++;
    poolInfo[poolinfos_id].poolFactor = _lpFactor;
    poolInfo[poolinfos_id].lastRewardBlock = _lastRewardBlock;
    poolInfo[poolinfos_id].LpTokenInfos.push(
        LpTokenInfo({
            lpToken: _lpToken,
            lpTokenType: _lpTokenType,
            lpFactor: _lpFactor,
            lpAccPerShare: 0
        })
    );
    //The index in storage starts with 1, then need sub(1)
    lpIndexInPool[keccak256(abi.encodePacked(poolinfos_id, _lpToken))] = 1;
    emit AddPool(poolinfos_id, address(_lpToken), _lpTokenType, _lpFactor);
}
```

```
function addLpTokenToPool(
    uint256 _pid,
    IERC20 _lpToken,
    uint256 _lpTokenType,
    uint256 _lpFactor
) public onlyCore poolExists(_pid) {
    require(_lpFactor > 0, "Lp Token Factor is zero");

    massUpdatePools();

    PoolInfo memory pool = poolInfo[_pid];
    require(
        lpIndexInPool[keccak256(abi.encodePacked(_pid, _lpToken))] == 0,
       "lp token already added"
    );

    //check lpToken count
    uint256 count = pool.LpTokenInfos.length;
    require(
        count >= LpTokenMinCount && count < LpTokenMaxCount,
       "pool lpToken length is bad"
    );

    totalXFactor = totalXFactor.add(_lpFactor);

    LpTokenInfo memory lpTokenInfo =
        LpTokenInfo({
            lpToken: _lpToken,
            lpTokenType: _lpTokenType,
            lpFactor: _lpFactor,
            lpAccPerShare: 0
        });
    poolInfo[_pid].poolFactor = pool.poolFactor.add(_lpFactor);
    poolInfo[_pid].LpTokenInfos.push(lpTokenInfo);

    //save lpToken index
    //The index in storage starts with 1, then need sub(1)
    lpIndexInPool[keccak256(abi.encodePacked(_pid, _lpToken))] = count + 1;

    emit AddLP(_pid, address(_lpToken), _lpTokenType, _lpFactor);
}
```

```
function setLpFactor(
    uint256 _pid,
    IERC20 _lpToken,
    uint256 _lpFactor,
    bool _withUpdate
) public onlyCore poolExists(_pid) {
    if (_withUpdate) {
        massUpdatePools();
    }

    PoolInfo storage pool = poolInfo[_pid];
    uint256 index = _getLpIndexInPool(_pid, _lpToken);
    //update poolFactor and totalXFactor
    uint256 poolFactorNew =
        pool.poolFactor.sub(pool.LpTokenInfos[index].lpFactor).add(
            _lpFactor
        );
    pool.LpTokenInfos[index].lpFactor = _lpFactor;

    totalXFactor = totalXFactor.sub(poolInfo[_pid].poolFactor).add(
        poolFactorNew
    );
    poolInfo[_pid].poolFactor = poolFactorNew;

    emit UpdateFactor(_pid, address(_lpToken), _lpFactor);
}
```

Fix Status: The Core authority has not been transferred to the timelock contract.

4.2.2 Enhancement Suggestions

4.2.2.1 Enhanced authentication access

The functions in the XHalfLife contract are all allowed to be called openly. According to the implementation of the project, the functions in the XHalfLife contract are currently called by the XdexStream contract, and then the functions on the XdexStream contract are authenticated, and only the xdexFarmMaster contract is allowed to be called. It is recommended that the XHalfLife contract can also be enhanced authentication access to control the scope of the calling authority, such as: only allowing the XdexStream contract to access the functions in the XHalfLife contract.

Fix Status: This issues has not been fixed yet, and it is an enhancement suggestion. The project party can evaluate whether to repair it according to the actual needs of the business.

5. Audit Result

5.1 Conclusion

Audit Result : Low Risk

Audit Number : 0X002102240002

Audit Date : Feb. 24, 2021

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, 2 low-risk vulnerabilities and 1 enhancement suggestions were found during the audit, the core authority has not been transferred to the timelock contract.

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.