Smart Contract Security Audit Report

Contents

1. Executive Summary

On March. 08, 2021, the SlowMist security team received the autofarm team's security audit application for autofarm project, 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

Audit version code:

https://github.com/autofarm-network/AutofarmV2_CrossChain/blob/56e3bde53c532a0faf114dceb51a23b526b33f8c/AutoFarmV2_CrossChain.sol

https://github.com/autofarm-network/AutofarmV2_CrossChain/blob/e8f460aad1dda0e840f5421818b5dbcaf68dfc33/StratX2_MDEX.sol

https://github.com/autofarm-network/AutofarmV2_CrossChain/blob/a22112979cb67aa61b5c2fd9da81705922e63d61/StratVLEV2.sol

Fixed version code:

https://github.com/autofarm-network/AutofarmV2_CrossChain/blob/b27f2cafcf51667726bd70220420707c89b75975/AutoFarmV2_CrossChain.sol

https://github.com/autofarm-network/AutofarmV2_CrossChain/blob/b27f2cafcf51667726bd70220420707c89b75975/StratX2_MDEX.sol

https://github.com/autofarm-network/AutofarmV2_CrossChain/blob/b27f2cafcf51667726bd7022

0420707c89b75975/StratVLEV2.sol

4. Code Overview

4.1 Contracts Description

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

StratX2_MDEX			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
noTimeLockFunc1	Public	Can Modify State	-
_convertMDXToEarned	Internal	Can Modify State	-

StratX2			
Function Name	Visibility	Mutability	Modifiers
deposit	Public	Can Modify State	onlyOwner nonReentrant whenNotPaused
farm	Public	Can Modify State	nonReentrant
_farm	Internal	Can Modify State	-
_unfarm	Internal	Can Modify State	-
withdraw	Public	Can Modify State	onlyOwner nonReentrant
earn	Public	Can Modify State	whenNotPaused
buyBack	Internal	Can Modify State	-
distributeFees	Internal	Can Modify State	-
convertDustToEarned	Public	Can Modify State	whenNotPaused
pause	Public	Can Modify State	-
unpause	Public	Can Modify State	-
setSettings	Public	Can Modify State	-
setGov	Public	Can Modify State	-
setOnlyGov	Public	Can Modify State	-
setUniRouterAddress	Public	Can Modify State	-
setBuyBackAddress	Public	Can Modify State	-

setRewardsAddress	Public	Can Modify State	-
inCaseTokensGetStuck	Public	Can Modify State	-
_wrapBNB	Internal	Can Modify State	-
wrapBNB	Public	Can Modify State	-
_safeSwap	Internal	Can Modify State	-

AutoFarmV2_CrossChain			
Function Name	Visibility	Mutability	Modifiers
poolLength	External	-	-
add	Public	Can Modify State	onlyOwner
stakedWantTokens	External	-	-
deposit	Public	Can Modify State	nonReentrant
withdraw	Public	Can Modify State	nonReentrant
inCaseTokensGetStuck	Public	Can Modify State	onlyOwner

StratVLEV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_supply	Internal	Can Modify State	-
_removeSupply	Internal	Can Modify State	-
_borrow	Internal	Can Modify State	-
_repayBorrow	Internal	Can Modify State	-
deposit	Public	Can Modify State	onlyOwner nonReentrant whenNotPaused
farm	Public	Can Modify State	nonReentrant
_farm	Internal	Can Modify State	-
_leverage	Internal	Can Modify State	-
deleverageOnce	Public	Can Modify State	-
_deleverageOnce	Internal	Can Modify State	-
deleverageUntilNotOverLevered	Public	Can Modify State	-
_deleverage	Internal	Can Modify State	-
rebalance	External	Can Modify State	-
earn	External	Can Modify State	whenNotPaused
buyBack	Internal	Can Modify State	-
distributeFees	Internal	Can Modify State	-
withdraw	External	Can Modify State	onlyOwner nonReentrant
pause	Public	Can Modify State	-
unpause	External	Can Modify State	-

_resetAllowances	Internal	Can Modify State	-
resetAllowances	Public	Can Modify State	-
updateBalance	Public	Can Modify State	-
wantLockedTotal	Public	-	-
wantLockedInHere	Public	-	-
setSettings	Public	Can Modify State	-
setGov	Public	Can Modify State	-
setOnlyGov	Public	Can Modify State	-
setUniRouterAddress	Public	Can Modify State	-
setBuyBackAddress	Public	Can Modify State	-
setRewardsAddress	Public	Can Modify State	-
inCaseTokensGetStuck	Public	Can Modify State	-
_wrapBNB	Internal	Can Modify State	-
_unwrapBNB	Internal	Can Modify State	-
wrapBNB	Public	Can Modify State	-
_safeSwap	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-

4.2 Code Audit

4.2.1 High-risk vulnerabilities

4.2.1.1 Sandwich attacks issues

The earn(), buyBack(), convertDustToEarned(), _convertMDXToEarned functions no limit slippage, there is a sandwich attacks issues. It is recommended to add a slippage limit, and the slippage parameter can only be modified by the Owner.

- AutofarmV2_CrossChain/StratVLEV2.sol

```
function earn() external whenNotPaused {
        if (onlyGov) {
            require(msg.sender == govAddress, "!gov");
        }
```

```
        IVenusDistribution(venusDistributionAddress).claimVenus(address(this));

        uint256 earnedAmt = IERC20(venusAddress).balanceOf(address(this));

        earnedAmt = distributeFees(earnedAmt);
        earnedAmt = buyBack(earnedAmt);

        if (venusAddress != wantAddress) {
            IPancakeRouter02(uniRouterAddress).swapExactTokensForTokens(
                earnedAmt,
                0,
                venusToWantPath,
                address(this),
                now.add(600)
            );
        }
        lastEarnBlock = block.number;

        _farm(false); // Supply wantToken without leverage, to cater for net -ve interest rates.
    }
```

- AutofarmV2_CrossChain/StratVLEV2.sol

```
function buyBack(uint256 _earnedAmt) internal returns (uint256) {
        if (buyBackRate <= 0) {
            return _earnedAmt;
        }

        uint256 buyBackAmt = _earnedAmt.mul(buyBackRate).div(buyBackRateMax);

        IPancakeRouter02(uniRouterAddress).swapExactTokensForTokens(
            buyBackAmt,
            0,
            earnedToAUTOPath,
            buyBackAddress,
            now + 600
        );

        return _earnedAmt.sub(buyBackAmt);
```

```
    }
```

- AutofarmV2_CrossChain/StratX2_MDEX.sol

```
function convertDustToEarned() public whenNotPaused {
        require(isAutoComp, "!isAutoComp");
        require(!isCAKEStaking, "isCAKEStaking");

        // Converts dust tokens into earned tokens, which will be reinvested on the next earn().

        // Converts token0 dust (if any) to earned tokens
        uint256 token0Amt = IERC20(token0Address).balanceOf(address(this));
        if (token0Address != earnedAddress && token0Amt > 0) {
            IERC20(token0Address).safeIncreaseAllowance(
                uniRouterAddress,
                token0Amt
            );

            // Swap all dust tokens to earned tokens
            IPancakeRouter02(uniRouterAddress)
                .swapExactTokensForTokensSupportingFeeOnTransferTokens(
                token0Amt,
                0,
                token0ToEarnedPath,
                address(this),
                now + 600
            );
        }

        // Converts token1 dust (if any) to earned tokens
        uint256 token1Amt = IERC20(token1Address).balanceOf(address(this));
        if (token1Address != earnedAddress && token1Amt > 0) {
            IERC20(token1Address).safeIncreaseAllowance(
                uniRouterAddress,
                token1Amt
            );

            // Swap all dust tokens to earned tokens
            IPancakeRouter02(uniRouterAddress)
                .swapExactTokensForTokensSupportingFeeOnTransferTokens(
```

```
                token1Amt,
                0,
                token1ToEarnedPath,
                address(this),
                now + 600
            );
        }
    }
```

- AutofarmV2_CrossChain/StratX2_MDEX.sol

```
function _convertMDXToEarned() internal {
        // Converts MDX (if any) to earned tokens
        uint256 MDXAmt = IERC20(MDXAddress).balanceOf(address(this));
        if (MDXAddress != earnedAddress && MDXAmt > 0) {
            IERC20(MDXAddress).safeIncreaseAllowance(uniRouterAddress, MDXAmt);

            // Swap all dust tokens to earned tokens
            IPancakeRouter02(uniRouterAddress)
                .swapExactTokensForTokensSupportingFeeOnTransferTokens(
                MDXAmt,
                0,
                MDXToEarnedPath,
                address(this),
                now + 60
            );
        }
    }
```

- AutofarmV2_CrossChain/StratX2_MDEX.sol

```
function buyBack(uint256 _earnedAmt) internal returns (uint256) {
        if (buyBackRate <= 0) {
            return _earnedAmt;
        }

        uint256 buyBackAmt = _earnedAmt.mul(buyBackRate).div(buyBackRateMax);

        if (earnedAddress == AUTOAddress) {
            IERC20(earnedAddress).safeTransfer(buyBackAddress, buyBackAmt);
        } else {
            IERC20(earnedAddress).safeIncreaseAllowance(
                uniRouterAddress,
```

```
                buyBackAmt
            );

            IPancakeRouter02(uniRouterAddress)
                .swapExactTokensForTokensSupportingFeeOnTransferTokens(
                buyBackAmt,
                0,
                earnedToAUTOPath,
                buyBackAddress,
                now + 600
            );
        }

        return _earnedAmt.sub(buyBackAmt);
    }
```

- AutofarmV2_CrossChain/StratVLEV2.sol

```
function earn() external whenNotPaused {
        if (onlyGov) {
            require(msg.sender == govAddress, "!gov");
        }

        IVenusDistribution(venusDistributionAddress).claimVenus(address(this));

        uint256 earnedAmt = IERC20(venusAddress).balanceOf(address(this));

        earnedAmt = distributeFees(earnedAmt);
        earnedAmt = buyBack(earnedAmt);

        if (venusAddress != wantAddress) {
            IPancakeRouter02(uniRouterAddress).swapExactTokensForTokens(
                earnedAmt,
                0,
                venusToWantPath,
                address(this),
                now.add(600)
            );
        }

        lastEarnBlock = block.number;
```

```
        _farm(false); // Supply wantToken without leverage, to cater for net -ve interest rates.
    }
```

Fix Status:

The issue has been fixed in this commit: b27f2cafcf51667726bd70220420707c89b75975

4.2.2 Medium-risk vulnerabilities

4.2.2.1 Excessive authority issues

The add function has excessive authority issues, Owner can add mining pool arbitrarily, there is a risk of stealing mining by himself, and _strat is the destination address of the final sending of funds, the owner can set this address arbitrarily, pay attention to compatibility issues with external contracts, it is recommended to set the ownership to the timelock contract, and add events to record in the add function.

- AutofarmV2_CrossChain/AutoFarmV2_CrossChain.sol

```
function add(
        uint256 _allocPoint,
        IERC20 _want,
        bool _withUpdate,
        address _strat
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }

        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(
            PoolInfo({
                want: _want,
                allocPoint: _allocPoint,
                lastRewardBlock: 0,
                accAUTOPerShare: 0,
```

```
                strat: _strat
            })
        );
    }
```

Fix Status: The ownership has been transfer to timelock contract.

Reference：

https://hecoinfo.com/tx/0xbe90b5dba8315b30a010ea957e9631154857b93d84cdb344c11b339b5f3e5421

The authority of the Gov role is large, and the address of the external contract can be set arbitrarily. Malicious and wrong external contracts will cause the user's funds to be lost, and there is a issues of excessive authority，it is recommended to set the gov to the timelock contract.

- AutofarmV2_CrossChain/StratX2.sol

```
function setEntranceFeeFactor(uint256 _entranceFeeFactor) public {
        require(msg.sender == govAddress, "!gov");
        require(_entranceFeeFactor >= entranceFeeFactorLL, "!safe - too low");
        require(_entranceFeeFactor <= entranceFeeFactorMax, "!safe - too high");
        entranceFeeFactor = _entranceFeeFactor;
    }

    function setWithdrawFeeFactor(uint256 _withdrawFeeFactor) public {
        require(msg.sender == govAddress, "!gov");
        require(_withdrawFeeFactor >= withdrawFeeFactorLL, "!safe - too low");
        require(_withdrawFeeFactor <= withdrawFeeFactorMax, "!safe - too high");
        withdrawFeeFactor = _withdrawFeeFactor;
    }

    function setControllerFee(uint256 _controllerFee) public {
        require(msg.sender == govAddress, "!gov");
        require(_controllerFee <= controllerFeeUL, "too high");
        controllerFee = _controllerFee;
    }

    function setbuyBackRate(uint256 _buyBackRate) public {
        require(msg.sender == govAddress, "!gov");
```

```
        require(buyBackRate <= buyBackRateUL, "too high");
        buyBackRate = _buyBackRate;
    }

    function setGov(address _govAddress) public {
        require(msg.sender == govAddress, "!gov");
        govAddress = _govAddress;
    }

    function setOnlyGov(bool _onlyGov) public {
        require(msg.sender == govAddress, "!gov");
        onlyGov = _onlyGov;
    }

    function setUniRouterAddress(address _uniRouterAddress) public {
        require(msg.sender == govAddress, "!gov");
        uniRouterAddress = _uniRouterAddress;
    }

    function setBuyBackAddress(address _buyBackAddress) public {
        require(msg.sender == govAddress, "!gov");
        buyBackAddress = _buyBackAddress;
    }

    function setRewardsAddress(address _rewardsAddress) public {
        require(msg.sender == govAddress, "!gov");
        rewardsAddress = _rewardsAddress;
    }
```

Fix Status: The Gov has been transfer to timelock contract.

Reference：

https://hecoinfo.com/tx/0xfdf183915b5659473f9e8e3438c295cb859e022faa073a0a8f12c38e0a4c257d

4.2.2.2 DoS issues

In the massUpdatePools function, if the length of poolInfo is too large, there is a risk of DoS. It is

recommended to limit poolInfo.length to avoid DoS caused by too large length.

- AutofarmV2_CrossChain/AutoFarmV2_CrossChain.sol

```
function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }
```

Fix Status:

The issue has been fixed in this commit: b27f2cafcf51667726bd70220420707c89b75975

4.2.3 Enhancement Suggestions

4.2.3.1 Missing event log

"setEntranceFeeFactor" function, "setWithdrawFeeFactor" function, "setControllerFee" function, "setbuyBackRate" function, "setGov" function, "setOnlyGov" function, "setUniRouterAddress" function, "setBuyBackAddress" function, "setRewardsAddress" function, no events are added to record. It is recommended to add events for recording.

- AutofarmV2_CrossChain/StratX2_MDEX.sol
- AutofarmV2_CrossChain/StratVLEV2.sol

```
    function setEntranceFeeFactor(uint256 _entranceFeeFactor) public {
        require(msg.sender == govAddress, "!gov");
        require(_entranceFeeFactor >= entranceFeeFactorLL, "!safe - too low");
        require(_entranceFeeFactor <= entranceFeeFactorMax, "!safe - too high");
        entranceFeeFactor = _entranceFeeFactor;
    }

    function setWithdrawFeeFactor(uint256 _withdrawFeeFactor) public {
        require(msg.sender == govAddress, "!gov");
        require(_withdrawFeeFactor >= withdrawFeeFactorLL, "!safe - too low");
```

```
        require(_withdrawFeeFactor <= withdrawFeeFactorMax, "!safe - too high");
        withdrawFeeFactor = _withdrawFeeFactor;
    }

    function setControllerFee(uint256 _controllerFee) public {
        require(msg.sender == govAddress, "!gov");
        require(_controllerFee <= controllerFeeUL, "too high");
        controllerFee = _controllerFee;
    }

    function setbuyBackRate(uint256 _buyBackRate) public {
        require(msg.sender == govAddress, "!gov");
        require(buyBackRate <= buyBackRateUL, "too high");
        buyBackRate = _buyBackRate;
    }

    function setGov(address _govAddress) public {
        require(msg.sender == govAddress, "!gov");
        govAddress = _govAddress;
    }

    function setOnlyGov(bool _onlyGov) public {
        require(msg.sender == govAddress, "!gov");
        onlyGov = _onlyGov;
    }

    function setUniRouterAddress(address _uniRouterAddress) public {
        require(msg.sender == govAddress, "!gov");
        uniRouterAddress = _uniRouterAddress;
    }

    function setBuyBackAddress(address _buyBackAddress) public {
        require(msg.sender == govAddress, "!gov");
        buyBackAddress = _buyBackAddress;
    }

    function setRewardsAddress(address _rewardsAddress) public {
        require(msg.sender == govAddress, "!gov");
        rewardsAddress = _rewardsAddress;
    }
```

Fix Status:

The issue has been fixed in this commit: b27f2cafcf51667726bd70220420707c89b75975

4.2.3.2 Missing nonReentrant modifier

The deposit function missing the nonReentrant modifier, it is recommended to add the nonReentrant modifier.

- AutofarmV2_CrossChain/StratX2_MDEX.sol

```
function deposit(address _userAddress, uint256 _wantAmt)
        public
        onlyOwner
        whenNotPaused
        returns (uint256)
    {
        IERC20(wantAddress).safeTransferFrom(
            address(msg.sender),
            address(this),
            _wantAmt
        );

        uint256 sharesAdded = _wantAmt;
        if (wantLockedTotal > 0 && sharesTotal > 0) {
            sharesAdded = _wantAmt
                .mul(sharesTotal)
                .mul(entranceFeeFactor)
                .div(wantLockedTotal)
                .div(entranceFeeFactorMax);
        }
        sharesTotal = sharesTotal.add(sharesAdded);

        if (isAutoComp) {
            _farm();
        } else {
            wantLockedTotal = wantLockedTotal.add(_wantAmt);
        }

        return sharesAdded;
    }
```

Fix Status:

The issue has been fixed in this commit: b27f2cafcf51667726bd70220420707c89b75975

5. Audit Result

5.1 Conclusion

Audit Result : Passed

Audit Number : 0X002103170002

Audit Date : March. 17, 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 vulnerabilities, 2 medium-risk vulnerabilities and 2 enhancement suggestions were found during the audit, the owner and gov authority has 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.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist