Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.09.24, the SlowMist security team received the VVS.Finance team's security audit application for VVS, 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 Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit version:

contracts.zip:

65fcfa2515d3ccca4eae16dee099b0534c6765f577d6e6fb48350363fcb1baa4

vvs-slowmist.zip:

407209b161ab970af6f99e50c119c3439b47867c7e41694a3d852f977dc8e3a0

Fixed version:

vvs-slowmist.zip:

d190a4cd55c4cc7d319a818e40f86a63ac7ebe404ae486cfc3d3c7a685ec2849

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	Confirmed
N2	Emergency withdrawal issue	Others	Low	Confirmed
N3	Dev address setting enhancement suggestions	Others	Suggestion	Ignored
N4	Risk of replay attack	Replay Vulnerability	Suggestion	Ignored
N5	Malleable attack risk	Replay Vulnerability	Suggestion	Fixed
N6	Compatibility issue	Others	Suggestion	Ignored
N7	Missing event records	Others	Suggestion	Fixed
N8	vvsAtLastUserAction parameter record error issue	Others	Suggestion	Fixed
N9	Failure to follow the Checks-Effects-Interactions principle	Reentrancy Vulnerability	Suggestion	Ignored

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:

VVSERC20			
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	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

VVSFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
createPair	External	Can Modify State	-
setFeeTo	External	Can Modify State	-
setFeeToSetter	External	Can Modify State	-

VVSPair

VVSPair			
Function Name	Visibility	Mutability	Modifiers
getReserves	Public	-	-
_safeTransfer	Private	Can Modify State	-
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
_update	Private	Can Modify State	-
_mintFee	Private	Can Modify State	-
mint	External	Can Modify State	lock
burn	External	Can Modify State	lock
swap	External	Can Modify State	lock
skim	External	Can Modify State	lock
sync	External	Can Modify State	lock

Craftsman			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateMultiplier	Public	Can Modify State	onlyOwner
poolLength	External	-	-
add	Public	Can Modify State	onlyOwner
set	Public	Can Modify State	onlyOwner

Craftsman			
setMigrator	Public	Can Modify State	onlyOwner
migrate	Public	Can Modify State	-
getMultiplier	Public	-	-
pendingVVS	External	-	-
massUpdatePools	Public	Can Modify State	-
updatePool	Public	Can Modify State	-
deposit	Public	Can Modify State	-
withdraw	Public	Can Modify State	-
enterStaking	Public	Can Modify State	-
leaveStaking	Public	Can Modify State	-
emergencyWithdraw	Public	Can Modify State	-
safeVVSTransfer	Internal	Can Modify State	-
dev	Public	Can Modify State	-
distributeSupply	Public	Can Modify State	onlyOwner

VVSToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	VVSInitMintable
mint	Public	Can Modify State	onlyOwner

VVSInitMintable

VVSInitMintable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
distributeSupply	Public	Can Modify State	onlyOwner
_perYearToPerBlock	Internal	-	-

Workbench			
Function Name	Visibility	Mutability	Modifiers
mint	Public	Can Modify State	onlyOwner
burn	Public	Can Modify State	onlyOwner
<Constructor>	Public	Can Modify State	-
safeVVSTransfer	Public	Can Modify State	onlyOwner

Timelock			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	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	-	-

VVSVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deposit	External	Can Modify State	whenNotPaused notContract
withdrawAll	External	Can Modify State	notContract
harvest	External	Can Modify State	notContract whenNotPaused
setAdmin	External	Can Modify State	onlyOwner
setTreasury	External	Can Modify State	onlyOwner
setPerformanceFee	External	Can Modify State	onlyAdmin
setCallFee	External	Can Modify State	onlyAdmin
setWithdrawFee	External	Can Modify State	onlyAdmin
setWithdrawFeePeriod	External	Can Modify State	onlyAdmin
emergencyWithdraw	External	Can Modify State	onlyAdmin
inCaseTokensGetStuck	External	Can Modify State	onlyAdmin
pause	External	Can Modify State	onlyAdmin whenNotPaused

VVSVault			
unpause	External	Can Modify State	onlyAdmin whenPaused
calculateHarvestVVSRewards	External	-	-
calculateTotalPendingVVSRewards	External	-	-
getPricePerFullShare	External	-	-
withdraw	Public	Can Modify State	notContract
available	Public	-	-
balanceOf	Public	-	-
_earn	Internal	Can Modify State	-
_isContract	Internal	-	-

VVSMigrator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
migrate	External	Can Modify State	-

VVSRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-

VVSRouter			
_addLiquidity	Internal	Can Modify State	-
addLiquidity	External	Can Modify State	ensure
addLiquidityETH	External	Payable	ensure
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-
removeLiquidityETHSupportingFeeOnTransferTokens	Public	Can Modify State	ensure
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	Can Modify State	-
_swap	Internal	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
_swapSupportingFeeOnTransferTokens	Internal	Can Modify State	-

VVSRouter			
swapExactTokensForTokensSupportingFeeOnTransferTokens	External	Can Modify State	ensure
swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	ensure
swapExactTokensForETHSupportingFeeOnTransferTokens	External	Can Modify State	ensure
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

VVSRouter01			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
_addLiquidity	Private	Can Modify State	-
addLiquidity	External	Can Modify State	ensure
addLiquidityETH	External	Payable	ensure
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-

VVSRouter01			
_swap	Private	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

WCRO			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
deposit	Public	Payable	-
withdraw	Public	Can Modify State	-
totalSupply	Public	-	-
approve	Public	Can Modify State	-

WCRO			
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-

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

VVSVesting			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
addVesting	Public	Can Modify State	onlyOwner onlyNewUser
withdraw	Public	Can Modify State	onlyExistingUser
revoke	Public	Can Modify State	onlyOwner onlyExistingUser
_withdraw	Private	Can Modify State	-
_hasVestingFunds	Private	-	-
vestingAmount	Public	-	-

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

Migrations			
setCompleted	Public	Can Modify State	restricted

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

1.The Owner role can mint arbitrarily through the mint function, and there is no upper limit on the number of minted tokens. The Owner role can also burn users' tokens through the burn function.

Code location:

/farm/contracts/Workbench.sol#L11-17

```
function mint(address _to, uint256 _amount) public onlyOwner {
    _mint(_to, _amount);
}

function burn(address _from ,uint256 _amount) public onlyOwner {
    _burn(_from, _amount);
}
```

2.The Owner role can mint arbitrarily through the mint function, and there is no upper limit on the number of minted tokens.

Code location:

/farm/contracts/VVSToken.sol#L14-16

```
function mint(address _to, uint256 _amount) public onlyOwner {
    _mint(_to, _amount);
}
```

3.Owner role can modify the `BONUS_MULTIPLIER` value through the updateMultiplier function to change the reward multiplier and change the migrator role through the setMigrator function, and there is no event record.

Code location:

/farm/contracts/Craftsman.sol#L101-103, L138-140

```
    function updateMultiplier(uint256 multiplierNumber) public onlyOwner {
        BONUS_MULTIPLIER = multiplierNumber;
    }
    function setMigrator(IMigratorChef _migrator) public onlyOwner {
        migrator = _migrator;
    }
```

4.Owner role can add a pool arbitrarily through the add function, and there is a risk that the Owner can add apool to mine by itself to obtain rewards. When calling the add function to add a pool, the `lastRewardBlock` and `totalAllocPoint` will be updated, and related information about the pool will be stored.

Code location:

/farm/contracts/Craftsman.sol#L111-123

```
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public
onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number :
startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accVVSPerShare: 0
        }));
    }
```

5.Owner role can update the distribution weight of the pool through the set function, and the updated weight will affect the user's mining reward. Owner calls add and in the set function, when the value of the `_withUpdate` parameter is passed as true, the mining pool will be updated. At this time, the set function will also update the distribution weight of the mining pool. After the update, the subsequent mining will be calculated according to the new weight. The modification of the reward distribution rate will affect the rewards of users for mining.

Code location:

/farm/contracts/Craftsman.so#L126-135

```
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public
onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 prevAllocPoint = poolInfo[_pid].allocPoint;
        poolInfo[_pid].allocPoint = _allocPoint;
        if (prevAllocPoint != _allocPoint) {
            totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint);
        }
    }
```

Solution

1. It is recommended to use a time lock mechanism or community governance to restrict.
2. It is recommended to use a time lock mechanism or community governance to restrict.
3. It is recommended to add the event record of setMigrator and updateMultiplier function, and the scope limit of `BONUS_MULTIPLIER`.
4. It is recommended to use a time lock mechanism or community governance to restrict.
5. It is recommended to use a time lock mechanism or community governance to restrict. And It is suggested to force all LP pools to be updated before the weights of LP pools are adjusted to avoid the impact of user income.

Status

Confirmed; After communicating with the project party, the owner role of the Workbench and VVSToken are Craftsman, and all the LP pools are updated before the weights of LP pools are adjusted through the updateStakingPool function.

[N2] [Low] Emergency withdrawal issue

Category: Others

Content

In the VVSVault contract, the admin role can make emergency withdrawals of tokens from the craftsman contract to the VVSVault contract via the emergencyWithdraw function. However, it should be noted that any user can obtain 0.25% of the token reward in the VVSVault contract through the harvest function, and re-stake the remaining tokens into the Craftsman contract. So if the emergencyWithdraw operation is performed while the contract is not suspended it may cause unintended results.

Code location：

/farm/contracts/VVSVault.sol#L148-163,L226-228

```
    function harvest() external notContract whenNotPaused {
        ICraftsman(craftsman).leaveStaking(0);

        uint256 bal = available();
        uint256 currentPerformanceFee = bal.mul(performanceFee).div(10000);
        token.safeTransfer(treasury, currentPerformanceFee);

        uint256 currentCallFee = bal.mul(callFee).div(10000);
        token.safeTransfer(msg.sender, currentCallFee);

        _earn();

        lastHarvestedTime = block.timestamp;

        emit Harvest(msg.sender, currentPerformanceFee, currentCallFee);
    }

    function emergencyWithdraw() external onlyAdmin {
```

```
        ICraftsman(craftsman).emergencyWithdraw(0);
    }
```

Solution

It is recommended to suspend the VVSVault contract before the emergencyWithdraw operation, or set the `_paused` parameter to true in the emergencyWithdraw function.

Status

Confirmed; After communicating with the project party, the project party thinks It's ok in some case, because someone just want to withdraw his fund without reward, if the project party first pauses the contract, user can not withdraw.

[N3] [Suggestion] Dev address setting enhancement suggestions

Category: Others

Content

If the dev address is an EOA address, in a scenario where the private key is leaked, the team's revenue will be stolen.

Code location：

/farm/contracts/Craftsman.sol#L301-305

```
    function dev(address _devaddr) public {
        require(msg.sender == devaddr, "dev: wut?");
        devaddr = _devaddr;
    }
```

Solution

It is recommended to set the development address as a multi-signature contract to avoid the leakage of private keys and the theft of team rewards.

Status

Ignored

[N4] [Suggestion] Risk of replay attack

Category: Replay Vulnerability

Content

`DOMAIN_SEPARATOR` is defined when the contract is initialized, but it is not reimplemented when `DOMAIN_SEPARATOR` is used in the permit function. So the `DOMAIN_SEPARATOR` contains the `chainId` and is defined at contract deployment instead of reconstructed for every signature, there is a risk of possible replay attacks between chains in the event of a future chain split.

Code location：

/core/contracts/VVSERC20.sol#L23-38

```
    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)'),
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }
```

Solution

It is recommended to redefine when using DOMAIN_SEPARATOR.

Reference: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2612.md

Status

Ignored

[N5] [Suggestion] Malleable attack risk

Category: Replay Vulnerability

Content

In the permit function, it restores the address of the signer through the ecrecover function, but does not check the value of v and s. Since EIP2 still allows the malleability for ecrecover, this will lead to the risk of transaction malleability attacks.

Code location：

/core/contracts/VVSERC20.sol#L81-94

```
    function permit(address owner, address spender, uint value, uint deadline, uint8
v, bytes32 r, bytes32 s) external {
        require(deadline >= block.timestamp, 'Vvs: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'Vvs:
INVALID_SIGNATURE');
        _approve(owner, spender, value);
    }
}
```

Solution

It is recommended to use the ECDSA library of openzeppelin to check the signature.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol

Status

Fixed

[N6] [Suggestion] Compatibility issue

Category: Others

Content

1.In the contract, users can stake/withdraw their tokens through the deposit function and withdraw function. It will directly record the amount parameter passed by the user into `user.amount`, and transfer the tokens to the contract through the safeTransferFrom function. If the contract receives deflationary tokens, the actual number of tokens received by the contract will not match the number of tokens recorded in the contract.

Code location：

/farm/contracts/Craftman.sol#L202-221

```
 function deposit(uint256 _pid, uint256 _amount) public {

        require (_pid != 0, 'deposit VVS by staking');

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pending =
user.amount.mul(pool.accVVSPerShare).div(1e12).sub(user.rewardDebt);
            if(pending > 0) {
                safeVVSTransfer(msg.sender, pending);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(address(msg.sender), address(this),
_amount);
            user.amount = user.amount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVVSPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
    }
```

```
    function withdraw(uint256 _pid, uint256 _amount) public {

        require (_pid != 0, 'withdraw VVS by unstaking');
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");

        updatePool(_pid);
        uint256 pending =
user.amount.mul(pool.accVVSPerShare).div(1e12).sub(user.rewardDebt);
        if(pending > 0) {
            safeVVSTransfer(msg.sender, pending);
        }
        if(_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVVSPerShare).div(1e12);
        emit Withdraw(msg.sender, _pid, _amount);
    }
```

2.Users can transfer the token into the vault contract through the deposit function. Under normal circumstances, the number of staking tokens transferred by the user is the same as the `_amount` parameter passed in. But if the staking token is a deflationary token, the number of tokens transferred by the user may be different from the number of tokens actually received in the contract.

Code location：

/farm/contracts/VVSVault.sol#L111-135

```
    function deposit(uint256 _amount) external whenNotPaused notContract {
        require(_amount > 0, "Nothing to deposit");

        uint256 pool = balanceOf();
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint256 currentShares = 0;
        if (totalShares != 0) {
            currentShares = (_amount.mul(totalShares)).div(pool);
        } else {
            currentShares = _amount;
        }
```

```
        UserInfo storage user = userInfo[msg.sender];

        user.shares = user.shares.add(currentShares);
        user.lastDepositedTime = block.timestamp;

        totalShares = totalShares.add(currentShares);

        user.vvsAtLastUserAction = user.shares.mul(balanceOf()).div(totalShares);
        user.lastUserActionTime = block.timestamp;

        _earn();

        emit Deposit(msg.sender, _amount, currentShares, block.timestamp);
    }
```

Solution

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

Status

Ignored; After communicating with the project party, the project party states that it will not use deflationary token.

[N7] [Suggestion] Missing event records

Category: Others

Content

In the contract, the Owner role can set Admin role and Treasury address through the setAdamin and aetTreasury function, but no event logging is preformed. And the Admin role can set `performanceFee`, `callFee`, `withdrawFee` and `withdrawFeePeriod` through the setPerformanceFee, setCallFee, setWithdrawFee and setWithdrawFeePeriod function, but no event logging is preformed.

Code location：

/farm/contracts/VVSVault.sol#L169-220

```
    function setAdmin(address _admin) external onlyOwner {
        require(_admin != address(0), "Cannot be zero address");
        admin = _admin;
```

```
    }

    function setTreasury(address _treasury) external onlyOwner {
        require(_treasury != address(0), "Cannot be zero address");
        treasury = _treasury;
    }

    function setPerformanceFee(uint256 _performanceFee) external onlyAdmin {
        require(_performanceFee <= MAX_PERFORMANCE_FEE, "performanceFee cannot be
more than MAX_PERFORMANCE_FEE");
        performanceFee = _performanceFee;
    }

    function setCallFee(uint256 _callFee) external onlyAdmin {
        require(_callFee <= MAX_CALL_FEE, "callFee cannot be more than
MAX_CALL_FEE");
        callFee = _callFee;
    }

    function setWithdrawFee(uint256 _withdrawFee) external onlyAdmin {
        require(_withdrawFee <= MAX_WITHDRAW_FEE, "withdrawFee cannot be more than
MAX_WITHDRAW_FEE");
        withdrawFee = _withdrawFee;
    }

    function setWithdrawFeePeriod(uint256 _withdrawFeePeriod) external onlyAdmin {
        require(
            _withdrawFeePeriod <= MAX_WITHDRAW_FEE_PERIOD,
            "withdrawFeePeriod cannot be more than MAX_WITHDRAW_FEE_PERIOD"
        );
        withdrawFeePeriod = _withdrawFeePeriod;
    }
```

Solution

It is recommended to record events when modifying sensitive parameters.

Status

Fixed

[N8] [Suggestion] vvsAtLastUserAction parameter record error issue

Category: Others

Content

In the VVSVault contract, the user can withdraw the funds staked by the user through the withdraw function. If the user does not withdraw all funds (user.shares> 0), this function will recalculate the user's `vvsAtLastUserAction` value. In the calculation process, the number of vvs tokens obtained by the balanceOf function is used to participate in the calculation. But at the end of this function, a certain amount of vvs tokens will be transferred to the user through the safeTransfer function, so the number of vvs tokens obtained by the balanceOf function used in the calculation of vvsAtLastUserAction is relatively large.`

Code location：

/farm/contracts/VVSVault.sol#L319-330

```
    if (user.shares > 0) {
        user.vvsAtLastUserAction = user.shares.mul(balanceOf()).div(totalShares);
    } else {
        user.vvsAtLastUserAction = 0;
    }

    user.lastUserActionTime = block.timestamp;

    token.safeTransfer(msg.sender, currentAmount);

    emit Withdraw(msg.sender, currentAmount, _shares);
}
```

Solution

It is recommended to perform the safeTransfer operation first and then calculate the user's vvsAtLastUserAction value.

Status

Fixed

[N9] [Suggestion] Failure to follow the Checks-Effects-Interactions principle

Category: Reentrancy Vulnerability

Content

In the event of an emergency, users can withdraw their staked assets through the emergencyWithdraw function. However, during the withdrawal process, it first transfers the staked funds to the user through the safeTransfer function, and then sets user.amount and user.rewardDebt to 0, which does not comply with the Checks-Effects-Interactions principle.

Code location：

/farm/contracts/Craftsman.sol#L286-293

```
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
}
```

Solution

It is recommended to follow the Checks-Effects-Interactions principle and change the status first before performing the transfer operation.

Status

Ignored; After communicating with the project party, the project party states that it will not use ERC777 token.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002110120001	SlowMist Security Team	2021.09.24 - 2021.10.12	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 medium risk, 1 low risk, 7 suggestion vulnerabilities. And 1 medium risk, 1 low risk vulnerabilities were confirmed and being fixed; 4 suggestion vulnerabilities were ignored; All other 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