Smart Contract Security Audit Report

Contents

1. Executive Summary

On Feb. 01, 2021, the SlowMist security team received the XDeFi team's security audit application for XDeFi-DEX, 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-base/tree/30d91cb3ce0570da9277f6071ca713a9f69e56b5

Fixed version code:

https://github.com/xdefilab/xdefi-base/releases/tag/v0.9.2

4. Code Overview

4.1 Contracts Description

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

XSwapProxyV1			
Function Name	Visibility	Mutability	Modifiers
constructor	Public	Can Modify State	-
function()	External	Payable	-
batchSwapExactIn	Public	Payable	-
batchSwapExactInRefer	Public	Payable	nonReentrant
batchSwapExactOut	Public	Payable	-
batchSwapExactOutRefer	Public	Payable	nonReentrant
multihopBatchSwapExactIn	Public	Payable	-
multihopBatchSwapExactInRefer	Public	Payable	nonReentrant
multihopBatchSwapExactOut	Public	Payable	-
multihopBatchSwapExactOutRefer	Public	Payable	nonReentrant
create	External	Payable	nonReentrant
joinPool	External	Payable	nonReentrant
joinswapExternAmountIn	External	Payable	nonReentrant

XPToken			
Function Name	Visibility	Mutability	Modifiers
name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
allowance	Public	-	-
balanceOf	Public	-	-
totalSupply	Public	-	-
approve	Public	Can Modify State	-
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-

XPool			
Function Name	Visibility	Mutability	Modifiers
isBound	External	-	-
getNumTokens	External	-	-
getFinalTokens	External	-	_viewlock_
getDenormalizedWeight	External	-	_viewlock_
getTotalDenormalizedWeight	External	-	_viewlock_
getNormalizedWeight	External	-	_viewlock_
getBalance	External	-	_viewlock_
setController	External	Can Modify State	logs
setExitFee	External	Can Modify State	-
updateSafu	External	Can Modify State	-
updateFarm	External	Can Modify State	-
bind	External	Can Modify State	_lock_
finalize	External	Can Modify State	_lock_
gulp	External	Can Modify State	_logs_ _lock_
getSpotPrice	External	-	_viewlock_
getSpotPriceSansFee	External	-	_viewlock_
joinPool	External	Can Modify State	_lock_
exitPool	External	Can Modify State	_lock_
swapExactAmountIn	External	Can Modify State	-
swapExactAmountInRefer	Public	Can Modify State	_lock_
swapExactAmountOut	External	Can Modify State	-
swapExactAmountOutRefer	Public	Can Modify State	_lock_
joinswapExternAmountIn	External	Can Modify State	_lock_
exitswapPoolAmountIn	External	Can Modify State	_logs_ _lock_
calcOutGivenIn	Public	-	-
calcInGivenOut	Public	-	-
_pullUnderlying	Internal	Can Modify State	-
_pushUnderlying	Internal	Can Modify State	-
_pullPoolShare	Internal	Can Modify State	-
_pushPoolShare	Internal	Can Modify State	-
_mintPoolShare	Internal	Can Modify State	-
_burnPoolShare	Internal	Can Modify State	-

XFactory			
Function Name	Visibility	Mutability	Modifiers

constructor	Public	Can Modify State	-
isPool	External	-	-
newXPool	External	Can Modify State	-
setPoolCreator	External	Can Modify State	-

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

WETH9			
Function Name	Visibility	Mutability	Modifiers
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	-
function()	External	Payable	-

XConfig			
Function Name	Visibility	Mutability	Modifiers
constructor()	Public	Can Modify State	-
getCore	External	-	-
getSAFU	External	-	-
getMaxExitFee	External	-	-
getSafuFee	External	-	-
getSwapProxy	External	-	-
ethAddress	External	-	-
XDEXAddress	External	-	-
hasPool	External	-	-
setCore	External	Can Modify State	onlyCore
setSAFU	External	Can Modify State	onlyCore
setMaxExitFee	External	Can Modify State	onlyCore
setSafuFee	External	Can Modify State	onlyCore

setSwapProxy	External	Can Modify State	onlyCore
addPoolSig	External	Can Modify State	-
removePoolSig	External	Can Modify State	-
isFarmPool	External	-	-
addFarmPool	External	Can Modify State	onlyCore
removeFarmPool	External	Can Modify State	onlyCore
updateSafu	External	Can Modify State	onlyCore
updateFarm	External	Can Modify State	onlyCore
collect	External	Can Modify State	onlyCore

4.2 Contract Information

The following is the smart contract address of the project that has been deployed to the main network. Some contracts have not been deployed yet.

Contract Name	Contract Address
XConfig	0xF8BE6916b13020785e3254403E17bEB1D8719Ae7
XMath	0x48f6E8C86816db1D92c4ba2EF4322A0f704454aD
XPoolCreator	0x47c3308a2F88cDEa19a99c27b6638767942aE16b
XFactory	0x79148393Cd39a5e1b80dC9218e7531e19529Edb7
XSwapProxy	0xA6Cf4AFD790c099788372c92dDc3cd975b51ECf2

4.3 Code Audit

4.3.1 High-risk vulnerabilities

4.3.1.1 Logic error in fee calculation

The swapExactAmountInRefer and swapExactAmountOutRefer functions did not require "_swapFee >= referFee + _safuFee" on the code when calculating the swapfee. So "_swapFee <referFee + _safuFee" may appear, and there is a issues that the calculation logic of the fee has a

error. After communication feedback, the XDeFi team added `require(_safuFee.badd(_referFee) <= _swapFee, "ERR_FEE_LIMIT");` to check.

- contracts/XPool.sol Line: 397-507

```
function swapExactAmountInRefer(
    address tokenIn,
    uint256 tokenAmountIn,
    address tokenOut,
    uint256 minAmountOut,
    uint256 maxPrice,
    address referrer
) public _lock_ returns (uint256 tokenAmountOut, uint256 spotPriceAfter) {
    require(_records[tokenIn].bound, "ERR_NOT_BOUND");
    require(_records[tokenOut].bound, "ERR_NOT_BOUND");
    require(finalized, "ERR_NOT_FINALIZED");

    Record storage inRecord = _records[address(tokenIn)];
    Record storage outRecord = _records[address(tokenOut)];

    require(
        tokenAmountIn <= (inRecord.balance).bmul(MAX_IN_RATIO),
        "ERR_MAX_IN_RATIO"
    );

    uint256 spotPriceBefore =
        XMath.calcSpotPrice(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            swapFee
        );
    require(spotPriceBefore <= maxPrice, "ERR_BAD_LIMIT_PRICE");

    tokenAmountOut = calcOutGivenIn(
        inRecord.balance,
        inRecord.denorm,
        outRecord.balance,
        outRecord.denorm,
        tokenAmountIn,
```

```
            swapFee
        );
        require(tokenAmountOut >= minAmountOut, "ERR_LIMIT_OUT");
        require(
            spotPriceBefore <= tokenAmountIn.bdiv(tokenAmountOut),
           "ERR_MATH_APPROX"
        );

        inRecord.balance = (inRecord.balance).badd(tokenAmountIn);
        outRecord.balance = (outRecord.balance).bsub(tokenAmountOut);

        spotPriceAfter = XMath.calcSpotPrice(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            swapFee
        );
        require(spotPriceAfter >= spotPriceBefore, "ERR_MATH_APPROX");
        require(spotPriceAfter <= maxPrice, "ERR_LIMIT_PRICE");

        emit LOG_SWAP(
            msg.sender,
            tokenIn,
            tokenOut,
            tokenAmountIn,
            tokenAmountOut
        );

        _pullUnderlying(tokenIn, msg.sender, tokenAmountIn);

        uint256 _swapFee = tokenAmountIn.bmul(swapFee);

        // to referral
        uint256 referFee = 0;
        if (
            referrer != address(0) &&
            referrer != msg.sender &&
            referrer != tx.origin
        ) {
            referFee = _swapFee / 5; // 20% to referrer
```

```
            _pushUnderlying(tokenIn, referrer, referFee);
            inRecord.balance = (inRecord.balance).bsub(referFee);
            emit LOG_REFER(msg.sender, referrer, tokenIn, referFee);
        }

        // to SAFU
        uint256 _safuFee = tokenAmountIn.bmul(safuFee);
        if (isFarmPool) {
            _safuFee = _swapFee.bsub(referFee);
        }
        _pushUnderlying(tokenIn, SAFU, _safuFee);
        inRecord.balance = (inRecord.balance).bsub(_safuFee);

        _pushUnderlying(tokenOut, msg.sender, tokenAmountOut);
        return (tokenAmountOut, spotPriceAfter);
    }
```

- contracts/XPool.sol Line: 509-600

```
function swapExactAmountOutRefer(
        address tokenIn,
        uint256 maxAmountIn,
        address tokenOut,
        uint256 tokenAmountOut,
        uint256 maxPrice,
        address referrer
    ) public _lock_ returns (uint256 tokenAmountIn, uint256 spotPriceAfter) {
        require(_records[tokenIn].bound, "ERR_NOT_BOUND");
        require(_records[tokenOut].bound, "ERR_NOT_BOUND");
        require(finalized, "ERR_NOT_FINALIZED");

        Record storage inRecord = _records[address(tokenIn)];
        Record storage outRecord = _records[address(tokenOut)];

        require(
            tokenAmountOut <= (outRecord.balance).bmul(MAX_OUT_RATIO),
            "ERR_MAX_OUT_RATIO"
        );

        uint256 spotPriceBefore =
            XMath.calcSpotPrice(
                inRecord.balance,
```

```
                inRecord.denorm,
                outRecord.balance,
                outRecord.denorm,
                swapFee
            );
        require(spotPriceBefore <= maxPrice, "ERR_BAD_LIMIT_PRICE");

        tokenAmountIn = calcInGivenOut(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            tokenAmountOut,
            swapFee
        );
        require(tokenAmountIn <= maxAmountIn, "ERR_LIMIT_IN");
        require(
            spotPriceBefore <= tokenAmountIn.bdiv(tokenAmountOut),
            "ERR_MATH_APPROX"
        );

        inRecord.balance = (inRecord.balance).badd(tokenAmountIn);
        outRecord.balance = (outRecord.balance).bsub(tokenAmountOut);

        spotPriceAfter = XMath.calcSpotPrice(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            swapFee
        );
        require(spotPriceAfter >= spotPriceBefore, "ERR_MATH_APPROX");
        require(spotPriceAfter <= maxPrice, "ERR_LIMIT_PRICE");

        emit LOG_SWAP(
            msg.sender,
            tokenIn,
            tokenOut,
            tokenAmountIn,
            tokenAmountOut
        );
```

```
        _pullUnderlying(tokenIn, msg.sender, tokenAmountIn);

        uint256 _swapFee = tokenAmountIn.bmul(swapFee);
        // to referral
        uint256 referFee = 0;
        if (
            referrer != address(0) &&
            referrer != msg.sender &&
            referrer != tx.origin
        ) {
            referFee = _swapFee / 5; // 20% to referrer
            _pushUnderlying(tokenIn, referrer, referFee);
            inRecord.balance = (inRecord.balance).bsub(referFee);
            emit LOG_REFER(msg.sender, referrer, tokenIn, referFee);
        }

        // to SAFU
        uint256 _safuFee = tokenAmountIn.bmul(safuFee);
        if (isFarmPool) {
            _safuFee = _swapFee.bsub(referFee);
        }
        _pushUnderlying(tokenIn, SAFU, _safuFee);
        inRecord.balance = (inRecord.balance).bsub(_safuFee);

        _pushUnderlying(tokenOut, msg.sender, tokenAmountOut);
        return (tokenAmountIn, spotPriceAfter);
    }
```

Fix Status: The issues has been fixed in this commit:

561e6ca38e0b38854ae034050f43a66cae302f55

4.3.2 Medium-risk vulnerabilities

4.3.2.1 Gas Token attack risk

In the transferAll function and the sendValue function, call.value is used for eth transfer, and there is no restriction on the gas limit of the call.value. There is a risk of arbitrary invocation of external logic

and Gas Token attacks. It is recommended to limit the gas limit of call.value.

- contracts/XSwapProxyV1.sol Line: 530-542

```
function transferAll(IERC20 token, uint256 amount) internal returns (bool) {
        if (amount == 0) {
            return true;
        }
        if (address(token) == xconfig.ethAddress()) {
            weth.withdraw(amount);
            (bool xfer, ) = msg.sender.call.value(amount)("");
            require(xfer, "ERR_ETH_FAILED");
        } else {
            token.safeTransfer(msg.sender, amount);
        }
        return true;
    }
```

The sendValue function is not used in the actual business logic. The project party can evaluate whether to fix it according to the business situation or directly delete the redundant code.

- contracts/lib/Address.sol Line: 72-85

```
function sendValue(address payable recipient, uint256 amount) internal {
        require(
            address(this).balance >= amount,
            "Address: insufficient balance"
        );

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(
            success,
            "Address: unable to send value, recipient may have reverted"
        );
    }
}
```

Fix Status: The issues has been fixed in this commit:

561e6ca38e0b38854ae034050f43a66cae302f55

4.3.3 Low-risk vulnerabilities

4.3.3.1 Excessive authority issues

The project will use the Core role address corresponding to the onlyCore modifier to manage the project, including setCore, mint, addFarmPool, removeFarmPool, updateSafu, updateFarm, collect, setSAFU, setMaxExitFee, setSafuFee, setSwapProxy and other governance operations, which have excessive authority .The mint function does not have an upper limit for minting coins. After communication and feedback, because the project needs to be trial run at the beginning of the project, immediately transferring the permissions to the timelock contract will affect the time plan of the trial run. After the project trial run ends, the project party will transfer the permissions of the Core role to timelock contract.

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

4.3.4 Enhancement Suggestions

4.3.4.1 Same address check is missing

The functions in the contracts/XPool.sol contract do not judge the situation that the addresses of tokenIn and tokenOut are the same. This operation should be meaningless in business, but it is allowed in the code. It is recommended to restrict the tokenIn and tokenOut to the same This situation avoids unknown risks caused by meaningless operations.

Fix Status: Since this issues does not directly affect the safety of the project, it is an enhancement point, so it is temporarily ignored.

4.3.4.2 Missing event log

The setSwapProxy function in the XConfig contract does not add events for recording. Since this method can directly modify the address of the proxy contract, it is recommended to add events for recording for the convenience of the community to review.

- contracts/XConfig.sol Line: 146-150

```
function setSwapProxy(address _proxy) external onlyCore {
        require(_proxy != address(0), "ERR_ZERO_ADDR");
        swapProxy = _proxy;
    }
```

Fix Status: The issues has been fixed in this commit:

561e6ca38e0b38854ae034050f43a66cae302f55

4.3.4.3 Token compatibility issues

Since the XDeFi project forks from balancer, XDeFi also has the issues of incompatibility with deflationary and inflationary tokens. It is recommended to remind users to avoid adding such tokens for trading.

Reference: https://mp.weixin.qq.com/s/sESfNRLN66w2OnFjs_PMuA

Fix Status: This issue requires the project party to remind users to pay attention to the issue of token compatibility.

5. Audit Result

5.1 Conclusion

Audit Result : Low Risk

Audit Number : 0X002102060001

Audit Date : Feb. 06, 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, 1 high-risk, 1 medium-risk, 1 low-risk vulnerabilities and 3 enhancement suggestions were found during the audit, the high-risk, medium-risk vulnerabilities identified have been fixed, the owner authority has not been transferred to the timelock contract. There is an enhancement suggestion has been ignore.

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

Github
https://github.com/slowmist