Making Web3 Space Safer for Everyone

KALOS

StableComp

Security Assessment

Published on : 30 Mar. 2023
Version v1.1

Security Report Published by KALOS

v1.1 30 Mar. 2023

Auditor : Jade

hojung han

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	1	1	-	-
High	3	3	-	-
Medium	1	1	-	-
Low	1	1	-	-
Tips	-	-	-	-

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

KALOS is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

Having secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges, KALOS is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the project developed by the StableComp team. KALOS conducted the audit focusing on whether the system created by the StableComp team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the project.

In detail, we have focused on the following

- Denial of Service
- Access Control of Various Storage Variables
- Access Control of Important Functions
- Freezing of User Assets
- Theft of User Assets
- Manipulation of Yield Calculation
- Unhandled Exceptions

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/stablecomp/stablecomp-contracts).

The commit of the code used for this Audit is “f5bb17e595a1320f3f107b039653f6d384c5ed45”,

Audit Timeline

Date	Event
2023/02/13	Audit Initiation (StableComp)
2023/03/13	Delivery of v1.0 report.
2023/03/30	Delivery of v1.1 report

Findings

KALOS found 1 Critical, 3 High, 1 medium and 1 Low severity issues.

Severity	Issue	Status
Critical	First Depositor Front-Running	(Fixed - v1.1)
High	minimum output parameter does not exist in UniswapV2's swapExactTokensForTokens function	(Fixed - v1.1)
High	The veScomp contract can freeze any arbitrary user's Scomp tokens by mismanaging allowances	(Fixed - v1.1)
High	minimum output parameter does not exist in Curve _add_liquidity_single_coin function	(Fixed - v1.1)
Medium	Potential Calculation Issue with Rewards in MasterchefScomp Contract after endBlock	(Fixed - v1.1)
Low	Potential Issue with Reward Token Management in MasterchefScomp Contract	(Fixed - v1.1)

Remarks

The KALOS team has not received information about the contract used as a converter in the SCompController contract, and the contract used as a converter is not within the scope of this audit. We have also excluded Farm.sol from the Audit Scope as the project team has confirmed that it will not be used.

OVERVIEW

Protocol overview

• **SComp**
SComp Token is the native token of StableComp with a limited supply of 200 million tokens. It supports all the basic functions defined in ERC20, such as transfers.

• **veScomp**
The veScomp token is issued on the StableComp platform and can be obtained by locking up Scomp for a certain period of time. By holding veScomp tokens, users can receive more rewards from the MasterchefScomp contract within the StableComp platform.

• **OneClick**
The OneClick Contract is a contract that enables token swapping and liquidity providing to be processed at a single endpoint in conjunction with the SCompVault Contract.

• **FeeDistribution**
The FeeDistribution contract is an Ethereum smart contract that distributes fees to users who lock their SComp tokens in the veScomp, based on a linear approximation of the amount of veScomp locked over time. Users can claim their share of the fees by calling the "claim" function or the "claim_many" function for multiple claims.

• **SurplusConverterUniV2Sushi**
Surplus Converter is a smart contract that converts the surplus of the Stablecomp protocol into a specific token via Uniswap or Sushiswap, providing the best price by comparing the prices of the two DEXs. It periodically executes and purchases a specific token using surplus funds from the Stablecomp protocol to transfer to the FeeDistributor or other Surplus Converter. It inherits the BaseSurplusConverter smart contract and is used in conjunction with the FeeDistributor. The Surplus Converter only works if the user has the Whitelisted role.

• **SCompStrategyV1.0**

The SCompStrategyV1 contract allows yield farming by depositing funds into Convex Finance's staking pools and depositing and staking the underlying assets into the appropriate vault. It includes functions for realizing gains and withdrawing funds, as well as several internal functions for interacting with assets, handling fees, and setting important variables.

• **SCompController**
The SCompController contract provides functionality to deposit sCOMP tokens into the Stablecomp protocol and generate revenue using the deposited tokens. It interacts with the SCompVault and ScompStrategy contracts to manage assets, deposit tokens held by SCompVault into SCompStrategy, generate revenue, and transfer back to SCompVault.

• **SCompVault**
SCompVault is a smart contract that enables users to deposit sCOMP tokens into the Stablecomp protocol and generate profits using the deposited tokens. It also includes a function to impose fees on deposits. This smart contract is implemented by inheriting the ERC20 token.

Scope

```
├── SCompController.sol
├── SCompVault.sol
├── abstract
│   └── BaseStrategy.sol
├── farmBooster
│   ├── MasterchefScomp.sol
│   ├── StableCompToken.sol
│   └── veScomp.vy
├── interface
│   ├── IBEP20.sol
│   ├── IBaseRewardsPool.sol
│   ├── IBoostContract.sol
│   ├── IBooster.sol
│   ├── ICakePool.sol
│   ├── IController.sol
│   ├── IConverter.sol
│   ├── ICrvDepositor.sol
│   ├── ICurveFi.sol
│   ├── ICurveGauge.sol
│   ├── ICurvePool.sol
│   ├── ICurveRegistry.sol
│   ├── ICurveRegistryAddressProvider.sol
│   ├── ICvxRewardsPool.sol
│   ├── IERC20.sol
│   ├── IFeeDistributor.sol
│   ├── IMasterChef.sol
│   ├── IMasterChefV2.sol
│   ├── IOneSplitAudit.sol
│   ├── ISCompVault.sol
│   ├── ISettV4.sol
│   ├── IStaker.sol
│   ├── IStrategy.sol
│   ├── IUniswapRouter.sol
│   ├── IUniswapRouterV2.sol
│   ├── IUniswapV2Factory.sol
│   ├── IVCake.sol
│   └── IVotingEscrow.sol
├── libraries
│   └── IterateMapping.sol
├── manageFee
│   ├── BaseSurplusConverter.sol
│   ├── FeeDistribution.vy
│   └── SurplusConverterUniV2Sushi.sol
├── oneClick
│   └── OneClick.sol
├── strategies
│   └── SCompStrategyV1.0.sol
└── utility
    ├── BaseSwapper.sol
    ├── CurveSwapper.sol
    ├── Faucet.sol
    ├── GenericERC20.sol
    ├── SCompAccessControl.sol
    ├── SCompTimeLockController.sol
    ├── SafeBEP20.sol
    ├── TokenSwapPathRegistry.sol
    └── UniswapSwapper.sol
```

Access Controls

Access control in contracts is achieved using modifiers and inline require statements. The access control of Stablecomp refers to the following variables.

- strategist
- governance
- controller
- owner

strategist : The address designated as a strategist can perform the following functions

- SCompController.sol#setVault(address, address)
- SCompController.sol#setStrategy(address, address)
- SCompController.sol#setConverter(address, address, address)
- SCompController.sol#withdrawAll(address)
- SCompController.sol#inCaseTokensGetStuck(address, uint256)
- SCompController.sol#inCaseStrategyTokenGetStuck(address, address)

governance : The address designated as a governance can perform the following functions

- BaseStrategy.sol#pause()
- SCompController.sol#approveStrategy(address, address)
- SCompController.sol#revokeStrategy(address, address)
- SCompController.sol#setRewards(address)
- SCompController.sol#setVault(address, address)
- SCompController.sol#setStrategy(address, address)
- SCompController.sol#setConverter(address, address, address)
- SCompController.sol#withdrawAll(address)
- SCompController.sol#inCaseTokensGetStuck(address, uint256)
- SCompController.sol#inCaseStrategyTokenGetStuck(address, address)
- BaseStrategy.sol#setController(address)
- BaseStrategy.sol#setWithdrawalMaxDeviationThreshold(uint256)
- BaseStrategy.sol#unpause()
- SCompStrategyV1.0.sol#setPid(uint256)
- SCompStrategyV1.0.sol#setCurvePoolSwap(address)
- SCompStrategyV1.0.sol#setTokenCompound(address, uint256)
- SCompAccessControl.sol#setStrategist(address)
- SCompAccessControl.sol#setGovernance(address)
- SCompAccessControl.sol#setTimeLockController(address)
- BaseStrategy.sol#deposit()

controller : The address designated as a controller can perform the following functions

- BaseStrategy.sol#withdrawAll()
- BaseStrategy.sol#withdraw(uint256)
- BaseStrategy.sol#withdrawOther(address)
- SCompVault.sol#harvest(address, uint256)
- BaseStrategy.sol#deposit()

timeLockController : The address designated as a owner can perform the following functions

- BaseStrategy.sol#setWithdrawalFee(uint256)
- BaseStrategy.sol#setPerformanceFeeStrategist(uint256)
- BaseStrategy.sol#setPerformanceFeeGovernance(uint256)
- OneClick.sol#setOneclickFee(address, uint256)

owner : The address designated as a owner can perform the following functions

- MasterchefScomp.sol#add(uint256, address, bool)
- MasterchefScomp.sol#set(uint256, uint256, bool)
- OneClick.sol#setTimeLockController(address)
- OneClick.sol#recoverTokens(address, uint256)

FINDINGS

1. minimum output parameter does not exist in UniswapV2's `swapExactTokensForTokens` function

ID: StableComp-01
Type: Logic Error
File: contracts/utility/UniswapSwapper.sol

Severity: High
Difficulty: Medium

Issue

During the process of reinvesting profits received, some profits may be lost.

The code below is problematic.

```
function harvest() external whenNotPaused returns (uint256) {
        uint256 idleWant = IERC20(want).balanceOf(address(this));
        uint256 totalWantBefore = balanceOf();

        // 1. Withdraw accrued rewards from staking positions (claim unclaimed positions as
well)
        baseRewardsPool.getReward(address(this), true);


        // 3. Sell 100% of accured rewards for underlying
        uint crvToSell = crvToken.balanceOf(address(this));
        if(crvToSell > 0)  {
            uint fee = takeFee(crv, crvToSell);
            crvToSell = crvToSell.sub(fee);

            _swapExactTokensForTokens(
                sushiswap,
                crv,
                crvToSell,
                getTokenSwapPath(crv, tokenCompoundAddress)
            );
        }

        uint cvxToSell = cvxToken.balanceOf(address(this));
        if(cvxToSell > 0)  {
            uint fee = takeFee(cvx, cvxToSell);
            cvxToSell = cvxToSell.sub(fee);

            _swapExactTokensForTokens(
                sushiswap,
                cvx,
                cvxToSell,
                getTokenSwapPath(cvx, tokenCompoundAddress)
            );
        }
        ...
}
...
function _swapExactTokensForTokens(
        address router,
        address startToken,
        uint256 balance,
        address[] memory path
    ) internal {
        _safeApproveHelper(startToken, router, balance);
        IUniswapRouterV2(router).swapExactTokensForTokens(
            balance,
            0, // This code becomes vulnerable to sandwich trading attacks by passing 0 as a
parameter
            path,
            address(this),
            block.timestamp
        );
    }
```

https://github.com/stablecomp/stablecomp-contracts/blob/f5bb17e595a1320f3f107b039653f6d384c5ed45/contracts/utility/UniswapSwapper.sol

If the minimum output is set, it is safe from sandwich trading via MEV. However, since the minimum output is not set, depending on the liquidity of the Dex pool, more than 90% of the tokens being swapped may be lost.

Recommendation

We recommend calculating the minimum output off-chain, passing it as a parameter to the `harvest` and `_swapExactTokensForTokens` functions, and ultimately setting the parameter related to minimum output of `swapExactTokensForTokens`.

Fix Comment

Initially, we recommended the Stablecomp Team to use the Chainlink Oracle to set a minimum output when the harvest function is executed. However, there was no Chainlink Price Feed available for the following stablecoins:

- agEur (0x1a7e4e63778B4f12a199C062f3eFdD288afCBce8)
- dola (0x865377367054516e17014CcdED1e7d814EDC9ce4)

- EUROC (0x1aBaEA1f7C830bD89Acc67eC4af516284b1bC33c)
- EURS (0xdB25f211AB05b1c97D595516F45794528a807ad8)
- ibEur (0x96E61422b6A9bA0e068B6c5ADd4fFaBC6a4aae27)
- sEur (0xD71eCFF9342A5Ced620049e616c5035F1dB98620)
- 3crv (0x6c3F90f043a72FA612cbac8115EE7e52BDe6E490)
- usdd (0x0C10bF8FcB7Bf5412187A595ab97a3609160b5c6)

As an alternative, we suggested the administrator investigate prices off-chain and then input the price data on-chain, using that price as the basis for calculating the minimum output.

2. First Depositor Front-Running

ID: StableComp-02
Type: Logic Error
File: contracts/SCompVault.sol

Severity: Critical
Difficulty: Medium

Issue

The first depositor of the SCompVault contract may lose most of the assets they deposit due to malicious MEV attackers.

The following code is the code that executes when a user deposits and withdraws funds in the SCompVault contract.

```
function balance() public view returns (uint256) {
        return
token.balanceOf(address(this)).add(IController(controller).balanceOf(address(token)));
    }
...
function deposit(uint256 _amount) public returns(uint) {
        uint256 _pool = balance();

        // deposit fee
        if(depositFee > 0) {
            uint256 amountFee =
            _amount.mul(depositFee).div(
                MAX_FEE
            );
            token.safeTransferFrom(msg.sender, treasuryFee, amountFee);
            _amount = _amount - amountFee;
        }

        uint256 _before = token.balanceOf(address(this));
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint256 _after = token.balanceOf(address(this));
        _amount = _after.sub(_before); // Additional check for deflationary tokens
        uint256 shares = 0;
        if (totalSupply() == 0) {
            shares = _amount;
        } else {
            shares = (_amount.mul(totalSupply())).div(_pool);
        }
        _mint(msg.sender, shares);

        emit Deposit(msg.sender, shares, block.timestamp);

        return shares;
    }
...
function withdraw(uint256 _shares) public returns(uint) {
        uint256 r = (balance().mul(_shares)).div(totalSupply());
        _burn(msg.sender, _shares);

        // Check balance
        uint256 b = token.balanceOf(address(this));
        if (b < r) {
            uint256 _withdraw = r.sub(b);
            IController(controller).withdraw(address(token), _withdraw);
            uint256 _after = token.balanceOf(address(this));
            uint256 _diff = _after.sub(b);
            if (_diff < _withdraw) {
                r = b.add(_diff);
            }
        }

        token.safeTransfer(msg.sender, r);
        emit Withdraw(msg.sender, r, block.timestamp);

        return r;
    }
```

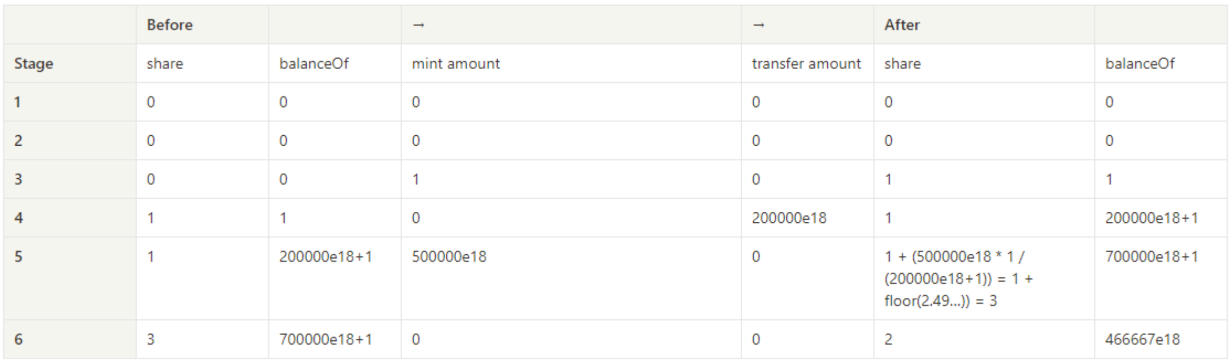
https://github.com/stablecomp/stablecomp-contracts/blob/f5bb17e595a1320f3f107b039653f6d384c5ed45/contracts/SCompVault.sol

Some MEV-unfriendly chains (such as Arbitrum) inherently block this vulnerability. However, MEV is allowed on the Ethereum Mainnet where StableComp contracts are deployed, and this vulnerability may occur.

In Solidity, integer division sometimes produces unintended results because it does not express decimal points. Let's assume that someone makes the first deposit to the pool.

In this code, the Attack Scenario consists of the below steps.

1. The victim deposits token that amount is 500000e18
2. attacker sees the victim's pending transaction in mempool.
3. The attacker deposits token that amount is 1 wei using `deposit` function
4. The attacker transfers 200000e18 tokens to the pool contract before the victim's pending transaction is executed.
5. the victim's pending transaction is executed.
6. The attacker withdraws the asset token more than the attacker's first deposit.



	Before		→	→	After	
Stage	share	balanceOf	mint amount	transfer amount	share	balanceOf
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	1	0	1	1
4	1	1	0	200000e18	1	200000e18+1
5	1	200000e18+1	500000e18	0	1 + (500000e18 * 1 / (200000e18+1)) = 1 + floor(2.49...)) = 3	700000e18+1
6	3	700000e18+1	0	0	2	466667e18

Recommendation

We recommend the following updates.

- Need to enforce a minimum deposit that can not be withdrawn.
- mint some of the initial amount to the zero address. Since most legit first depositors will mint thousands of shares, this will not be a big cost for them.

The above recommendation increases the cost for attack greatly.

Uniswap V2 also uses the above recommendation. Refer to the codes below.
https://github.com/Uniswap/v2-core/blob/ee547b17853e71ed4e0101ccfd52e70d5acded58/contracts/UniswapV2Pair.sol#L109-L131

Fix Comment

Enforced a minimum deposit requirement that cannot be withdrawn, following the recommendation to mint some of the initial amount to the zero address.

This approach was deemed reasonable since most legitimate first depositors will mint thousands of shares, making it a negligible cost for them.

Confirmed successful implementation of the suggestion.

3. The veScomp contract can freeze any arbitrary user's Scomp tokens by mismanaging allowances

ID: StableComp-03	Severity: High
Type: Logic Error	Difficulty: Low
File: contracts/farmBooster/veScomp.vy	

Issue

If other contracts or EOAs referring to the veScomp contract maintain allowances of Scomp tokens greater than zero, the veScomp contract can impose a lock-up for up to two years on the user without their consent.

The following code retrieves Scomp tokens from a specific EOA or contract by a third party and issues veTokens to the owner of Scomp tokens.

```
@external
@nonreentrant('lock')
def deposit_for(_addr: address, _value: uint256):
    """
    @notice Deposit `_value` tokens for `_addr` and add to the lock
    @dev Anyone (even a smart contract) can deposit for someone else, but
         cannot extend their locktime and deposit for a brand new user
    @param _addr User's wallet address
    @param _value Amount to add to user's lock
    """
    _locked: LockedBalance = self.locked[_addr]

    assert _value > 0  # dev: need non-zero value
    assert _locked.amount > 0, "No existing lock found"
    assert _locked.end > block.timestamp, "Cannot add to expired lock. Withdraw"

    self._deposit_for(_addr, _value, 0, self.locked[_addr], DEPOSIT_FOR_TYPE)
...

@internal
def _deposit_for(_addr: address, _value: uint256, unlock_time: uint256, locked_balance:
LockedBalance, type: int128):
    """
    @notice Deposit and lock tokens for a user
    @param _addr User's wallet address
    @param _value Amount to deposit
    @param unlock_time New time when to unlock the tokens, or 0 if unchanged
    @param locked_balance Previous locked amount / timestamp
    """
    _locked: LockedBalance = locked_balance
    supply_before: uint256 = self.supply

    self.supply = supply_before + _value
    old_locked: LockedBalance = _locked
    # Adding to existing lock, or if a lock is expired - creating a new one
    _locked.amount += convert(_value, int128)
    if unlock_time != 0:
        _locked.end = unlock_time
    self.locked[_addr] = _locked

    # Possibilities:
    # Both old_locked.end could be current or expired (>/< block.timestamp)
    # value == 0 (extend lock) or value > 0 (add to lock or extend lock)
    # _locked.end > block.timestamp (always)
    self._checkpoint(_addr, old_locked, _locked)

    if _value != 0:
        assert ERC20(self.token).transferFrom(_addr, self, _value)

    log Deposit(_addr, _value, _locked.end, type, block.timestamp)
    log Supply(supply_before, supply_before + _value)
```

https://github.com/stablecomp/stablecomp-contracts/blob/f5bb17e595a1320f3f107b039653f6d384c5ed45/contracts/farmBooster/veScomp.vy

Often, there are cases where the allowance is set to a value greater than zero instead of being kept at zero.

In such cases, a user may suffer losses if their assets in a specific EOA or contract are frozen for up to two years by malicious users.

Recommendation

We recommend removing the `deposit_for` function.

Fix Comment

Implemented the recommendation to remove the deposit_for function in order to prevent asset freezing.

After careful consideration, this approach was deemed necessary to ensure the security and reliability of the platform.

Confirmed successful removal of the function in accordance with the suggestion.

4. Potential Calculation Issue with Rewards in MasterchefScomp Contract after endBlock

ID: StableComp-04
Severity: Medium
Type: Logic Error
Difficulty: Low
File: contracts/farmBooster/MasterchefScomp.sol

Issue

Users may receive additional rewards beyond the predetermined rewards, which may result in losses for the project team.

The following code contains a potential issue where users may receive additional rewards after endBlock.

```
function updatePool(uint256 _pid) public returns (PoolInfo memory pool) {
        pool = poolInfo[_pid];
        uint256 lastBlock = block.number < endBlock ? block.number : endBlock;

        if (lastBlock > pool.lastRewardBlock) {
            uint256 lpSupply = pool.totalBoostedShare;

            if (lpSupply > 0 && totalAllocPoint > 0) {
                uint256 multiplier = block.number.sub(pool.lastRewardBlock);
                uint256 tokenReward =
multiplier.mul(tokenPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
                pool.accTokenPerShare =
pool.accTokenPerShare.add((tokenReward.mul(ACC_TOKEN_PRECISION).div(lpSupply)));
            }
            pool.lastRewardBlock = block.number;
            poolInfo[_pid] = pool;
            emit UpdatePool(_pid, pool.lastRewardBlock, lpSupply, pool.accTokenPerShare);
        }
    }
```

https://github.com/stablecomp/stablecomp-contracts/blob/f5bb17e595a1320f3f107b039653f6d384c5ed45/contracts/farmBooster/MasterchefScomp.sol

The multiplier calculation includes blocks beyond `endBlock` because the `pool.lastRewardBlock` is subtracted from `block.number`, when `lastBlock` should be used instead of `block.number`.

Recommendation

We recommend subtracting `pool.lastRewardBlock` from `lastBlock` instead of subtracting it from `block.number` when calculating the multiplier.

```
function updatePool(uint256 _pid) public returns (PoolInfo memory pool) {
        pool = poolInfo[_pid];
        uint256 lastBlock = block.number < endBlock ? block.number : endBlock;

        if (lastBlock > pool.lastRewardBlock) {
            uint256 lpSupply = pool.totalBoostedShare;

            if (lpSupply > 0 && totalAllocPoint > 0) {
                uint256 multiplier = lastBlock.sub(pool.lastRewardBlock);
                uint256 tokenReward =
multiplier.mul(tokenPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
                pool.accTokenPerShare =
pool.accTokenPerShare.add((tokenReward.mul(ACC_TOKEN_PRECISION).div(lpSupply)));
            }
            pool.lastRewardBlock = block.number;
            poolInfo[_pid] = pool;
            emit UpdatePool(_pid, pool.lastRewardBlock, lpSupply, pool.accTokenPerShare);
        }
    }
```

Fix Comment

Applied the suggestion to subtract pool.lastRewardBlock from lastBlock instead of subtracting it from block.number when calculating the multiplier in the updatePool function.

This change was deemed necessary to improve accuracy in the calculation and ensure consistency with the intended logic of the code.

Confirmed successful implementation of the recommendation.

5. Potential Issue with Reward Token Management in MasterchefScomp Contract

ID: StableComp-05
Severity: Tips
Type: N/A
Difficulty: N/A
File: contracts/farmBooster/MasterchefScomp.sol

Issue

Users may receive less or no rewards in specific situations.

The following code is a function that transfers rewards to users.

```
function settlePendingToken(
        address _user,
        uint256 _pid,
        uint256 _boostMultiplier
    ) internal {
        UserInfo memory user = userInfo[_pid][_user];

        uint256 boostedAmount = user.amount.mul(_boostMultiplier).div(BOOST_PRECISION);
        uint256 accToken =
boostedAmount.mul(poolInfo[_pid].accTokenPerShare).div(ACC_TOKEN_PRECISION);
        uint256 pending = accToken.sub(user.rewardDebt);
        // SafeTransfer TOKEN
        _safeTransfer(_user, pending);
    }

    /// @notice Safe Transfer TOKEN.
    /// @param _to The TOKEN receiver address.
    /// @param _amount transfer TOKEN amounts.
    function _safeTransfer(address _to, uint256 _amount) internal {
        if (_amount > 0) {
            uint256 balance = TOKEN.balanceOf(address(this));
            if (balance < _amount) {
                _amount = balance;
            }
            TOKEN.safeTransfer(_to, _amount);
        }
    }
```

https://github.com/stablecomp/stablecomp-contracts/blob/f5bb17e595a1320f3f107b039653f6d384c5ed45/contracts/farmBooster/MasterchefScomp.sol

If the amount in the `_safeTransfer` function is greater than the amount of tokens owned by the current contract, only the amount of tokens owned by the contract is sent.

In this case, the user may receive a small amount of tokens they are entitled to or may not receive any at all.

Recommendation

To `_safeTransfer` function, add the following require statement.

We recommend adding the following require statement to the `_safeTransfer` function. This ensures that the `_safeTransfer` function fails when there is insufficient balance in the MasterchefScomp contract or the amount to be paid is too small.

```
function _safeTransfer(address _to, uint256 _amount) internal {
        if (_amount > 0) {
            uint256 balance = TOKEN.balanceOf(address(this));
            require(balance>=_amount, "insufficient balance in contract");
            TOKEN.safeTransfer(_to, _amount);
        }
    }
```

Fix Comment

Implemented the recommendation to add a require statement to the _safeTransfer function to ensure it fails when there is insufficient balance in the MasterchefScomp contract or the amount to be paid is too small.

This change was deemed necessary to prevent potential errors and ensure the safety of the transaction. Confirmed successful implementation of the suggestion.

6. minimum output parameter does not exist in Curve `_add_liquidity_single_coin` function

ID: StableComp-06
Type: Logic Error
File: contracts/utility/CurveSwapper.sol

Severity: High
Difficulty: Medium

Issue

During the process of reinvesting profits received, some profits may be lost.

The code below is problematic.

```
function harvest() external whenNotPaused returns (uint256) {
        ...
        if (tokenCompoundToDeposit > 0) {
            // Add liquidity
            _add_liquidity_single_coin(
                curvePool.swap,
                want,
                tokenCompoundAddress,
                tokenCompoundToDeposit,
                curvePool.tokenCompoundPosition,
                curvePool.numElements,
                0
            );
            wantGained = IERC20(want).balanceOf(address(this)).sub(
                idleWant
            );
        }

        // Deposit remaining want (including idle want) into strategy position
        uint256 wantToDeposited =
        IERC20(want).balanceOf(address(this));

        if (wantToDeposited > 0) {
            _deposit(wantToDeposited);
        }

        uint256 totalWantAfter = balanceOf();
        require(totalWantAfter >= totalWantBefore, "SCompStrategy: want-decreased");

        emit Harvest(wantGained, block.number);
        return wantGained;
    }
...
function _add_liquidity_single_coin(
        address swap,
        address pool,
        address inputToken,
        uint256 inputAmount,
        uint256 inputPosition,
        uint256 numPoolElements,
        uint256 min_mint_amount
    ) internal {
        _safeApproveHelper(inputToken, swap, inputAmount);
        if (numPoolElements == 2) {
            uint256[2] memory convertedAmounts;
            convertedAmounts[inputPosition] = inputAmount;
            ICurveFi(swap).add_liquidity(convertedAmounts, min_mint_amount);
        } else if (numPoolElements == 3) {
            uint256[3] memory convertedAmounts;
            convertedAmounts[inputPosition] = inputAmount;
            ICurveFi(swap).add_liquidity(convertedAmounts, min_mint_amount);
        } else if (numPoolElements == 4) {
            uint256[4] memory convertedAmounts;
            convertedAmounts[inputPosition] = inputAmount;
            ICurveFi(swap).add_liquidity(convertedAmounts, min_mint_amount);
        } else {
            revert("Bad numPoolElements");
        }
    }
```

https://github.com/stablecomp/stablecomp-contracts/blob/f5bb17e595a1320f3f107b039653f6d384c5ed45/contracts/utility/CurveSwapper.sol

If the minimum output is set, it is safe from sandwich trading via MEV.

However, since the minimum output is not set, the contract may lose a portion of the interest income.

Recommendation

We recommend calculating the minimum output off-chain, passing it as a parameter to the harvest and `_add_liquidity_single_coin` functions, and ultimately setting the parameter related to minimum output of `_add_liquidity_single_coin`.

Fix Comment

We recommended calculating the minimum output using the get_virtual_price function within the curve.fi contract. However, for certain LP tokens, the get_virtual_price function was not available, so the contract was ultimately patched to use the lp_price function instead.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure codes, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	**Medium**	**High**
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document