SLOWMIST

Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.05.08, the SlowMist security team received the Sable Finance team's security audit application for Sable 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 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.
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:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This is a modified protocol based on liquidy

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing event record	Malicious Event Log Audit	Suggestion	Fixed
N2	Missing Logic Judgments for	Design Logic Audit	Suggestion	Fixed

NO	Title	Category	Level	Status
	Other Cases			
N3	Award calculation omission	Design Logic Audit	High	Fixed
N4	The Dos issue	Denial of Service Vulnerability	Medium	Fixed
N5	Reward calculation exception problem	Design Logic Audit	Critical	Fixed
N6	Memory variable exploit	Design Logic Audit	Critical	Fixed
N7	Compiler version is too low	Integer Overflow and Underflow Vulnerability	Suggestion	Fixed
N8	Variable Setting Scope Recommendations	Design Logic Audit	Suggestion	Fixed
N9	Risk of initial operation	Authority Control Vulnerability Audit	Low	Acknowledged
N10	Race Conditions Vulnerability issue	Race Conditions Vulnerability	Critical	Fixed
N11	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N12	Stake and unstake recommendations	Design Logic Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/Sable-Finance/sable_audit

commit: d1112c91adae149c1a7b62f2891e4324b8dfce34

Project:https://github.com/Sable-Finance/sable_audit

commit: d1112c91adae149c1a7b62f2891e4324b8dfce34

Audit scope: The part that differs from liquidity.

(PriceFeed.sol,SystemState.sol,StabilityPool.sol,OracleRateCalculation.sol. TroveManager.sol, BorrowerOperations.sol, TroveHelper.sol, TimeLock.sol, CommunityIssuance.sol, SableStakingV2.sol , SableRewarder.sol)

ReviewCommit:

e13f33f6bff3d6f1d4875a816b7606381d1b33e4

The main network address of the contract is as follows:

Deployed on bsc:

"timeLock": "0x638675B7C2e056917567571307c6F6a7D69A258A",

"activePool": "0x0cCb12C9fB1e1252E60d29aC5c4fDc0640edD72C",

"borrowerOperations": "0xa49BEC2146fBeeA7314cdbe0Fd222419B0c0602f",

"troveManager": "0xEC035081376ce975Ba9EAF28dFeC7c7A4c483B85",

"collSurplusPool": "0xbe40060aef1A2acb4425823c82978f976Fd93cd0",

"communityIssuance": "0x7fd517b06b898F1a6081E0891265516F83Dc9C9E",

"defaultPool": "0x654Ed83ab231550001Fc1d2281B78fcD84121088",

"hintHelpers": "0x08E260d3e5EA4fB09fFa264DD4129593fD5405e8",

"sableRewarder": "0x23d253F1Ab38a1Ec8c05103232B4eFaFB6A1bdEb",

"sableStaking": "0xFbc81aEB7e5c11d4A60a0690Db9F36F93E25B16C",

"priceFeed": "0xA5220fd82C098b7f1C711e2F1C1d599ccfbCDCB3",

"sortedTroves": "0x97C131C309A04BFa1AAE82856d64b696b89dC87C",

"stabilityPool": "0x598913568093AB9F3d549236EB98388271073F18",

"gasPool": "0xe9Bc9ADBdf67343b5A66d73cf2E521BB3F088d01", 已验证

"systemState": "0x698ad77E62679c8E6aCfAfea03547C38fC5Ec0aD", "oracleCalc": "0x76Dcd40843C1dE96839bf83790257A36011E6632",

"troveHelper": "0xd1BF4d208028CBFe65c6b4D68C12e68F5F3D80F8",

"usdsToken": "0x0c6Ed1E73BA73B8441868538E210ebD5DD240FA0",

"sableToken": "0x1eE098cBaF1f846d5Df1993f7e2d10AFb35A878d",

"multiTroveGetter": "0x97C984497b81FA38BAaF684E7Afd2685052804E9"

4.2 Visibility Description

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

PriceFeed			
Function Name	Visibility	Mutability	Modifiers
_setBNBFeed	Internal	Can Modify State	-
setAddresses	External	Payable	onlyOwner
_restrictCaller	Internal	-	-
fetchPrice	External	Can Modify State	-
_chainlinkIsBroken	Internal	-	-
_badChainlinkResponse	Internal	-	-
_chainlinkIsFrozen	Internal	-	-
_chainlinkPriceChangeAboveMax	Internal	-	-
_pythIsBroken	Internal	-	-
_pythResponseFailed	Internal	-	-
_pythIsFrozen	Internal	-	-
_bothOraclesLiveAndUnbrokenAndSimilarPrice	Internal	-	-
_bothOraclesSimilarPrice	Internal	-	-

PriceFeed			
_scaleChainlinkPriceByDigits	Internal	-	-
_scalePythPrice	Internal	-	-
_changeStatus	Internal	Can Modify State	-
_storePrice	Internal	Can Modify State	-
_storePythPrice	Internal	Can Modify State	-
_storeChainlinkPrice	Internal	Can Modify State	-
_getCurrentPythResponse	Internal	Can Modify State	-
_getCurrentChainlinkResponse	Internal	-	-
_getPrevChainlinkResponse	Internal	-	-
<Receive Ether>	External	Payable	-

TroveManager			
Function Name	Visibility	Mutability	Modifiers
setAddresses	External	Can Modify State	onlyOwner
getTroveOwnersCount	External	-	-
getTroveFromTroveOwnersArray	External	-	-
liquidate	External	Can Modify State	-
_liquidateNormalMode	Internal	Can Modify State	-
_liquidateRecoveryMode	Internal	Can Modify State	-
_getOffsetAndRedistributionVals	Internal	-	-

TroveManager			
liquidateTroves	External	Can Modify State	-
_getTotalsFromLiquidateTrovesSequence_RecoveryMode	Internal	Can Modify State	-
_getTotalsFromLiquidateTrovesSequence_NormalMode	Internal	Can Modify State	-
batchLiquidateTroves	Public	Can Modify State	-
_getTotalFromBatchLiquidate_RecoveryMode	Internal	Can Modify State	-
_getTotalsFromBatchLiquidate_NormalMode	Internal	Can Modify State	-
_addLiquidationValuesToTotals	Internal	-	-
_sendGasCompensation	Internal	Can Modify State	-
_movePendingTroveRewardsToActivePool	Internal	Can Modify State	-
_redeemCollateralFromTrove	Internal	Can Modify State	-
_redeemCloseTrove	Internal	Can Modify State	-
redeemCollateral	External	Can Modify State	-
getNominalICR	Public	-	-
getCurrentICR	Public	-	-
_getCurrentTroveAmounts	Internal	-	-
applyPendingRewards	External	Can Modify State	-
_applyPendingRewards	Internal	Can Modify State	-
updateTroveRewardSnapshots	External	Can Modify State	-
_updateTroveRewardSnapshots	Internal	Can Modify State	-

TroveManager			
getPendingETHReward	Public	-	-
getPendingLUSDDebtReward	Public	-	-
hasPendingRewards	Public	-	-
getEntireDebtAndColl	Public	-	-
removeStake	External	Can Modify State	-
_removeStake	Internal	Can Modify State	-
updateStakeAndTotalStakes	External	Can Modify State	-
_updateStakeAndTotalStakes	Internal	Can Modify State	-
_computeNewStake	Internal	-	-
_redistributeDebtAndColl	Internal	Can Modify State	-
closeTrove	External	Can Modify State	-
_closeTrove	Internal	Can Modify State	-
_updateSystemSnapshots_excludeCollRemainder	Internal	Can Modify State	-
addTroveOwnerToArray	External	Can Modify State	-
_removeTroveOwner	Internal	Can Modify State	-
getTCR	External	-	-
checkRecoveryMode	External	-	-
_checkPotentialRecoveryMode	Internal	-	-
_updateBaseRateFromRedemption	Internal	Can Modify State	-
getRedemptionRate	Public	-	-

TroveManager			
getRedemptionRateWithDecay	Public	-	-
_calcRedemptionRate	Internal	-	-
_getRedemptionFee	Internal	-	-
getRedemptionFeeWithDecay	External	-	-
_calcRedemptionFee	Internal	-	-
getBorrowingRate	Public	-	-
getBorrowingRateWithDecay	Public	-	-
_calcBorrowingRate	Internal	-	-
getBorrowingFee	External	-	-
getBorrowingFeeWithDecay	External	-	-
_calcBorrowingFee	Internal	-	-
decayBaseRateFromBorrowing	External	Can Modify State	-
_updateLastFeeOpTime	Internal	Can Modify State	-
_calcDecayedBaseRate	Internal	-	-
_requireCallerIsBorrowerOperations	Internal	-	-
_requireTroveIsActive	Internal	-	-
getTroveStatus	External	-	-
getTroveStake	External	-	-
getTroveDebt	External	-	-
getTroveColl	External	-	-
setTroveStatus	External	Can Modify State	-

TroveManager			
increaseTroveColl	External	Can Modify State	-
decreaseTroveColl	External	Can Modify State	-
increaseTroveDebt	External	Can Modify State	-
decreaseTroveDebt	External	Can Modify State	-

