Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.08.09, the SlowMist security team received the Vee Finance team's security audit application for Vee Finance, 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

Audit version:

https://github.com/VeeFinance/vee-protocol-v3-audit

commit: cfcda7efd894e4e7ab97652d504bf6ac63825cb1

Fixed version:

https://github.com/VeeFinance/veefinance-protocol

commit: 6828b257473011afd16e97082343a199eb2fc5f1

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status

NO	Title	Category	Level	Status
N1	Token allowance issue	Design Logic Audit	Low	Confirmed
N2	Slippage check issue	Design Logic Audit	Critical	Confirmed
N3	cToken counterfeiting issue	Design Logic Audit	Suggestion	Ignored
N4	Price record manipulation issue	Others	Medium	Ignored
N5	Token transfer issue	Others	Suggestion	Fixed
N6	Malleable attack issue	Others	Suggestion	Fixed
N7	Relative price manipulation	Design Logic Audit	Medium	Ignored
N8	Bad debt issue	Design Logic Audit	Medium	Ignored
N9	Return value check issue	Design Logic Audit	Medium	Fixed
N10	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N11	Initialization issue	Design Logic Audit	Suggestion	Ignored
N12	Access control issue	Authority Control Vulnerability	Suggestion	Confirmed
N13	Add schedule issue	Design Logic Audit	Medium	Ignored
N14	Transfer issue	Others	Suggestion	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:

Vee			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allowance	External	-	-
approve	External	Can Modify State	-
balanceOf	External	-	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
delegate	Public	Can Modify State	-
delegateBySig	Public	Can Modify State	-
getCurrentVotes	External	-	-
getPriorVotes	Public	-	-
_delegate	Internal	Can Modify State	-
_transferTokens	Internal	Can Modify State	-
_moveDelegates	Internal	Can Modify State	-
_writeCheckpoint	Internal	Can Modify State	-

Vee			
safe32	Internal	-	-
safe96	Internal	-	-
add96	Internal	-	-
sub96	Internal	-	-
getChainId	Internal	-	-

VeeLens			
Function Name	Visibility	Mutability	Modifiers
cTokenMetadata	Public	Can Modify State	-
cTokenMetadataAll	External	Can Modify State	-
cTokenBalances	Public	Can Modify State	-
cTokenBalancesAll	External	Can Modify State	-
cTokenUnderlyingPrice	Public	Can Modify State	-
cTokenUnderlyingPriceAll	External	Can Modify State	-
getAccountLimits	Public	Can Modify State	-
getVeeBalanceMetadata	External	-	-
getVeeBalanceMetadataExt	External	Can Modify State	-
getVeeVotes	External	-	-
compareStrings	Internal	-	-
add	Internal	-	-

VeeLens			
sub	Internal	-	-

BaseJumpRateModelV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateJumpRateModel	External	Can Modify State	-
utilizationRate	Public	-	-
getBorrowRateInternal	Internal	-	-
getSupplyRate	External	-	-
updateJumpRateModelInternal	Internal	Can Modify State	-

CErc20			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	-
mint	External	Can Modify State	-
redeem	External	Can Modify State	-
redeemUnderlying	External	Can Modify State	-
borrow	External	Can Modify State	-
repayBorrow	External	Can Modify State	-
repayBorrowBehalf	External	Can Modify State	-
liquidateBorrow	External	Can Modify State	-

CErc20			
sweepToken	External	Can Modify State	-
_addReserves	External	Can Modify State	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can Modify State	-
doTransferOut	Internal	Can Modify State	-
doDeposit	Internal	Can Modify State	-
borrowAndCall	External	Payable	-
repayLeverageAndBorrow	External	Can Modify State	-
mintBehalf	External	Can Modify State	-
requireNoError	Internal	-	-

CEther			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mint	External	Payable	-
redeem	External	Can Modify State	-
redeemUnderlying	External	Can Modify State	-
borrow	External	Can Modify State	-
repayBorrow	External	Payable	-
repayBorrowBehalf	External	Payable	-

CEther			
liquidateBorrow	External	Payable	-
<Fallback>	External	Payable	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can Modify State	-
doTransferOut	Internal	Can Modify State	-
requireNoError	Internal	-	-
doDeposit	Internal	Can Modify State	-
borrowAndCall	External	Payable	-
repayLeverageAndBorrow	External	Payable	-
mintBehalf	External	Payable	-

Comptroller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getAssetsIn	External	-	-
checkMembership	External	-	-
enterMarkets	Public	Can Modify State	-
addToMarketInternal	Internal	Can Modify State	-
exitMarket	External	Can Modify State	-
mintAllowed	External	Can Modify State	-

Comptroller			
mintVerify	External	Can Modify State	-
redeemAllowed	External	Can Modify State	-
redeemAllowedInternal	Internal	-	-
redeemVerify	External	Can Modify State	-
borrowAllowed	External	Can Modify State	-
borrowVerify	External	Can Modify State	-
repayBorrowAllowed	External	Can Modify State	-
repayBorrowVerify	External	Can Modify State	-
liquidateBorrowAllowed	External	Can Modify State	-
liquidateBorrowVerify	External	Can Modify State	-
seizeAllowed	External	Can Modify State	-
seizeVerify	External	Can Modify State	-
transferAllowed	External	Can Modify State	-
transferVerify	External	Can Modify State	-
getAccountLiquidity	Public	-	-
getAccountLiquidityInternal	Internal	-	-
getHypotheticalAccountLiquidity	Public	-	-
getHypotheticalAccountLiquidityInternal	Internal	-	-
liquidateCalculateSeizeTokens	External	-	-
_setPriceOracle	Public	Can Modify State	-

Comptroller			
_setCloseFactor	External	Can Modify State	-
_setCollateralFactor	External	Can Modify State	-
_setLiquidationIncentive	External	Can Modify State	-
_supportMarket	External	Can Modify State	-
_setTokenAddress	External	Can Modify State	-
_addMarketInternal	Internal	Can Modify State	-
_setMarketBorrowCaps	External	Can Modify State	-
_setBorrowCapGuardian	External	Can Modify State	-
_setPauseGuardian	Public	Can Modify State	-
_setMintPaused	Public	Can Modify State	-
_setBorrowPaused	Public	Can Modify State	-
_setTransferPaused	Public	Can Modify State	-
_setSeizePaused	Public	Can Modify State	-
_become	Public	Can Modify State	-
adminOrInitializing	Internal	-	-
setVeeSpeedInternal	Internal	Can Modify State	-
updateVeeSupplyIndex	Internal	Can Modify State	-
updateVeeBorrowIndex	Internal	Can Modify State	-
distributeSupplierVee	Internal	Can Modify State	-
distributeBorrowerVee	Internal	Can Modify State	-

Comptroller			
updateContributorRewards	Public	Can Modify State	-
claimVee	Public	Can Modify State	-
claimVee	Public	Can Modify State	-
claimVee	Public	Can Modify State	-
grantVeeInternal	Internal	Can Modify State	-
_grantVee	Public	Can Modify State	-
_setVeeSpeed	Public	Can Modify State	-
_setContributorVeeSpeed	Public	Can Modify State	-
getAllMarkets	Public	-	-
getBlockNumber	Public	-	-
getVeeAddress	Public	-	-
mintBehalf	Public	Can Modify State	-
mintBehalf	Public	Payable	-

CToken			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	-
transferTokens	Internal	Can Modify State	-
transfer	External	Can Modify State	nonReentrant
transferFrom	External	Can Modify State	nonReentrant

CToken			
approve	External	Can Modify State	-
allowance	External	-	-
balanceOf	External	-	-
balanceOfUnderlying	External	Can Modify State	-
getAccountSnapshot	External	-	-
getBlockNumber	Internal	-	-
borrowRatePerBlock	External	-	-
supplyRatePerBlock	External	-	-
totalBorrowsCurrent	External	Can Modify State	nonReentrant
borrowBalanceCurrent	External	Can Modify State	nonReentrant
borrowBalanceStored	Public	-	-
borrowBalanceStoredInternal	Internal	-	-
accrueBalanceInternal	Internal	-	-
deltaAccrueFeeInternal	Internal	-	-
exchangeRateCurrent	Public	Can Modify State	nonReentrant
exchangeRateStored	Public	-	-
exchangeRateStoredInternal	Internal	-	-
getCash	External	-	-
accrueInterest	Public	Can Modify State	-
mintInternal	Internal	Can Modify State	nonReentrant

CToken			
mintFresh	Internal	Can Modify State	-
redeemInternal	Internal	Can Modify State	nonReentrant
redeemUnderlyingInternal	Internal	Can Modify State	nonReentrant
redeemFresh	Internal	Can Modify State	-
borrowInternal	Internal	Can Modify State	nonReentrant
borrowFresh	Internal	Can Modify State	-
repayBorrowInternal	Internal	Can Modify State	nonReentrant
repayBorrowBehalfInternal	Internal	Can Modify State	nonReentrant
repayBorrowFresh	Internal	Can Modify State	-
liquidateBorrowInternal	Internal	Can Modify State	nonReentrant
liquidateBorrowFresh	Internal	Can Modify State	-
seize	External	Can Modify State	nonReentrant
seizeInternal	Internal	Can Modify State	-
_setPendingAdmin	External	Can Modify State	-
_acceptAdmin	External	Can Modify State	-
_setComptroller	Public	Can Modify State	-
_setReserveFactor	External	Can Modify State	nonReentrant
_setReserveFactorFresh	Internal	Can Modify State	-
_addReservesInternal	Internal	Can Modify State	nonReentrant
_addReservesFresh	Internal	Can Modify State	-

CToken			
_reduceReserves	External	Can Modify State	nonReentrant
_reduceReservesFresh	Internal	Can Modify State	-
_setInterestRateModel	Public	Can Modify State	-
_setInterestRateModelFresh	Internal	Can Modify State	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can Modify State	-
doTransferOut	Internal	Can Modify State	-
doDeposit	Internal	Can Modify State	-
setOrderProxy	External	Can Modify State	-
getOrderProxy	External	-	-
_getRevertMsg	Internal	-	-
callOrderProxyInternal	Internal	Can Modify State	-
source	Internal	-	-
decodeMessage	Internal	-	-
borrowLeverageInternal	Internal	Can Modify State	nonReentrant
mintBehalfInternal	Internal	Can Modify State	nonReentrant
repayLeverageFresh	Internal	Can Modify State	-
repayLeverageInternal	Internal	Can Modify State	-

JumpRateModel

JumpRateModel			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
utilizationRate	Public	-	-
getBorrowRate	Public	-	-
getSupplyRate	Public	-	-

JumpRateModelV2			
Function Name	Visibility	Mutability	Modifiers
getBorrowRate	External	-	-
getSupplyRate	External	-	-
<Constructor>	Public	Can Modify State	BaseJumpRateModelV2

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	-

Timelock			
executeTransaction	Public	Payable	-
getBlockTimestamp	Internal	-	-

Unitroller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_setPendingImplementation	Public	Can Modify State	-
_acceptImplementation	Public	Can Modify State	-
_setPendingAdmin	Public	Can Modify State	-
_acceptAdmin	Public	Can Modify State	-
<Fallback>	External	Payable	-

WhitePaperInterestRateModel			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
utilizationRate	Public	-	-
getBorrowRate	Public	-	-
getSupplyRate	Public	-	-

SwapHelper			
Function Name	Visibility	Mutability	Modifiers

SwapHelper			
swapERC20ToETH	External	Payable	-
swapETHToERC20	External	Payable	-
swapERC20ToERC20	External	Payable	-
getAmountOut	Internal	-	-
_router	Internal	-	-
getExecutionPrice	Public	-	-
getPairPrice	External	Payable	-
getTokenA2TokenBRate	Internal	-	-
getTokenA2TokenBPrice	Internal	-	-
getPairReserves	Internal	-	-
calcPrices	Internal	-	-

VeeProxyController			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
initialize	Public	Can Modify State	initializer
createOrderERC20ToERC20	External	Payable	veeLock
createOrderERC20ToETH	External	Payable	veeLock
createOrderETHToERC20	External	Payable	veeLock
onOrderCreate	Internal	Can Modify State	-

VeeProxyController			
calcSwapAmount	Internal	-	-
checkAllowance	Internal	-	-
addPlatformFees	Internal	Can Modify State	nonReentrant
queryPlatformFees	External	-	-
withdrawPlatformFees	External	Can Modify State	onlyAdmin nonReentrant
sendTokenPrices2BaseToken	Internal	Can Modify State	-
commonCreateRequire	Internal	Can Modify State	-
checkOrder	External	Can Modify State	-
executeOrder	External	Can Modify State	onlyExecutor nonReentrant veeLock
getStopLowPrice	Internal	-	-
calcAveragePrice	Internal	-	-
cancelOrder	External	Can Modify State	nonReentrant veeLock
settlementOrder	Internal	Can Modify State	-
getOrderDetail	External	-	-
setExecutor	External	Can Modify State	onlyAdmin
removeExecutor	External	Can Modify State	onlyAdmin
setCETH	External	Can Modify State	onlyAdmin
setRouter	External	Can Modify State	onlyAdmin

VeeProxyController			
repayLeverageAndBorrow	Internal	Can Modify State	-
min	Internal	-	-
getBorrowRemains	Internal	-	-
transferProfit	Internal	Can Modify State	-
swapERC20ToETH	Internal	Can Modify State	-
swapETHToERC20	Internal	Can Modify State	-
swapERC20ToERC20	Internal	Can Modify State	-
getPairPrice	Internal	Can Modify State	-
getExecutionPrice	Internal	Can Modify State	-
getRandom	Private	Can Modify State	-
deposit	External	Can Modify State	-
deposit	External	Payable	-
addToMarketInternal	Internal	Can Modify State	-
getAccountAssetBalance	External	-	-
getAssetsIn	External	-	-
source	Private	-	-
decodeMessage	Internal	-	-
subAndSaveBalance	Internal	Can Modify State	-

VeeProxyController			
subAndSaveLeverageBalance	Internal	Can Modify State	-
addAndSaveBalance	Internal	Can Modify State	-
addAndSaveLeverageBalance	Internal	Can Modify State	-
delegateTo	Internal	Can Modify State	-
setSwapHelper	External	Can Modify State	onlyAdmin
_setRouter	Internal	Can Modify State	-
getRouter	External	-	-
_router	Internal	-	-
setServiceFee	External	Can Modify State	onlyAdmin
setOriginFee	External	Can Modify State	onlyAdmin
setMaxleverage	External	Can Modify State	onlyAdmin
setMaxExpire	External	Can Modify State	onlyAdmin

VeeSystemController			
Function Name	Visibility	Mutability	Modifiers
setState	External	Can Modify State	onlyAdmin
setBaseToken	External	Can Modify State	onlyAdmin

4.3 Vulnerability Summary

[N1] [Low] Token allowance issue

Category: Design Logic Audit

Content

In the SwapHelper contract, users can swap tokens through the swapERC20ToETH function and the swapERC20ToERC20 function. When calling this function, it will check whether the allowance is greater than the number of tokens that need to be swap. If the allowance is insufficient, the Token contract will be called for approve operation. However, if the approve function of this token restricts the number of allowance (that is, only 0 can be approved when there is an authorization quota, and then other target quotas can be approved), the authorization operation will not succeed.

Code location：

```
uint256 allowance = IERC20(tokenA).allowance(address(this), router);
if(allowance < amountA){
    require(IERC20(tokenA).approve(router, amountA), "failed to approve");
}
```

Solution

It is recommended to call approve to 0 first, and then approve the required target quota.

Status

Confirmed

[N2] [Critical] Slippage check issue

Category: Design Logic Audit

Content

In the SwapHelper contract, users can swap tokens through the swapERC20ToETH function, swapETHToERC20 function, and swapERC20ToERC20 function. It will use the PangolinRouter to call the target DEX for swap, but the minimum number of received tokens passed in is 1, which will pose risks such as sandwich attacks. And when the

order is executed, token swap will still be carried out, and malicious users can use the slippage issue caused by the order execution to carry out arbitrage attacks.

Code location：

```
    function swapERC20ToETH(address tokenA, uint256 amountA) external payable returns
(uint256[] memory){
        require(tokenA != address(0), "invalid token A");

        address router = _router();
        address[] memory path = new address[](2);
        path[0] = tokenA;
        path[1] = IPangolinRouter(router).WAVAX();

        uint256 allowance = IERC20(tokenA).allowance(address(this), router);
        if(allowance < amountA){
            require(IERC20(tokenA).approve(router, amountA), "failed to approve");
        }

        uint256 deadline = (block.timestamp + 99999999);
        uint256[] memory amounts;
        uint256 amountOutMin = 1;
        amounts = IPangolinRouter(router).swapExactTokensForAVAX(amountA,
amountOutMin, path, payable(address(this)), deadline);
        return amounts;
    }
```

Solution

It is recommended to conduct a slippage check to avoid the impact of large slippage in the business scenario of large capital swap.

Status

Confirmed; After communicating with the project party, the project party stated that it will compare the number of tokens in the pool with the price of the off-chain oracle when creating an order to check slippage. When the order is executed, the business scenario depends on the market behavior in real time, which is the expected design, so the slippage check will not be performed when the order is executed.

[N3] [Suggestion] cToken counterfeiting issue

Category: Design Logic Audit

Content

In the VeeProxyController contract, the createOrderERC20ToERC20/createOrderERC20ToETH/createOrderETHToERC20 functions can be used to create an order, but the cToken address passed in is not checked when the order is created, which will cause the creator to pass in a fake cToken contract and cause subsequent order execution , token exchange, price check environment has unexpected situation.

Code location：

```
    function createOrderERC20ToERC20(address orderOwner, address ctokenA, address
tokenA, address tokenB, uint256 amountA, uint256 stopHighPairPrice, uint256
stopLowPairPrice, uint256 expiryDate, uint8 leverage) external
veeLock(uint8(VeeLockState.LOCK_CREATE)) payable returns (bytes32 orderId){
        require(tokenA != tokenB,"tokenA and tokenB can't be same");
        require(tokenA != address(0), "invalid tokenA");
        require(ctokenA != address(0), "invalid ctoken A");
        commonCreateRequire(orderOwner, stopHighPairPrice, stopLowPairPrice, amountA,
expiryDate, leverage);
        addPlatformFees(tokenA, amountA, leverage);
        checkAllowance(orderOwner,tokenA,amountA,leverage);
        uint256[] memory amounts = swapERC20ToERC20(tokenA, tokenB,
calcSwapAmount(amountA, leverage));
        orderId = onOrderCreate(orderOwner, ctokenA, tokenA, tokenB, amountA,
amounts, stopHighPairPrice, stopLowPairPrice, expiryDate, leverage);
    }
```

Solution

It is recommended to check the validity of the passed cToken and token when creating an order.

Status

Ignored; After communicating with the project party, the project party stated that the cToken address directly uses msg.sender and checked that msg.sender should be the VEETOKEN_ROLE role.

[N4] [Medium] Price record manipulation issue

Category: Others

Content

In the SwapHelper contract, getPairPrice is used to obtain the price of the token transaction pair, but the price calculation method of this function is calculated by the number of tokens obtained through the getReserves interface of the Pair contract. However, the number of tokens acquired by this interface is real-time, and malicious users can change the token price through the swap operation. This will cause the price obtained by the getPairPrice function to be inaccurate and easily manipulated.

Code location：

```
    function getTokenA2TokenBPrice(address tokenA, address tokenB) internal view
returns(uint256 price){
        if(tokenA == tokenB){
            price = 1e18;
        }else{
            (uint256 Res0, uint256 Res1) = getPairReserves(tokenA, tokenB);
            price = calcPrices(tokenA, tokenB, Res0, Res1);
        }
    }
```

Solution

It is recommended to use ChainLink or other delayed price feed oracles.

Status

Ignored; After communicating with the project party, the project party stated that the price obtained by the getPairPrice method is only used to display the historical price in the website's pending order list, and will not have any impact on the user's profit.

[N5] [Suggestion] Token transfer issue

Category: Others

Content

In the VeeProxyController contract, users of the transferProfit function and withdrawPlatformFees function transfer profits and fees, and they directly use the transfer function to transfer tokens. If the docked token has a false top-up problem, or the return value does not meet the EIP20 specification, it may cause unexpected the result of.

Code location：

```
    function transferProfit(Order memory order, uint256 profit) internal returns
(bool){
        if(profit == 0){
            return true;
        }
        if(order.tokenA == VETH){
            payable(order.orderOwner).transfer(profit);
        }else{
            assert(IERC20(order.tokenA).transfer(order.orderOwner, profit));
        }
        emit TransferProfit(order.tokenA, order.tokenB, profit);
        return true;
    }
```

Solution

It is recommended to use the safeTransfer function for transfer operations.

Status

Fixed

[N6] [Suggestion] Malleable attack issue

Category: Others

Content

In the VeeProxyController contract, the source function is used to recover the address of the signer. It uses ecrecover to obtain it. The v and s values are not checked, which may pose a potential risk of malleable attacks.

Code location：

```
    function source(bytes memory message, bytes memory signature) private pure
returns (address) {
        (bytes32 r, bytes32 s, uint8 v) = abi.decode(signature, (bytes32, bytes32,
uint8));
        bytes32 hash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32",
keccak256(message)));
        return ecrecover(hash, v, r, s);
    }
```

Solution

It is recommended to use the ECDSA library of openzeppelin for signature address recovery. At the same time, pay attention to using a secure signature library when signing off-chain to avoid using the same random number K to cause the same r value.

Status

Fixed

[N7] [Medium] Relative price manipulation

Category: Design Logic Audit

Content

In the VeeProxyController contract, when the time expires or the price reaches the stop profit and stop loss price, the order will be executed through the executeOrder function, but when the price is taken, the getExecutionPrice function of the SwapHelper contract is used. This function obtains the quantity through the getAmountsIn or getAmountsOut interface of DEX. But these two interfaces are also calculated based on the real-time number of tokens in the contract, so there is an issue of malicious manipulation. As a result, the final price obtained is inconsistent with the actual price.

Code location：

```
    function executeOrder(bytes32 orderId) external onlyExecutor nonReentrant
veeLock(uint8(VeeLockState.LOCK_EXECUTEDORDER)) returns (bool){
        Order memory order = orders[orderId];
        require(order.orderOwner != address(0), "invalid order id");
```

```
        require(order.expiryDate > block.timestamp, "expiryDate");

        uint256 price = getExecutionPrice(order.tokenA, order.tokenB, order.amountB,
false);
        uint256 stopLowPrice = getStopLowPrice(order.tokenA, order.tokenB,
order.amountA, order.amountB, order.stopLowPairPrice, order.leverage);

        require(price <= stopLowPrice || price >= order.stopHighPairPrice, "The price
is not reached");
        sendTokenPrices2BaseToken(orderId,uint8(1));
        uint256[] memory amounts;
        if(order.tokenA != VETH && order.tokenB != VETH ){
            amounts = swapERC20ToERC20(order.tokenB, order.tokenA, order.amountB);
        }else if(order.tokenA == VETH && order.tokenB != VETH ){
            amounts = swapERC20ToETH(order.tokenB, order.amountB);
        }else{
            amounts = swapETHToERC20(order.tokenA, order.amountB);
        }
        require(amounts[1] != 0, "failed to swap tokens");
        settlementOrder(orderId, amounts);
        emit OnOrderExecuted(orderId,
order.amountA,order.stopLowPairPrice,order.stopHighPairPrice,price);
        return true;
    }
```

```
    function getExecutionPrice(
        address _buyAsset,
        address _sellAsset,
        uint256 _tradeSize,
        bool _isBuyingCollateral
    )
        public
        view
        returns (uint256)
    {
        IPangolinRouter router = IPangolinRouter(_router());
        address[] memory path = new address[](2);
        path[0] = _sellAsset;
        path[1] = _buyAsset;

        uint256[] memory flows = _isBuyingCollateral ?
router.getAmountsIn(_tradeSize, path) : router.getAmountsOut(_tradeSize, path);
```

```
        uint256 buyDecimals = uint256(10)**IERC20(_buyAsset).decimals();
        uint256 sellDecimals = uint256(10)**IERC20(_sellAsset).decimals();
        uint256 price =
flows[1].preciseDiv(buyDecimals).preciseDiv(flows[0].preciseDiv(sellDecimals));
        return price;
    }
```

Solution

It is recommended to obtain and calculate through the delayed price feed oracle machine.

Status

Ignored; After communicating with the project party, the project party stated that the real-time business scenario relies on market behavior and the real-time use of the number of tokens in the pool to participate in the calculation is an expected design. In the product design, it is considered that the price is reached and the user must perform the swap operation for the user.

[N8] [Medium] Bad debt issue

Category: Design Logic Audit

Content

The parameters for creating an order in the VeeProxyController contract come from the order passed in by the user from the borrowAndCall function of the cToken contract, so the stopHighPairPrice, stopLowPairPrice, and expiryDate parameters are all passed in as defined by the user. If the stopLowPairPrice passed in by the user is 1 and the stopHighPairPrice is uint(-1), then the price check on the executeOrder operation may never pass, resulting in bad debts.

Note: ExpiryDate check and price check are separate and require check. Even if it times out, executeOrder cannot be executed if the price does not reach the trigger point.

Code location：

```
    function executeOrder(bytes32 orderId) external onlyExecutor nonReentrant
veeLock(uint8(VeeLockState.LOCK_EXECUTEDORDER)) returns (bool){
```

```
        Order memory order = orders[orderId];
        require(order.orderOwner != address(0), "invalid order id");
        require(order.expiryDate > block.timestamp, "expiryDate");

        uint256 price = getExecutionPrice(order.tokenA, order.tokenB, order.amountB,
false);
        uint256 stopLowPrice = getStopLowPrice(order.tokenA, order.tokenB,
order.amountA, order.amountB, order.stopLowPairPrice, order.leverage);
        require(price <= stopLowPrice || price >= order.stopHighPairPrice, "The price
is not reached");
        sendTokenPrices2BaseToken(orderId,uint8(1));
        uint256[] memory amounts;
        if(order.tokenA != VETH && order.tokenB != VETH ){
            amounts = swapERC20ToERC20(order.tokenB, order.tokenA, order.amountB);
        }else if(order.tokenA == VETH && order.tokenB != VETH ){
            amounts = swapERC20ToETH(order.tokenB, order.amountB);
        }else{
            amounts = swapETHToERC20(order.tokenA, order.amountB);
        }
        require(amounts[1] != 0, "failed to swap tokens");
        settlementOrder(orderId, amounts);
        emit OnOrderExecuted(orderId,
order.amountA,order.stopLowPairPrice,order.stopHighPairPrice,price);
        return true;
    }
```

Solution

It is recommended to check the incoming stopHighPairPrice and stopLowPairPrice and limit them to a certain range, or allow the order to be executed directly when the price reaches the stop-profit and stop-loss price when the order expires.

Status

Ignored; After communicating with the project, the project stated that its pending order service will continue to monitor the order's stop-profit and stop-loss conditions, execute the executeOrder when the conditions are met, and the service is also checking overtime, so the order will be cancelled up to 30 days.

[N9] [Medium] Return value check issue

Category: Design Logic Audit

Content

In the CEther contract, the borrowLeverageInternal function is called in the borrowAndCall function and its return value is assigned to the ret parameter, but this parameter is not checked subsequently.

Code location：

```
    function borrowAndCall(uint borrowAmount, uint8 leverage, bytes4 signature, bytes
calldata order) external payable returns (bytes memory) {
        uint ret = borrowLeverageInternal(borrowAmount, leverage);
        (bool success, bytes memory returnData) = callOrderProxyInternal(signature,
order);
        require(success, _getRevertMsg(returnData));
        return returnData;
    }
```

Solution

It is recommended to check the execution return value to avoid unexpected issues.

Status

Fixed

[N10] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the Comptroller and cToken contracts, the admin role can modify key sensitive parameters such as the oracle, the rate model, and the market, which will lead to the risk of excessive authority of the admin role.

Solution

It is recommended that the authority division processing be carried out in the early stage of the project:

1. The authority related to user funds should be transferred to the timelock contract or community governance. The timelock contract admin can use multi-signature to avoid the risk of private key leakage.

2. In the early stage of the project, some parameters may be frequently modified. This part of the permissions can be controlled separately.
3. Consider retaining the authority to temporarily suspend the project in order to respond to an emergency in the early stage of the project, which can quickly suspend the project and stop the loss in time.
4. After the project has passed the early stage of smooth operation, the authority can be transferred to community governance.

Status

Confirmed; After communicating with the project and feedback, the project stated that it will transfer the ownership to the timelock contract, and in order to deal with emergency situations, it will retain the right to perform any operation immediately without time delay within half a year of the transfer of the authority. Half a year after the transfer of ownership, the immediate execution interface of the timelock contract will no longer be available.

[N11] [Suggestion] Initialization issue

Category: Design Logic Audit

Content

The initialize function in the VestingEscrow contract does not control permissions.

Code location：

```
function initialize(address _token) public initializer {
    vestingToken = IERC20(_token);
    _notEntered = true;
}
```

Solution

It is recommended to call this function of the logic contract to initialize the logic contract when setting up the logic contract in the proxy contract to avoid the problem of initial front-running.

Status

Ignored; After communicating with the project party, the project stated that it will use @openzeppelin/hardhat-upgrades to deploy the contract, and initialize will be automatically executed on the proxy contract when the contract is deployed.

[N12] [Suggestion] Access control issue

Category: Authority Control Vulnerability

Content

In the cToken contract, the user can return the leveraged loan through the repayLeverageAndBorrow function, which is to trigger the repayLeverageAndBorrow function call of the cToken contract through the settlementOrder function when the order is executed. However, in the cToken contract, the repayLeverageAndBorrow function does not limit that only the VeeProxyController contract can be called. This will cause any user to repay the loan through this function. If this happens, it will cause problems with cToken and VeeProxyController's accounting.

Code location：

```
    function repayLeverageAndBorrow(address borrower, uint repayAmount, uint
expectLeverageAmount, uint realLeverageAmount) external returns (uint err, uint
result) {
        if (realLeverageAmount > 0) {
            (err, result) = repayLeverageInternal(borrower, expectLeverageAmount,
realLeverageAmount);
            requireNoError(err, "repayLeverage failed");
        }
        if(repayAmount > 0) {
            (err, result) = repayBorrowBehalfInternal(borrower, repayAmount);
            requireNoError(err, "repayBorrow failed");
        }
    }
```

Solution

It is recommended to increase the call permission control according to the expected design.

Status

Confirmed; After communicating with the project party, the project party stated that this is an expected design, which allows users to return leverage and borrowing through repayLeverageAndBorrow.

[N13] [Medium] Add schedule issue

Category: Design Logic Audit

Content

The _addSchedule function in the estingEscrow contract is used to add user lock information. If `vestingSchedules[account].length <1` during the adding process, a new space will be created for `vestingSchedules[account]` to record data. It will not be created again during the second addition, which will cause only the first _addSchedule to be successful, and other additions cannot be successfully added.

Code location：

```
    function _addSchedule(address account, uint8 slot, uint amount, uint
startTimestamp, uint endTimestamp, uint accelerateFactor) internal
returns(VestingSchedule memory) {
        if (vestingSchedules[account].length < 1) { //create 1 slots for users
            vestingSchedules[account].push();
        }
        require(vestingSchedules[account][slot].totalAmount == 0, "slot used");
        vestingSchedules[account][slot] = VestingSchedule(amount, 0, startTimestamp,
endTimestamp, account, accelerateFactor);
        lockingBalances[account] -= amount;
        return vestingSchedules[account][slot];
    }
```

Solution

It is recommended to modify the judgment condition to `vestingSchedules[account].length = slot`.

Status

Ignored; After communicating with the project, the project team stated that VestingEscrow will add the item function in the later planning. The current version only opens 1 slot for users by default, and subsequent slots need to be opened by purchasing props.

[N14] [Suggestion] Transfer issue

Category: Others

Content

VestingEscrow is integrated, and safeTransferFrom is used to transfer tokens during deposit. When unlocking, use the transfer function to transfer out.

Code location：

```
    function withdrawVestedTokens() nonReentrant external {
        require(block.timestamp > lastWithdrawTime[msg.sender] + 3600, "withdraw too
frequent");
        VestingSchedule[] storage schedules = vestingSchedules[msg.sender];
        uint totalUnlocked = unlockedBalances[msg.sender];
        for (uint i = 0; i < schedules.length; i++) {
            uint totalAmountVested = estimateVested(schedules[i]);
            uint amountUnlocked = totalAmountVested - schedules[i].amountWithdrawn;
            schedules[i].amountWithdrawn = totalAmountVested;

            if (amountUnlocked > 0) {
                totalUnlocked += amountUnlocked;
                emit Unlock(msg.sender, uint8(i), amountUnlocked,
schedules[i].totalAmount - totalAmountVested);
            }
        }
        vestingToken.safeTransfer(msg.sender, totalUnlocked);
        emit Withdraw(msg.sender, totalUnlocked, unlockedBalances[msg.sender]);
        unlockedBalances[msg.sender] = 0;
        lastWithdrawTime[msg.sender] = block.timestamp;

    }
```

Solution

It is recommended to use the safeTransfer function to transfer out.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002109080004	SlowMist Security Team	2021.08.09 - 2021.09.08	Medium Risk

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, 6 medium risks, 1 low risk, 6 suggestion vulnerabilities. And 1 critical risk, 1 medium risk, 1 low risk, 1 suggestion vulnerability were confirmed and being fixed; 4 medium risks, 2 suggestion vulnerabilities were ignored; All other findings were fixed. At present, the code has not been deployed to the mainnet, and the ownership of some contracts has not been transferred to the timelock contract on the mainnet, so there is still a risk of excessive authority.

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