Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.11.10, the SlowMist security team received the team's security audit application for DeSyn Iterative Audit, 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

Audit Version:

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

commit: 3878d72c7c2092c7866902f87c0449e24ec9d10f

Fixed Version:

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

commit: 868170d818094be81d05d9d0886c3deb4f8c3ecc

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Repeated initialization handle defect	Design Logic Audit	Medium	Fixed
N2	Redundant logic in whitelist addition	Design Logic Audit	Suggestion	Fixed
N3	The rebalance operation is missing tokenA check	Design Logic Audit	Suggestion	Fixed
N4	ERC20-conforming token verification flaws	Design Logic Audit	Low	Acknowledged
N5	Redundant fee calculation for exit pool	Others	Suggestion	Fixed
N6	Token List Inconsistency Issue	Design Logic Audit	High	Fixed
N7	Risk of performance fee being covered	Design Logic Audit	High	Fixed
N8	managerClaim token list inconsistency issue	Design Logic Audit	High	Fixed
N9	KOL gets rewards repeatedly	Design Logic Audit	Suggestion	Acknowledged
N10	Transfer Compatibility Issues	Design Logic Audit	Medium	Acknowledged
N11	Front Running set pool parameters issue	Design Logic Audit	Low	Fixed
N12	Oracle price period issue	Design Logic Audit	Medium	Fixed
N13	kols_list duplicate push issue	Design Logic Audit	Critical	Fixed
N14	Oracle price decimal issue	Design Logic Audit	High	Fixed
N15	Token decimal compatibility issue	Design Logic Audit	Low	Fixed

NO	Title	Category	Level	Status
N16	Token symbol conflict issue	Design Logic Audit	Medium	Acknowledged
N17	function type issue	Others	Suggestion	Fixed
N18	Wrong event log	Others	Low	Fixed
N19	closurePeriod setting issue	Design Logic Audit	Suggestion	Fixed
N20	exitPoolHandleB logic defect issue	Design Logic Audit	Medium	Fixed
N21	Missing event record	Others	Suggestion	Fixed
N22	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

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:

BBronze			
Function Name	Visibility	Mutability	Modifiers
getColor	External	-	-

ConfigurableRightsPool			
Function Name	Visibility	Mutability	Modifiers

ConfigurableRightsPool			
<Constructor>	Public	Can Modify State	PCToken
init	Public	Can Modify State	-
setCap	External	Can Modify State	logs lock needsBPool onlyOwner
execute	External	Can Modify State	logs lock needsBPool
claimManagerFee	External	Can Modify State	logs lock onlyManager needsBPool
createPool	External	Can Modify State	onlyOwner logs lock
createPool	External	Can Modify State	onlyOwner logs lock
rebalance	External	Can Modify State	logs lock onlyAdmin needsBPool
joinPool	External	Can Modify State	logs lock needsBPool
_closeEtfCollectCompletedToClaimeIssueFee	Internal	Can Modify State	-
exitPool	External	Can Modify State	logs lock needsBPool
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	-

ConfigurableRightsPool			
pushPoolShareFromLib	Public	Can Modify State	-
pullPoolShareFromLib	Public	Can Modify State	-
burnPoolShareFromLib	Public	Can Modify State	-
createPoolInternal	Internal	Can Modify State	-
addTokenToWhitelist	External	Can Modify State	onlyOwner
_verifyWhiteToken	Internal	-	-
_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	-

ExchangeProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setRegistry	External	Can Modify State	onlyOwner
batchSwapExactIn	Public	Payable	-
batchSwapExactOut	Public	Payable	-

ExchangeProxy			
multihopBatchSwapExactIn	Public	Payable	-
multihopBatchSwapExactOut	Public	Payable	-
smartSwapExactIn	Public	Payable	-
smartSwapExactOut	Public	Payable	-
viewSplitExactIn	Public	-	-
viewSplitExactOut	Public	-	-
getPoolData	Internal	-	-
calcEffectiveLiquidity	Internal	-	-
calcTotalOutExactIn	Internal	-	-
calcTotalOutExactOut	Internal	-	-
transferFromAll	Internal	Can Modify State	-
getBalance	Internal	-	-
transferAll	Internal	Can Modify State	-
isETH	Internal	-	-
<Fallback>	External	Payable	-

LiquidityPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
isPublicSwap	External	-	-
isFinalized	External	-	-
isBound	External	-	-
getNumTokens	External	-	-

LiquidityPool			
getCurrentTokens	External	-	_viewlock_
getFinalTokens	External	-	_viewlock_
getDenormalizedWeight	External	-	_viewlock_
getTotalDenormalizedWeight	External	-	_viewlock_
getNormalizedWeight	External	Can Modify State	_viewlock_
getBalance	Public	-	_viewlock_
getSwapFee	External	-	_viewlock_
getController	External	-	_viewlock_
setSwapFee	External	Can Modify State	_logs_ _lock_
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_
rebindSmart	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_
gulp	External	Can Modify State	_logs_ _lock_
joinPool	External	Can Modify State	_logs_ _lock_
exitPool	External	Can Modify State	_logs_ _lock_
_safeApprove	Internal	Can Modify State	-

LiquidityPool			
_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	-

LpTokenBase			
Function Name	Visibility	Mutability	Modifiers
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_move	Internal	Can Modify State	-
_push	Internal	Can Modify State	-
_pull	Internal	Can Modify State	-

LpToken			
Function Name	Visibility	Mutability	Modifiers
name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
allowance	External	-	-
balanceOf	External	-	-

LpToken			
totalSupply	Public	-	-
approve	External	Can Modify State	-
increaseApproval	External	Can Modify State	-
decreaseApproval	External	Can Modify State	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-

Math			
Function Name	Visibility	Mutability	Modifiers
calcSpotPrice	Public	-	-
calcOutGivenIn	Public	-	-
calcInGivenOut	Public	-	-
calcPoolOutGivenSingleIn	Public	-	-
calcSingleInGivenPoolOut	Public	-	-
calcSingleOutGivenPoolIn	Public	-	-
calcPoolInGivenSingleOut	Public	-	-

Multicall			
Function Name	Visibility	Mutability	Modifiers
aggregate	Public	Can Modify State	-
getEthBalance	Public	-	-
getBlockHash	Public	-	-
getLastBlockHash	Public	-	-

Multicall			
getCurrentBlockTimestamp	Public	-	-
getCurrentBlockDifficulty	Public	-	-
getCurrentBlockGasLimit	Public	-	-
getCurrentBlockCoinbase	Public	-	-

Num			
Function Name	Visibility	Mutability	Modifiers
btoi	Internal	-	-
bfloor	Internal	-	-
badd	Internal	-	-
bsub	Internal	-	-
bsubSign	Internal	-	-
bmul	Internal	-	-
bdiv	Internal	-	-
bpowi	Internal	-	-
bpow	Internal	-	-
bpowApprox	Internal	-	-

PCToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allowance	External	-	-
balanceOf	External	-	-

PCToken			
approve	External	Can Modify State	-
increaseApproval	External	Can Modify State	-
decreaseApproval	External	Can Modify State	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
totalSupply	External	-	-
name	External	-	-
symbol	External	-	-
decimals	External	-	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_move	Internal	Can Modify State	-
_push	Internal	Can Modify State	-
_pull	Internal	Can Modify State	-

sorMultiCall			
Function Name	Visibility	Mutability	Modifiers
getUint	Internal	-	-
getPoolInfo	External	-	-

WhiteToken			
Function Name	Visibility	Mutability	Modifiers
_queryIsTokenWhitelisted	Internal	-	-

WhiteToken			
_isTokenWhitelistedForVerify	Internal	-	-
_addTokenToWhitelist	Internal	Can Modify State	-
_removeTokenFromWhitelist	Internal	Can Modify State	-
_countListener	Private	Can Modify State	-
_initWhiteTokenState	Internal	-	-

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

DesynOwnable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Internal	Can Modify State	-
initHandle	External	Can Modify State	onlyOwner
setController	External	Can Modify State	onlyOwner
setAddAdminList	External	Can Modify State	onlyOwner
removeOwner	External	Can Modify State	onlyOwner
getOwners	Public	-	-
getOwnerPercentage	Public	-	-
getController	External	-	-

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

Actions			
Function Name	Visibility	Mutability	Modifiers
create	External	Can Modify State	-
createSmartPool	External	Can Modify State	-
joinPool	External	Can Modify State	-
joinSmartPool	External	Can Modify State	-
setPublicSwap	External	Can Modify State	-
setSwapFee	External	Can Modify State	-
setController	External	Can Modify State	-

Actions			
setTokens	External	Can Modify State	-
finalize	External	Can Modify State	-
rebalance	External	Can Modify State	-
setCap	External	Can Modify State	-
whitelistLiquidityProvider	External	Can Modify State	-
removeWhitelistedLiquidityProvider	External	Can Modify State	-
_safeApprove	Internal	Can Modify State	-
_join	Internal	Can Modify State	-

CRPFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
createPool	Internal	Can Modify State	-
newCrp	External	Can Modify State	-
setUserVault	External	Can Modify State	onlyBlabs
setByteCodes	External	Can Modify State	onlyBlabs
isCrp	External	-	-

DesynChainlinkOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getPrice	Public	Can Modify State	-
getAllPrice	External	Can Modify State	-

DesynChainlinkOracle			
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

DSProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Fallback>	External	Payable	-
execute	Public	Payable	-
execute	Public	Payable	auth note
setCache	Public	Can Modify State	auth note

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

DSProxyCache			
Function Name	Visibility	Mutability	Modifiers
read	Public	-	-
write	Public	Can Modify State	-

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

Factory			
Function Name	Visibility	Mutability	Modifiers
addTokenToWhitelist	External	Can Modify State	onlyBlabs
removeTokenFromWhitelist	External	Can Modify State	onlyBlabs
isTokenWhitelistedForVerify	External	-	-
isTokenWhitelistedForVerify	External	-	-
isLiquidityPool	External	-	-
createPool	Internal	Can Modify State	-
newLiquidityPool	External	Can Modify State	-
<Constructor>	Public	Can Modify State	-
getBLabs	External	-	-
setBLabs	External	Can Modify State	onlyBlabs
getSwapRouter	External	-	-

Factory			
getModuleStatus	External	-	-
getOracleAddress	External	-	-
setSwapRouter	External	Can Modify State	onlyBlabs
registerModule	External	Can Modify State	-
removeModule	External	Can Modify State	-
setOracle	External	Can Modify State	-
collect	External	Can Modify State	onlyBlabs
getVault	External	-	-
setVault	External	Can Modify State	onlyBlabs
getUserVault	External	-	-
setUserVault	External	Can Modify State	onlyBlabs
getManagerOwner	External	-	-
setManagerOwner	External	Can Modify State	onlyBlabs

UserVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
depositToken	Public	Can Modify State	onlyVault
claimKolReward	Public	Can Modify State	-
claimManagersReward	Public	Can Modify State	-
isClosePool	Public	-	-
getUnManagerReward	Public	-	-

UserVault			
getManagerReward	Public	-	-
getAllManagersReward	Public	-	-
getPoolAllFee	Public	-	-
getAllKolReward	Public	-	-
kolUnClaimAmount	Public	-	-
kolClaimTotal	Public	-	-
computeReward	Public	-	-
levelJudge	Internal	-	-
setResult	Internal	Can Modify State	-
createTokenParams	Internal	-	-
communaldepositToken	Internal	Can Modify State	-
recordTokenInfo	Public	Can Modify State	-
setPoolParams	Public	Can Modify State	-
getKolsAdr	Public	-	-
getPoolUserList	Public	-	-
getPoolUserKolAdr	Public	-	-
getPoolKolUserInfo	Public	-	-
getPoolKolTotalAmounts	Public	-	-
poolManagerTokenList	Public	-	-
poolManagerTokenAmount	Public	-	-
poolIssueTokenList	Public	-	-
poolRedeemTokenList	Public	-	-

UserVault			
poolIssueTokenAmount	Public	-	-
poolRedeemTokenAmount	Public	-	-
poolPerfermanceTokenList	Public	-	-
poolPerfermanceTokenAmount	Public	-	-
getManagerClaimBool	Public	-	-
setBlackList	Public	Can Modify State	onlyOwner
setCrpFactory	Public	Can Modify State	onlyOwner
adminClaimToken	Public	Can Modify State	onlyOwner
getBNB	Public	Payable	onlyOwner
setVaultAdr	Public	Can Modify State	onlyOwner

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

Vault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-

Vault			
depositManagerToken	Public	Can Modify State	-
depositIssueRedeemPToken	Public	Can Modify State	-
communaldepositToken	Internal	Can Modify State	-
communaldepositTokenNew	Internal	Can Modify State	-
poolManagerTokenList	Public	-	-
poolManagerTokenAmount	Public	-	-
poolIssueTokenList	Public	-	-
poolRedeemTokenList	Public	-	-
poolIssueTokenAmount	Public	-	-
poolRedeemTokenAmount	Public	-	-
poolPerfermanceTokenList	Public	-	-
poolPerfermanceTokenAmount	Public	-	-
getManagerClaimBool	Public	-	-
setBlackList	Public	Can Modify State	onlyOwner
setUserVaultAdr	Public	Can Modify State	onlyOwner
setCrpFactory	Public	Can Modify State	onlyOwner
adminClaimToken	Public	Can Modify State	onlyOwner
getBNB	Public	Payable	onlyOwner
setManagerRatio	Public	Can Modify State	onlyOwner
setIssueRatio	Public	Can Modify State	onlyOwner
setRedeemRatio	Public	Can Modify State	onlyOwner
setPerfermanceRatio	Public	Can Modify State	onlyOwner

Vault			
managerClaim	Public	Can Modify State	-
recordUserVault	Internal	Can Modify State	-
transferHandle	Internal	Can Modify State	-
computeBalance	Internal	-	-
isClosePool	Public	-	-
clearPool	Internal	Can Modify State	-
managerClaimRecordList	Public	-	-
managerClaimList	Public	-	-

4.3 Vulnerability Summary

[N1] [Medium] Repeated initialization handle defect

Category: Design Logic Audit

Content

In the DesynOwnable contract, the owner can set allOwnerPercentage, adminList, owners and ownerPercentage parameters through the initHandle function. But the initHandle function can be called repeatedly. When the owner role is called multiple times, the allOwnerPercentage parameter will continue to increase.

Code location: contracts/utils/DesynOwnable.sol

```
    function initHandle(address[] memory _owners, uint[] memory _ownerPercentage)
external onlyOwner {
        require(_owners.length == _ownerPercentage.length, "ownerP");
        for (uint i = 0; i < _owners.length; i++) {
            allOwnerPercentage += _ownerPercentage[i];
            adminList[_owners[i]] = true;
        }
        owners = _owners;
        ownerPercentage = _ownerPercentage;
    }
```

Solution

It is recommended to forbid the owner to repeatedly call the initHandle function, because allOwnerPercentage, adminList, owners and ownerPercentage parameters can be set in other functions of this contract.

Status

Fixed

[N2] [Suggestion] Redundant logic in whitelist addition

Category: Design Logic Audit

Content

In the WhiteToken contract, the `_addTokenToWhitelist` function is used to add whitelist tokens. When the add operation is performed for the first time, the `_hasInit` parameter will be set to true, but the `_countListener` function is called to set the `_hasInit` parameter. The logic for duplicate settings is redundant.

Code location: contracts/base/WhiteToken.sol

```
    function _addTokenToWhitelist(uint sort, address token) internal {
        require(token != address(0), "ERR_INVALID_TOKEN_ADDRESS");
        require(_queryIsTokenWhitelisted(token) == false,
"ERR_HAS_BEEN_ADDED_WHITE");

        _tokenWhitelistedInfo[sort][token] = true;
        _isTokenWhitelisted[token] = true;
        if (_hasInit == false) {
            _hasInit = true;
        }
        _whiteTokenCount++;
        _countListener();

        emit LOG_WHITELIST(address(this), sort, msg.sender, token);
    }
```

Solution

It is recommended to remove the call to `_countListener` in the `_addTokenToWhitelist` function.

Status

Fixed

[N3] [Suggestion] The rebalance operation is missing tokenA check

Category: Design Logic Audit

Content

In CRP, the rebalance function is used to adjust the weight of the pool. When performing the rebindSmart operation, it is necessary to ensure that tokenA has been bound, but the legality of tokenA is not checked in the contract.

Code location: contracts/base/ConfigurableRightsPool.sol

```
    function rebalance(
        address tokenA,
        address tokenB,
        uint deltaWeight,
        uint minAmountOut
    ) external virtual logs lock onlyAdmin needsBPool {
        ...
    }
```

Solution

Although rebalance will not succeed when using tokenA that does not meet the requirements, it is still recommended to check tokenA to save gas.

Status

Fixed

[N4] [Low] ERC20-conforming token verification flaws

Category: Design Logic Audit

Content

In the SmartPoolManager library, the verifyTokenCompliance function user checks whether a token complies with the EIP20 standard. verifyTokenCompliance will call the transfer function of the specified token through the verifyTokenComplianceInternal function to test the amount of transfer 0. And check if the transfer was successful. However, the transfer function of some tokens checks that the transfer amount must be greater than 0, and all of its interfaces conform to the EIP20 standard. Therefore, this kind of inspection method is still not compatible with some tokens that meet the EIP20 standard.

Code location: contracts/libraries/SmartPoolManager.sol

```
function verifyTokenCompliance(address token) external {
    verifyTokenComplianceInternal(token);
}

function verifyTokenCompliance(address[] calldata tokens) external {
    for (uint i = 0; i < tokens.length; i++) {
        verifyTokenComplianceInternal(tokens[i]);
    }
}

function verifyTokenComplianceInternal(address token) internal {
    bool returnValue = IERC20(token).transfer(msg.sender, 0);
    require(returnValue, "ERR_NONCONFORMING_TOKEN");
}
```

Solution

If you need to check by transfer, it is recommended to transfer more than 0 tokens between the caller and the contract.

Status

Acknowledged; After communicating with the project team, the project team stated that they also have platform token whitelist to ensure that all the tokens added to our protocol are good.

[N5] [Suggestion] Redundant fee calculation for exit pool

Category: Others

Content

In the SmartPoolManager library, the exitPool function is used to calculate the actual number of each token that can be received when exiting the pool. When the exit fee is 0, the exitPool function still subtracts 0 from poolAmountIn to calculate pAiAfterExitFee, which is a redundant calculation.

Code location: contracts/libraries/SmartPoolManager.sol

```
function exitPool(
    IConfigurableRightsPool self,
    IBPool bPool,
    uint poolAmountIn,
    uint[] calldata minAmountsOut
) external view returns (uint pAiAfterExitFee, uint[] memory actualAmountsOut) {
    ...
```

```
        // Calculate exit fee and the final amount in
        pAiAfterExitFee = DesynSafeMath.bsub(poolAmountIn, 0);

        ...
    }
```

Solution

If the exit fee is 0 is the expected design, it is recommended to remove the calculation of pAiAfterExitFee.

Status

Fixed

[N6] [High] Token List Inconsistency Issue

Category: Design Logic Audit

Content

In the vault contract, the communaldepositToken function is used to update the stored tokenList and tokenAmount. But if the previously stored list is inconsistent with the current list that needs to be updated (the tokens in the pool may be replaced by rebalance). This will cause the communaldepositToken function to only update previously stored tokens and ignore new ones.

The same is true of the communaldepositTokenNew function in Vault contracts.

The same is true of the communaldepositToken function in the UserVault contract.

Code location: contracts/deploy/Vault.sol

```
    function communaldepositToken(
        address[] calldata poolTokens,
        uint[] calldata tokensAmount,
        address poolAdr,
        address[] memory _pool_tokenList,
        uint[] memory _pool_tokenAmount
    ) internal returns (address[] memory new_pool_tokenList, uint[] memory
new_pool_tokenAmount) {
        ...

        } else {
            for (uint k = 0; k < poolTokens.length; k++) {
                if (_pool_tokenList[k] == poolTokens[k]) {
                    address t = poolTokens[k];
```

```
                    uint tokenBalance = tokensAmount[k];
                    IERC20(t).transferFrom(msg.sender, address(this), tokenBalance);
                    new_pool_tokenList[k] = poolTokens[k];
                    new_pool_tokenAmount[k] = _pool_tokenAmount[k].add(tokenBalance);
                }
            }
        }
        return (new_pool_tokenList, new_pool_tokenAmount);
    }
```

Solution

It is recommended that if the token list is inconsistent, clear the list data stored in the contract through the managerClaim function first, and then update the new list.

Status

Fixed

[N7] [High] Risk of performance fee being covered

Category: Design Logic Audit

Content

In the Vault contract, the performance fee and redeem fee can be updated through the communaldepositTokenNew function. When updating new_pool_tokenListP may be inconsistent with the previously stored token list, which will cause the token list to be directly overwritten.

Code location: contracts/deploy/Vault.sol

```
    function communaldepositTokenNew(
        address[] calldata poolTokens,
        uint[] calldata tokensAmount,
        uint[] calldata tokensAmountIR,
        address poolAdr
    )
        internal
        returns (
            address[] memory new_pool_tokenList,
            uint[] memory new_pool_tokenAmount,
            address[] memory new_pool_tokenListP,
            uint[] memory new_pool_tokenAmountP
        )
    {
        ...
```

```
        //perfermance
        if (
            (pool_perfermance_tokenList[poolAdr].length ==
pool_perfermance_tokenAmount[poolAdr].length &&
pool_perfermance_tokenList[poolAdr].length == 0) ||
            !pool_manager_isClaim[poolAdr]
        ) {
            for (uint i = 0; i < poolTokens.length; i++) {
                new_pool_tokenListP[i] = poolTokens[i];
                new_pool_tokenAmountP[i] = tokensAmount[i].sub(tokensAmountIR[i]);
            }
        } else {
            for (uint k = 0; k < poolTokens.length; k++) {
                new_pool_tokenListP[k] = poolTokens[k];
                new_pool_tokenAmountP[k] = pool_perfermance_tokenAmount[poolAdr]
[k].add(tokensAmount[k].sub(tokensAmountIR[k]));
            }
        }

        return (new_pool_tokenList, new_pool_tokenAmount, new_pool_tokenListP,
new_pool_tokenAmountP);
    }
```

Solution

It is recommended to perform the managerClaimList operation before updating the token list to clean up the old list.

Status

Fixed

[N8] [High] managerClaim token list inconsistency issue

Category: Design Logic Audit

Content

In the Vault contract, users can perform recordUserVault operations through the managerClaim function. However, since managerTokenAmount and other fees are updated through depositManagerToken and depositIssueRedeemPToken functions respectively, pool_manager_tokenList may be inconsistent with other fee token lists. This will result in a possible mismatch between the token list and the fee amount when doing a recordUserVault operation.

Code location: contracts/deploy/Vault.sol

```
    function managerClaim(address pool) public {
        require(crpFactory.isCrp(pool), "invalid address");
        address manager_address = ICRPPool(pool).getController();
        address[] memory _pool_manager_tokenList = pool_manager_tokenList[pool].length
!= 0
            ? pool_manager_tokenList[pool]
            : (
                pool_issue_tokenList[pool].length != 0
                    ? pool_issue_tokenList[pool]
                    : (pool_redeem_tokenList[pool].length != 0 ?
pool_redeem_tokenList[pool] : pool_perfermance_tokenList[pool])
            );
        ...
        if (isClosePool(pool)) {
            recordUserVault(pool, _pool_manager_tokenList, managerTokenAmount,
issueTokenAmount, redeemTokenAmount, perfermanceTokenAmount);
        }

        ...
    }
```

Solution

It is recommended to perform the managerClaim operation before performing the depositManagerToken and depositIssueRedeemPToken operations to clear the list of each fee token.

Status

Fixed

[N9] [Suggestion] KOL gets rewards repeatedly

Category: Design Logic Audit

Content

In the UserVault contract, KOL can claim rewards through the claimKolReward function. KOL's reward calculation is related to the amount of liquidity added by users. When the isCompletedCollect status of CRP is fasle, the user does not need to pay the IssueFee when performing the joinPool operation. And when the CRP is not ClosePool, the user will theoretically only be charged RedeemFee when exitPool. On the other hand, KOLs can obtain commissions from issueFee, redeemFee, managerFee, and performanceFee. Therefore, if the KOL accumulates rewards for himself through the joinPool/exitPool operation repeatedly, the commissions he gets may be able to cover his costs.

Solution

N/A

Status

Acknowledged; After communicating with the project team, the project team stated that it is okay, cause the rewards the KOLs get will depend on the total fees they contribute.

[N10] [Medium] Transfer Compatibility Issues

Category: Design Logic Audit

Content

In the UserVault contract, when the user makes a claim fee, if the user is a contract, UserVault will first obtain the owner address of the contract, and then transfer the fee to the owner. However, if the contract does not have an owner interface, its fees will not be successfully claimed.

Code location: contracts/deploy/UserVault.sol

```
    function claimKolReward(address pool) public {
        ...
            if (address(msg.sender).isContract()) {
                IERC20(kol_token_list[pool]
[0]).transfer(IDSProxy(msg.sender).owner(), totalAmount);
            } else {
                IERC20(kol_token_list[pool][0]).transfer(msg.sender, totalAmount);
            }
          }
    }

    function claimManagersReward(address pool) public {
        ...
            if (address(msg.sender).isContract()) {
                IERC20(kol_token_list[pool]
[0]).transfer(IDSProxy(msg.sender).owner(), totalAmount);
            } else {
                IERC20(kol_token_list[pool][0]).transfer(msg.sender, totalAmount);
            }
        }
    }
```

Solution

It is recommended to send the fee directly to msg.sender.

Status

Acknowledged; After communicating with the project team, the project team stated that usually the address of KOL is automatically generated by the front end, if a KOL needs to provide the address by himself, he needs to be very careful.

[N11] [Low] Front Running set pool parameters issue

Category: Design Logic Audit

Content

In the UserVault contract, the setPoolParams function is used to set the KOL commission ratio. It can only be set once, but without permission control, all users can call the setPoolParams function to set it. This creates the risk of a conditional race, where a malicious user could front-run and preemptively set an unintended fee ratio.

Code location: contracts/deploy/UserVault.sol

```
    function setPoolParams(address pool, SmartPoolManager.KolPoolParams memory
_poolParams) public {
        require(crpFactory.isCrp(pool), "invalid address");
        require(!is_set_params[pool], "Params is seted");
        is_set_params[pool] = true;
        kol_pool_params[pool] = _poolParams;
    }
```

Solution

It is recommended to perform permission control on the setPoolParams function.

Status

Fixed

[N12] [Medium] Oracle price period issue

Category: Design Logic Audit

Content

In the agreement, the Oracle contract is used to provide Uniswap v2 pair pool price feed. However, its initial PERIOD is only 60s. If the owner has not changed the PERIOD in time, the agreement will start to run, which will lead to the risk of the oracle machine being manipulated.

Code location: contracts/deploy/Oracle.sol

```
uint public PERIOD = 60 seconds;
```

Solution

It is recommended to set the initial PERIOD to 1 hour.

Status

Fixed

[N13] [Critical] kols_list duplicate push issue

Category: Design Logic Audit

Content

In the protocol, when CRP performs the recordTokenInfo operation, it will push the KOL address passed in by the user into the kols_list. However, it does not check whether the KOL already exists. If the same KOL address is pushed, it will affect the reward calculation.

Code location: contracts/deploy/UserVault.sol

```
    function recordTokenInfo(
        address kol,
        address user,
        address[] calldata poolTokens,
        uint[] calldata tokensAmount
    ) public {
        ...
        if (user_kol_list[msg.sender][user] == address(0)) {
            user_kol_list[msg.sender][user] = kol;
            kols_list[msg.sender].push(kol);
            newKol = kol;
        }
        ...
    }
```

Solution

It is recommended to check whether the KOL already exists before pushing the KOL parameter.

Status

Fixed

[N14] [High] Oracle price decimal issue

Category: Design Logic Audit

Content

In the DesynChainlinkOracle contract, the getPrice function is used to get the token price from prices, getChainlinkPrice, getUniswapPrice. The getChainlinkPrice function will return the decimal of 1e18, and getUniswapPrice will return the decimal of the corresponding token. However, the getPrice function performs precision processing. When the decimal of the token is less than 1e18, the price will be multiplied by (10**decimalDelta). This will cause the decimal of the token price to be amplified.

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 = bsub(uint(18), uint(token.decimals()));
        // Ensure that we don't multiply the result by 0
        if (decimalDelta > 0) {
            return price.mul(10**decimalDelta);
        } else {
            return price;
        }
    }
```

Solution

It is recommended that after the decimal is processed in the getChainlinkPrice and getUniswapPrice functions, there is no need to process it in the getPrice function.

Status

Fixed

[N15] [Low] Token decimal compatibility issue

Category: Design Logic Audit

Content

In the DesynChainlinkOracle contract, the getPrice and getChainlinkPrice functions are used to obtain the token price, which will process the decimal of the price to return a decimal of 1e18. However, if the token's decimal exceeds 1e18, it will cause overflow and fail to obtain the price.

Code location: contracts/deploy/DesynChainlinkOracle.sol

```
    function getPrice(address tokenAddress) public returns (uint price) {
        ...

        uint decimalDelta = bsub(uint(18), uint(token.decimals()));
        ...
    }

    function getChainlinkPrice(AggregatorV2V3Interface feed) internal view returns
(uint) {
        // Chainlink USD-denominated feeds store answers at 8 decimals
        uint decimalDelta = bsub(uint(18), feed.decimals());
        ...
    }
```

Solution

It is recommended to be compatible with tokens whose decimal exceeds 1e18.

Status

Fixed; After communicating with the project team, the project team stated that it is okay, we also consider the token decimal to normalize the token values.

[N16] [Medium] Token symbol conflict issue

Category: Design Logic Audit

Content

In the DesynChainlinkOracle contract, the getPrice function obtains the price feed source through the getFeed function. The getFeed function is obtained through the symbol of the token, but the symbol of the token may be repeated. If different tokens in CRP have the same symbol, it will lead to disorder in price acquisition.

Code location: contracts/deploy/DesynChainlinkOracle.sol

```
    function getPrice(address tokenAddress) public returns (uint price) {
        IERC20 token = IERC20(tokenAddress);
        AggregatorV2V3Interface feed = getFeed(token.symbol());
        ...
    }

    function getFeed(string memory symbol) public view returns
(AggregatorV2V3Interface) {
        return feeds[keccak256(abi.encodePacked(symbol))];
    }
```

Solution

It is recommended to obtain the price feed source through the token address.

Status

Acknowledged; After communicating with the project team, the project team stated that because chainlink uses the symbol to reference the token price, we also use it.

[N17] [Suggestion] function type issue

Category: Others

Content

In the UserVault contract, the communaldepositToken function is used to update pool_tokenList and pool_tokenAmount, but it does not directly modify the parameters stored in the contract but assigns the value to a temporary variable. So this function type can be a view function.

Code location: contracts/deploy/UserVault.sol

```
    function communaldepositToken(
        address[] calldata poolTokens,
        uint[] calldata tokensAmount,
        address poolAdr,
        address[] memory _pool_tokenList,
        uint[] memory _pool_tokenAmount
    ) internal returns (address[] memory new_pool_tokenList, uint[] memory
new_pool_tokenAmount) {
        ...
    }
```

Solution

It is recommended to modify the function to a view function.

Status

Fixed

[N18] [Low] Wrong event log

Category: Others

Content

In CRP, the sizeChanged event will record oldSize and newSize. While the upperCap will be set to `DesynConstants.MAX_UINT` when CRP is initialized, and the oldSize of the sizeChanged event record is 0 when the upperCap is modified in the createPool function.

Code location: contracts/base/ConfigurableRightsPool.sol

```
    function createPool(
        uint initialSupply,
        uint collectPeriod,
        SmartPoolManager.Period closurePeriod,
        SmartPoolManager.PoolTokenRange memory tokenRange
    ) external virtual onlyOwner logs lock {
        ...

            etfStatus.upperCap =
initialSupply.bmul(tokenRange.bspCap).bdiv(_initialBalances[0]);
            etfStatus.floorCap =
initialSupply.bmul(tokenRange.bspFloor).bdiv(_initialBalances[0]);
            emit sizeChanged(msg.sender, "UPPER", 0, etfStatus.upperCap);
            emit sizeChanged(msg.sender, "FLOOR", 0, etfStatus.floorCap);
        ...
    }
```

Solution

It is suggested that oldSize should be the old etfStatusCap instead of 0 when recording the sizeChanged event.

Status

Fixed

[N19] [Suggestion] closurePeriod setting issue

Category: Design Logic Audit

Content

In the createPool function of the ConfigurableRightsPool contract, the closurePeriod will be set when the closePool is created. There are three types of periods: HALF, ONE and others. ONE means that the period is 365 days, but HALF is 30s.

Code location: contracts/base/ConfigurableRightsPool.sol

```
    function createPool(
        uint initialSupply,
        uint collectPeriod,
        SmartPoolManager.Period closurePeriod,
        SmartPoolManager.PoolTokenRange memory tokenRange
    ) external virtual onlyOwner logs lock {
        ...
            if (closurePeriod == SmartPoolManager.Period.HALF) {
                // TODO
                period = 30 seconds;
                // period = 1 seconds; // TEST CONFIG：for test only
            } else if (closurePeriod == SmartPoolManager.Period.ONE) {
                period = 365 days;
            } else {
                period = 730 days;
            }
            ...
        }

        createPoolInternal(initialSupply);
    }
```

Solution

If it is not the expected design, it is recommended to modify the HALF period to a more contractual time.

Status

Fixed

[N20] [Medium] exitPoolHandleB logic defect issue

Category: Design Logic Audit

Content

In the SmartPoolManager library, the exitPoolHandleB function is used to calculate the new fundAmount and

etfAmount parameters. When the CRP is closePool, its isCloseEtfCollectEndWithFailure and isCloseEtfClosureEnd will be checked. To pass this check, block.timestamp must be greater than collectEndTime or greater than closureEndTime. After that, exitPoolHandleB checks `bools && block.timestamp <= collectEndTime`. When CRP is closePool, these two checks can only pass successfully when `block.timestamp == collectEndTime`. This condition is extremely harsh.

Code location: contracts/libraris/SmartPoolManager.sol

```
    function exitPoolHandleB(
        bool bools,
        bool isCompletedCollect,
        uint closureEndTime,
        uint collectEndTime,
        uint _etfAmount,
        uint _fundAmount,
        uint poolAmountIn
    ) external view returns (uint etfAmount, uint fundAmount) {
        if (bools) {
            bool isCloseEtfCollectEndWithFailure = isCompletedCollect == false &&
block.timestamp >= collectEndTime;
            bool isCloseEtfClosureEnd = block.timestamp >= closureEndTime;
            require(isCloseEtfCollectEndWithFailure || isCloseEtfClosureEnd,
"ERR_CLOSURE_TIME_NOT_ARRIVED!");
        }
        fundAmount = _fundAmount;
        etfAmount = _etfAmount;
        if (bools && block.timestamp <= collectEndTime) {
            fundAmount =
DesynSafeMath.bmul(DesynSafeMath.bdiv(DesynSafeMath.bsub(_etfAmount, poolAmountIn),
_etfAmount), _fundAmount);
            etfAmount = DesynSafeMath.bsub(_etfAmount, poolAmountIn);
        }
    }
```

Solution

It is recommended to re-evaluate whether this is the intended design.

Status

Fixed

[N21] [Suggestion] Missing event record

Category: Others

Content

In the CRPFactory contract, the Blabs role can modify the bytecodes parameter through the setByteCodes function, but the event is not recorded.

In the UserVault contract, the owner role can modify the black_list and crpFactory parameters through the setBlackList and setCrpFactory functions respectively, but no event recording is performed.

In the Vault contract, the owner role can modify contract parameters through setBlackList, setCrpFactory, and setPerfermanceRatio, but no event recording is performed.

Code location:

contracts/deploy/CRPFactory.sol

```
  function setByteCodes(bytes memory _bytecodes) external onlyBlabs {
    bytecodes = _bytecodes;
}
```

contracts/deploy/UserVault.sol

```
function setBlackList(address pool, bool bools) public onlyOwner {
    black_list[pool] = bools;
}

function setCrpFactory(address adr) public onlyOwner {
    crpFactory = ICRPFactory(adr);
}
```

contracts/deploy/Vault.sol

```
function setBlackList(address pool, bool bools) public onlyOwner {
    black_list[pool] = bools;
}
function setCrpFactory(address adr) public onlyOwner {
    crpFactory = ICRPFactory(adr);
}
function setPerfermanceRatio(uint amount) public onlyOwner {
    perfermance_ratio = amount;
}
```

Solution

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

Status

Fixed

[N22] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the protocol, privileged roles can modify various sensitive parameters in the contract at will. And in the Vault and UserVault contracts, the owner can withdraw the funds in the contract through the adminClaimToken and getBNB functions. This will lead to the risk of excessive owner permissions.

Solution

It is inappropriate and unsafe for sensitive permissions involving user funds to be managed by EOA addresses.
In the short term, the protocol can transfer permissions to multi-signature wallets to deal with single-point risks and various parameter changes in the early stages of the protocol.
But in the long run, it is more appropriate to hand over permissions to timelock and manage them by the community.
In order to deal with emergencies, the project team should still retain the authority to suspend the protocol in an emergency.

Status

Acknowledged; After communicating with the project team, the project team stated that the ownership of the protocol will be transferred to the multisig contract for management in the early stage of the operation of the protocol, and will be transferred to community governance after the protocol runs stably.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002211250001	SlowMist Security Team	2022.11.10 - 2022.11.25	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 1 critical risk, 4 high risks, 6 medium risks, 4 low risks, and 7 suggestions. All the findings were fixed or acknowledged. 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