Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.12.14, the SlowMist security team received the Vee Finance team's security audit application for Vee Finance Iterative Audit, 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 team 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 Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

This audit is only for `commit 264b792e1c531dfa53233091f73bdda6d8fcc0e8` and does not include subsequent iterations.

Audit Version:

https://github.com/VeeFinance/vee-finance-protocol-v3.0

commit: 264b792e1c531dfa53233091f73bdda6d8fcc0e8

Fixed Version:

https://github.com/VeeFinance/vee-finance-protocol-v3.0

commit: a05c7bf218ad0f2514d6ff7b738a6a8ab90c7706

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability	Medium	Fixed
N2	`initialized` issue	Others	Medium	Fixed
N3	Compatibility issue	Others	Suggestion	Fixed
N4	Comptroller forgery risk	Design Logic Audit	Critical	Fixed
N5	Receiving extra reward issue	Design Logic Audit	Low	Fixed

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:

PangolinERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-

PangolinERC20			
_burn	Internal	Can Modify State	-
_approve	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
approve	External	Can Modify State	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

StakeBank			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setVoteToken	External	Can Modify State	onlyOwner
setNewRewardsAPR	External	Can Modify State	onlyOwner
_setNewRewardsAPR	Internal	Can Modify State	-
getHistoryAPR	External	-	-
refreshUserTotal	External	Can Modify State	-
stake	External	Can Modify State	-
withdraw	External	Can Modify State	-
estimateRewards	External	-	-
_estimateRewards	Internal	-	-

StakeBank			
_distrubuteRewards	Internal	Can Modify State	-
_interestFactor	Internal	-	-
_matchStakeAPR	Internal	-	-
userStakePoolSize	External	-	-
userTotalStaked	External	-	-

VeeLPFarm			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
updateMultiplier	Public	Can Modify State	onlyOwner
add	Public	Can Modify State	onlyOwner
set	Public	Can Modify State	onlyOwner
updateStakingPool	Internal	Can Modify State	-
getMultiplier	Internal	-	-
pendingRewards	External	-	-
updateAllPools	External	Can Modify State	-
_updateAllPools	Internal	Can Modify State	-
updatePool	Internal	Can Modify State	-
deposit	External	Can Modify State	-
claimVee	External	Can Modify State	nonReentrant

VeeLPFarm			
depositBehalf	External	Can Modify State	-
withdraw	External	Can Modify State	-
withdrawDuplex	External	Can Modify State	-
emergencyWithdraw	External	Can Modify State	-
safeRewardsTransfer	Internal	Can Modify State	-
getPoolSize	External	-	-
setVeeHub	External	Can Modify State	onlyOwner
setRewardsPerBlock	External	Can Modify State	onlyOwner
_stakeToDex	Internal	Can Modify State	-
_withdrawFromDex	Internal	Can Modify State	-
upgradePatch	External	Can Modify State	onlyOwner

VoteToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
approve	External	Can Modify State	-

VoteToken			
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-
adjustSpeed	External	Can Modify State	-
_increaseFrozen	Internal	Can Modify State	-
_decreaseFrozen	Internal	Can Modify State	-
setMinter	External	Can Modify State	onlyOwner
_setMinter	Internal	Can Modify State	-
mint	External	Can Modify State	-
burn	External	Can Modify State	-
setRewardPool	External	Can Modify State	onlyOwner

RewardPool			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
_setVeeHub	External	Can Modify State	onlyOwner
claimVee	Public	Can Modify State	-
claimVee	Public	Can Modify State	-
setVeeSpeedInternal	Internal	Can Modify State	-
updateVeeSupplyIndex	External	Can Modify State	-

RewardPool			
_updateVeeSupplyIndex	Internal	Can Modify State	-
updateVeeBorrowIndex	External	Can Modify State	-
_updateVeeBorrowIndex	Internal	Can Modify State	-
distributeSupplierVee	External	Can Modify State	-
_distributeSupplierVee	Internal	Can Modify State	-
distributeBorrowerVee	External	Can Modify State	-
_distributeBorrowerVee	Internal	Can Modify State	-
grantVeeInternal	Internal	Can Modify State	-
getBlockTimestamp	Public	-	-
getVeeAddress	Public	-	-
_setVeeSpeed	External	Can Modify State	onlyOwner
setVoteToken	External	Can Modify State	onlyOwner
setVeeAccSpeed	External	Can Modify State	-
accSpeedIncrease	Internal	Can Modify State	-
accSpeedDecrease	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the VoteToken contract, the owner can change the minterAllowed status of any account through the setMinter function, and the account whose minterAllowed is true can perform mint and burn operations at will. This will lead to the risk of excessive owner permissions.

Code location: contracts/periphery/VoteToken.sol

```
function setMinter(address account, bool isMinter) external onlyOwner {
    _setMinter(account, isMinter);
}

function _setMinter(address account, bool isMinter) internal {
    bool oldIsMinter = minterAllowed[account];
    require(oldIsMinter != isMinter, "not change");
    minterAllowed[account] = isMinter;
    emit SetMinter(account, isMinter);
}

function mint(address to, uint value) external {
    require(minterAllowed[msg.sender], "minter not allowed");
    _mint(to, value);
}

function burn(address from, uint value) external {
    require(minterAllowed[msg.sender], "burner not allowed");
    _burn(from, value);
}
```

Solution

It is recommended that the ownership of the owner be transferred to community governance.

Status

Fixed; Fixed in commit a05c7bf218ad0f2514d6ff7b738a6a8ab90c7706

[N2] [Medium] `initialized` issue

Category: Others

Content

In the StakeBank contract, when `userAssets.initialized` is false, the user can update `userAssets.balance` through the refreshUserTotal function, but in the end, `userAssets.initialized` is not set to true.

Code location: contracts/periphery/StakeBank.sol

```
    function refreshUserTotal(address account) external returns(bool) {
        UserAssets storage userAssets = userTotals[account];
        if (userAssets.initialized == false) {
            uint totalBalance;
            for (uint i = 0; i < stakePools[account].length; i++) {
                totalBalance += stakePools[account][i].userBalance;
            }
            userAssets.balance = totalBalance;
        }

    }
```

Solution

It is recommended to set `userAssets.initialized` to true after completing the status update.

Status

Fixed; Fixed in commit 1106af3ba5b110e12c6146faf41aae26991c58ff

[N3] [Suggestion] Compatibility issue

Category: Others

Content

In the StakeBank contract, any user can perform a stake operation. It will first transfer the user-specified amount of stakeToken into the StakeBank contract through the safeTransferFrom function and use the userBalance parameter to record the amount passed in by the user. If stakeToken is a deflationary token, then the actual token received by the contract will not match the number recorded.

Code location: contracts/periphery/StakeBank.sol

```
    function stake(uint pid, uint amount) external {
        if (stakePools[msg.sender].length == 0) {
            for (uint i = 0; i < USER_POOL_SIZE; i++) {
                stakePools[msg.sender].push();
            }
        }
        UserStakeStatement storage statement = stakePools[msg.sender][pid];
        require(statement.userBalance == 0, "already staked");
        TransferHelper.safeTransferFrom(stakeToken, msg.sender, address(this),
amount);
        // solhint-disable-next-line not-rely-on-time
        statement.userStakeTime = block.timestamp;
        statement.userBalance = amount;
        statement.aprIndex = latestAPRUpdateTime;
        totalStake += amount;
        emit Stake(msg.sender, amount);
        IVoteToken(voteToken).mint(msg.sender, amount);
    }
```

Solution

It is recommended to record the difference between the user's contract balance before and after the transfer.

Status

Fixed; Fixed in commit 1106af3ba5b110e12c6146faf41aae26991c58ff

[N4] [Critical] Comptroller forgery risk

Category: Design Logic Audit

Content

In the RewardPool contract, the user can obtain rewards through the claimVee function, but it does not check the validComptroller status of the comptroller address passed in by the user. If the user passes in fake comptroller and cToken, it will lead to the risk of receiving the reward maliciously.

Code location: contracts/compound/RewardPool.sol

```
    function claimVee(address comptroller, address[] memory holders, CToken[] memory
cTokens, bool borrowers, bool suppliers) public {
```

```
        for (uint i = 0; i < cTokens.length; i++) {
            CToken cToken = cTokens[i];
            // require(markets[address(cToken)].isListed, "market must be listed");
            (bool isListed, , ) =
ComptrollerInterfaceLite(comptroller).markets(address(cToken));
            if(!isListed){
                continue;
            }
            if (borrowers) {
                // Exp memory borrowIndex = Exp({mantissa: cToken.borrowIndex()});
                // updateVeeBorrowIndex(address(cToken), borrowIndex);
                uint exchangeRate = CToken(cToken).exchangeRateStored();
                uint borrowIndex = cToken.borrowIndex();
                _updateVeeBorrowIndex(address(cToken), borrowIndex, exchangeRate);
                for (uint j = 0; j < holders.length; j++) {
                    _distributeBorrowerVee(address(cToken), holders[j], borrowIndex,
0, 0, exchangeRate);
                }
            }
            if (suppliers) {
                _updateVeeSupplyIndex(address(cToken));
                for (uint j = 0; j < holders.length; j++) {
                    _distributeSupplierVee(address(cToken), holders[j], 0, 0);
                }
            }
        }
        for (uint j = 0; j < holders.length; j++) {
            veeAccrued[holders[j]] = grantVeeInternal(holders[j],
veeAccrued[holders[j]]);
        }
    }
```

Solution

Should check whether the parameters passed in by the user are expected.

Status

Fixed; Fixed in commit 1106af3ba5b110e12c6146faf41aae26991c58ff

[N5] [Low] Receiving extra reward issue

Category: Design Logic Audit

Content

In the RewardPool contract, the user of the setVeeSpeedInternal function sets the vee distribution rate. When a cToken is removed, its veeSpeed will be set to 0, and then supplyState.timestamp will not be updated. If the cToken is put on the shelf again, veeSpeed will be set to non-zero. At this time, _distributeSupplierVee after the _updateVeeSupplyIndex operation will cause the time elapsed during the cToken's delisting period to be included in the reward.

Code location: contracts/compound/RewardPool.sol

```
    function setVeeSpeedInternal(CToken cToken, uint veeSpeed) internal {
        uint currentVeeSpeed = veeSpeeds[address(cToken)];
        if (currentVeeSpeed != 0) {
            // note that VEE speed could be set to 0 to halt liquidity rewards for a
market
            // Exp memory borrowIndex = Exp({mantissa: cToken.borrowIndex()});
            uint exchangeRate = CToken(cToken).exchangeRateStored();
            _updateVeeSupplyIndex(address(cToken));
            // updateVeeBorrowIndex(address(cToken), borrowIndex);
            _updateVeeBorrowIndex(address(cToken), cToken.borrowIndex(),
exchangeRate);
        } else if (veeSpeed != 0) {
            // Add the VEE market
            // Market storage market = markets[address(cToken)];
            // require(market.isListed, "vee market is not listed");
            bool isListed;
            for(uint i = 0; i < comptrollers.length; i++) {
                (isListed, , ) =
ComptrollerInterfaceLite(comptrollers[i]).markets(address(cToken));
                if (isListed) {
                    break;
                }
            }
            require(isListed, "market is not listed");

            if (veeSupplyState[address(cToken)].index == 0 &&
veeSupplyState[address(cToken)].timestamp == 0) {
                veeSupplyState[address(cToken)] = VeeMarketState({
```

```
                    index: veeInitialIndex,
                    timestamp: safe32(getBlockTimestamp(), "block timestamp exceeds
32 bits")
                });
            }

            if (veeBorrowState[address(cToken)].index == 0 &&
veeBorrowState[address(cToken)].timestamp == 0) {
                veeBorrowState[address(cToken)] = VeeMarketState({
                    index: veeInitialIndex,
                    timestamp: safe32(getBlockTimestamp(), "block timestamp exceeds
32 bits")
                });
            }
        }

        if (currentVeeSpeed != veeSpeed) {
            veeSpeeds[address(cToken)] = veeSpeed;
            emit VeeSpeedUpdated(cToken, veeSpeed);
        }
    }
```

Solution

It is recommended to check if cToken is listed and update the `Index` before updating veeSpeed.

Status

Fixed; Fixed in commit 1106af3ba5b110e12c6146faf41aae26991c58ff

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002112160002	SlowMist Security Team	2021.12.14 - 2021.12.16	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 2 medium risk, 1 low risk, 1 suggestion vulnerabilities. All the

findings were fixed. 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist