SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.06.30, the SlowMist security team received the Sigma Money team's security audit application for SigmaMoney, 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 protocol is forked from Fx Protocol and Pendle finance.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Mathematical Errors in Price Conversion	Design Logic Audit	Medium	Fixed
N2	The price query function returns the	Others	Information	Fixed

NO	Title	Category	Level	Status
	same value			
N3	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N4	Read-Only Parameter Principle Violation	Others	Suggestion	Fixed
N5	Error message mismatched function operation	Design Logic Audit	Low	Fixed
N6	SY token deposit and redeem operations lack slippage protection	Design Logic Audit	High	Fixed
N7	The amount of sy tokens actually received does not match the amount of deposit	Design Logic Audit	High	Fixed
N8	Missing Position ID Verification	Design Logic Audit	Low	Fixed
N9	The amount of slisBNB transferred and actually received does not match	Design Logic Audit	High	Fixed
N10	Redundant code	Others	Suggestion	Fixed
N11	Token swaps lack slippage protection	Design Logic Audit	Low	Fixed
N12	Lack of validation of swap call data	Others	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/SigmaMoney/contracts/

Initial audit version: a7047fba8f7c79f5b9dc8a83fc97c09da11a1bc4

Final audit version: 9bdf0754d8257ce0a92cc542365a64aa759a024e

Audit Scope:

```
- contracts/core/BNBUSDBasePool.sol
- contracts/core/PoolManager.sol
- contracts/core/pool/BasePool.sol
- contracts/core/pool/SigmaClisBNBPool.sol
- contracts/helpers/RevenuePool.sol
- contracts/interfaces/ISigmaClisBNBSYPool.sol
- contracts/interfaces/Pancake/IPancakeV3Pool.sol
- contracts/interfaces/Pancake/IV3SwapRouter.sol
- contracts/interfaces/sigma/IListaStakeManager.sol
- contracts/interfaces/sigma/ISigmaController.sol
- contracts/libraries/PancakeLib.sol
- contracts/price-oracle/BNBPriceOracle.sol
- contracts/rate-provider/SigmaClisBNBSYBNBRateProvider.sol
- contracts/rate-provider/SigmaClisBNBSYSlisBNBRateProvider.sol
- contracts/sigma/SigmaController.sol
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

BNBUSDBasePool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
previewDeposit	Public	-	onlyValidToken
previewRedeem	External	-	-
nav	External	-	-

BNBUSDBasePool			
getStableTokenPrice	Public	-	-
getStableTokenPriceWithScale	Public	-	-
deposit	External	Can Modify State	nonReentrant onlyValidToken sync
requestRedeem	External	Can Modify State	-
redeem	External	Can Modify State	nonReentrant sync
instantRedeem	External	Can Modify State	nonReentrant sync
rebalance	External	Can Modify State	onlyValidToken nonReentrant sync
rebalance	External	Can Modify State	onlyValidToken nonReentrant sync
liquidate	External	Can Modify State	onlyValidToken nonReentrant sync
arbitrage	External	Can Modify State	onlyValidToken onlyPegKeeper nonReentrant sync
updateStableDepegPrice	External	Can Modify State	onlyRole
updateRedeemCoolDownPeriod	External	Can Modify State	onlyRole
updateInstantRedeemFeeRatio	External	Can Modify State	onlyRole
_update	Internal	Can Modify State	-
_updateStableDepegPrice	Internal	Can Modify State	-
_updateRedeemCoolDownPeriod	Internal	Can Modify State	-
_updateInstantRedeemFeeRatio	Internal	Can Modify State	-
_deposit	Internal	Can Modify State	-

BNBUSDBasePool

_beforeRebalanceOrLiquidate	Internal	-	-
_afterRebalanceOrLiquidate	Internal	Can Modify State	-

PoolManager

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
initializeV2	External	Can Modify State	onlyRegisteredPool reinitializer
getPoolInfo	External	-	-
operate	External	Can Modify State	onlyRegisteredPool nonReentrant whenNotPaused
redeem	External	Can Modify State	onlyRegisteredPool nonReentrant whenNotPaused
rebalance	External	Can Modify State	onlyRegisteredPool nonReentrant whenNotPaused onlyFxUSDSave
rebalance	External	Can Modify State	onlyRegisteredPool nonReentrant whenNotPaused onlyFxUSDSave
liquidate	External	Can Modify State	onlyRegisteredPool nonReentrant whenNotPaused onlyFxUSDSave
harvest	External	Can Modify State	onlyRegisteredPool onlyRole nonReentrant
setPause	External	Can Modify State	onlyRole
registerPool	External	Can Modify State	onlyRole
updateRateProvider	External	Can Modify State	onlyRole
updateRewardSplitter	External	Can Modify State	onlyRole onlyRegisteredPool

PoolManager			
updatePoolCapacity	External	Can Modify State	onlyRole onlyRegisteredPool
updateThreshold	External	Can Modify State	onlyRole
_updateRewardSplitter	Internal	Can Modify State	-
_updatePoolCapacity	Internal	Can Modify State	-
_updateThreshold	Internal	Can Modify State	-
_scaleUp	Internal	-	-
_scaleUp	Internal	-	-
_scaleDown	Internal	-	-
_scaleDownRoundingUp	Internal	-	-
_scaleDown	Internal	-	-
_beforeRebalanceOrLiquidate	Internal	-	-
_afterRebalanceOrLiquidate	Internal	Can Modify State	-
_changePoolCollateral	Internal	Can Modify State	-
_changePoolDebts	Internal	Can Modify State	-
_getTokenScalingFactor	Internal	-	-

BasePool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
__BasePool_init	Internal	Can Modify State	onlyInitializing
operate	External	Can Modify State	onlyPoolManager

BasePool			
redeem	External	Can Modify State	onlyPoolManager
rebalance	External	Can Modify State	onlyPoolManager
rebalance	External	Can Modify State	onlyPoolManager
liquidate	External	Can Modify State	onlyPoolManager
updateBorrowAndRedeemStatus	External	Can Modify State	onlyRole
updateDebtRatioRange	External	Can Modify State	onlyRole
updateMaxRedeemRatioPerTick	External	Can Modify State	onlyRole
updateRebalanceRatios	External	Can Modify State	onlyRole
updateLiquidateRatios	External	Can Modify State	onlyRole
updatePriceOracle	External	Can Modify State	onlyRole
_getRawDebtToRebalance	Internal	-	-
_getTickRawCollAndDebts	Internal	-	-
_rebalanceTick	Internal	Can Modify State	-
_liquidateTick	Internal	Can Modify State	-
_updateCollAndDebtIndex	Internal	Can Modify State	-
_deductProtocolFees	Internal	-	-

SigmaClisBNBPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BasePool
initialize	External	Can Modify State	initializer
getOpenRatio	External	-	-
getFundingRatio	External	-	-

SigmaClisBNBPool

getOpenFeeRatio	Public	-	-
getCloseFeeRatio	External	-	-
updateOpenRatio	External	Can Modify State	onlyRole
updateCloseFeeRatio	External	Can Modify State	onlyRole
updateFundingRatio	External	Can Modify State	onlyRole
_getOpenRatio	Internal	-	-
_updateOpenRatio	Internal	Can Modify State	-
_getCloseFeeRatio	Internal	-	-
_updateCloseFeeRatio	Internal	Can Modify State	-
_getFundingRatio	Internal	-	-
_updateFundingRatio	Internal	Can Modify State	-
_getAverageInterestRate	Internal	-	-
_updateInterestRate	Internal	Can Modify State	-
_updateCollAndDebtIndex	Internal	Can Modify State	-
_deductProtocolFees	Internal	-	-

RevenuePool

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
getRewardCount	External	-	-
claim	External	Can Modify State	-
updateTreasury	External	Can Modify State	onlyOwner
updateEcosystem	External	Can Modify State	onlyOwner

RevenuePool

updateStaker	External	Can Modify State	onlyOwner
addRewardToken	External	Can Modify State	onlyOwner
updateRewardTokenRatio	External	Can Modify State	onlyOwner
updateRewardTokenBurner	External	Can Modify State	onlyOwner
removeRewardToken	External	Can Modify State	onlyOwner
_checkRatioRange	Internal	-	-
_ensureNonZeroAddress	Internal	-	-

BNBPriceOracle

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	SpotPriceOracleBase
getBNBUSDSpotPrice	External	-	-
getPrice	Public	-	-
getExchangePrice	Public	-	-
getLiquidatePrice	External	-	-
getRedeemPrice	External	-	-
updateMaxPriceDeviation	External	Can Modify State	onlyOwner
_updateMaxPriceDeviation	Private	Can Modify State	-
_getBNBUSDSpotPrice	Internal	-	-

SigmaClisBNBSYBNBRateProvider

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

SigmaClisBNBSYBNBRateProvider			
getRate	External	-	-

SigmaController			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
deposit	External	Payable	nonReentrant
redeemInstant	External	Payable	nonReentrant
registerSwapTarget	External	Can Modify State	onlyOwner
unregisterSwapTarget	External	Can Modify State	onlyOwner
_transferInCollAndConvert	Internal	Can Modify State	-
_transferInDebt	Internal	Can Modify State	-
_transferOutColl	Internal	Can Modify State	-
_onlySupportedCollToken	Internal	-	-
_onlySupportedDebtToken	Internal	-	-
_onlySupportedSwapTarget	Internal	-	-
_redeem	Internal	Can Modify State	-
_isEmptyPosition	Internal	-	-
_updateEntryPosition	Internal	Can Modify State	-
_scaleUp	Internal	-	-
_scaleDown	Internal	-	-
_swap	Internal	Can Modify State	-
_balanceOf	Internal	-	-

SigmaController			
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Medium] Mathematical Errors in Price Conversion

Category: Design Logic Audit

Content

In the PancakeLib library, the `calculateSqrtX96` function has a mathematical error and lacks a square root operation. The current implementation `sqrtPriceX96 = (price * E96) / HALF_PRECISION` calculates a linear scaling instead of a square root, which causes the result to not match the expected √price × 2^96 format and does not form a correct inverse relationship with the `calculatePrice` function, which results in incorrect price conversion calculations.

- contracts/libraries/PancakeLib.sol#L17-L24

```
  function calculateSqrtX96(uint256 price, bool reverse) internal pure returns
(uint256 sqrtPriceX96) {
    if (reverse) {
      sqrtPriceX96 = (E96 * E96) / price;
    } else {
      sqrtPriceX96 = (price * E96) / HALF_PRECISION;
    }
    return sqrtPriceX96;
  }
```

Solution

It is recommended to use Math.sqrt(price) to correctly calculate the square root before format conversion.

Status

Fixed

[N2] [Information] The price query function returns the same value

Category: Others

Content

There is only a single price source in the BNBPriceOracle contract. Its `_getBNBUSDSpotPrice` function only obtains price data from Chainlink and assigns the same price to `minPrice` and `maxPrice`, causing all price query functions (`getBNBUSDSpotPrice`, `getPrice`, `getExchangePrice`, `getLiquidatePrice`, and `getRedeemPrice`) to return the same result.

- contracts/contracts/price-oracle/BNBPriceOracle.sol#L54-L56, L60-L73, L76-L79, L82-L84, L87-L90, L123-L128

```
  function getBNBUSDSpotPrice() external view returns (uint256 chainlinkPrice,
uint256 minPrice, uint256 maxPrice) {}
  function getPrice() public view override returns (uint256 anchorPrice, uint256
minPrice, uint256 maxPrice) {}
  function getExchangePrice() public view returns (uint256) {}
  function getLiquidatePrice() external view returns (uint256) {}
  function getRedeemPrice() external view returns (uint256) {}
  function _getBNBUSDSpotPrice() internal view returns (uint256 chainlinkPrice,
uint256 minPrice, uint256 maxPrice) {}
```

Solution

It is recommended that other contracts be aware of this limitation when using all price query functions.

Status

Fixed

[N3] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the BNBPriceOracle contract, the Owner role can modify the maximum price deviation.

- contracts/price-oracle/BNBPriceOracle.sol

```
  function updateMaxPriceDeviation(uint256 newMaxPriceDeviation) external onlyOwner
{}
```

2.The `Owner` has key permissions in the SigmaController contract and can control the target contract that allows users to swap tokens through the `registerSwapTarget` and `unregisterSwapTarget` functions.

- contracts/sigma/SigmaController.sol#L233-L235, L237-L240

```
function registerSwapTarget(address swapTarget) external onlyOwner {}
function unregisterSwapTarget(address swapTarget) external onlyOwner {}
```

Solution

In the short term, transferring the ownership of core roles to time-locked contracts and managing them by multi-signatures is an effective solution to avoid single-point risks. However, in the long run, a more reasonable solution is to implement a permission separation strategy and set up multiple privileged roles to manage each privileged function separately. Permissions involving user funds and contract core parameter updates should be managed by the community, while permissions involving emergency contract suspensions can be managed by EOA addresses. This ensures the safety of user funds while responding quickly to threats.

Status

Acknowledged

[N4] [Suggestion] Read-Only Parameter Principle Violation

Category: Others

Content

In the PancakeV3SpotPriceReader contract, the `getSpotPrice` function, as a read-only function, modifies the `encoding` parameter, which does not comply with the best programming principle that function parameters are read-only inputs.

- contracts/base/price-oracle/PancakeV3SpotPriceReader.sol#L30-L38

```
function getSpotPrice(uint256 encoding) external view returns (uint256 spotPrice) {
  uint256 poolType = encoding & 0xff;
  if (poolType != 13) {
    revert("Invalid pool type");
  }

  encoding >>= 8;
  spotPrice = _getSpotPriceByAerodromeCL(encoding);
}
```

Solution

It is recommended to introduce a local variable to store the modified encoding value instead of directly modifying the function parameter.

Status

Fixed

[N5] [Low] Error message mismatched function operation

Category: Design Logic Audit

Content

The `redeemInstant` function in the SigmaController contract implements the functions of repaying debts and redeeming collateral. The function requires the user to perform at least one operation (repayment or redemption). However, when both operation parameters (`opDebt.debtAmount` and `opColl.collAmount`) are 0 at the same time, the error message `"No supply and no borrow"` returned does not match the actual purpose of the function.

- contracts/sigma/SigmaController.sol#L199-L231

```
  function redeemInstant(
    OpPosition memory opPosition,
    OpColl memory opColl,
    OpDebt memory opDebt,
    OpParameter memory opParam
  ) external payable nonReentrant returns (int256 newColl, int256 newDebt, uint256
tokenOut) {
        //...
    require(!(opDebt.debtAmount == 0 && opColl.collAmount == 0), "No supply and no
borrow");
        //...
  }
```

Solution

It is recommended to change the error message to "No repay and no redeem" to accurately reflect the actual intention of the function.

Status

Fixed

[N6] [High] SY token deposit and redeem operations lack slippage protection

Category: Design Logic Audit

Content

In the `_transferInCollAndConvert` and `_redeem` functions of the SigmaController contract, the minimum output parameters of `sy.deposit` and `sy.redeem` are set to 0, respectively, abandoning the slippage protection mechanism. This design may expose transactions to the risks of sandwich attacks and price manipulation. Attackers can manipulate prices through pre-transactions, causing users to receive far less tokens than expected.

- contracts/contracts/sigma/SigmaController.sol#L242-L268, L338-L362

```
  function _transferInCollAndConvert(
    OpColl memory opColl,
    OpDebt memory opDebt,
    bytes memory data
  ) internal returns (int256 newColl, int256 newRawColl, int256 newDebt) {
        //...
    sy.deposit(address(this), address(slisBNB), amountOut, 0);
        //...
  }

  function _redeem(
    OpPosition memory opPosition,
    OpColl memory opColl,
    OpDebt memory opDebt,
    OpParameter memory opParam
  ) internal returns (int256 newColl, int256 newRawColl, int256 newDebt, uint256
amountOut) {
        //...
      sy.redeem(address(this), amountOut, address(slisBNB), 0, false);
        //...
  }
```

Solution

It is recommended to set reasonable minimum output parameters and verify whether the return value meets expectations.

Status

Fixed

[N7] [High] The amount of sy tokens actually received does not match the amount of deposit

Category: Design Logic Audit

Content

In the SigmaController contract, the `_transferInCollAndConvert` function directly uses the input `amountOut` as the amount of sy tokens obtained (`newColl`) after depositing slisBNB into the sy contract, rather than using the actual amount returned by the `sy.deposit` function. When the market fluctuates or is manipulated, the actual amount of sy tokens obtained may be lower than `amountOut`, causing the amount authorized and used (`newColl`) in subsequent calls to `fxPoolManager.operate` to be greater than the actual amount of sy tokens held by the contract, thereby causing transaction rollbacks or contract fund losses.

- contracts/contracts/sigma/SigmaController.sol#L146-L181, L242-L268

```
  function deposit(
    OpPosition memory opPosition,
    OpColl memory opColl,
    OpDebt memory opDebt,
    OpParameter memory opParam,
    bytes memory data
  ) external payable nonReentrant returns (uint256 positionId, int256 newColl, int256
newDebt) {
    require(opPosition.pool != address(0), "Pool address cannot be zero");
    require(opParam.receiver == msg.sender, "Receiver must be the sender");
    require(!(opColl.collAmount == 0 && opDebt.debtAmount == 0), "No supply and no
borrow");

    _onlySupportedCollToken(opColl.collToken);
    _onlySupportedDebtToken(opPosition.pool, opDebt.debtToken);
    bool isOpen = _isEmptyPosition(opPosition.pool, opPosition.positionId);

    uint256 debtBalance = IERC20(opDebt.debtToken).balanceOf(address(this));

    if (opPosition.positionId != 0) {

      IERC721(opPosition.pool).transferFrom(msg.sender, address(this),
opPosition.positionId);
    }

    int256 newRawColl;
    (newColl, newRawColl, newDebt) = _transferInCollAndConvert(opColl, opDebt, data);

    // deposit sy to fx pool
    IERC20(sy).forceApprove(address(fxPoolManager), uint256(newColl));
    positionId = fxPoolManager.operate(opPosition.pool, opPosition.positionId,
newColl, newDebt);

    uint256 debtDelta = IERC20(opDebt.debtToken).balanceOf(address(this)) -
debtBalance;

    _updateEntryPosition(opPosition.pool, positionId, newRawColl);

    if (opPosition.positionId != 0) {
      require(positionId == opPosition.positionId, "PositionId mismatch");
    }

    // transfer out debt
    if (debtDelta > 0) {
      IERC20(opDebt.debtToken).safeTransfer(opParam.receiver, debtDelta);
    }

    // transfer xBNB to the user
    IERC721(opPosition.pool).transferFrom(address(this), opParam.receiver,
positionId);

    emit Deposit(opPosition.pool, positionId, opParam.receiver, newColl, newRawColl,
newDebt, isOpen);
  }

  function _transferInCollAndConvert(
    OpColl memory opColl,
    OpDebt memory opDebt,
    bytes memory data
  ) internal returns (int256 newColl, int256 newRawColl, int256 newDebt) {
    newDebt = int256(opDebt.debtAmount);
    if (opColl.collAmount == 0) {
      return (newColl, newRawColl, newDebt);
    }

    if (opColl.collToken != address(0)) {
      IERC20(opColl.collToken).safeTransferFrom(msg.sender, address(this),
opColl.collAmount);
    }

    uint256 amountOut = opColl.collAmount;
    if (opColl.collToken != address(slisBNB)) {
      (uint256 tokenInAmount, address swapTarget, bytes memory swapData) =
abi.decode(data, (uint256, address, bytes));
      require(tokenInAmount == opColl.collAmount, "Token in amount mismatch");
      amountOut = _swap(opColl.collToken, address(slisBNB), opColl.collAmount,
swapTarget, swapData);
    }

    slisBNB.forceApprove(address(sy), amountOut);
    sy.deposit(address(this), address(slisBNB), amountOut, 0);
    newColl = int256(amountOut);
    newRawColl = _scaleUp(newColl);
  }
```

Solution

It is recommended to use the return value of the sy.deposit function as the amount to be deposited in the fx pool.

Status

Fixed

[N8] [Low] Missing Position ID Verification

Category: Design Logic Audit

Content

In the `_redeem` function of the SigmaController contract, the return value `actualPositionId` of `fxPoolManager.operate` is ignored, and it is not verified whether the return value matches the passed `opPosition.positionId`.

- contracts/sigma/SigmaController.sol#L339-L362

```
  function _redeem(
    OpPosition memory opPosition,
    OpColl memory opColl,
    OpDebt memory opDebt,
    OpParameter memory opParam
  ) internal returns (int256 newColl, int256 newRawColl, int256 newDebt, uint256
amountOut) {
    (newColl, newRawColl, newDebt) = _transferInDebt(opPosition, opColl, opDebt);

    // transfer xBNB to this contract
    IERC721(opPosition.pool).transferFrom(msg.sender, address(this),
opPosition.positionId);

    // withdraw sy from fx pool
    uint256 syBalance = sy.balanceOf(address(this));
    fxPoolManager.operate(opPosition.pool, opPosition.positionId, newColl, newDebt);
    amountOut = sy.balanceOf(address(this)) - syBalance;
    if (amountOut > 0) {
      // sy => slisBNB
      sy.redeem(address(this), amountOut, address(slisBNB), 0, false);
    }
    _updateEntryPosition(opPosition.pool, opPosition.positionId, newRawColl);

    // transfer xBNB to the user
    IERC721(opPosition.pool).transferFrom(address(this), opParam.receiver,
opPosition.positionId);
  }
```

Solution

It is recommended to get the return value of the fxPoolManager.operate function and verify whether it matches the passed in opPosition.positionId.

Status

Fixed

[N9] [High] The amount of slisBNB transferred and actually received does not match

Category: Design Logic Audit

Content

In the `_redeem` function of SigmaController, the return value of `sy.redeem` (the actual amount of slisBNB redeemed) was ignored. The function incorrectly passed the redeemed amount of sy tokens `amountOut` to the `redeemInstant` function as the amount of slisBNB to be transferred out, instead of using the actual amount of slisBNB obtained from `sy.redeem`. Since the exchange rate between sy tokens and slisBNB may not be 1:1, this will cause the amount of slisBNB transferred to the user to be inconsistent with the actual redemption amount, which may cause financial losses under certain conditions.

- contracts/sigma/SigmaController.sol#L199-L231, L338-L362

```
  function redeemInstant(
    OpPosition memory opPosition,
    OpColl memory opColl,
    OpDebt memory opDebt,
    OpParameter memory opParam
  ) external payable nonReentrant returns (int256 newColl, int256 newDebt, uint256
tokenOut) {
    require(opPosition.pool != address(0), "Pool address cannot be zero");
    require(opPosition.positionId != 0, "PositionId cannot be zero");
    require(opParam.receiver == msg.sender, "Receiver must be the sender");
    require(!(opDebt.debtAmount == 0 && opColl.collAmount == 0), "No supply and no
borrow");
```

```
    require(opColl.collToken == address(slisBNB), "Only support instant redeem with
slisBNB");
    _onlySupportedDebtToken(opPosition.pool, opDebt.debtToken);

    int256 newRawColl;
    uint256 amountOut;
    (newColl, newRawColl, newDebt, amountOut) = _redeem(opPosition, opColl, opDebt,
opParam);

    // transfer out collaterals
    tokenOut = _transferOutColl(opParam.receiver, amountOut, opColl);

    bool isClose = _isEmptyPosition(opPosition.pool, opPosition.positionId);
    emit RedeemInstant(
      opPosition.pool,
      opPosition.positionId,
      opParam.receiver,
      newColl,
      newRawColl,
      newDebt,
      isClose,
      tokenOut
    );
  }

  function _redeem(
    OpPosition memory opPosition,
    OpColl memory opColl,
    OpDebt memory opDebt,
    OpParameter memory opParam
  ) internal returns (int256 newColl, int256 newRawColl, int256 newDebt, uint256
amountOut) {
    (newColl, newRawColl, newDebt) = _transferInDebt(opPosition, opColl, opDebt);

    // transfer xBNB to this contract
    IERC721(opPosition.pool).transferFrom(msg.sender, address(this),
opPosition.positionId);

    // withdraw sy from fx pool
    uint256 syBalance = sy.balanceOf(address(this));
    fxPoolManager.operate(opPosition.pool, opPosition.positionId, newColl, newDebt);
    amountOut = sy.balanceOf(address(this)) - syBalance;
    if (amountOut > 0) {
      // sy => slisBNB
      sy.redeem(address(this), amountOut, address(slisBNB), 0, false);
    }

    _updateEntryPosition(opPosition.pool, opPosition.positionId, newRawColl);
    // transfer xBNB to the user
    IERC721(opPosition.pool).transferFrom(address(this), opParam.receiver,
opPosition.positionId);
  }
```

Solution

It is recommended to obtain the return value amountTokenOut of the sy.redeem function and use this value as the amount of slisBNB tokens transferred to the user.

Status

Fixed

[N10] [Suggestion] Redundant code

Category: Others

Content

In the SigmaController contract, the `MIN_COLLATERAL`, `MIN_DEBT`, `FEE_PRECISION`, `SLIPPAGE_TOLERANCE_PRECISION`, `MAX_SLIPPAGE`, `MIN_SLIPPAGE` variables are not used.

- contracts/sigma/SigmaController.sol#L42, L44, L48, L50, L51, L52

```
  int256 internal constant MIN_COLLATERAL = 1e9;
  int256 internal constant MIN_DEBT = 1e9;
  uint256 internal constant PRECISION = 1e18;
  uint256 internal constant FEE_PRECISION = 1e9;
  uint256 internal constant SLIPPAGE_TOLERANCE_PRECISION = 1e9;
  uint256 internal constant MAX_SLIPPAGE = 1e8;
  uint256 internal constant MIN_SLIPPAGE = 1e5;
```

Solution

It is recommended to delete the redundant code.

Status

Fixed

[N11] [Low] Token swaps lack slippage protection

Category: Design Logic Audit

Content

In the `_swap` function of the SigmaController contract, the `amountOut` value obtained after the token swap is only calculated by the balance difference before and after the swap, without comparing and verifying with the minimum swap amount expected by the user. If the `swapTarget` contract itself does not implement a slippage protection mechanism, the transaction may face the risk of a sandwich attack, especially in trading pairs with low liquidity.

- contracts/sigma/SigmaController.sol#L407-L445

```
function _swap(
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  address swapTarget,
  bytes memory swapData
) internal returns (uint256 amountOut) {
  _onlySupportedSwapTarget(swapTarget);

  if (amountIn == 0) return 0;

  amountOut = _balanceOf(tokenOut, address(this));
  if (tokenIn != address(0)) {
    IERC20(tokenIn).forceApprove(swapTarget, amountIn);
    (bool success, ) = swapTarget.call(swapData);
    // below lines will propagate inner error up
    if (!success) {
      // solhint-disable-next-line no-inline-assembly
      assembly {
        let ptr := mload(0x40)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)
        revert(ptr, size)
      }
    }
  } else {
    (bool success, ) = swapTarget.call{ value: amountIn }(swapData);
    if (!success) {
      // solhint-disable-next-line no-inline-assembly
      assembly {
        let ptr := mload(0x40)
        let size := returndatasize()
        returndatacopy(ptr, 0, size)
        revert(ptr, size)
      }
    }
  }
  amountOut = _balanceOf(tokenOut, address(this)) - amountOut;
}
```

Solution

It is recommended to compare and verify the amountOut parameter with the minimum redemption amount expected by the user in the function.

Status

Fixed

[N12] [Information] Lack of validation of swap call data

Category: Others

Content

In the `_swap` function of the SigmaController contract, users can pass in any `swapData` bytecode through the `data` parameter, and the bytecode will be directly used to call the swap contract in the whitelist. Due to the lack of validity verification and format checking of the swapData content, malicious users may construct special call data to trigger unexpected functions of the swap contract.

- contracts/sigma/SigmaController.sol#L407-L445

```
function _swap(
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  address swapTarget,
  bytes memory swapData
) internal returns (uint256 amountOut) {
  _onlySupportedSwapTarget(swapTarget);

  if (amountIn == 0) return 0;

  amountOut = _balanceOf(tokenOut, address(this));
  if (tokenIn != address(0)) {
    IERC20(tokenIn).forceApprove(swapTarget, amountIn);
    (bool success, ) = swapTarget.call(swapData);
    // below lines will propagate inner error up
    if (!success) {
      // solhint-disable-next-line no-inline-assembly
      assembly {
        let ptr := mload(0x40)
```

```
          let size := returndatasize()
          returndatacopy(ptr, 0, size)
          revert(ptr, size)
        }
      }
    } else {
      (bool success, ) = swapTarget.call{ value: amountIn }(swapData);
      if (!success) {
        // solhint-disable-next-line no-inline-assembly
        assembly {
          let ptr := mload(0x40)
          let size := returndatasize()
          returndatacopy(ptr, 0, size)
          revert(ptr, size)
        }
      }
    }
    amountOut = _balanceOf(tokenOut, address(this)) - amountOut;
  }
```

Solution

It is recommended that when adding supportedSwapTargets mapping, the Owner role needs to pay attention to whether there are special function interfaces in the swap contract.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002507010001	SlowMist Security Team	2025.06.30 - 2025.07.01	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 3 high risk, 2 medium risk, 3 low risk, 2 suggestion, 2 Information.

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