Smart Contract Security Audit Report

Contents

1. Executive Summary

On Mar. 10, 2021, the SlowMist security team received the Smoothy Finance team's security audit application for SmoothyV1, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack

- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

Smoothy.finance - single pool with low-cost zero-slippage swapping and maximum interest earning Smoothy.finance is a novel automated market maker (AMM) for same-backed assets (such as stablecoins) in a single pool. The unique features of Smoothy.finance are single pool for all stablecoins, extremely low swapping fee, and maximum interest earning. Smoohty.finance develops a unique swapping protocol for stablecoins using bonding curve, which supports 10+ stablecoins in the same pool. The single pool feature of Smoothy.finance greatly maximizes the liquidity of all stablecoins without worrying about fragmented liquidity caused by multiple pools - a common way used by existing protocols (such as curve). Moreover, thanks to the bonding curve of Smoothy.finance, the gas cost of swapping in Smoothy.finance can be extremely low - even about 10x lower than that of mStable/curve yPool. Finally, the dynamic cash reserve (DSR) algorithm invented by Smoothy.finance can further maximize the interest earned from the underlying lending platforms without incurring extra gas cost for normal transactions.

Audit files:

https://github.com/smoothyfinance/smoothy-contract

commit: f50450aa09f5031be5f7f330f527333571982d65

3.2 Project Structure

```
├── ReentrancyGuardPausable.sol
├── Root.sol
├── SmoothyV1.sol
├── UpgradeableOwnable.sol
├── UpgradeableOwnableProxy.sol
├── YERC20.sol
├── liquidity-mining
   ├── SMTYToken.sol
   ├── SmoothyMasterRoot.sol
   ├── SmoothyMasterV1.sol
   ├── VotingEscrow.sol
   └── VotingEscrowRoot.sol
```

4. Code Overview

4.1 Main Contract address

Contract Name	Contract Address
Root	0xe5859f4EFc09027A9B718781DCb2C6910CAc6E91
SmoothyV1	0x965cC658158a7689FBB6C4Df735aA435C500C29B
Timelock	0xa13c1A5fdFBBe60a71a2c1822de97000EC8e4079
SMTYToken.sol	Not yet deployed on the mainnet
SmoothyMasterRoot.sol	Not yet deployed on the mainnet
SmoothyMasterV1.sol	Not yet deployed on the mainnet
VotingEscrow.sol	Not yet deployed on the mainnet
VotingEscrowRoot.sol	Not yet deployed on the mainnet

4.2 Contracts Description

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

SmoothyMasterV1			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can modify state	onlyOwner
poolLength	External	-	-
add	Public	Can modify state	onlyOwner
set	Public	Can modify state	onlyOwner
getSmtyBlockReward	Public	-	-
pendingSMTY	External	-	-
massUpdatePools	Public	Can modify state	-
updatePool	Public	Can modify state	-
_updatePool	Internal	Can modify state	-
_updateWorkingAmount	Internal	Can modify state	-
deposit	External	Can modify state	claimSmty
createLock	External	Can modify state	-
_createLock	Internal	Can modify state	claimSmty
extendLock	External	Can modify state	claimSmty
increaseAmount	External	Can modify state	claimSmty
withdraw	Public	Can modify state	claimSmty
claim	Public	Can modify state	claimSmty
safeSMTYTransfer	Internal	Can modify state	-
getUserInfo	Public	-	-

SMTYToken			
Function Name	Visibility	Mutability	Modifiers
changeMinter	Public	Can modify state	onlyOwner
pause	Public	Can modify state	onlyOwner
mint	Public	Can modify state	-

UpgradeableOwnable

Function Name	Visibility	Mutability	Modifiers
_setOwner	Private	Can modify state	-
owner	Public	-	-
renounceOwnership	Public	Can modify state	onlyOwner
transferOwnership	Public	Can modify state	onlyOwner

SMTYToken			
Function Name	Visibility	Mutability	Modifiers
upgradeTo	External	Can modify state	onlyOwner
implementation	External	-	-

VotingEscrow			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can modify state	onlyOwner
decimals	Public	-	-
totalSupply	Public	-	-
balanceOf	Public	-	-
transfer	Public	Can modify state	-
allowance	Public	-	-
approve	Public	Can modify state	-
transferFrom	Public	Can modify state	-
amountOf	Public	-	-
endOf	Public	-	-
maxEnd	Public	-	-
createLock	External	Can modify state	-
_createLock	Internal	Can modify state	claimReward
addAmount	External	Can modify state	claimReward
extendLock	External	Can modify state	-
_extendLock	Internal	Can modify state	claimReward
withdraw	External	Can modify state	claimReward
claim	External	Can modify state	claimReward
_updateBalance	Internal	Can modify state	-
collectReward	Public	Can modify state	-
pendingReward	Public	-	-

SmoothyV1			
Function Name	Visibility	Mutability	Modifiers
name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
_getSoftWeight	Internal	-	-
_setSoftWeight	Internal	-	-
_getHardWeight	Internal	-	-
_setHardWeight	Internal	-	-
_getDecimalMulitiplier	Internal	-	-
_setDecimalMultiplier	Internal	-	-
_isYEnabled	Internal	-	-
_setYEnabled	Internal	-	-
_setTID	Internal	-	-
_getTID	Internal	-	-
pause	External	Can modify state	onlyOwner
unpause	External	Can modify state	onlyOwner
changeRewardCollector	External	Can modify state	onlyOwner
adjustWeights	External	Can modify state	onlyOwner
changeSwapFee	External	Can modify state	onlyOwner
changeRedeemFee	External	Can modify state	onlyOwner
changeAdminFeePct	External	Can modify state	onlyOwner
changeAdminInterestPct	External	Can modify state	onlyOwner
initialize	External	Can modify state	onlyOwner
addTokens	External	Can modify state	onlyOwner
setYEnabled	External	Can modify state	onlyOwner
lg2	Internal	-	-
_safeToInt128	Internal	-	-
_logApprox	Internal	-	-
_log	Internal	-	-
_getBalancesAndWeights	Internal	-	-
_getBalancesAndInfos	Internal	-	-
_getBalance	Internal	-	-
getBalance	Public	-	-
_normalizeBalance	Internal	-	-
_getCashBalance	Internal	-	-

_getBalanceDetail	Internal	-	-
_updateTotalBalanceWithNewYBalance	Internal	Can modify state	-
_rebalanceReserve	Internal	Can modify state	-
rebalanceReserve	External	Can modify state	nonReentrantAndUnpaused
_rebalanceReserveSubstract	Internal	Can modify state	-
_transferOut	Internal	Can modify state	-
_transferIn	Internal	Can modify state	-
_getMintAmount	Internal	-	-
getMintAmount	Public	-	-
mint	External	Can modify state	nonReentrantAndUnpaused
_redeemPenaltyFor	Internal	-	-
_redeemPenaltyForAll	Internal	-	-
_redeemPenaltyDerivativeForOne	Internal	-	-
_redeemPenaltyDerivativeForAll	Internal	-	-
_redeemFindOne	Internal	-	-
_redeemFind	Internal	-	-
_getRedeemByLpTokenAmount	Internal	-	-
getRedeemByLpTokenAmount	Public	-	-
redeemByLpToken	External	Can modify state	nonReentrantAndUnpaused
getSwapAmount	External	-	-
_getSwapAmount	Internal	-	-
swap	External	Can modify state	nonReentrantAndUnpaused
swapAll	External	Can modify state	nonReentrantAndUnpaused
_collectReward	Internal	Can modify state	-
collectReward	External	Can modify state	nonReentrantAndUnpaused
getTokenStats	Public	-	-

SmoothyV1Full			
Function Name	Visibility	Mutability	Modifiers
redeem	External	Can modify state	nonReentrantAndUnpaused

4.3 Code Audit

4.3.1 High-risk vulnerabilities

4.3.1.1 Risk of repeated contract initialization

In the SmoothyMasterV1 contract, the owner can initialize the contract through the initialize function to set the address of key parameters such as SMTYToken, startTime, and communityAddr. However, there is no restriction on the initialize function to prevent repeated initialization calls, which will cause the owner role to repeatedly initialize the contract through the initialize function. The same goes for VotingEscrow and SmoothyV1 contracts.

Fix suggestion: It is suggested to restrict the initialization function that does not allow repeated calls.

Code location: SmoothyMasterV1.sol, VotingEscrow.sol, SmoothyV1.sol

```
function initialize(
    SMTYToken _smty,
    IERC20 _veSMTY,
    address _teamAddr,
    address _communityAddr,
    uint256 _startTime
)
    external
    onlyOwner
{
    smty = _smty;
    veSMTY = _veSMTY;
    teamAddr = _teamAddr;
    communityAddr = _communityAddr;
    startTime = _startTime;
}
```

```
function initialize(IERC20 smty, IERC20 syUSD, address collector) external onlyOwner {
    _smty = smty;
    _syUSD = syUSD;
    _collector = collector;
}
```

```
function initialize(
    uint8 tid,
    uint256 bTokenAmount
)
    external
    onlyOwner
{
    require(tid < _ntokens, "Backed token not exists");
    uint256 info = _tokenInfos[tid];
    address addr = address(info);

    IERC20(addr).safeTransferFrom(
        msg.sender,
        address(this),
        bTokenAmount
    );
    _totalBalance = _totalBalance.add(bTokenAmount.mul(_normalizeBalance(info)));
    _mint(msg.sender, bTokenAmount.mul(_normalizeBalance(info)));
}
```

Fix status: The project party has transferred the owner authority of the SmoothyV1 contract to the timelock contract, and the VotingEscrow contract and the SmoothyMasterV1 contract have not yet been deployed on the mainnet.

4.3.2 Medium-risk vulnerabilities

4.3.2.1 Risk of excessive authority

In the SMTYToken contract, the minter role can mint tokens arbitrarily through the mint function. The owner role can arbitrarily modify the minter role address through the changeMinter function, which

will lead to the risk of excessive owner authority.

Fix suggestion: It is suggested to transfer the owner authority to community governance.

Code location: liquidity-mining/SMTYToken.sol

```
function changeMinter(address newMinter) public onlyOwner {
    _minter = newMinter;
}

function mint(address _to, uint256 _amount) public {
    require(_minter == msg.sender, "Only minter can mint");
    _mint(_to, _amount);
}
```

Fix status: The project party stated that the owner authority will be transferred to the timelock contract in the future, and the contract has not yet been deployed on the mainnet.

4.3.2.2 Denial of Service Risk

In the SmoothyMasterV1 contract, the user can update all pools through the massUpdatePools function, but it uses the for loop to update cyclically. If the number of pools exceeds too much, it will cause a DoS risk.

Fix suggestion: It is suggested to limit the number of pools to avoid this problem.

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

Fix status: No Fixed.

4.3.3 Low-risk vulnerabilities

4.3.3.1 The lockDuration does not match the lockEnd

In the SmoothyMasterV1 contract, the user can extend the mortgage lock period through the extendLock function. When reconfirming the lockDuration, take the new lock duration and the smaller value of MAX_TIME, but in the end, when determining the lockEnd, the _end parameter is still directly passed in. Assigned to lockEnd, if the new lock duration is greater than MAX_TIME, this will cause the lockDuration to not match the lockEnd.

Fix suggestion: It is suggested to recalculate lockEnd based on lockDuration.

Code location: liquidity-mining/SmoothyMasterV1.sol

```
function extendLock(uint256 _pid, uint256 _end) external claimSmty(_pid, msg.sender, block.timestamp) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = pool.userInfo[msg.sender];

    require(user.lockDuration != 0, "must be locked");
    require(_end <= block.timestamp + MAX_TIME, "end too long");
    require(_end > user.lockEnd, "new end must be greater");
    require(user.amount != 0, "user amount must be non-zero");

    user.lockDuration = Math.min(user.lockDuration.add(_end.sub(user.lockEnd)), MAX_TIME);
    user.lockEnd = _end;

    emit LockExtend(msg.sender, _pid, user.amount, user.lockEnd, user.lockDuration);
}
```

Fix status: No Fixed.

4.3.3.1 Inaccurate calculation of LP amount

In the SmoothyV1 contract, in order to save gas in mint, redeem, and swap operations, the

calculation using getMintAmount uses cached data for calculation, which will cause the final calculation result to be inconsistent with expectations.

Fix suggestion: Due to project design requirements, it is suggested that the project party manually invoke the update when the update is not performed to avoid this issue.

Code location: SmoothyV1.sol

```
function swap(
    uint256 bTokenIdxIn,
    uint256 bTokenIdxOut,
    uint256 bTokenInAmount,
    uint256 bTokenOutMin
)
    external
    nonReentrantAndUnpaused
{
    uint256 infoIn = _tokenInfos[bTokenIdxIn];
    uint256 infoOut = _tokenInfos[bTokenIdxOut];
    (
        uint256 bTokenOutAmount,
        uint256 adminFee
    ) = _getSwapAmount(infoIn, infoOut, bTokenInAmount);
    require(bTokenOutAmount >= bTokenOutMin, "Returned bTokenAmount < asked");

    _transferIn(infoIn, bTokenInAmount);
    _transferOut(infoOut, bTokenOutAmount, adminFee);

    emit Swap(
        msg.sender,
        bTokenIdxIn,
        bTokenIdxOut,
        bTokenInAmount,
        bTokenOutAmount
    );
}

function mint(
    uint256 bTokenIdx,
```

```
        uint256 bTokenAmount,
        uint256 lpTokenMintedMin
    )
        external
        nonReentrantAndUnpaused
    {
        uint256 lpTokenAmount = getMintAmount(bTokenIdx, bTokenAmount);
        require(
            lpTokenAmount >= lpTokenMintedMin,
            "lpToken minted should >= minimum lpToken asked"
        );

        _transferIn(_tokenInfos[bTokenIdx], bTokenAmount);
        _mint(msg.sender, lpTokenAmount);
        emit Mint(msg.sender, bTokenAmount, lpTokenAmount);
    }

    function redeemByLpToken(
        uint256 bTokenIdx,
        uint256 lpTokenAmount,
        uint256 bTokenMin
    )
        external
        nonReentrantAndUnpaused
    {
        (uint256 bTokenAmount, uint256 totalBalance, uint256 adminFee) = _getRedeemByLpTokenAmount(
            bTokenIdx,
            lpTokenAmount
        );
        require(bTokenAmount >= bTokenMin, "bToken returned < min bToken asked");

        // Make sure _totalBalance == sum(balances)
        _collectReward(totalBalance);

        _burn(msg.sender, lpTokenAmount);
        _transferOut(_tokenInfos[bTokenIdx], bTokenAmount, adminFee);

        emit Redeem(msg.sender, bTokenAmount, lpTokenAmount);
    }
```

Fix status: No Fixed.

4.3.4 Enhancement Suggestions

4.3.4.1 The change of LP pool weights affects users' income

In the SmoothyMasterV1 contract, when the Owner calls the add function and the set function to add a new pool or reset the pool weight, all LP pool weights will change accordingly. The Owner can update all pools before adjusting the weight by passing in the _withUpdate parameter with a value of true to ensure that the user's income before the pool weight is changed will not be affected by the adjustment of the pool weight, but if the value of the _withUpdate parameter is false, then All pools will not be updated before the pool weight is adjusted, which will cause the user's income to be affected before the pool weight is changed.

Fix suggestion: 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.

Code location: liquidity-mining/SmoothyMasterV1.sol

```
    function add(
        uint256 _allocPoint,
        IERC20 _lpToken,
        bool _withUpdate
    )
        public
        onlyOwner
    {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardTime = block.timestamp > startTime ? block.timestamp : startTime;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
```

```
            lastRewardTime: lastRewardTime,
            accSMTYPerShare: 0,
            workingSupply: 0
        }));
    }
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    )
        public
        onlyOwner
    {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }
```

Fix status: No Fixed.

4.3.4.2 Loss of precision issue

In the SmoothyMasterV1 contract, when using the _updateWorkingAmount function to calculate the number of workingAmount users participate in mining, divide first and then multiply, which will result in loss of accuracy.

Fix suggestion: It is suggested to multiply and then divide to avoid this issue

Code location: SmoothyMasterV1.sol

```
    function _updateWorkingAmount(
        uint256 _pid,
        address _account
    ) internal
    {
        PoolInfo storage pool = poolInfo[_pid];
```

```
        UserInfo storage user = pool.userInfo[_account];

        uint256 lim = user.amount.mul(4) / 10;

        uint256 votingBalance = veSMTY.balanceOf(_account);
        uint256 totalBalance = veSMTY.totalSupply();

        if (totalBalance != 0) {
            uint256 lsupply = pool.lpToken.totalSupply();
            lim = lim.add(lsupply.mul(votingBalance).div(totalBalance).mul(6) / 10);
        }

        uint256 veAmount = Math.min(user.amount, lim);

        uint256 timelockBoost = user.lockDuration.mul(MAX_EXTRA_BOOST).div(MAX_TIME).add(1e18);
        uint256 newWorkingAmount = veAmount.mul(timelockBoost).div(1e18);

        pool.workingSupply = pool.workingSupply.sub(user.workingAmount).add(newWorkingAmount);
        user.workingAmount = newWorkingAmount;

        emit WorkingAmountUpdate(_account, _pid, user.workingAmount, pool.workingSupply);
    }
```

Fix status: No Fixed.

4.3.4.3 Unrecoverable issue of pool imbalance

In the SmoothyV1 contract, when the user performs operations such as recharge, redemption, and exchange, the penalty mechanism will be triggered when the weight of the coin exceeds the soft cap, but the contract does not have an incentive mechanism to perform exchange operations to reduce the proportion of the token pool. If the token pool is maliciously manipulated to exceed the soft cap, it may be difficult for the token pool to return to normal due to no incentive mechanism, which will affect normal business use.

Fix suggestion: It is suggested to add an incentive mechanism in an unbalanced state to avoid this problem.

Code location: SmoothyV1.sol

Fix status: No Fixed.

4.3.4.4 Risk of Potential Token Transfer Failure

In the SmoothyV1 contract, when the user deposits the token, the safeTransferFrom function is used to transfer the corresponding token, and the safeTransfer function is used to transfer the token when withdrawToken. The safeTransferFrom function and safeTransfer function will check the returned success and data , If the connected token defines the return value, but does not return according to the EIP20 specification, the user will not be able to pass the check here, resulting in the tokens being unable to be transferred in or out.

Fix suggestion: It is suggested that when docking new tokens, the project party should check whether its writing complies with EIP20 specifications.

Code location: SmoothyV1.sol

Fix status: No Fixed.

4.3.4.5 Token compatibility issue

In the SmoothyV1 contract, under the condition that each pool is stable, the exchange operation will be performed in a 1:1 manner. However, if the project is connected to a stable rebase algorithm, the number of tokens in the pool will be changed when it undergoes deflation, resulting in an unexpected

number of users during the exchange.

Fix suggestion: It is suggested to strictly evaluate the algorithm model of stablecoins to avoid this risk when accessing stablecoins.

Code location: SilMaster.sol

Fix status: No Fixed.

5. Audit Result

5.1 Conclusion

Audit Result : Some Risks

Audit Number : 0X002103230001

Audit Date : Mar. 23, 2021

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist Team analysis tool audit of the codes for security issues. There are ten security issues found during the audit. There are one high-risk vulnerabilities, two medium-risk vulnerabilities and two low-risk vulnerabilities. We also provide five enhancement suggestions. The project party has fixed the issue of excessive owner authority of the SmoothyV1 contract. The remaining issues have not been fixed yet, so there are still some risks.

6. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist