Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2023.07.31, the SlowMist security team received the DeSyn Protocol team's security audit application for DeSyn Phase4, 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

Desyn is a web3 asset management platform that provides a decentralized asset management infrastructure for everyone around the world. The scope of this audit is to buy/sell the Short-term Treasury Bill Token (STBT) module and the Actions module iterations. Managers can use the stable currency in the closed ETF to buy STBT through the swap function of the bill module, and can sell it at any time. For the new Actions module, users can use the smart join/exit function to convert a token into the token required by the ETF when adding liquidity, or convert the ETF token into a specified token when removing liquidity.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Recommendations for Parameter Declarations	Gas Optimization Audit	Suggestion	Fixed
N2	Modify `tbill` status issue	Others	Low	Fixed
N3	Invalid function return value	Others	Low	Fixed
N4	The transfer return value that does not conform to the EIP20 interface standard	Others	Low	Fixed
N5	Redundant Period enum	Others	Suggestion	Fixed
N6	Missing error message	Others	Suggestion	Fixed
N7	Hardcoded testnet address issue	Others	Suggestion	Acknowledged
N8	collectEndTime is not checked	Design Logic Audit	Critical	Fixed
N9	Missing 0 balance check	Others	Suggestion	Fixed
N10	Decimal conversion causes data mismatch	Arithmetic Accuracy Deviation Vulnerability	Low	Acknowledged
N11	Incorrect DTBT amount update	Design Logic Audit	Critical	Fixed
N12	Invalid execution result capture	Design Logic Audit	Suggestion	Fixed
N13	Typo issue	Others	Suggestion	Fixed
N14	The length of minAmountsOut is not checked	Others	Suggestion	Fixed
N15	Redundant approval operation	Others	Suggestion	Fixed
N16	The validity of the aggregator is not checked	Design Logic Audit	High	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/Meta-DesynLab/desyn-modules/tree/stbt-pro/contracts/bill

commit: 65ef8dd5a34277f945cdd4c8ac1968b7905f2dd0

https://github.com/Meta-DesynLab/desyn-contracts-fork/tree/smart

commit: c66c4d5dc18bd0d6712d14c866a4d6b5abb0ec4b

Fixed Version:

https://github.com/Meta-DesynLab/desyn-modules/tree/stbt-pro/contracts/bill

commit: c835b74531dc44bce3b83f5d49e47f77e2d30e01

https://github.com/Meta-DesynLab/desyn-contracts-fork/tree/smart

commit: 1745a407dbddd1c19dec234004bc5fe52ab39584

Audit Scope:

- desyn-modules/contracts/bill/*.sol
- desyn-contracts-fork/contracts/deploy/Actions.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:

DTBT			
Function Name	Visibility	Mutability	Modifiers
addAdmin	Public	Can Modify State	-
add	Internal	-	-

DTBT			
sub	Internal	-	-
name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
_move	Internal	Can Modify State	-
_mint	Internal	Can Modify State	-
updateDecimals	Public	Can Modify State	_onlyOwner_
allowance	External	-	-
balanceOf	External	-	-
totalSupply	Public	-	-
approve	External	Can Modify State	-
mint	Public	Can Modify State	_onlyOwner_
burn	Public	Can Modify State	_onlyOwner_
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-

TBillSimple			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
swap	External	Can Modify State	-
rebalance	External	Can Modify State	_checkTx
_sendStableCoin	Internal	Can Modify State	_checkTx
_sendSTBT	Internal	Can Modify State	_checkTx

TBillSimple			
_updateDtbtAmount	Internal	Can Modify State	-
_decimalsHandle	Internal	-	-
_invokeRebind	Internal	Can Modify State	-
_invokeUnbind	Internal	Can Modify State	-
updateETFaddr	External	Can Modify State	onlyOwner _config_
updateStbtAddr	External	Can Modify State	onlyOwner _config_
updateStableCoinAddr	External	Can Modify State	onlyOwner _config_
updateStableCoinReceiverAddr	External	Can Modify State	onlyOwner _config_
updateStbtReceiverAddr	External	Can Modify State	onlyOwner _config_

Actions			
Function Name	Visibility	Mutability	Modifiers
create	External	Can Modify State	-
createSmartPool	External	Can Modify State	-
joinPool	External	Can Modify State	-
autoExitSmartPool	External	Can Modify State	-
joinSmartPool	External	Can Modify State	-
autoJoinSmartPool	External	Can Modify State	-
exitPool	External	Can Modify State	-
setPublicSwap	External	Can Modify State	-
setController	External	Can Modify State	-
setManagersInfo	Public	Can Modify State	-
_beforeOwnerChange	Internal	Can Modify State	-

Actions			
snapshotBeginAssets	External	Can Modify State	-
snapshotEndAssets	External	Can Modify State	-
approveUnderlying	External	Can Modify State	-
rebalance	External	Can Modify State	-
finalize	External	Can Modify State	-
setCap	External	Can Modify State	-
whitelistLiquidityProvider	External	Can Modify State	-
removeWhitelistedLiquidityProvider	External	Can Modify State	-
addTokenToWhitelist	Public	Can Modify State	-
claimManagementFee	Public	Can Modify State	-
_safeApprove	Internal	Can Modify State	-
_join	Internal	Can Modify State	-
_exit	Internal	Can Modify State	-
claimKolReward	Public	Can Modify State	-
claimManagersReward	External	Can Modify State	-
_claimManagersReward	Internal	Can Modify State	-
_makeSwap	Public	Can Modify State	-
_getVault	Internal	-	-
_getUserVault	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Suggestion] Recommendations for Parameter Declarations

Category: Gas Optimization Audit

Content

In the DTBT contract, the `_name` and `_symbol` parameters are hardcoded with fixed values, but they are not declared as constants, which will result in additional gas consumption.

The same is true for `stableCoinReceiver`, `sTBTReceiver`, `stbt`, `stableCoin` parameters in the TBillSimple contract.

Code location:

contracts/bill/DTBT.sol

```
    string private _name = 'DTBT';
    string private _symbol = 'DTBT';
```

contracts/bill/TBillSimple.sol

```
    address public stableCoinReceiver = 0x981B62de4864ed5b2c762A4e20bDA01b70EeBb2E;
    address public sTBTReceiver = 0x199E9C9A58e0CF6D26c4e753693644Ca65A4c497;
    IERC20Detailed stbt = IERC20Detailed(0x0f539454d2Effd45E9bFeD7C57B2D48bFd04CB32);
    IERC20Detailed stableCoin =
IERC20Detailed(0x43c7181e745Be7265EB103c5D69F1b7b4EF8763f);
```

Solution

It is recommended to declare these parameters as `constant`.

Status

Fixed

[N2] [Low] Modify `tbill` status issue

Category: Others

Content

In the DTBT contract, the owner role can set the specified user's tbill status to true through the addAdmin function, and `tbill` can perform any mint/burn operations. However, there is no function in the contract to remove the user's tbill status. If the `tbill` role is controlled by a malicious user, the protocol cannot perform permission recovery.

Code location: contracts/bill/DTBT.sol

```
function addAdmin(address tbill) public returns (bool) {
  require(msg.sender == _owner, 'ERR_NOT_OWNER');
  _tbills[tbill] = true;
  emit AdminsUpdate(msg.sender, tbill);
  return true;
}
```

Solution

It is recommended to add the function of removing admin to deal with the above risks.

Status

Fixed

[N3] [Low] Invalid function return value

Category: Others

Content

In the DTBT contract, the owner role can modify the decimals of the contract through the updateDecimals function.

This function defines the return value, but it does not return any value.

Code location: contracts/bill/DTBT.sol

```
function updateDecimals(uint8 newDecimals) public _onlyOwner_ returns (bool) {
  emit UpdateDecimals(_decimals, newDecimals);
  _decimals = newDecimals;
}
```

Solution

It is recommended to return true after the function is executed or remove the return value definition.

Status

Fixed

[N4] [Low] The transfer return value that does not conform to the EIP20 interface standard

Category: Others

Content

In the DTBT contract, users can transfer DTBT tokens through the transferFrom function. However, the implementation of this function does not comply with the return value standard specified in EIP20.

Code location: contracts/bill/DTBT.sol

```
  function transferFrom(address src, address dst, uint amt) external {
    require(msg.sender == src || amt <= _allowance[src][msg.sender],
'ERR_BTOKEN_BAD_CALLER');
    _move(src, dst, amt);
    if (msg.sender != src && _allowance[src][msg.sender] != uint(-1)) {
      _allowance[src][msg.sender] = sub(_allowance[src][msg.sender], amt);
      emit Approval(msg.sender, dst, _allowance[src][msg.sender]);
    }
  }
```

Solution

It is recommended to implement transfer functions according to the standards in EIP20.

Status

Fixed

[N5] [Suggestion] Redundant Period enum

Category: Others

Content

The Period enum is defined in the TBillSimple contract, but it is not used in the contract.

Code location: contracts/bill/TBillSimple.sol

```
  enum Period {
    NOTSTRAT,
    DEPOSIT,
    REDEEM
  }
```

Solution

It is recommended to remove the redundant Period enumeration.

Status

Fixed

[N6] [Suggestion] Missing error message

Category: Others

Content

In the DTBT contract, the add/sub function users add and subtract mathematically. It will perform overflow checks through `require`, but no error message will be thrown when the check fails, which will prevent users from intuitively obtaining the cause of the error.

Code location: contracts/bill/DTBT.sol

```
function add(uint a, uint b) internal pure returns (uint c) {
  require((c = a + b) >= a);
}

function sub(uint a, uint b) internal pure returns (uint c) {
  require((c = a - b) <= a);
}
```

Solution

It is recommended that error messages be thrown during `require` checks.

Status

Fixed

[N7] [Suggestion] Hardcoded testnet address issue

Category: Others

Content

In the TBillSimple contract, the parameters of stableCoinReceiver, sTBTReceiver, stbt, and stableCoin are all hardcoded with the address of the Goerli testnet. If the hard-coded address is not modified when the protocol is launched on the mainnet, the protocol will not work properly.

Code location: contracts/bill/TBillSimple.sol

```
address public stableCoinReceiver = 0x981B62de4864ed5b2c762A4e20bDA01b70EeBb2E;
address public sTBTReceiver = 0x199E9C9A58e0CF6D26c4e753693644Ca65A4c497;
IERC20Detailed stbt = IERC20Detailed(0x0f539454d2Effd45E9bFeD7C57B2D48bFd04CB32);
```

```
  IERC20Detailed stableCoin =
IERC20Detailed(0x43c7181e745Be7265EB103c5D69F1b7b4EF8763f);
```

Solution

Please note that the hard-coded test address should be modified when the mainnet is launched.

Status

Acknowledged; After communicating with the project team, the project team stated that it will carefully check whether the hard-coded address is correct when the mainnet is deployed.

[N8] [Critical] collectEndTime is not checked

Category: Design Logic Audit

Content

In the TBillSimple contract, managers can buy/sell STBT through the swap function. It checks whether the current time is less than the closing end time of the ETF, but does not check whether the current time is greater than the fundraising end time of the ETF. Although it will check whether the isCompletedCollect status of the ETF is true through `_checkTx`, users can still add liquidity to the ETF at this time. If the swap operation is performed at this time, the stableCoin will be unbound, and users will not be able to add liquidity during the fundraising period, which is inconsistent with the design expectation.

Code location: contracts/bill/TBillSimple.sol

```
function swap() external {
  if (currentPeriod == CurrentPeriod.WaitSendStableCoin) {
    (, , , uint256 closureEndTime, , , , , , , ) = etf.etfStatus();
    require(closureEndTime > block.timestamp, 'ERR_ETF_HAS_CLOSURE');
    _sendStableCoin();
    currentPeriod = CurrentPeriod.WaitSendSTBT;
  } else {
    _sendSTBT();
    currentPeriod = CurrentPeriod.WaitSendStableCoin;
  }
}
```

Solution

It is recommended to check that the current time is greater than collectEndTime and less than closureEndTime

during the swap operation.

Status

Fixed

[N9] [Suggestion] Missing 0 balance check

Category: Others

Content

In the TBillSimple contract, managers can synchronize the stableCoin position data in bPool through the rebalance function. But it does not check whether the stableCoin balance in bPool is greater than 0, and if the balance is 0, the rebind operation is meaningless.

Code location: contracts/bill/TBillSimple.sol

```
  function rebalance() external _checkTx {
    require(currentPeriod == CurrentPeriod.WaitSendStableCoin,
'ERR_NOT_WAIT_SEND_STABLECOIN');
    (, , , uint256 closureEndTime, , , , , , , ) = etf.etfStatus();
    require(closureEndTime < block.timestamp, 'ERR_ETF_HAS_NOT_CLOSURE');

    uint bal = stableCoin.balanceOf(address(etf.bPool()));
    _updateDtbtAmount(0);
    _invokeRebind(address(stableCoin), bal, 50e18);
  }
```

Solution

It is recommended to check whether the stableCoin balance in bPool is greater than 0 during rebalance.

Status

Fixed

[N10] [Low] Decimal conversion causes data mismatch

Category: Arithmetic Accuracy Deviation Vulnerability

Content

In the TBillSimple contract, managers can buy/sell STBT through the swap function. It will adjust the ETF position through `_sendStableCoin` and `_sendSTBT` functions respectively, and use `_decimalsHandle` function to

perform decimal conversion between stableCoin and stbt tokens. However, due to the fact that solidity will truncate decimals when performing division operations, there will be some precision loss when converting large decimals to small decimals. This will cause the calculation result of `_decimalsHandle` to be smaller than the actual amount of tokens received by the ETF, causing the `Record` data of the ETF to be inconsistent with the actual balance.

In fact, the rebalance function in the TBillSimple contract can alleviate the problem of the mismatch between the stableCoin balance and `Record` data, but the contract does not implement the rebalance operation for STBT tokens.

Code location: contracts/bill/TBillSimple.sol

```
function _sendStableCoin() internal _checkTx {
  ...
  uint tbtNewBal = _decimalsHandle(bal, decimalsStableCoin, decimalsDtbt);
  ...
}

function _sendSTBT() internal _checkTx {
  ...
  uint stableCoinNewBal = _decimalsHandle(bal, decimalsStbt, decimalsStableCoin);
  ...
}

function _decimalsHandle(
  uint256 currentValue,
  uint currentDecimals,
  uint targetDecimals
) internal pure returns (uint256) {
  if (currentDecimals >= targetDecimals)
    return currentValue / 10 ** (currentDecimals - targetDecimals);
  if (currentDecimals < targetDecimals)
    return currentValue * 10 ** (targetDecimals - currentDecimals);
}
```

Solution

It is recommended to implement the function of synchronizing the STBT actual balance and `Record` data in the ETF.

Status

Acknowledged; After communicating with the project team, the project team said that because of the rebase

mechanism, the amount of stbt will change every day, so we will use balanceOf to track the actual balance of stbt instead, and after the closure period, the stbt needs to be converted to stable coin.

[N11] [Critical] Incorrect DTBT amount update

Category: Design Logic Audit

Content

In the TBillSimple contract, it uses DTBT instead of STBT for timely accounting operations. When ETF sells STBT, it will burn the balance of DTBT tokens in bPool and unbond them, but it does update the DTBT with the same balance through `_updateDtbtAmount` function by mistake, which will make the amount of DTBT tokens in bPool not decrease , which is not as expected by design.

Code location: contracts/bill/TBillSimple.sol

```
function _sendSTBT() internal _checkTx {
  ...
  _updateDtbtAmount(bal);
  ...
}
```

Solution

It is recommended to perform `_updateDtbtAmount(0)` operation when selling STBT.

Status

Fixed

[N12] [Suggestion] Invalid execution result capture

Category: Design Logic Audit

Content

In the TBillSimple contract, the `_invokeUnbind` function is used to unbind the specified token in the ETF. It captures the result of the operation through `try-catch`, but does not process the result of the operation in the catch.

Code location: contracts/bill/TBillSimple.sol

```
  function _invokeUnbind(IETF _etf, address _token) internal {
    bytes memory callData = abi.encodeWithSignature('unbindPure(address)', _token);

    try _etf.execute(address(_etf.bPool()), 0, callData, false) returns (bytes
memory) {} catch (
      bytes memory
    ) {}
  }
```

Solution

If you do not check the execution result, it is recommended to remove the `try-catch` operation. If you need to check the execution result, implement the specific checking logic in the catch.

Status

Fixed

[N13] [Suggestion] Typo issue

Category: Others

Content

In the smart join/exit pool function of the Actions contract, it will receive the handleToken parameter to specify the source token or target token of the token swap. But it is wrong to write this parameter as handleToekn.

Code location: contracts/deploy/Actions.sol

```
    function autoExitSmartPool(
        IConfigurableRightsPool pool,
        uint poolAmountIn,
        uint[] memory minAmountsOut,
        uint minSwapReturn,
        address handleToekn,
        IAggregator.SwapInfoBase calldata swapBase,
        IAggregator.SwapData[] memory swapDatas) external {
        ...
        IERC20 receiveToken = IERC20(handleToekn);
        ...
        for(uint j = 0; j < len; j++) {
            ...
            if(tokens[j] != address(receiveToken)){
                ...
                _makeSwap(swapBase, swapData, IERC20(handleToekn));
            }
```

```
        }
        ...
    }

    function autoJoinSmartPool(
        IConfigurableRightsPool pool,
        address kol,
        uint issueAmount,
        uint poolAmountOut,
        address handleToekn,
        IAggregator.SwapInfoBase calldata swapBase,
        IAggregator.SwapData[] memory swapDatas
    ) external {
        ...
        IERC20 issueToken = IERC20(handleToekn);
        ...
        for (uint i; i < poolTokens.length; i++) {
            ...
            poolTokens[i] == handleToekn
            ...
        }
        ...
        emit SmartJoinPool(msg.sender, handleToekn, issueAmount, address(pool),
pool.balanceOf(address(this)));
    }
```

Solution

It is recommended to change typos to improve code readability.

Status

Fixed

[N14] [Suggestion] The length of minAmountsOut is not checked

Category: Others

Content

In the Actions contract, the autoExitSmartPool function user removes the user's liquidity and converts the obtained tokens into the specified tokens, and performs a slippage check through the minAmountsOut parameter passed in by the user. However, the function does not check whether the length of the minAmountsOut list is the same as the length of the tokens. If the length of the incoming list is shorter than the expected length, the transaction may fail and waste gas.

Code location: contracts/deploy/Actions.sol

```
    function autoExitSmartPool(
        IConfigurableRightsPool pool,
        uint poolAmountIn,
        uint[] memory minAmountsOut,
        uint minSwapReturn,
        address handleToekn,
        IAggregator.SwapInfoBase calldata swapBase,
        IAggregator.SwapData[] memory swapDatas) external {
        address[] memory tokens = pool.bPool().getCurrentTokens();
        uint len = swapDatas.length;
        require(tokens.length == swapDatas.length,"ERR_TOKENLENGTH_MISMATCH");
        ...
    }
```

Solution

It is recommended to check that the minAmountsOut list has the same length as the tokens.

Status

Fixed

[N15] [Suggestion] Redundant approval operation

Category: Others

Content

In the Actions contract, the autoExitSmartPool function user removes user liquidity and converts the obtained tokens into specified tokens. It will first approve the pool tokens in this contract to the contract itself, and then perform `_exit` operation. But the self-approval operation is meaningless. If the contract needs to transfer tokens, it can be transferred directly through transfer without approval, which is a waste of gas.

Code location: contracts/deploy/Actions.sol

```
    function autoExitSmartPool(
        IConfigurableRightsPool pool,
        uint poolAmountIn,
        uint[] memory minAmountsOut,
        uint minSwapReturn,
        address handleToekn,
        IAggregator.SwapInfoBase calldata swapBase,
        IAggregator.SwapData[] memory swapDatas) external {
```

```
        ...
        IERC20(pool).safeApprove(address(this), 0);
        IERC20(pool).safeApprove(address(this), type(uint256).max);
        ...
    }
```

Solution

It is recommended to remove redundant self-approval actions.

Status

Fixed

[N16] [High] The validity of the aggregator is not checked

Category: Design Logic Audit

Content

In the Actions contract, the smart join/exit pool mode will help users exchange tokens through the `_makeSwap` function. It performs token exchange operations by calling the aggregator address specified by the user. It stipulates that the aggregator is composed of UNISWAPV2, UNISWAPV3, and ONEINCH through the SwapType enumeration, but in fact, the contract does not check whether the aggregator called by the user is a real and valid address. If the user is phished or the wrong address is passed in, it will result in a loss of funds or use the Actions contract to perform sensitive operations on the ETF.

For example: when performing the swap through ONEINCH, it will not check the address of the caller and can pass in any call data. This allows the attacker to perform operations such as setController, approveUnderlying, addTokenToWhitelist, etc. on the pool when the user is phished.

Code location: contracts/deploy/Actions.sol

```
    function _makeSwap(IAggregator.SwapInfoBase calldata swapBase,
IAggregator.SwapData memory swapData, IERC20 swapAcceptToken) public returns (uint256
postSwap) {
        // IERC20 swapAcceptToken = IERC20(swapData.token1);
        uint preSwap = swapAcceptToken.balanceOf(address(this));

        if (swapBase.swapType == IAggregator.SwapType.UNISWAPV3) {
            (uint256 minReturn, uint256[] memory pools) = abi.decode(swapData.data,
(uint256, uint256[]));
            IAggregator(swapBase.aggregator).uniswapV3Swap(swapData.quantity,
```

```
minReturn, pools);
        } else if (swapBase.swapType == IAggregator.SwapType.UNISWAPV2) {
            (uint256 minReturn, address[] memory paths) = abi.decode(swapData.data,
(uint256, address[]));

IAggregator(swapBase.aggregator).swapExactTokensForTokens(swapData.quantity,
minReturn, paths, address(this), SafeMath.add(block.timestamp, 1800));
        } else {
            Address.functionCallWithValue(swapBase.aggregator, swapData.data, 0);
            // _validateData(swapBase.quantity, swapBase.data, address(this));
        }

        postSwap = SafeMath.sub(swapAcceptToken.balanceOf(address(this)), preSwap);
    }
```

Solution

It is recommended to set valid UNISWAPV2, UNISWAPV3, and ONEINCH addresses in the contract. When the contract calls the aggregator, it is necessary to check whether the parameters passed in by the user are consistent with the actual ones.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002308020001	SlowMist Security Team	2023.07.31 - 2023.08.02	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 critical risks, 1 high risk, 4 low risks, and 9 suggestions. All the findings were fixed or acknowledged. The code was not deployed to the mainnet.

6 Statement

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

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

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist