Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2021.08.19, the SlowMist security team received the WaultFinance team's security audit application for WUSD, 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 party 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:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

The audit version:

https://github.com/WaultFinance/WUSD/tree/91c541c2f1c0ac781ddcfb2be6a62555a5e1e8d1

The fixed version:

https://github.com/WaultFinance/WUSD/tree/5f50a2c7ffff7828c70299e8a9217cfbb926b8c1

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Transaction reordering issues	Reordering Vulnerability	Low	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:

WexWithdrawer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
withdraw	External	Can Modify State	onlyOwner
deposit	External	Can Modify State	onlyOwner
initiateMasterChange	External	Can Modify State	onlyOwner
cancelMasterChange	External	Can Modify State	onlyOwner
changeMaster	External	Can Modify State	onlyOwner

Context			
Function Name	Visibility	Mutability	Modifiers
_msgSender	Internal	-	-
_msgData	Internal	-	-

Ownable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
owner	Public	-	-
renounceOwnership	Public	Can Modify State	onlyOwner
transferOwnership	Public	Can Modify State	onlyOwner
_setOwner	Private	Can Modify State	-

Mintable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
minter	Public	-	-
transferMintership	Public	Can Modify State	onlyOwner

ERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
totalSupply	Public	-	-
balanceOf	Public	-	-

ERC20			
transfer	Public	Can Modify State	-
allowance	Public	-	-
approve	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
increaseAllowance	Public	Can Modify State	-
decreaseAllowance	Public	Can Modify State	-
_transfer	Internal	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Internal	Can Modify State	-
_beforeTokenTransfer	Internal	Can Modify State	-
_afterTokenTransfer	Internal	Can Modify State	-

WUSD			
Function Name	Visibility	Mutability	Modifiers
mint	External	Can Modify State	onlyMinter
burn	External	Can Modify State	onlyMinter

WUSDMaster			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

WUSDMaster			
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
setSwapPath	External	Can Modify State	onlyOwner
setWexPermille	External	Can Modify State	onlyOwner
setTreasuryPermille	External	Can Modify State	onlyOwner
setFeePermille	External	Can Modify State	onlyOwner
setTreasuryAddress	External	Can Modify State	onlyOwner
setStrategistAddress	External	Can Modify State	onlyOwner
setMaxStakeAmount	External	Can Modify State	onlyOwner
setMaxRedeemAmount	External	Can Modify State	onlyOwner
setMaxStakePerBlock	External	Can Modify State	onlyOwner
stake	External	Can Modify State	nonReentrant whenNotPaused
claimWusd	External	Can Modify State	nonReentrant whenNotPaused
redeem	External	Can Modify State	nonReentrant whenNotPaused
claimUsdt	External	Can Modify State	nonReentrant whenNotPaused
emergencyRedeemAll	External	Can Modify State	nonReentrant whenPaused
emergencyClaimUsdtAll	External	Can Modify State	nonReentrant whenPaused
withdrawUsdt	External	Can Modify State	onlyOwner
withdrawWex	External	Can Modify State	onlyWithdrawer

4.3 Vulnerability Summary

[N1] [Low] Transaction reordering issues

Category: Reordering Vulnerability

Content

(1) In commit: 91c541c2f1c0ac781ddcfb2be6a62555a5e1e8d1, the

swapExactTokensForTokensSupportingFeeOnTransferTokens in the stake function is not checked for slippage.

- https://github.com/WaultFinance/WUSD/blob/91c541c2f/WUSDMaster.sol#L716-L722

```
function stake(uint256 amount) external nonReentrant {
        require(amount > 0, 'amount cant be zero');
        require(wusdClaimAmount[msg.sender] == 0, 'you have to claim first');
        require(amount <= maxStakeAmount, 'amount too high');

        usdt.safeTransferFrom(msg.sender, address(this), amount);
        if(feePermille > 0) {
            uint256 feeAmount = amount * feePermille / 1000;
            usdt.safeTransfer(treasury, feeAmount);
            amount = amount - feeAmount;
        }
        uint256 wexAmount = amount * wexPermille / 1000;
        usdt.approve(address(wswapRouter), wexAmount);
        wswapRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            wexAmount,
            0,
            swapPath,
            address(this),
            block.timestamp
        );

        wusdClaimAmount[msg.sender] = amount;
        wusdClaimBlock[msg.sender] = block.number;

        emit Stake(msg.sender, amount);
    }
```

(2) In commit: de61d93cd7a35213484827cf32533919c34e732e amountOutMin is the parameter that limits the slippage, but it is entered by the user, the maxStakeAmount is added, but this limit can still be bypassed by sorting multiple transactions.

- https://github.com/WaultFinance/WUSD/blob/de61d93cd7a35213484827cf32533919c34e732e/WUSDMaster.sol#L808-L834

```
    function stake(uint256 amount, uint256 amountOutMin) external nonReentrant
whenNotPaused {
        require(amount > 0, 'amount cant be zero');
        require(wusdClaimAmount[msg.sender] == 0, 'you have to claim first');
        require(amount <= maxStakeAmount, 'amount too high');

        usdt.safeTransferFrom(msg.sender, address(this), amount);
        if(feePermille > 0) {
            uint256 feeAmount = amount * feePermille / 1000;
            usdt.safeTransfer(treasury, feeAmount);
            amount = amount - feeAmount;
        }
        wusd.mint(address(this), amount);
        uint256 wexAmount = amount * wexPermille / 1000;
        usdt.approve(address(wswapRouter), wexAmount);
        wswapRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            wexAmount,
            amountOutMin,
            swapPath,
            address(this),
            block.timestamp
        );

        wusdClaimAmount[msg.sender] = amount;
        wusdClaimBlock[msg.sender] = block.number;

        emit Stake(msg.sender, amount);
    }
```

(3) In commit: 5f50a2c7ffff7828c70299e8a9217cfbb926b8c1, the maxStakePerBlock is added, but this limit can still be bypassed by sorting multiple transactions in multiple blocks.

- https://github.com/WaultFinance/WUSD/blob/5f50a2c7ffff7828c70299e8a9217cfbb926b8c1/WUSDMaster.sol#L819-L851

```
function stake(uint256 amount, uint256 amountOutMin) external nonReentrant
whenNotPaused {
        require(amount > 0, 'amount cant be zero');
        require(wusdClaimAmount[msg.sender] == 0, 'you have to claim first');
        require(amount <= maxStakeAmount, 'amount too high');
        if(lastBlock != block.number) {
            lastBlockUsdtStaked = 0;
            lastBlock = block.number;
        }
        lastBlockUsdtStaked += amount;
        require(lastBlockUsdtStaked <= maxStakePerBlock, 'maximum stake per block
exceeded');

        usdt.safeTransferFrom(msg.sender, address(this), amount);
        if(feePermille > 0) {
            uint256 feeAmount = amount * feePermille / 1000;
            usdt.safeTransfer(treasury, feeAmount);
            amount = amount - feeAmount;
        }
        wusd.mint(address(this), amount);
        uint256 wexAmount = amount * wexPermille / 1000;
        usdt.approve(address(wswapRouter), wexAmount);
        wswapRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            wexAmount,
            amountOutMin,
            swapPath,
            address(this),
            block.timestamp
        );

        wusdClaimAmount[msg.sender] = amount;
        wusdClaimBlock[msg.sender] = block.number;

        emit Stake(msg.sender, amount);
    }
```

Solution

This is a big fund attack, the following processing is recommended:

1. It is recommended to use require(tx.origin == msg.sender) to determine the EOA address.(but after EIP-3074 takes effect, there will be a security issue.)
2. It is recommended to limit the amountOutMin in swapExactTokensForTokensSupportingFeeOnTransferTokens, and it cannot be controlled by the user, check slippage with point 3.
3. It is recommended to use the usdt_wex delayed price to calculate the swap quantity before swapExactTokensForTokensSupportingFeeOnTransferTokens, and then check the slippage.

Status

Confirmed; It is a low-risk issue that is difficult to exploit and profit.

WaultFinance team response:

With the introduction of max mint per block team can control the issue.

For example if liquidity drops or arbitrage is possible we can reduce maxmintperblock to 100k wusd.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0x002108230001	SlowMist Security Team	2021.08.19 - 2021.08.23	Low Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found a low-risk vulnerability. The issue has been confirmed. 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.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist