Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.03.16, the SlowMist security team received the DeSyn Protocol team's security audit application for Desyn Phase3, 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	-

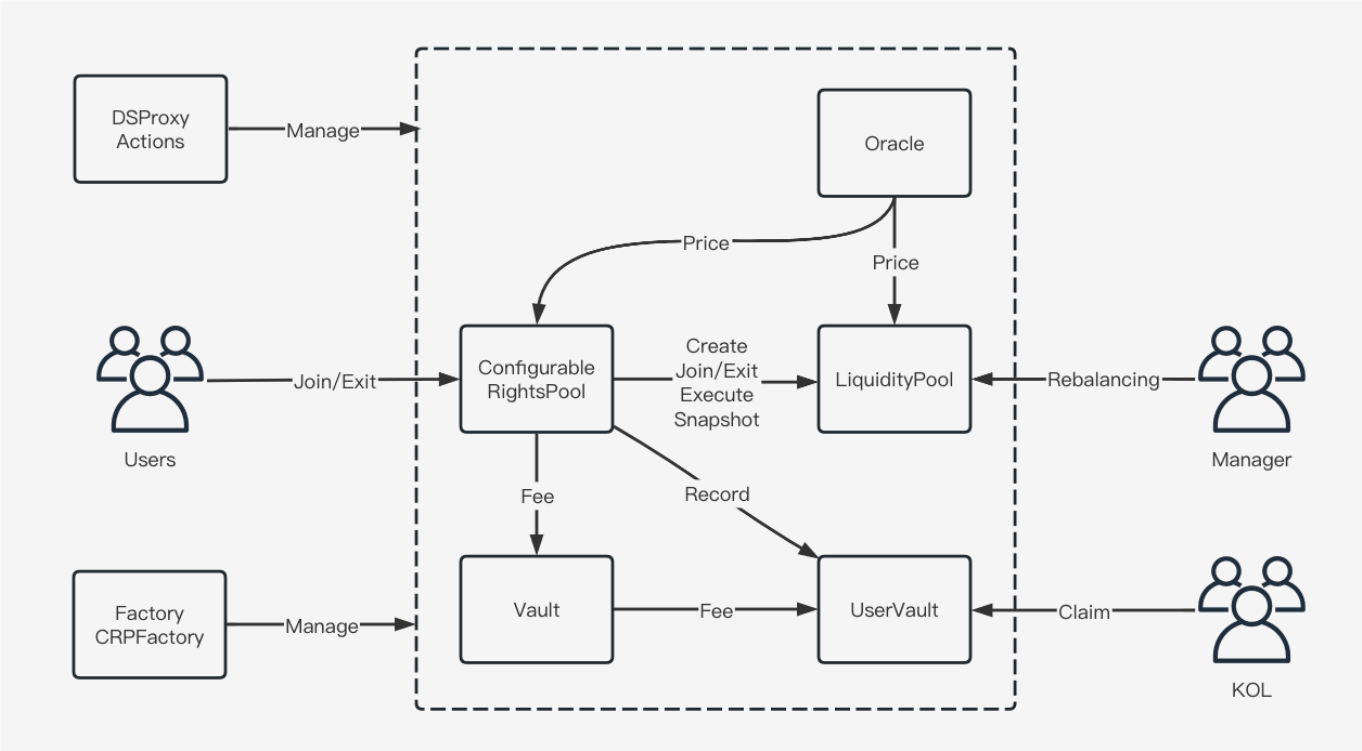
3 Project Overview

3.1 Project Introduction

Desyn is a web3 asset management platform that provides a decentralized asset management infrastructure for everyone around the world.

The Desyn protocol consists mainly of Factory, CRP, LiquidityPool, Actions, Oracle, Vault, UserVault and Rebalance Adapter. Factory is used to create Pool and CRP and manage privileged roles. CRP is used for pool management, including liquidity add/remove, asset snapshot, fee management, etc. Privileged Roles can call the Actions module via DSProxy to perform sensitive operations. Oracles are used to provide reliable token prices to the protocol. Vault is used for fee management of the protocol and a portion of the fee is transferred to UserVault for commission. The Rebalance Adapter is used by managers to manage ETF rebalancing. The

following is a brief architecture diagram:



3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Redundant collectEndTime check	Others	Suggestion	Acknowledged
N2	Redundant closureEndTime check on exitPool	Others	Suggestion	Acknowledged
N3	Redundant performance fee calculation	Others	Suggestion	Fixed
N4	The time allowed for snapshots is too short	Design Logic Audit	Low	Acknowledged
N5	Duplicate decimal processing issue	Design Logic Audit	Critical	Fixed
N6	Decimal processing issue in AllPrice calculation	Design Logic Audit	Critical	Fixed

NO	Title	Category	Level	Status
N7	Redundant code issue	Others	Suggestion	Fixed
N8	Token list conflict in recordTokenInfo	Design Logic Audit	Critical	Fixed
N9	couldManagerClaim not checked when managerClaim	Design Logic Audit	Low	Fixed
N10	Some redundant invoke functions	Others	Suggestion	Acknowledged
N11	Risk of ETF falsification	Design Logic Audit	Critical	Fixed
N12	Risk of the Manager role potentially disrupting the protocol	Design Logic Audit	Medium	Fixed
N13	Double slippage check	Design Logic Audit	Low	Acknowledged
N14	Does not follow the Checks-Effects-Interactions specification	Others	Suggestion	Fixed
N15	Perform strict parameter checking	Others	Suggestion	Acknowledged
N16	Redundant aggregator parameter	Others	Suggestion	Fixed
N17	Incorrect `_validateData` function visibility	Gas Optimization Audit	Low	Fixed
N18	Incorrect approval operation	Design Logic Audit	Medium	Fixed
N19	Risk of excessive authority	Authority Control Vulnerability Audit	High	Acknowledged
N20	Enforce strict permission controls	Design Logic Audit	Medium	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

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

commit: 07a077d6f886aaee8acc7591ce2dd7bb87ce526b

https://github.com/Meta-DesynLab/desyn-modules

commit: 896dd9808e4f21082c2b97332bb4217ea71cffc9

Fixed Version:

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

commit: 9dc4e8ffe5a100c2da77edcfd85b86db5be93376

https://github.com/Meta-DesynLab/desyn-modules

commit: 669776385b7f9a80a57ddefefb3395ab7bee7a17

The main network address of the contract is as follows:

Contract Name	Contract Address	Chain
rebalanceAdapter	0x7ea4b98c00ba7cb8dd3f6e88c8c84957545fc921	Merlin
rebalanceAdapter	0x26C42e15fe6288152F91ea88F0a0e35B78bC94BC	Bitlayer
rebalanceAdapter	0xE7dfA1607F286E37f86915b791282Ccdd2535952	Mode

4.2 Visibility Description

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

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

ConfigurableRightsPool			
init	Public	Can Modify State	-
setCap	External	Can Modify State	logs lock needsBPool onlyOwner
execute	External	Can Modify State	logs lock needsBPool
couldClaimManagerFee	Public	-	-
claimManagerFee	Public	Can Modify State	logs lock onlyAdmin needsBPool
_claimManagerFee	Internal	Can Modify State	-
createPool	External	Can Modify State	onlyOwner logs lock notPaused
joinPool	External	Can Modify State	logs lock needsBPool notPaused
exitPool	External	Can Modify State	logs lock needsBPool notPaused
whitelistLiquidityProvider	External	Can Modify State	onlyOwner lock logs
removeWhitelistedLiquidityProvider	External	Can Modify State	onlyOwner lock logs
canProvideLiquidity	External	-	-
hasPermission	External	-	-
getRightsManagerVersion	External	-	-
getDesynSafeMathVersion	External	-	-
getSmartPoolManagerVersion	External	-	-
mintPoolShareFromLib	Public	Can Modify State	-
pushPoolShareFromLib	Public	Can Modify State	-
pullPoolShareFromLib	Public	Can Modify State	-

ConfigurableRightsPool			
burnPoolShareFromLib	Public	Can Modify State	-
createPoolInternal	Internal	Can Modify State	-
addTokenToWhitelist	External	Can Modify State	onlyOwner
_verifyWhiteToken	Public	-	-
_pullUnderlying	Internal	Can Modify State	needsBPool
_pushUnderlying	Internal	Can Modify State	needsBPool
_mint	Internal	Can Modify State	-
_mintPoolShare	Internal	Can Modify State	-
_pushPoolShare	Internal	Can Modify State	-
_pullPoolShare	Internal	Can Modify State	-
_burnPoolShare	Internal	Can Modify State	-
snapshotBeginAssets	External	Can Modify State	-
beginFundAssets	External	-	-
endFundAssets	External	-	-
snapshotEndAssets	Public	Can Modify State	-
snapshotAssets	Public	Can Modify State	-
_getPoolTokensInfo	Internal	-	-

LiquidityPool			
Function Name	Visibility	Mutability	Modifiers

LiquidityPool			
<Constructor>	Public	Can Modify State	-
isPublicSwap	External	-	-
isFinalized	External	-	-
isBound	External	-	-
getNumTokens	External	-	-
getCurrentTokens	External	-	_viewlock_
getFinalTokens	External	-	_viewlock_
getDenormalizedWeight	External	-	_viewlock_
getTotalDenormalizedWeight	External	-	_viewlock_
getNormalizedWeight	External	Can Modify State	_viewlock_
getBalance	Public	-	_viewlock_
getController	External	-	_viewlock_
setController	External	Can Modify State	_logs_ _lock_
setPublicSwap	External	Can Modify State	_logs_ _lock_
finalize	External	Can Modify State	_logs_ _lock_
bind	External	Can Modify State	_logs_
rebind	Public	Can Modify State	_logs_ _lock_
execute	External	Can Modify State	_logs_ _lock_
unbind	External	Can Modify State	_logs_ _lock_
unbindPure	External	Can Modify State	_logs_ _lock_
rebindPure	Public	Can Modify State	_logs_ _lock_
gulp	External	Can Modify State	_logs_ _lock_

LiquidityPool			
joinPool	External	Can Modify State	_logs_ _lock_
exitPool	External	Can Modify State	_logs_ _lock_
_pullUnderlying	Internal	Can Modify State	-
_pushUnderlying	Internal	Can Modify State	-
_pullPoolShare	Internal	Can Modify State	-
_pushPoolShare	Internal	Can Modify State	-
_mintPoolShare	Internal	Can Modify State	-
_burnPoolShare	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-

DesynChainlinkOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getPrice	Public	Can Modify State	-
getAllPrice	External	Can Modify State	-
getChainlinkPrice	Internal	-	-
getUniswapPrice	Internal	-	-
setDirectPrice	External	Can Modify State	onlyAdmin
setFeed	External	Can Modify State	onlyAdmin
getFeed	Public	-	-
assetPrices	External	-	-
compareStrings	Internal	-	-
setAdmin	External	Can Modify State	onlyAdmin

Oracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setNewTokensInfo	External	Can Modify State	onlyOwner
setNewTokenInfo	Public	Can Modify State	onlyOwner
update	External	Can Modify State	-
consult	External	-	-
setPERIOD	External	Can Modify State	onlyOwner

UserVault			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
getManagerClaimBool	External	-	-
depositToken	External	Can Modify State	onlyVault
getPoolReward	External	-	-
getKolReward	External	-	-
kolClaim	External	Can Modify State	-
managerClaim	External	Can Modify State	-
getManagerReward	External	-	-
getUnManagerReward	External	Can Modify State	-
getPoolFeeTypes	External	-	-
getManagerFeeTypes	External	-	-
_getKolsFeeTypes	Internal	-	-
getKolFeeType	External	-	-

UserVault			
getKolsReward	External	-	-
getUnKolReward	External	Can Modify State	-
recordTokenInfo	External	Can Modify State	-
setPoolParams	External	Can Modify State	onlyCrpFactory
getKolsAdr	External	-	-
getPoolConfig	External	-	-
setBlackList	External	Can Modify State	onlyOwner _logs_
setCrpFactory	External	Can Modify State	onlyOwner _logs_
claimToken	External	Can Modify State	onlyOwner
claimEther	External	Payable	onlyOwner
setVaultAdr	External	Can Modify State	onlyOwner _logs_
getKolHasClaimed	External	-	-
getManageHasClaimed	External	-	-
getKolUserInfo	External	-	-
getUserKolInfo	External	-	-
_updatePool	Internal	Can Modify State	-
_arrIncludeAddr	Internal	-	-
_transferHandle	Internal	Can Modify State	-
_levelJudge	Internal	-	-
_isClosePool	Internal	-	-
_computeKolTotalReward	Internal	-	-

Vault			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
depositManagerToken	External	Can Modify State	-
depositIssueRedeemPToken	External	Can Modify State	-
getManagerClaimBool	External	-	-
setBlackList	External	Can Modify State	onlyOwner _logs_
setUserVaultAdr	External	Can Modify State	onlyOwner _logs_
setCrpFactory	External	Can Modify State	onlyOwner _logs_
claimToken	External	Can Modify State	onlyOwner
claimEther	External	Payable	onlyOwner
setManagerRatio	External	Can Modify State	onlyOwner _logs_
setIssueRatio	External	Can Modify State	onlyOwner _logs_
setRedeemRatio	External	Can Modify State	onlyOwner _logs_
setPerfermanceRatio	External	Can Modify State	onlyOwner _logs_
managerClaim	Public	Can Modify State	-
getManagerFeeTypes	External	-	-
getUnManagerReward	External	-	-
_addTokenInPool	Internal	Can Modify State	-
_updateTokenAmountInPool	Internal	Can Modify State	-
_updateTokenAmountInPool	Internal	Can Modify State	-
_depositTokenIM	Internal	Can Modify State	-
_arrIncludeAddr	Internal	-	-

Vault			
_depositTokenRP	Internal	Can Modify State	-
_isClosePool	Internal	-	-
_computeBalance	Internal	-	-
_clearPool	Internal	Can Modify State	-
_recordUserVault	Internal	Can Modify State	-
_transferHandle	Internal	Can Modify State	-
_updateManageHasClaimed	Internal	Can Modify State	-
_getManagerHasClaimed	Internal	-	-

SmartPoolManager			
Function Name	Visibility	Mutability	Modifiers
initRequire	External	-	-
verifyTokenCompliance	External	Can Modify State	-
verifyTokenCompliance	External	Can Modify State	-
createPoolInternalHandle	External	-	-
createPoolHandle	External	-	-
exitPoolHandle	External	-	-
exitPoolHandleA	External	Can Modify State	-
exitPoolHandleB	External	-	-
joinPoolHandle	External	-	-
joinPool	External	-	-
exitPool	External	-	-
verifyTokenComplianceInternal	Internal	Can Modify State	-

SmartPoolManager			
handleTransferInTokens	External	Can Modify State	-
handleClaim	External	Can Modify State	-
handleFeeClaim	External	Can Modify State	-
WhitelistHandle	External	-	-
_pullUnderlying	Internal	Can Modify State	-
_pushUnderlying	Internal	Can Modify State	-

Invoke			
Function Name	Visibility	Mutability	Modifiers
invokeApprove	Internal	Can Modify State	-
invokeTransfer	Internal	Can Modify State	-
strictInvokeTransfer	Internal	Can Modify State	-
invokeUnwrapWETH	Internal	Can Modify State	-
invokeWrapWETH	Internal	Can Modify State	-
invokeMint	Internal	Can Modify State	-
invokeUnbind	Internal	Can Modify State	-
invokeRebind	Internal	Can Modify State	-

RebalanceAdapter			
Function Name	Visibility	Mutability	Modifiers
getUnderlyingInfo	External	-	-
getUnderlyingAllowance	External	-	-
getSig	Private	-	-

RebalanceAdapter			
approveSwapRouter	External	Can Modify State	onlyOwner
approve	External	Can Modify State	onlyManager
rebalance	External	Can Modify State	onlyManager
_rebalance	Internal	Can Modify State	-
_makeSwap	Internal	Can Modify State	-
_validateData	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Suggestion] Redundant collectEndTime check

Category: Others

Content

In the ConfigurableRightsPool contract, users can redeem underlying assets through the exitPool function. When the pool is a closed ETF, it will check isCompletedCollect and collectEndTime to decide whether to perform `_claimManagerFee` operation. But when isCompletedCollect is true, the collectEndTime check will be performed in snapshotEndAssets, and when isCompletedCollect is false, the collectEndTime check will be performed in exitPoolHandleB. Hence the collectEndTime check before the `_claimManagerFee` operation is redundant.

Code location: contracts/base/ConfigurableRightsPool.sol

```
    function exitPool(uint poolAmountIn, uint[] calldata minAmountsOut) external logs
lock needsBPool notPaused {
        uint actualPoolAmountIn;
        bool isCloseEtfCollectEndWithFailure;

        if(etype == SmartPoolManager.Etypes.CLOSED){
            isCloseEtfCollectEndWithFailure = isCompletedCollect == false &&
block.timestamp >= etfStatus.collectEndTime;
            if(!isCloseEtfCollectEndWithFailure){
```

```
                if(hasClaimManageFee == false) {
                    _claimManagerFee();
                    hasClaimManageFee = true;
                }
                if(hasSetBeginFund && !hasSetEndFund) snapshotEndAssets();
            }
        }
        ...
    }
```

Solution

It is recommended to remove the redundant collectEndTime check.

Status

Acknowledged

[N2] [Suggestion] Redundant closureEndTime check on exitPool

Category: Others

Content

In the ConfigurableRightsPool contract, users can redeem underlying assets through the exitPool function. When the pool is a closed ETF and isCloseEtfCollectEndWithFailure is false, it will check that the current time must be greater than `closureEndTime + 5 minutes` before allowing the user to exit.

However, it should be noted that the snapshotEndAssets operation will be performed before this. The snapshotEndAssets function will also check whether the current time is greater than `closureEndTime + 5 minutes`. Only the admin and owner can execute snapshotEndAssets within 5 minutes after the closure period ends. Otherwise, the transaction will be reverted. Therefore, the closureEndTime check in the exitPool function is redundant. When snapshotEndAssets cannot be performed, the entire transaction will be reverted, and subsequent closureEndTime checks will not be performed.

Code location: contracts/base/ConfigurableRightsPool.sol

```
    function exitPool(uint poolAmountIn, uint[] calldata minAmountsOut) external logs
lock needsBPool notPaused {
        ...
            if(!isCloseEtfCollectEndWithFailure){
                if(hasClaimManageFee == false) {
                    _claimManagerFee();
```

```
                    hasClaimManageFee = true;
                }
                if(hasSetBeginFund && !hasSetEndFund) snapshotEndAssets();
            }
        }

        ...
            if (isCloseEtfCollectEndWithFailure) {
                _actualRedeemFee = 0; // collect failure
            } else {
                // TODO
                require(etfStatus.closureEndTime + 5 minutes <=
block.timestamp,"ERR_CLOSURE_TIME_NOT_ARRIVED!");
            }
        }

        ...
    }

    function snapshotEndAssets() public {
        ...
        bool inT1 = (etfStatus.closureEndTime + 5 minutes) >= nowTime ;
        if(inT1) require(adminList[msg.sender] || msg.sender == _owner, "onlyAdmin");
        ...
    }
```

Solution

It is recommended to remove this check, or increase the time interval from when the closure period ends to when the pool can be exited.

Status

Acknowledged; After communicating with the project team, the project team stated that all users including the admin role cannot exit the pool before T+1.

[N3] [Suggestion] Redundant performance fee calculation

Category: Others

Content

In the exitPool function of the ConfigurableRightsPool contract, redeemAndPerformanceFeeReceived, finalAmountOut and redeemFeeReceived are calculated through exitPoolHandleA. However, since the

performance fee will be calculated uniformly in the snapshotEndAssets operation, there is no need to process the performance fee in the exitPool function.

Code location:

contracts/base/ConfigurableRightsPool.sol

```
    function exitPool(uint poolAmountIn, uint[] calldata minAmountsOut) external logs
lock needsBPool notPaused {
        ...
        for (uint i = 0; i < poolTokens.length; i++) {
            (uint redeemAndPerformanceFeeReceived, uint finalAmountOut, uint
redeemFeeReceived) = SmartPoolManager.exitPoolHandleA(
                IConfigurableRightsPool(address(this)),
                bPool,
                poolTokens[i],
                actualAmountsOut[i],
                _actualRedeemFee
            );
            redeemFeesReceived[i] = redeemFeeReceived;
            redeemAndPerformanceFeesReceived[i] = redeemAndPerformanceFeeReceived;

            emit LogExit(msg.sender, poolTokens[i], finalAmountOut);
        }
        ...
    }
```

contracts/libraries/SmartPoolManager.sol

```
    function exitPoolHandleA(
        IConfigurableRightsPool self,
        IBPool bPool,
        address poolToken,
        uint _tokenAmountOut,
        uint redeemFee
    )
        external
        returns (
            uint redeemAndPerformanceFeeReceived,
            uint finalAmountOut,
            uint redeemFeeReceived
        )
    {
        ...
        uint performanceFeeReceived = 0;
        redeemAndPerformanceFeeReceived = DesynSafeMath.badd(performanceFeeReceived,
```

```
redeemFeeReceived);
        finalAmountOut = DesynSafeMath.bsub(_tokenAmountOut,
redeemAndPerformanceFeeReceived);

        ...
    }
```

Solution

It is recommended to be clear about the handling of performance fees in the exitPool function.

Status

Fixed

[N4] [Low] The time allowed for snapshots is too short

Category: Design Logic Audit

Content

In the ConfigurableRightsPool contract when the closed ETF completes collect and the collection period has ended, the current total value of the ETF can be recorded through the snapshotBeginAssets function. However, users can only call the snapshotBeginAssets function within 15 minutes after the end of the collection period. If there is congestion on the chain, it may not be possible to call the snapshot in time within 15 minutes.

Code location: contracts/base/ConfigurableRightsPool.sol

```
    function snapshotBeginAssets() external {
        uint nowTime = block.timestamp;
        // TODO
        require(!hasSetBeginFund && isCompletedCollect && etype ==
SmartPoolManager.Etypes.CLOSED && nowTime <= (etfStatus.collectEndTime + 15 minutes) ,
"ERR_CONDITIONS_NOT_MET");

        ...
    }
```

Solution

It is recommended to relax the time for snapshot calls to ensure that the ETF can enter the closing period normally.

Status

Acknowledged; After communicating with the project team, the project team indicated that this is a period of time that only exists for the testing phase, which is actually longer.

[N5] [Critical] Duplicate decimal processing issue

Category: Design Logic Audit

Content

In the DesynChainlinkOracle contract, the getPrice function is used to obtain the corresponding token price from prices, getChainlinkPrice, and getUniswapPrice, and perform decimal processing. However, decimal has been processed in getChainlinkPrice and getUniswapPrice, and theoretically, the returned decimal will be 1e18. Therefore, processing decimals through decimalDelta will cause decimals to be enlarged.

Note that the amountIn passed in for the consult in the getUniswapPrice function is 1e18, which needs to be ensured that the token decimal in twapOracle matches it in practice

Code location: contracts/deploy/DesynChainlinkOracle.sol

```
    function getPrice(address tokenAddress) public returns (uint price) {
        IERC20 token = IERC20(tokenAddress);
        AggregatorV2V3Interface feed = getFeed(token.symbol());
        if (prices[address(token)] != 0) {
            price = prices[address(token)];
        } else if (address(feed) != address(0)) {
            price = getChainlinkPrice(feed);
        } else {
            try twapOracle.update(address(token)) {} catch {}
            price = getUniswapPrice(tokenAddress);
        }

        (uint decimalDelta, bool isUnderFlow18) =
uint(18).abs(uint(token.decimals()));

        if(isUnderFlow18){
            return price.mul(10**decimalDelta);
        }

        if(!isUnderFlow18){
            return price.div(10**decimalDelta);
        }
    }

    function getUniswapPrice(address tokenAddress) internal view returns (uint) {
```

```
        IERC20 token = IERC20(tokenAddress);
        uint price = twapOracle.consult(tokenAddress, 1e18);
        return price;
    }
```

Solution

It is recommended to uniformly process the decimal of the price to avoid repeated operations that cause the price decimal to be inconsistent with expectations.

Status

Fixed

[N6] [Critical] Decimal processing issue in AllPrice calculation

Category: Design Logic Audit

Content

In the DesynChainlinkOracle contract, the getAllPrice function is used to calculate the total value of the specified amount of tokens. It is calculated by `badd(fundAll, bmul(getPrice(t), tokenAmountOut))`, theoretically the decimal returned by getPrice is 1e18, and the decimal of tokenAmountOut is consistent with the decimal of the token itself. Therefore, multiplying these two values will result in a very large decimal in the final result.

The getNormalizedWeight function is also affected by this, but this is not harmful to normal business.

Note:

Code location:

contracts/deploy/DesynChainlinkOracle.sol

```
    function getAllPrice(address[] calldata poolTokens, uint[] calldata
actualAmountsOut) external returns (uint fundAll) {
        require(poolTokens.length == actualAmountsOut.length, "Invalid Length");

        for (uint i = 0; i < poolTokens.length; i++) {
            address t = poolTokens[i];
            uint tokenAmountOut = actualAmountsOut[i];
            fundAll = badd(fundAll, bmul(getPrice(t), tokenAmountOut));
        }
    }
```

contracts/base/LiquidityPool.sol

```
function getNormalizedWeight(address token) external _viewlock_ returns (uint) {
    require(_records[token].bound, "ERR_NOT_BOUND");
    Oracles oracle = Oracles(_factory.getOracleAddress());
    uint denorm = _records[token].denorm;
    uint price = oracle.getPrice(token);

    uint[] memory _balances = new uint[](_tokens.length);
    for (uint i = 0; i < _tokens.length; i++) {
        _balances[i] = getBalance(_tokens[i]);
    }
    uint totalValue = oracle.getAllPrice(_tokens, _balances);
    uint currentValue = bmul(price, getBalance(token));
    return bdiv(currentValue, totalValue);
}
```

Solution

If it is not expected, it is recommended to return a unified decimal.

Status

Fixed

[N7] [Suggestion] Redundant code issue

Category: Others

Content

In the SmartPoolManager library, the exitPoolHandle function has been deprecated since closed ETF profits are calculated in snapshotEndAssets.

Code location: contracts/libraries/SmartPoolManager.sol

```
function exitPoolHandle(
    uint _endEtfAmount,
    uint _endFundAmount,
    uint _beginEtfAmount,
    uint _beginFundAmount,
    uint poolAmountIn,
    uint totalEnd
)
    external
    pure
    returns (
        uint endEtfAmount,
        uint endFundAmount,
```

```
            uint profitRate
        )
    {
        ...
    }
```

Solution

It is recommended to remove redundant code.

Status

Fixed

[N8] [Critical] Token list conflict in recordTokenInfo

Category: Design Logic Audit

Content

When the user performs createPool and joinPool, if the Pool is a closed ETF and the current time is within the collection period, the KOL invitation amount will be recorded through UserVault's recordTokenInfo interface. But unfortunately, the tokens in the Pool can be Bind/unBind at any time, which will cause the list of tokens supported by the Pool to change.

If the recordTokenInfo operation is performed when the Pool token list changes, the amount recorded by variables such as poolInviteTotal, kolTotalAmountList, and kolUserInfo may be disturbed.

Code location: contracts/deploy/UserVault.sol

```
    function recordTokenInfo(
        address kol,
        address user,
        address[] calldata poolTokens,
        uint[] calldata tokensAmount
    ) external {
        ...
        poolInviteTotal[pool] = poolInviteTotal[pool].add(tokensAmount[0]);
        uint[] memory totalAmounts = new uint[](len);
        for (uint i; i < len; i++) {
            bool kolHasInvitations = kolTotalAmountList[pool][newKol].length == 0;
            kolHasInvitations
                ? totalAmounts[i] = tokensAmount[i]
                : totalAmounts[i] = tokensAmount[i].add(kolTotalAmountList[pool]
[newKol][i]);
        }
```

```
        kolTotalAmountList[pool][newKol] = totalAmounts;
        //kol user info record
        KolUserInfo[] storage userInfoArray = kolUserInfo[pool][newKol];
        uint index = userKolBind.index;
        if (index == 0) {
            KolUserInfo memory userInfo;
            userInfo.userAdr = user;
            userInfo.userAmount = tokensAmount;
            userInfoArray.push(userInfo);
            userKolBind.index = userInfoArray.length;
        } else {
            KolUserInfo storage userInfo = kolUserInfo[pool][newKol][index - 1];
            for (uint a; a < userInfo.userAmount.length; a++) {
                userInfo.userAmount[a] = userInfo.userAmount[a].add(tokensAmount[a]);
            }
        }
    }
```

Solution

It is recommended that the list of tokens supported by the closed ETF is not allowed to be changed during the collection cycle.

Status

Fixed

[N9] [Low] couldManagerClaim not checked when managerClaim

Category: Design Logic Audit

Content

In the UserVault contract, the Manager role can claim management fees through the managerClaim function, but it does not check whether the couldManagerClaim parameter is true.

Code location: contracts/deploy/UserVault.sol

```
    function managerClaim(address pool) external {
        // try  {} catch {}
        if (_isClosePool(pool)) {
            bool isManager = IDesynOwnable(pool).adminList(msg.sender) ||
IDesynOwnable(pool).getController() == msg.sender;
            bool isCollectSuccee = ICRPPool(pool).isCompletedCollect();
            require(isCollectSuccee, "ERR_NOT_COMPLETED_COLLECT");
            require(isManager, "ERR_NOT_MANAGER");
            (address[] memory tokens, uint[] memory amounts) =
```

```
this.getUnManagerReward(pool);
            poolsStatus[pool].couldManagerClaim = false;

            ...
        }
    }
```

Solution

It is recommended to check couldManagerClaim when doing managerClaim operations.

Status

Fixed

[N10] [Suggestion] Some redundant invoke functions

Category: Others

Content

In the Invoke library, the strictInvokeTransfer, invokeUnwrapWETH, invokeWrapWETH, and invokeMint functions are not used.

Code location: modules/contracts/rebalance/Invoke.sol

```
  function strictInvokeTransfer(
    IETF _etf,
    address _token,
    address _to,
    uint256 _quantity
  ) internal {
    ...
  }

  function invokeUnwrapWETH(IETF _etf, address _weth, uint256 _quantity) internal {
    ...
  }

  function invokeWrapWETH(IETF _etf, address _weth, uint256 _quantity) internal {
    ...
  }

  function invokeMint(IETF _etf, address _token, address _receiver, uint256 amount)
internal {
    ...
  }
```

Solution

Unless this is a function reserved for future features, it is recommended to remove these unused functions.

Status

Acknowledged; After communicating with the project team, the project team indicated that these interfaces are reserved.

[N11] [Critical] Risk of ETF falsification

Category: Design Logic Audit

Content

In the RebalanceAdapter contract, the Manager role can adjust the position of the pool through the rebalance function. But the address of the ETF is obtained from the rebalanceInfo passed in by the user. If a malicious user passes in a fake ETF, the check in the onlyManager decorator will be bypassed, and the _verifyWhiteToken check of token1 and the isCompletedCollect and collectEndTime checks will be useless. Malicious users can steal bPool funds through _makeSwap and bind malicious tokens.

The approve function also has this risk, but it doesn't break the protocol too much

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
modifier onlyManager(address _etf) {
  require(
    IETF(_etf).adminList(msg.sender) || msg.sender == IETF(_etf).getController(),
    'onlyAdmin'
  );
  _;
}

function approve(
  IETF etf,
  address token,
  address spender,
  uint256 amount
) external override onlyManager(address(etf)) {
  etf.invokeApprove(token, spender, amount);

  emit TokenApproved(address(etf), token, spender, amount);
}
```

```
  function rebalance(
    IRebalanceAdapter.RebalanceInfo calldata rebalanceInfo
  ) external override onlyManager(rebalanceInfo.etf) {
    IETF etf = IETF(rebalanceInfo.etf);
    IBpool bPool = IBpool(etf.bPool());

    etf._verifyWhiteToken(rebalanceInfo.token1);

    require(bPool.isBound(rebalanceInfo.token0), 'TOKEN_NOT_BOUND');

    (, uint256 collectEndTime, , uint256 closureEndTime, , , , , , , ) =
etf.etfStatus();
    if (etf.etype() == 1) {
      require(etf.isCompletedCollect(), 'COLLECTION_FAILED');
      require(
        block.timestamp > collectEndTime && block.timestamp < closureEndTime,
        'NOT_REBALANCE_PERIOD'
      );
    }

    ...
  }
```

Solution

Check whether the ETF passed in by the user is the correct CRP.

Status

Fixed

[N12] [Medium] Risk of the Manager role potentially disrupting the protocol

Category: Design Logic Audit

Content

In the RebalanceAdapter contract, the Manager role can adjust the position of the ETF through the rebalance function. It will use the `_makeSwap` function to call Uniswap, 1inch and other DEXs for token swaps to adjust token positions. But unfortunately, the swap path of the token is not checked during the token swap process, which will cause the Manager role to pass in a carefully constructed malicious swap path to steal the middle token0 of the ETF.

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
  function _makeSwap(
    IRebalanceAdapter.RebalanceInfo calldata rebalanceInfo,
    address bPool
  ) internal returns (uint256 postSwap) {
    ...
    if (rebalanceInfo.swapType == IRebalanceAdapter.SwapType.UNISWAPV3) {
      (uint256 minReturn, uint256[] memory pools) = abi.decode(
        rebalanceInfo.data,
        (uint256, uint256[])
      );

      bytes memory swapData = abi.encodeWithSignature(
        'uniswapV3Swap(uint256,uint256,uint256[])',
        rebalanceInfo.quantity,
        minReturn,
        pools
      );

      IETF(rebalanceInfo.etf).execute(rebalanceInfo.aggregator, 0, swapData, true);
    } else if (rebalanceInfo.swapType == IRebalanceAdapter.SwapType.UNISWAPV2) {
      (uint256 minReturn, address[] memory paths) = abi.decode(
        rebalanceInfo.data,
        (uint256, address[])
      );

      bytes memory swapData = abi.encodeWithSignature(
        'swapExactTokensForTokens(uint256,uint256,address[],address,uint256)',
        rebalanceInfo.quantity,
        minReturn,
        paths,
        bPool,
        block.timestamp.add(1800)
      );

      IETF(rebalanceInfo.etf).execute(rebalanceInfo.aggregator, 0, swapData, true);
    } else {
      IETF(rebalanceInfo.etf).execute(rebalanceInfo.aggregator, 0, rebalanceInfo.data,
true);

      _validateData(rebalanceInfo.quantity, rebalanceInfo.aggregator,
rebalanceInfo.data, bPool);
    }

    postSwap = IERC20(rebalanceInfo.token1).balanceOf(bPool).sub(preSwap);
  }

  function _validateData(
    uint256 quantity,
```

```
    address aggregator,
    bytes calldata data,
    address expectedReceiver
  ) internal {
    ... //SlowMist// The code is omitted, but it should check whether the pools or the final swaped token is token1
  }
```

Solution

It is recommended to check the parsed `pools` and `paths` paths to ensure that the pools in the swap path are in the whitelist. Or check whether the token that is finally swapped out is token1.

Status

Fixed

[N13] [Low] Double slippage check

Category: Design Logic Audit

Content

In the RebalanceAdapter contract, the Manager role can adjust the position of the ETF through the rebalance function. It will use the `_makeSwap` function to call Uniswap, 1inch and other DEXs for token swaps to adjust token positions. During the swap process, the slippage will be checked by passing minReturn to the external DEXs, but the implementation of the slippage check of the external DEXs is uncontrollable. If the slippage protection of the external DEXs fails, it will affect the funds of users in the protocol.

(1inch users encountered invalid slippage check before)

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
function _makeSwap(
  IRebalanceAdapter.RebalanceInfo calldata rebalanceInfo,
  address bPool
) internal returns (uint256 postSwap) {
  ...
  postSwap = IERC20(rebalanceInfo.token1).balanceOf(bPool).sub(preSwap);
}
```

Solution

It is recommended to check whether postSwap is greater than or equal to minReturn after the swap is

completed.

Status

Acknowledged; After communicating with the project team, the project team indicated that the slippage check will rely on external DEXs to simplify the code.

[N14] [Suggestion] Does not follow the Checks-Effects-Interactions specification

Category: Others

Content

In the RebalanceAdapter contract, the `_makeSwap` function is used for token swap. When the swap type is not UNISWAPV3 and UNISWAPV2, the parameters passed in by the user will be checked through `_validateData`. However, the execute operation is performed first, and the `_validateData` operation is performed after the token exchange is completed. This is not in line with the follow the `Checks-Effects-Interactions` principle.

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
  function _makeSwap(
    IRebalanceAdapter.RebalanceInfo calldata rebalanceInfo,
    address bPool
  ) internal returns (uint256 postSwap) {
    ...
    } else {
      IETF(rebalanceInfo.etf).execute(rebalanceInfo.aggregator, 0, rebalanceInfo.data,
true);

      _validateData(rebalanceInfo.quantity, rebalanceInfo.aggregator,
rebalanceInfo.data, bPool);
    }

    postSwap = IERC20(rebalanceInfo.token1).balanceOf(bPool).sub(preSwap);
  }
```

Solution

In the current business scenario, this does not pose a security risk, but we still strongly recommend following the `Checks-Effects-Interactions` principle, first perform the `_validateData` operation, and then execute the `execute` operation.

Status

Fixed

[N15] [Suggestion] Perform strict parameter checking

Category: Others

Content

In the RebalanceAdapter contract, the `_validateData` function is used to check the exchange parameters, but it only parses the recipient and amountIn for checking, but does not check whether other key parameters such as srcToken, dstToken, clipperExchange, makerAsset, takerAsset are in line with expectations.

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
  function _validateData(
    uint256 quantity,
    address aggregator,
    bytes calldata data,
    address expectedReceiver
  ) internal {
    ... //SlowMist// A lot of code display is omitted, and it is strongly recommended to fully parse and check the key parameters in data.
  }
```

Solution

We strongly recommend checking the necessary parameters to ensure that the operations of the Manager role are in line with expectations, so as to increase the trust of the community and reduce the cost of trust.

Status

Acknowledged; After communicating with the project team, the project team stated that it will only check key parameters in this protocol to keep the code concise.

[N16] [Suggestion] Redundant aggregator parameter

Category: Others

Content

In the RebalanceAdapter contract, the `_validateData` function is used to check the conversion parameters, but the `aggregator` parameter it receives is not used.

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
function _validateData(
  uint256 quantity,
  address aggregator,
  bytes calldata data,
  address expectedReceiver
) internal {
  ...
}
```

Solution

If it is not the expected design, it is recommended to remove the aggregator parameter.

Status

Fixed

[N17] [Low] Incorrect `_validateData` function visibility

Category: Gas Optimization Audit

Content

In the RebalanceAdapter contract, the `_validateData` function is used to check the exchange parameters and does not involve any storage changes. But the visibility of this function is not view or pure.

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
function _validateData(
  uint256 quantity,
  address aggregator,
  bytes calldata data,
  address expectedReceiver
) internal {
  ...
}
```

Solution

It is recommended to change the visibility of the `_validateData` function to view or pure.

Status

Fixed

[N18] [Medium] Incorrect approval operation

Category: Design Logic Audit

Content

In the RebalanceAdapter contract, the Manager role can adjust the position of the ETF through the rebalance function. If token1 has not been bound in bPool, it will be approved first. However, the subsidy of this contract was wrongly approved to bPool, which will cause bPool to be unable to transfer tokens from CRP in the future.

Code location: modules/contracts/rebalance/RebalanceAdapter.sol

```
function rebalance(
  IRebalanceAdapter.RebalanceInfo calldata rebalanceInfo
) external override onlyManager(rebalanceInfo.etf) {
  ...

  if (!bPool.isBound(rebalanceInfo.token1)) {
    IERC20(rebalanceInfo.token1).safeApprove(address(bPool), 0);
    IERC20(rebalanceInfo.token1).safeApprove(address(bPool), uint256(-1));
  }

  ...
}
```

Solution

It is recommended to approve the CRP allowance to bPool through the invokeApprove function, please note that the parameter isUnderlying should be false.

Status

Fixed

[N19] [High] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

There is an execute function in the CRP and bPool contracts so that the Manager can manage the ETF. In bFactory, the Blabs role can arbitrarily register modules to gain control over CRP. The registered modules can use the execute function in CRP to call the execute function in bPool to perform any operations. This would pose a huge risk to users' funds.

And the Manager can approve the tokens in the bPool through the approve function of the RebalanceAdapter contract, which will also bring huge risks to the user's funds.

The above problems lead to the risk of excessive permissions of the Blabs role and the Manager role.

Code location:

contracts/base/LiquidityPool.sol

```
    function execute(
        address _target,
        uint _value,
        bytes calldata _data
    ) external _logs_ _lock_ returns (bytes memory _returnValue) {
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(!_finalized, "ERR_IS_FINALIZED");

        _returnValue = _target.functionCallWithValue(_data, _value);

        return _returnValue;
    }
```

contracts/base/ConfigurableRightsPool.sol

```
    function execute(
        address _target,
        uint _value,
        bytes calldata _data,
        bool isUnderlying
    ) external logs lock needsBPool returns (bytes memory _returnValue) {
        require(bFactory.getModuleStatus(address(this), msg.sender), 'MODULE IS NOT
REGISTER');
        if (isUnderlying) {
        _returnValue = bPool.execute(_target, _value, _data);
        } else {
        _returnValue = _target.functionCallWithValue(_data, _value);
        }
    }
```

contracts/deploy/Factory.sol

```
    function setBLabs(address b) external onlyBlabs {
        require(b != address(0), "ERR_ZERO_ADDRESS");
        emit LOG_BLABS(msg.sender, b);
```

```
        _blabs = b;
    }

    function setSystemModule(address module, bool state) external onlyBlabs {
        require(module != address(0), "ZERO ADDRESS");

        _isSystemModule[module] = state;

        emit SYSTEM_MODULE_CHANGED(module, state);
    }

    function registerModule(address etf, address module) external onlyBlabs {
        require(etf != address(0), "ZERO ETF ADDRESS");
        require(module != address(0), "ZERO ADDRESS");

        _isModuleRegistered[etf][module] = true;

        emit MODULE_STATUS_CHANGE(etf, module, true);
    }
```

modules/contracts/rebalance/RebalanceAdapter.sol

```
function approve(
  IETF etf,
  address token,
  address spender,
  uint256 amount
) external override onlyManager(address(etf)) {
  etf.invokeApprove(token, spender, amount);

  emit TokenApproved(address(etf), token, spender, amount);
}
```

Solution

It is recommended that in the early stages of the project, both the Blabs role and the Manager role should use multi-signatures to avoid single-point risks. After the project is running stably, the addition of the Blabs role and the Manager role should be handed over to community governance for management, and strict identity authentication should be performed when adding roles.

Status

Acknowledged; After communicating with the project team, the project team stated that they would follow the

plan suggested in the proposal and that the early stages of the protocol would be controlled by multi-sig. Once the project is stable, the protocol will be handed over to the community for governance.

[N20] [Medium] Enforce strict permission controls

Category: Design Logic Audit

Content

In the LiquidityPool contract, any user can call the joinPool, exitPool, and gulp functions to add/remove liquidity/record token balances, which will make it impossible to charge various fees to users in CRP.

Code location: contracts/base/LiquidityPool.sol

```
    function gulp(address token) external _logs_ _lock_ {
        ...
    }

    function joinPool(uint poolAmountOut, uint[] calldata maxAmountsIn) external
_logs_ _lock_ {
        ...
    }

    function exitPool(uint poolAmountIn, uint[] calldata minAmountsOut) external
_logs_ _lock_ {
        ...
    }
```

Solution

It is recommended to restrict the joinPool and exitPool functions in the LiquidityPool contract from being called by CRP. gulp functions are called by admin.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002303220002	SlowMist Security Team	2023.03.16 - 2023.03.22	Medium Risk

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 4 critical risks, 1 high risk, 3 medium risks, 4 low risks, and 8 suggestions. All the findings were fixed. The code was not deployed to the mainnet. Since the ownership of the protocol has not yet been handed over to community governance, the protocol still has the risk of excessive owner authority.

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