BorrowerOperations			
Function Name	Visibility	Mutability	Modifiers
setAddresses	External	Can Modify State	onlyOwner
openTrove	External	Payable	-
addColl	External	Payable	-
moveETHGainToTrove	External	Payable	-
withdrawColl	External	Can Modify State	-
withdrawLUSD	External	Can Modify State	-
repayLUSD	External	Can Modify State	-
adjustTrove	External	Payable	-
_adjustTrove	Internal	Can Modify State	-
closeTrove	External	Can Modify State	-
claimCollateral	External	Can Modify State	-
_triggerBorrowingFee	Internal	Can Modify State	-
_getUSDValue	Internal	-	-
_getCollChange	Internal	-	-
_updateTroveFromAdjustment	Internal	Can Modify State	-

BorrowerOperations			
_moveTokensAndETHfromAdjustment	Internal	Can Modify State	-
_activePoolAddColl	Internal	Can Modify State	-
_withdrawLUSD	Internal	Can Modify State	-
_repayLUSD	Internal	Can Modify State	-
_requireSingularCollChange	Internal	-	-
_requireCallerIsBorrower	Internal	-	-
_requireNonZeroAdjustment	Internal	-	-
_requireTroveisActive	Internal	-	-
_requireTroveisNotActive	Internal	-	-
_requireNonZeroDebtChange	Internal	-	-
_requireNotInRecoveryMode	Internal	-	-
_requireNoCollWithdrawal	Internal	-	-
_requireValidAdjustmentInCurrentMode	Internal	-	-
_requireICRisAboveMCR	Internal	-	-
_requireICRisAboveCCR	Internal	-	-
_requireNewICRisAboveOldICR	Internal	-	-
_requireNewTCRisAboveCCR	Internal	-	-
_requireAtLeastMinNetDebt	Internal	-	-
_requireValidLUSDRepayment	Internal	-	-
_requireCallerIsStabilityPool	Internal	-	-
_requireSufficientLUSDBalance	Internal	-	-
_requireValidMaxFeePercentage	Internal	-	-
_getNewNominalICRFromTroveChange	Internal	-	-

BorrowerOperations

_getNewICRFromTroveChange	Internal	-	-
_getNewTroveAmounts	Internal	-	-
_getNewTCRFromTroveChange	Internal	-	-
getCompositeDebt	External	-	-

TroveHelper

Function Name	Visibility	Mutability	Modifiers
setAddresses	External	Can Modify State	onlyOwner
getCappedOffsetVals	External	-	-
isValidFirstRedemptionHint	External	-	-
requireValidMaxFeePercentage	External	-	-
requireAfterBootstrapPeriod	External	-	-
requireLUSDBalanceCoversRedemption	External	-	-
requireMoreThanOneTroveInSystem	External	-	-
requireAmountGreaterThanZero	External	-	-
requireTCRoverMCR	External	-	-
checkPotentialRecoveryMode	External	-	-
_requireCallerIsTroveManager	Internal	-	-

TimeLock

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
isOperation	Public	-	-
isOperationPending	Public	-	-
isOperationReady	Public	-	-
isOperationDone	Public	-	-

TimeLock			
getTimestamp	Public	-	-
getMinDelay	Public	-	-
hashOperation	Public	-	-
hashOperationBatch	Public	-	-
schedule	Public	Can Modify State	onlyRole
scheduleBatch	Public	Can Modify State	onlyRole
_schedule	Private	Can Modify State	-
cancel	Public	Can Modify State	onlyRole
execute	Public	Payable	onlyRole
executeBatch	Public	Payable	onlyRole
_beforeCall	Private	-	-
_afterCall	Private	Can Modify State	-
_call	Private	Can Modify State	-
updateDelay	External	Can Modify State	-

SystemState			
Function Name	Visibility	Mutability	Modifiers
setConfigs	Public	Can Modify State	onlyOwner
setLUSDGasCompensation	External	Can Modify State	onlyTimeLock
setBorrowingFeeFloor	External	Can Modify State	onlyTimeLock
setRedemptionFeeFloor	External	Can Modify State	onlyTimeLock
setMinNetDebt	External	Can Modify State	onlyTimeLock
setMCR	External	Can Modify State	onlyTimeLock

SystemState			
setCCR	External	Can Modify State	onlyTimeLock
getLUSDGasCompensation	External	-	-
getBorrowingFeeFloor	External	-	-
getRedemptionFeeFloor	External	-	-
getMinNetDebt	External	-	-
getMCR	External	-	-
getCCR	External	-	-
_setLUSDGasCompensation	Internal	Can Modify State	-
_setBorrowingFeeFloor	Internal	Can Modify State	-
_setRedemptionFeeFloor	Internal	Can Modify State	-
_setMinNetDebt	Internal	Can Modify State	-
_setMCR	Internal	Can Modify State	-
_setCCR	Internal	Can Modify State	-

StabilityPool			
Function Name	Visibility	Mutability	Modifiers
setParams	External	Can Modify State	onlyOwner
getETH	External	-	-
getTotalLUSDDeposits	External	-	-
provideToSP	External	Can Modify State	-
withdrawFromSP	External	Can Modify State	-
setRewardsPerBlock	External	Can Modify State	-
withdrawETHGainToTrove	External	Can Modify State	-

StabilityPool			
offset	External	Can Modify State	-
_computeRewardsPerUnitStaked	Internal	Can Modify State	-
_updateRewardSumAndProduct	Internal	Can Modify State	-
_moveOffsetCollAndDebt	Internal	Can Modify State	-
_decreaseLUSD	Internal	Can Modify State	-
getDepositorETHGain	Public	-	-
_getETHGainFromSnapshots	Internal	-	-
getDepositorLQTYGain	Public	-	-
getFrontEndLQTYGain	Public	-	-
getRewarsPerBlock	External	-	-
getCompoundedLUSDDeposit	Public	-	-
getCompoundedFrontEndStake	Public	-	-
_getCompoundedStakeFromSnapshots	Internal	-	-
_sendLUSDtoStabilityPool	Internal	Can Modify State	-
_sendETHGainToDepositor	Internal	Can Modify State	-
_sendLUSDToDepositor	Internal	Can Modify State	-
registerFrontEnd	External	Can Modify State	-
_setFrontEndTag	Internal	Can Modify State	-
_updateDepositAndSnapshots	Internal	Can Modify State	-
_updateRewardDebt	Internal	Can Modify State	-
_resetAccPerShareAndPayOutProfit	Internal	Can Modify State	-
updatePoolRewards	Internal	Can Modify State	-
_updateFrontEndStakeAndSnapshots	Internal	Can Modify State	-

StabilityPool

_payOutLQTYGainsDepositor	Internal	Can Modify State	-
_payOutLQTYGainsFrontEnd	Internal	Can Modify State	-
_requireCallerIsActivePool	Internal	-	-
_requireCallerIsTimeLock	Internal	-	-
_requireCallerIsTroveManager	Internal	-	-
_requireNoUnderCollateralizedTroves	Internal	Can Modify State	-
_requireUserHasDeposit	Internal	-	-
_requireUserHasNoDeposit	Internal	-	-
_requireNonZeroAmount	Internal	-	-
_requireUserHasTrove	Internal	-	-
_requireUserHasETHGain	Internal	-	-
_requireFrontEndNotRegistered	Internal	-	-
_requireFrontEndIsRegisteredOrZero	Internal	-	-
_requireValidKickbackRate	Internal	-	-
<Receive Ether>	External	Payable	-

OracleRateCalculation

Function Name	Visibility	Mutability	Modifiers
getOracleRate	External	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event record

Category: Malicious Event Log Audit

Content

- contracts/contracts/TroveManager.sol

`setAddresses` functions do not log events.

- contracts/contracts/PriceFeed.sol

`setAddresses` functions do not log events.

Solution

It is recommended to record the modification of sensitive parameters for subsequent community review or self-examination.

Status

Fixed

[N2] [Suggestion] Missing Logic Judgments for Other Cases

Category: Design Logic Audit

Content

- contracts/contracts/PriceFeed.sol

If `publishTimePyth > block.timestamp` . If this happens, there must be an exception. This situation must be dealt with, because this situation returns false, noFrozen

```
function _pythIsFrozen(PythResponse  memory _pythResponse) internal view returns
(bool) {
    // If the block.timestamp - publishTimePyth > 30, use Chainlink.
    if (block.timestamp > _pythResponse.publishTimePyth) {
        return block.timestamp.sub(_pythResponse.publishTimePyth) > 30;
    }
}
```

Solution

It is recommended to clarify the logic implementation.

Status

Fixed

[N3] [High] Award calculation omission

Category: Design Logic Audit

Content

- contracts/contracts/StabilityPool.sol

In the `_payOutLQTYGainsDepositor` and `_payOutLQTYGainsFrontEnd` functions. If `uint256 balance = _communityIssuance.balanceLQTY()`; If balance = 0, the user will not receive the money and the contract will record the corresponding reward. If the balance is 0, it should be recorded in `user.unpaidRewards` instead of directly not Rewarding users directly records user liabilities.

This will cause users to lose their rewards.

```
    function _payOutLQTYGainsDepositor(
        ICommunityIssuance _communityIssuance,
        address _depositor,
        uint256 _compoundedLUSDDeposit
    ) internal {
        updatePoolRewards();
        Deposit memory user = deposits[_depositor];
        if (_compoundedLUSDDeposit == 0 && user.unpaidRewards == 0) {
            return;
        }
        address frontEndTag = deposits[_depositor].frontEndTag;
        /*
         * If not tagged with a front end, the depositor gets a 100% cut of what
their deposit earned.
         * Otherwise, their cut of the deposit's earnings is equal to the
kickbackRate, set by the front end through
         * which they made their deposit.
         */
        uint256 kickbackRate = frontEndTag == address(0)
            ? DECIMAL_PRECISION
            : frontEnds[frontEndTag].kickbackRate;

        uint256 depositorLQTYGain = kickbackRate
            .mul(
                _compoundedLUSDDeposit.mul(accumulatedRewardsPerShare) /
                    DECIMAL_PRECISION
            )
            .sub(user.rewardDebt)
            .div(DECIMAL_PRECISION)
            .add(user.unpaidRewards);
        //SLOWMIST//If there is no balance in balance, there will be no rewards.
        uint256 balance = _communityIssuance.balanceLQTY()
        if (balance > 0) {
            if (depositorLQTYGain > balance) {
                _communityIssuance.sendLQTY(_depositor, balance);
                user.unpaidRewards = depositorLQTYGain - balance;
            } else {
                _communityIssuance.sendLQTY(_depositor, depositorLQTYGain);
                user.unpaidRewards = 0;
            }
        }
        //
        user.rewardDebt = kickbackRate.mul(
            _compoundedLUSDDeposit.mul(accumulatedRewardsPerShare) /
                DECIMAL_PRECISION
        );
        emit LQTYPaidToDepositor(_depositor, depositorLQTYGain);
    }
```

Solution

Correctly record the rewards that the user has not claimed.

Status

Fixed; Revert to the liquid version to ensure the stability of the project

[N4] [Medium] The Dos issue

Category: Denial of Service Vulnerability

Content

- contracts/contracts/StabilityPool.sol

In the `_resetAccPerShareAndPayOutProfit` function, if the array length of `stakerSets` and `fronEndSets` is too long, This will cause `offset` to always fail when called.And since this offset is called by `TroveManager` liquidateTroves and `batchLiquidateTroves`, this will cause liquidation to fail.

Solution

This function needs to confirm whether it is necessary to reset the rewards of all users during liquidation.

Status

Fixed; Revert to the liquid version to ensure the stability of the project.

[N5] [Critical] Reward calculation exception problem

Category: Design Logic Audit

Content

- contracts/contracts/StabilityPool.sol

When the `offset` function is executed, the function `_resetAccPerShareAndPayOutProfit` will be called, which will reset the `deposit.rewardDebt` of all users, which will cause the user to receive an excess reward immediately.

```
function _resetAccPerShareAndPayOutProfit() internal {
    updatePoolRewards();
    uint256 stakerLength = stakerSets.length();
    for (uint256 i = 0; i < stakerLength; i++) {
        address stakerAddress = stakerSets.at(i);
        Deposit storage deposit = deposits[stakerAddress];
        address frontEndTag = deposits[stakerAddress].frontEndTag;
        uint256 kickbackRate = frontEndTag == address(0)
            ? DECIMAL_PRECISION
            : frontEnds[frontEndTag].kickbackRate;
        deposit.unpaidRewards = kickbackRate
            .mul(
                deposit.initialValue.mul(accumulatedRewardsPerShare) /
                    DECIMAL_PRECISION
            )
            .sub(deposit.rewardDebt)
            .div(DECIMAL_PRECISION)
            .add(deposit.unpaidRewards);
        deposit.rewardDebt = 0;
    }

    uint256 fronEndLength = fronEndSets.length();
    for (uint256 i = 0; i < fronEndLength; i++) {
        address frontEndAddress = fronEndSets.at(i);
        FrontEndStake storage frontEnd = frontEndStakes[frontEndAddress];
        uint256 kickbackRate = frontEnds[frontEndAddress].kickbackRate;
        uint256 frontEndShare = uint256(DECIMAL_PRECISION).sub(
            kickbackRate
        );
        frontEnd.unpaidRewards = frontEndShare
            .mul(
                frontEnd.totalDeposits.mul(accumulatedRewardsPerShare) /
                    DECIMAL_PRECISION
            )
            .sub(frontEnd.rewardDebt)
            .div(DECIMAL_PRECISION)
            .add(frontEnd.unpaidRewards);
        frontEnd.rewardDebt = 0;
    }
```

```
        accumulatedRewardsPerShare = 0;
    }
```

Solution

Confirm whether the cleaning logic is normal.

Status

Fixed; Revert to the liquid version to ensure the stability of the project

[N6] [Critical] Memory variable exploit

Category: Design Logic Audit

Content

- contracts/contracts/StabilityPool.sol

`Deposit memory user = deposits[_depositor];` This user uses a temporary variable, so the `user.rewardDebt` of the user will not be recorded in the future and the `user.unpaidRewards` that may not be received by the user will be missed.

```
function _payOutLQTYGainsDepositor(
       ICommunityIssuance _communityIssuance,
       address _depositor,
       uint256 _compoundedLUSDDeposit
   ) internal {
       updatePoolRewards();
       Deposit memory user = deposits[_depositor];//SLOWMIST//
       if (_compoundedLUSDDeposit == 0 && user.unpaidRewards == 0) {
           return;
       }
       address frontEndTag = deposits[_depositor].frontEndTag;
       /*
        * If not tagged with a front end, the depositor gets a 100% cut of what their
deposit earned.
        * Otherwise, their cut of the deposit's earnings is equal to the
kickbackRate, set by the front end through
        * which they made their deposit.
        */
       uint256 kickbackRate = frontEndTag == address(0)
           ? DECIMAL_PRECISION
           : frontEnds[frontEndTag].kickbackRate;

       uint256 depositorLQTYGain = kickbackRate
```

```
            .mul(
                _compoundedLUSDDeposit.mul(accumulatedRewardsPerShare) /
                    DECIMAL_PRECISION
            )
            .sub(user.rewardDebt)
            .div(DECIMAL_PRECISION)
            .add(user.unpaidRewards);

        uint256 balance = _communityIssuance.balanceLQTY();
        if (balance > 0) {
            if (depositorLQTYGain > balance) {
                _communityIssuance.sendLQTY(_depositor, balance);
                user.unpaidRewards = depositorLQTYGain - balance;
            } else {
                _communityIssuance.sendLQTY(_depositor, depositorLQTYGain);
                user.unpaidRewards = 0;
            }
        }
        //
        user.rewardDebt = kickbackRate.mul(
            _compoundedLUSDDeposit.mul(accumulatedRewardsPerShare) /
                DECIMAL_PRECISION
        );
        emit LQTYPaidToDepositor(_depositor, depositorLQTYGain);
    }
```

`FrontEndStake memory frontEnd = frontEndStakes[_frontEnd];` This user uses a temporary variable, so the `frontEnd.rewardDebt` of the user will not be recorded in the future and the `frontEnd.unpaidRewards` that may not be received by the user will be missed.

```
     function _payOutLQTYGainsFrontEnd(
        ICommunityIssuance _communityIssuance,
        address _frontEnd,
        uint256 _compoundedLUSDDeposit
    ) internal {
        updatePoolRewards();
        FrontEndStake memory frontEnd = frontEndStakes[_frontEnd];//SLOWMIST//
        if (
            _compoundedLUSDDeposit == 0 ||
            (_frontEnd == address(0) && frontEnd.unpaidRewards == 0)
        ) {
            return;
        }

        /*
         * If not tagged with a front end, the depositor gets a 100% cut of what their
```

```
deposit earned.
         * Otherwise, their cut of the deposit's earnings is equal to the
kickbackRate, set by the front end through
         * which they made their deposit.
         */
        uint256 kickbackRate = frontEnds[_frontEnd].kickbackRate;
        uint256 frontEndShare = uint256(DECIMAL_PRECISION).sub(kickbackRate);

        uint256 LQTYGain = frontEndShare
            .mul(
                _compoundedLUSDDeposit.mul(accumulatedRewardsPerShare) /
                    DECIMAL_PRECISION
            )
            .sub(frontEnd.rewardDebt)
            .div(DECIMAL_PRECISION)
            .add(frontEnd.unpaidRewards);
        uint256 balance = _communityIssuance.balanceLQTY();
        if (balance > 0) {
            if (LQTYGain > balance) {
                _communityIssuance.sendLQTY(_frontEnd, balance);
                frontEnd.unpaidRewards = LQTYGain - balance;
            } else {
                _communityIssuance.sendLQTY(_frontEnd, LQTYGain);
                frontEnd.unpaidRewards = 0;
            }
        }
        frontEnd.rewardDebt =
            _compoundedLUSDDeposit.mul(accumulatedRewardsPerShare) /
            DECIMAL_PRECISION;
        emit LQTYPaidToFrontEnd(_frontEnd, LQTYGain);
    }
```

Solution

Use `storage` to save data.

Status

Fixed; Revert to the liquid version to ensure the stability of the project.

[N7] [Suggestion] Compiler version is too low

Category: Integer Overflow and Underflow Vulnerability

Content

The calculation of `+ - * /` is useful in the contract. Since the compiled version is lower than 8.0, `safemath` is not used by default. It is recommended to use the `safemath` calculation method.

Solution

Use safemath for calculations

Status

Fixed

[N8] [Suggestion] Variable Setting Scope Recommendations

Category: Design Logic Audit

Content

- contracts/contracts/SystemState.sol

It is necessary to ensure that `minNetDebt>0` to avoid the situation where the protocol is unavailable.

```
function _setMinNetDebt(uint256 _value) internal {
    uint256 oldValue = minNetDebt;
    minNetDebt = _value;
    emit MinNetDebtChanged(oldValue, _value);
}
```

Solution

Make sure to set minNetDebt > 0

Status

Fixed

[N9] [Low] Risk of initial operation

Category: Authority Control Vulnerability Audit

Content

Since `Timelock` contract role can set several key parameters, it is recommended that Timelock roles use multi-signature management to be more secure.

- SystemState.sol

`TimeLock` can `setLUSDGasCompensation`

`TimeLock` can `setBorrowingFeeFloor`

`TimeLock` can `setRedemptionFeeFloor`

`TimeLock` can `setMinNetDebt`

`TimeLock` can `setMCR`

`TimeLock` can `setCCR`

Solution

Use multi-signature to manage the role of Timelock.

Status

Acknowledged

[N10] [Critical] Race Conditions Vulnerability issue

Category: Race Conditions Vulnerability

Content

- contracts/contracts/PriceFeed.sol

After executing a branch condition, it does not directly return the obtained price and will call `_changeStatus` to modify the status at the end of the execution, so that it will enter multiple branches when fetching the price and eventually lead to the wrong price of the obtained oracle.

```
function fetchPrice(
    bytes[] calldata priceFeedUpdateData
) external override returns (FetchPriceResult memory result) {
    _restrictCaller();
    // Get current and previous price data from Pyth, and current price data from
Chainlink
    ChainlinkResponse memory chainlinkResponse = _getCurrentChainlinkResponse();
    ChainlinkResponse memory prevChainlinkResponse =
_getPrevChainlinkResponse(chainlinkResponse.roundId, chainlinkResponse.decimals);
    PythResponse memory pythResponse = _getCurrentPythResponse(priceFeedUpdateData);

    result.price = 0;
    result.deviationPyth = 0;
    result.publishTimePyth = 0;
    result.oracleKey = bytes32("PYTH");

    // --- CASE 1: System fetched last price from Pyth  ---
    if (status == Status.pythWorking) {
        // If Pyth is broken, try Chainlink
        if (_pythIsBroken(pythResponse)) {
            // If Chainlink is broken then both oracles are untrusted, so return the
last good price
            if (_chainlinkIsBroken(chainlinkResponse, prevChainlinkResponse)) {
                _changeStatus(Status.bothOraclesUntrusted);
                result.price = lastGoodPrice;
            }
            /*
            * If Chainlink is only frozen but otherwise returning valid data, return
the last good price.
            */
            else if (_chainlinkIsFrozen(chainlinkResponse)) {
                _changeStatus(Status.usingChainlinkPythUntrusted);
                result.price = lastGoodPrice;
                result.oracleKey = bytes32("LINK");
            }

            else {
                // If Pyth is broken and Chainlink is working, switch to Chainlink
and return current Chainlink price
                _changeStatus(Status.usingChainlinkPythUntrusted);
                result.price = _storeChainlinkPrice(chainlinkResponse);
                result.oracleKey = bytes32("LINK");
            }

        }

        // If Pyth is frozen, try Chainlink
        else if (_pythIsFrozen(pythResponse)) {
            // If Chainlink is broken too, remember Chainlink broke, and return last
good price
            if (_chainlinkIsBroken(chainlinkResponse, prevChainlinkResponse)) {
                _changeStatus(Status.usingPythChainlinkUntrusted);
                result.price = lastGoodPrice;
            } else {
                // If Chainlink is frozen or working, remember Pyth froze, and switch
to Chainlink
                _changeStatus(Status.usingChainlinkPythFrozen);
                result.oracleKey = bytes32("LINK");

                if (_chainlinkIsFrozen(chainlinkResponse)) {
                    result.price = lastGoodPrice;
                } else {
                    // If Chainlink is working, use it
                    result.price = _storeChainlinkPrice(chainlinkResponse);
                }
            }
        }

        // If Pyth is working
        else {
            // If Pyth is working and Chainlink is broken, remember Chainlink is broken
            if (_chainlinkIsBroken(chainlinkResponse, prevChainlinkResponse)) {
                _changeStatus(Status.usingPythChainlinkUntrusted);//SLOWMIST//The status is set to usingPythChainlinkUntrusted, so the function branch of if (status == Status.usingPythChainlinkUntrusted) will continue to be executed after the value is completed.
            }

            // If Pyth is working, return Pyth current price (no status change)
            result.price = _storePythPrice(pythResponse);
            result.deviationPyth = pythResponse.deviationPyth;
            result.publishTimePyth = pythResponse.publishTimePyth;
        }

    }


    /....../


    // --- CASE 5: Using Pyth, Chainlink is untrusted ---

    //SLOWMIST//After the execution of the first branch, it is possible to enter the judgment of this branch due to state changes. This is wrong.

    if (status == Status.usingPythChainlinkUntrusted) {
        // If Pyth breaks, now both oracles are untrusted
        if (_pythIsBroken(pythResponse)) {
            _changeStatus(Status.bothOraclesUntrusted);
            result.price = lastGoodPrice;
        }

        // If Pyth is frozen, return last good price (no status change)
        else if (_pythIsFrozen(pythResponse)) {
            result.price = lastGoodPrice;
        }

        // If Pyth and Chainlink are both live, unbroken and similar price, switch back to Pyth and return Pyth price
        else if (_bothOraclesLiveAndUnbrokenAndSimilarPrice(chainlinkResponse, prevChainlinkResponse, pythResponse)) {
            _changeStatus(Status.pythWorking);
            result.price = _storePythPrice(pythResponse);
            result.deviationPyth = pythResponse.deviationPyth;
            result.publishTimePyth = pythResponse.publishTimePyth;
        }
        else {
            // return Pyth price (no status change)
            result.price = _storePythPrice(pythResponse);
            result.deviationPyth = pythResponse.deviationPyth;
            result.publishTimePyth = pythResponse.publishTimePyth;
        }
    }

}
```

Solution

Return the value after getting the data

Status

Fixed

[N11] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

- contracts/contracts/SABLE/CommunityIssuance.sol

The `owner` can set rewards per second. If the owner's private key is leaked, it will cause serious losses.

```
    function updateRewardPerSec(uint newRewardPerSec) external override onlyOwner {
        stabilityPool.ownerTriggerIssuance();
        require(lastIssuanceTime == block.timestamp);
        latestRewardPerSec = newRewardPerSec;
        emit RewardPerSecUpdated(newRewardPerSec);
    }
```

- contracts/contracts/SABLE/SableRewarder.sol

The owner has too much authority and can take money from the `SableRewarder` contract by updatingRewardPerSec to increase the reward if the private key is compromised.

```
    function updateRewardPerSec(uint newRewardPerSec) external override onlyOwner {
        _issueSABLE();
        require(lastIssuanceTime == block.timestamp);
        latestRewardPerSec = newRewardPerSec;
        emit RewardPerSecUpdated(newRewardPerSec);
    }
```

Solution

In the short term, these privileged addresses can be controlled by the project team in the early stages of the project to ensure the stable operation of the project. However, in order to avoid single-point risks, privileged roles can be managed by timelock, and timelock permissions can be managed by multisig contracts. At the same time, in order to ensure rapid response to emergency situations, the authority of the emergency suspension agreement can be independently managed by the EOA of the project team. But in the long run, when the project is running stably, transferring the ownership to community governance can effectively avoid the risk of centralization and gain the trust of community users.

Status

Acknowledged; Owners of CommunityIssuance.sol are managed using time locks.

[N12] [Suggestion] Stake and unstake recommendations

Category: Design Logic Audit

Content

- contracts/contracts/SABLE/SableStakingV2.sol

If there are not enough USDS, SABLE and BNB in the contract the user will not be able to use the take and unstake functions.

- contracts/contracts/SABLE/SableRewarder.sol

If the SableRewarder contract does not have enough SABLE, then the stake and unstake functions of SableStakingV2 will also not work.

Solution

To ensure that SableStakingV2 has enough USDS, SABLE, BNB. Also ensure that SableRewarder has enough SABLE.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002305160002	SlowMist Security Team	2023.05.08 - 2023.05.16	Low 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 3 critical risk, 1 high risk, 1 medium risk, 1 low risk, 4 suggestion vulnerabilities.

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.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist