Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.06.28, the SlowMist security team received the DeSyn Protocol team's security audit application for DeSyn Protocol, 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-smart-contracts-v2

commit: 4c961026c441fd562eb958b36b757cfbfe63aa74

Fixed Version:

https://github.com/Meta-DesynLab/desyn-smart-contracts-v2

commit: cbac4e12e0a54d83e0b84ef9bf8eb80b475fab2f

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing event record	Others	Suggestion	Confirmed
N2	Swap path issue	Design Logic Audit	Low	Confirmed
N3	Issue with tokens outside of ETF	Design Logic Audit	Medium	Confirmed
N4	Risk of over-approval	Design Logic Audit	Low	Confirmed
N5	Insecure Token Transfer	Others	Suggestion	Confirmed
N6	Compatibility issue	Others	Suggestion	Confirmed
N7	Unlimited mint issue	Authority Control Vulnerability Audit	Medium	Confirmed
N8	Burn contract's token issue	Authority Control Vulnerability Audit	Low	Confirmed
N9	Access control issue	Authority Control Vulnerability Audit	Suggestion	Confirmed
N10	Arbitrary ratio issue	Authority Control Vulnerability Audit	Medium	Fixed
N11	Claim arbitrary ManagerFee issue	Authority Control Vulnerability Audit	Medium	Confirmed
N12	rebalance an unbound token issue	Design Logic Audit	Medium	Fixed

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
<Constructor>	Public	Can Modify State	PCToken
setSwapFee	External	Can Modify State	logs lock onlyOwner needsBPool
isPublicSwap	External	-	viewlock needsBPool
setCap	External	Can Modify State	logs lock needsBPool onlyOwner
setPublicSwap	External	Can Modify State	logs lock onlyOwner needsBPool
claimManagerFee	External	Can Modify State	logs lock managerFeeOwner needsBPool
createPool	External	Can Modify State	onlyOwner logs lock
createPool	External	Can Modify State	onlyOwner logs lock
rebalance	External	Can Modify State	logs lock onlyOwner needsBPool
commitAddToken	External	Can Modify State	logs lock onlyOwner needsBPool
applyAddToken	External	Can Modify State	logs lock onlyOwner needsBPool
removeToken	External	Can Modify State	logs lock onlyOwner needsBPool
joinPool	External	Can Modify State	logs lock needsBPool lockUnderlyingPool
exitPool	External	Can Modify State	logs lock needsBPool lockUnderlyingPool

ConfigurableRightsPool			
whitelistLiquidityProvider	External	Can Modify State	onlyOwner lock logs
removeWhitelistedLiquidityProvider	External	Can Modify State	onlyOwner lock logs
canProvideLiquidity	External	-	-
hasPermission	External	-	-
getDenormalizedWeight	External	-	viewlock needsBPool
getRightsManagerVersion	External	-	-
getDesynSafeMathVersion	External	-	-
getSmartPoolManagerVersion	External	-	-
mintPoolShareFromLib	Public	Can Modify State	-
pushPoolShareFromLib	Public	Can Modify State	-
pullPoolShareFromLib	Public	Can Modify State	-
burnPoolShareFromLib	Public	Can Modify State	-
createPoolInternal	Internal	Can Modify State	-
_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	-

ConfigurableRightsPool			
_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	-
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	-

ExchangeProxy			

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	-

LiquidityPool			
Function Name	Visibility	Mutability	Modifiers
<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	-	_viewlock_
getBalance	External	-	_viewlock_
getSwapFee	External	-	_viewlock_

LiquidityPool			
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_
unbind	External	Can Modify State	_logs_ _lock_
unbindPure	External	Can Modify State	_logs_ _lock_
gulp	External	Can Modify State	_logs_ _lock_
getSpotPrice	External	-	_viewlock_
getSpotPriceSansFee	External	-	_viewlock_
joinPool	External	Can Modify State	_logs_ _lock_
exitPool	External	Can Modify State	_logs_ _lock_
_safeApprove	Internal	Can Modify State	-
_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	-

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

Token			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20Detailed
mintTokens	Public	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	-
setTokens	External	Can Modify State	-
finalize	External	Can Modify State	-
increaseWeight	External	Can Modify State	-
rebalance	External	Can Modify State	-
decreaseWeight	External	Can Modify State	-
setCap	External	Can Modify State	-
commitAddToken	External	Can Modify State	-
applyAddToken	External	Can Modify State	-
removeToken	External	Can Modify State	-
whitelistLiquidityProvider	External	Can Modify State	-

Actions			
removeWhitelistedLiquidityProvider	External	Can Modify State	-
_safeApprove	Internal	Can Modify State	-
_join	Internal	Can Modify State	-

CRPFactory			
Function Name	Visibility	Mutability	Modifiers
newCrp	External	Can Modify State	-
isCrp	External	-	-

DSAuthEvents			
Function Name	Visibility	Mutability	Modifiers

DSAuth			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setOwner	Public	Can Modify State	auth
setAuthority	Public	Can Modify State	auth
isAuthorized	Internal	-	-

DSNote			
Function Name	Visibility	Mutability	Modifiers

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

DSProxy			
<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

Factory			
isLiquidityPool	External	-	-
newLiquidityPool	External	Can Modify State	-
<Constructor>	Public	Can Modify State	-
getBLabs	External	-	-
setBLabs	External	Can Modify State	-
getSwapRouter	External	-	-
setSwapRouter	External	Can Modify State	-
collect	External	Can Modify State	-

Vault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
depositManagerToken	Public	Can Modify State	-
depositIssueRedeemToken	Public	Can Modify State	-
communaldepositToken	Internal	Can Modify State	-
poolManagerTokenList	Public	-	-
poolManagerTokenAmount	Public	-	-
poolIssueRedeemTokenList	Public	-	-
poolIssueRedeemTokenAmount	Public	-	-
getManagerClaimBool	Public	-	-
setBlackList	Public	Can Modify State	onlyOwner
adminClaimToken	Public	Can Modify State	onlyOwner

Vault			
getBNB	Public	Payable	onlyOwner
setManagerRatio	Public	Can Modify State	onlyOwner
setIssueRedeemRatio	Public	Can Modify State	onlyOwner
managerClaim	Public	Can Modify State	-
managerClaimRecordList	Public	-	-
managerClaimList	Public	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event record

Category: Others

Content

In the LiquidityPool contract, the controller can modify the _swapFee, _controller and _publicSwap parameters through the setSwapFee, setController and setPublicSwap functions respectively, but no event recording is performed.

The same is true for the setManagerRatio and setIssueRedeemRatio functions in the Vault contract.

Code location:

contracts/base/LiquidityPool.sol

```
    function setSwapFee(uint swapFee)
        external
        _logs_
        _lock_
    {
        ...
        _swapFee = swapFee;
    }

    function setController(address manager)
        external
        _logs_
        _lock_
```

```
    {
        ...
        _controller = manager;
    }

    function setPublicSwap(bool public_)
        external
        _logs_
        _lock_
    {
        ...
        _publicSwap = public_;
    }
```

contracts/depoly/Vault.sol

```
    function setManagerRatio(uint amount) public onlyOwner{
        manager_ratio = amount;
    }
      function setIssueRedeemRatio(uint amount) public onlyOwner{
        issue_redeem_ratio = amount;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for subsequent self-inspection or community review.

Status

Confirmed

[N2] [Low] Swap path issue

Category: Design Logic Audit

Content

In the LiquidityPool contract, the controller role can adjust the token weight and balance through the rebindSmart function. It will first combine tokenA and tokenB into a swap path, but does not check whether the tokenA-tokenB pool in the DEX exists. If there is no tokenA-tokenB pool, it should use the native token as a relay path for swap.

Code location: contracts/base/LiquidityPool.sol

```
function rebindSmart(
    address tokenA,
    address tokenB,
    uint deltaWeight,
    uint deltaBalance,
    bool isSoldout,
    uint minAmountOut
    )
    public
    _logs_
    _lock_
{
    require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
    require(!_finalized, "ERR_IS_FINALIZED");

    address[] memory paths = new address[](2);
    paths[0] = tokenA;
    paths[1] = tokenB;

    ...
}
```

Solution

It is recommended to check whether the corresponding pool exists in the DEX when constructing the swap path.

Status

Confirmed

[N3] [Medium] Issue with tokens outside of ETF

Category: Design Logic Audit

Content

Any user can call LiquidityPool's rebindSmart function through the rebalance function to adjust the token weight and balance. If the tokenB passed in by the user is a worthless token, a malicious user can exchange the valuable tokenA for tokenB to complete arbitrage during the rebindSmart operation.

Code location:

contracts/libraries/SmartPoolManager.sol

contracts/base/LiquidityPool.sol

```
    function rebalance(
        IConfigurableRightsPool self,
        IBPool bPool,
        address tokenA,
        address tokenB,
        uint deltaWeight,
        uint minAmountOut
    )
        external
    {
        ...
        bPool.rebindSmart(tokenA, tokenB, deltaWeight, deltaBalanceA, soldout,
minAmountOut);
    }


    function rebindSmart(
        address tokenA,
        address tokenB,
        uint deltaWeight,
        uint deltaBalance,
        bool isSoldout,
        uint minAmountOut
        )
        public
        _logs_
        _lock_
    {
        ...

            swapRouter.swapExactTokensForTokens(
                deltaBalance,
                minAmountOut,
                paths,
                address(this),
                badd(block.timestamp, 1800)
            );
            uint balanceBAfter = IERC20(tokenB).balanceOf(address(this));

            uint newBalanceB = badd(oldBalanceB, bsub(balanceBAfter, balanceBBefore));

            _records[tokenB].balance = newBalanceB;
            _records[tokenB].denorm = newWeightB;
        }
         ...

            swapRouter.swapExactTokensForTokens(
                deltaBalance,
```

```
                minAmountOut,
                paths,
                address(this),
                badd(block.timestamp, 1800)
            );
            ...
    }
```

Solution

It is recommended to perform a whitelist check on the target tokens of the swap.

Status

Confirmed; After communicating with the project team, the project team stated that it will add a whitelist check in the future.

[N4] [Low] Risk of over-approval

Category: Design Logic Audit

Content

In the LiquidityPool contract, tokenA is approved to the router contract during the rebindSmart operation, and the approved amount is `uint256(-1)`, but the approved amount is not recovered after the swap is completed. This would lead to the risk of over-approval.

Code location: contracts/base/LiquidityPool.sol

```
    function rebindSmart(
        address tokenA,
        address tokenB,
        uint deltaWeight,
        uint deltaBalance,
        bool isSoldout,
        uint minAmountOut
        )
        public
        _logs_
        _lock_
    {
        ...

            _safeApprove(IERC20(tokenA), address(swapRouter), uint256(-1));

            swapRouter.swapExactTokensForTokens(
```

```
                deltaBalance,
                minAmountOut,
                paths,
                address(this),
                badd(block.timestamp, 1800)
            );
        ...

    }
```

Solution

It is recommended to set the approval limit to 0 after the approved contract has completed its operation.

Status

Confirmed

[N5] [Suggestion] Insecure Token Transfer

Category: Others

Content

In the LiquidityPool contract, the `_pullUnderlying` function is used to transfer the user's tokens into this contract, and the `_pushUnderlying` function is used to transfer the tokens to the specified user. But it just uses the standard EIP20 transfer function and checks the return value. However, if the user adds a token that does not meet the EIP20 standard (such as: ERC-BNB), it will cause problems with the transfer of the token.

Code location: contracts/base/LiquidityPool.sol

```
    function _pullUnderlying(address erc20, address from, uint amount)
        internal
    {
        bool xfer = IERC20(erc20).transferFrom(from, address(this), amount);
        require(xfer, "ERR_ERC20_FALSE");
    }

    function _pushUnderlying(address erc20, address to, uint amount)
        internal
    {
        bool xfer = IERC20(erc20).transfer(to, amount);
        require(xfer, "ERR_ERC20_FALSE");
    }
```

Solution

It is recommended to use OpenZeppelin's SafeERC20 library for token transfers.

Status

Confirmed

[N6] [Suggestion] Compatibility issue

Category: Others

Content

In the LiquidityPool contract, the `_pullUnderlying` function is used to transfer the user's tokens into this contract, but the LiquidityPool contract does not check how many tokens the contract actually received. If LiquidityPool lists deflationary tokens, the tokens transferred through the `_pullUnderlying` function do not match the tokens actually received by the contract. This will lead to a bias in the calculation of the number of tokens when doing the joinPool.

Code location: contracts/base/LiquidityPool.sol

```
    function _pullUnderlying(address erc20, address from, uint amount)
        internal
    {
        bool xfer = IERC20(erc20).transferFrom(from, address(this), amount);
        require(xfer, "ERR_ERC20_FALSE");
    }

    function joinPool(uint poolAmountOut, uint[] calldata maxAmountsIn)
        external
        _logs_
        _lock_
    {
        require(_finalized, "ERR_NOT_FINALIZED");

        uint poolTotal = totalSupply();
        uint ratio = bdiv(poolAmountOut, poolTotal);
        require(ratio != 0, "ERR_MATH_APPROX");

        for (uint i = 0; i < _tokens.length; i++) {
            address t = _tokens[i];
            uint bal = _records[t].balance;
            uint tokenAmountIn = bmul(ratio, bal);
            require(tokenAmountIn != 0, "ERR_MATH_APPROX");
            require(tokenAmountIn <= maxAmountsIn[i], "ERR_LIMIT_IN");
```

```
            _records[t].balance = badd(_records[t].balance, tokenAmountIn);
            emit LOG_JOIN(msg.sender, t, tokenAmountIn);
            _pullUnderlying(t, msg.sender, tokenAmountIn);
        }
        _mintPoolShare(poolAmountOut);
        _pushPoolShare(msg.sender, poolAmountOut);
    }
```

Solution

Check the token balance difference before and after the transfer when performing the token transfer operation to this contract.

Status

Confirmed

[N7] [Medium] Unlimited mint issue

Category: Authority Control Vulnerability Audit

Content

In the TToken contract, the user owner can freely mint tokens without upper limit through the mint function, which will lead to the risk of excessive owner authority.

Code location: contracts/T/TToken.sol

```
    function mint(address dst, uint256 amt) public _onlyOwner_ returns (bool) {
        _mint(dst, amt);
        return true;
    }
```

Solution

It is recommended to transfer the owner to community governance, or to set a cap on the total supply of tokens.

Status

Confirmed

[N8] [Low] Burn contract's token issue

Category: Authority Control Vulnerability Audit

Content

In the TToken contract, the user can arbitrarily burn the tokens of this contract through the burn function.

Code location: contracts/T/TToken.sol

```
    function burn(uint amt) public returns (bool) {
        require(_balance[address(this)] >= amt, "ERR_INSUFFICIENT_BAL");
        _balance[address(this)] = sub(_balance[address(this)], amt);
        _totalSupply = sub(_totalSupply, amt);
        emit Transfer(address(this), address(0), amt);
        return true;
    }
```

Solution

It is recommended to control the permissions of the burn function, which can only be burned by privileged roles.

Status

Confirmed

[N9] [Suggestion] Access control issue

Category: Authority Control Vulnerability Audit

Content

In the Vault contract, any user can perform the managerClaim operation, which will cause any user to perform the managerClaim operation without the pool without being restricted by various conditions in the ConfigurableRightsPool contract.

depositManagerToken and depositIssueRedeemToken should also only be called by the ConfigurableRightsPool contract.

Code location: contracts/depoly/Vault.sol

```
    function managerClaim(address pool) public {
        address manager_address =  ICRPPool(pool).getController();
         ...
    }

    function depositManagerToken(address[] calldata poolTokens,uint[] calldata
tokensAmount) public {
        ...
    }
```

```
    function depositIssueRedeemToken(address[] calldata poolTokens,uint[] calldata
tokensAmount) public {
        ...
    }
```

Solution

It is recommended to restrict the managerClaim、depositManagerToken and depositIssueRedeemToken function to only be called by the ConfigurableRightsPool contract.

Status

Confirmed; After communicating with the project team, the project team stated that this is the expected design.

[N10] [Medium] Arbitrary ratio issue

Category: Authority Control Vulnerability Audit

Content

In the vault contract, the user can arbitrarily modify the manager_ratio and issue_redeem_ratio parameters through the setManagerRatio and setIssueRedeemRatio functions. If the ratio is set higher than expected, it will cause an error in the number of tokens transferred during the managerClaim operation.

Code location: contracts/depoly/Vault.sol

```
    function setManagerRatio(uint amount) public onlyOwner{
        manager_ratio = amount;
    }
      function setIssueRedeemRatio(uint amount) public onlyOwner{
        issue_redeem_ratio = amount;
    }
```

Solution

A range limit is recommended for the setting of ratio.

Status

Fixed

[N11] [Medium] Claim arbitrary ManagerFee issue

Category: Authority Control Vulnerability Audit

Content

In the ConfigurableRightsPool contract, the managerFeeOwner role can charge management fees through the claimManagerFee function, and the management fees are calculated based on the time passed in by the owner. Since time can be passed in arbitrarily, this will lead to the risk of excessive owner permissions.

Code location: contracts/base/ConfigurableRightsPool.sol

```
    function claimManagerFee(uint time)
        external
        logs
        lock
        managerFeeOwner
        needsBPool
        virtual
    {
       address[] memory poolTokens = bPool.getCurrentTokens();
       uint[] memory tokensAmount = new uint[](poolTokens.length);
        bool returnValue;
        for (uint i = 0; i < poolTokens.length; i++) {
            address t = poolTokens[i];
            uint tokenBalance = bPool.getBalance(t);
            uint tokenAmountOut = DesynSafeMath.bmul(tokenBalance,managerFee*time/12);
            _pushUnderlying(t, address(this), tokenAmountOut);
           returnValue = IERC20(t).safeApprove(vault_Address, tokenAmountOut);
           tokensAmount[i] = tokenAmountOut;
        }
        IVault(vault_Address).depositManagerToken(poolTokens,tokensAmount);
        startClaimFeeTime = block.timestamp;
    }
```

Solution

It is recommended to calculate time in the contract, and use the time recorded in the contract to participate in the calculation.

Status

Confirmed

[N12] [Medium] rebalance an unbound token issue

Category: Design Logic Audit

Content

In the LiquidityPool contract, the owner can bind tokenB through the rebalance function. But the

ConfigurableRightsPool contract does not approve tokenB to the LiquidityPool contract, which will cause problems when joinPool.

Solution

If tokenB is not bound during rebalance, the ConfigurableRightsPool contract will approve tokenB to the LiquidityPool contract during rebalance.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002207180001	SlowMist Security Team	2022.06.28 - 2022.07.18	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 5 medium risks, 3 low risks, 4 suggestions. And 3 medium risks, 3 low risks, 4 suggestions were confirmed; All other findings were fixed. The code was not deployed to the mainnet.

6 Statement

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

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

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist