Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.09.26, the SlowMist security team received the DeSyn Protocol team's security audit application for DeSyn Phase5, 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. This is an audit of new features since the last audit. The structure of the protocol has not changed.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Potential denial of service risk due to gulp operations	Denial of Service Vulnerability	Medium	Fixed
N2	Incorrect event logging	Others	Suggestion	Fixed
N3	Incorrect WETH address	Others	High	Fixed
N4	Incorrect poolTokens check	Design Logic Audit	Medium	Fixed
N5	Redundant STBT token check	Others	Suggestion	Fixed
N6	Potential risk of slot conflict	Variable Coverage Vulnerability	Suggestion	Acknowledged
N7	Risk of event forgery	Malicious Event Log Audit	Low	Acknowledged
N8	Potential risk of denial of service due to large CRPFactory array	Denial of Service Vulnerability	Suggestion	Acknowledged
N9	Potential risk of endless loop	Denial of Service Vulnerability	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

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

commit: b6eb5476d5fa89388d7ce300090d69b05654954f

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

commit: 069722d464d0903a4a124be149c9f3ce0198b495

Fixed Version

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

commit: 06ee533f2872f7c2e723bf0a4d8a4b934ec5e6a8

The main network address of the contract is as follows:

Contract Name	Contract Address	Chain
Actions	0x6AF3054ABFfD253635f2F2E181BDe0E2eB6A4Ea7	Merlin
Vault	0x3a7F0E0a07037B366A5E414724147D7F30E8bAac	Merlin
UserVault	0x768bb4773F2048299Fb4ee5231D54D5865c20e18	Merlin
Oracle	0xec4AB14186416752e79C7CF32F009Cd15CBC65aD	Merlin
DesynChainlinkOracle	0xde21a6cc567a0c4bEC33f7Cc25F826c188F4F235	Merlin
Factory	0x790b4ee7998A93702f29e56f8b615eF35BE5af43	Merlin
DesynSafeMath	0x84Ce6eBE261f6Af7d0a50b5C9c21Df1700a012b1	Merlin
RightsManager	0xcfFbB141118DCE7B2adAc82b248294A61fD7cD44	Merlin
SmartPoolManager	0x15b719537a6ECF0fF2eed27F007082d070EaA0eB	Merlin
CRPFactory	0x1a5B48000D74ca588148038374254b629e7ac42E	Merlin
Actions	0x82327d64b43a6ee922a0f0393ce99f0203c7da39	Bitlayer
Vault	0x301Be34Da27088f2a81F344904c5384F212b132d	Bitlayer
UserVault	0xb5068dA710D6Ba6D79a9E6Fd8a9e80b1bFdf9164	Bitlayer
Oracle	0x0B3D68F0646D0AFB2CE625B146eB99FE941ba8BC	Bitlayer
DesynChainlinkOracle	0x6AF58b55B4eec887Ca39946842Fb463e9Fb25Ed4	Bitlayer
Factory	0x09eFC8C8F08B810F1F76B0c926D6dCeb37409665	Bitlayer
DesynSafeMath	0xdE6b117384452b21F5a643E56952593B88110e78	Bitlayer
RightsManager	0x5C3027D8Cb28A712413553206A094213337E88c5	Bitlayer
SmartPoolManager	0x770c9d0851b21df8A84943EdE4f487D30d9741ba	Bitlayer
CRPFactory	0xe788511225632ffdA2c532d65ede98aF047282e8	Bitlayer

4.2 Visibility Description

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

Actions			
Function Name	Visibility	Mutability	Modifiers
create	External	Can Modify State	-
createSmartPool	External	Can Modify State	-
joinPool	External	Can Modify State	-
autoExitSmartPool	External	Payable	lock
joinSmartPool	External	Can Modify State	-
autoJoinSmartPool	External	Payable	lock
exitPool	External	Can Modify State	-
setPublicSwap	External	Can Modify State	-
setController	External	Can Modify State	-
setManagersInfo	Public	Can Modify State	-
_beforeOwnerChange	Internal	Can Modify State	-
snapshotBeginAssets	External	Can Modify State	-
snapshotEndAssets	External	Can Modify State	-
approveUnderlying	External	Can Modify State	-
rebalance	External	Can Modify State	-
finalize	External	Can Modify State	-
setCap	External	Can Modify State	-
whitelistLiquidityProvider	External	Can Modify State	-
removeWhitelistedLiquidityProvider	External	Can Modify State	-

Actions			
addTokenToWhitelist	Public	Can Modify State	-
claimManagementFee	Public	Can Modify State	-
_safeApprove	Internal	Can Modify State	-
_join	Internal	Can Modify State	-
_exit	Internal	Can Modify State	-
claimKolReward	Public	Can Modify State	-
claimManagersReward	External	Can Modify State	-
_claimManagersReward	Internal	Can Modify State	-
_makeSwap	Internal	Can Modify State	-
_getVault	Internal	-	-
_getUserVault	Internal	Can Modify State	-
_calculateShare	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-

CRPFactory			
Function Name	Visibility	Mutability	Modifiers
createPool	Internal	Can Modify State	-
newCrp	External	Can Modify State	-
setUserVault	External	Can Modify State	onlyBlabs
setByteCodes	External	Can Modify State	onlyBlabs _logs_
isCrp	External	-	-

ConfigurableRightsPool			
Function Name	Visibility	Mutability	Modifiers
<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
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	Public	-	-
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	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	logs
beginFundAssets	External	-	-
endFundAssets	External	-	-
snapshotEndAssets	Public	Can Modify State	logs
snapshotAssets	Public	Can Modify State	-
_getPoolTokensInfo	Internal	-	-

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	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_

LiquidityPool			
gulp	External	Can Modify State	_logs_ _lock_
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	-

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

SmartPoolManager			
verifyTokenComplianceInternal	Internal	Can Modify State	-
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	-

4.3 Vulnerability Summary

[N1] [Medium] Potential denial of service risk due to gulp operations

Category: Denial of Service Vulnerability

Content

In the Actions contract, when the user performs the autoJoinSmartPool operation, the number of shares the user can obtain will be calculated through the `_calculateShare` function, and minPoolAmountOut will be checked at the end. The `_calculateShare` function obtains the number of tokens recorded in the pool through bPool's getBalance when calculating the share, but unfortunately any user can update this parameter through bPool's gulp function.

Therefore, when an ordinary user performs an autoJoinSmartPool operation, a malicious user directly transfers funds to bPool and calls the gulp function to update the token balance recorded in the pool. At this time, ordinary users will not be able to successfully add liquidity due to the minPoolAmountOut check. If the minPoolAmountOut value passed in by an ordinary user is 0, it may cause an interest rate inflation attack.

Code location:

contracts/deploy/Actions.sol#L604-L609

contracts/deploy/Actions.sol#L407

contracts/base/LiquidityPool.sol#L296

```
    function gulp(address token) external _logs_ _lock_ {
        // require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(_records[token].bound, "ERR_NOT_BOUND");
        _records[token].balance = IERC20(token).balanceOf(address(this));
    }

    function autoJoinSmartPool(
        ...
    ) external payable lock {
        ...
        uint shareAmountOutPerToken =
_calculateShare(joinVar.bPool,joinVar.totalPoolShares,joinVar.issueFee, poolTokens[i],
maxAmountsIn[i]);
        ...
        require(minPoolAmountOut <=
joinVar.actualShareAmountOut,"ERR_SHARE_AMOUNT_TO_SMALL");
    }

    function _calculateShare(LiquidityPoolActions bPool,uint totalPoolShares, uint
issueFee, address t, uint actualTokenAmountIn) internal returns(uint) {
        uint totalTokenBalance = bPool.getBalance(t);
        uint issueFeeRate = issueFee.bmul(1000);
        uint share = (totalPoolShares.bsub(1).bmul(actualTokenAmountIn) *
(uint(1000).bsub(issueFeeRate))).bdiv((1000 * totalTokenBalance.badd(1)));
        return share;
    }
```

Solution

It is recommended to control the permissions of gulp functions so that only privileged roles can be called.

Status

Fixed

[N2] [Suggestion] Incorrect event logging

Category: Others

Content

In the CRP contract, the createPool function adds the creator parameter as the real creator of the pool. However, the corresponding events were not modified accordingly.

Code location:

contracts/base/ConfigurableRightsPool.sol#L228-L230

```
    function createPool(
        address creator,
        uint initialSupply,
        uint collectPeriod,
        SmartPoolManager.Period closurePeriod,
        SmartPoolManager.PoolTokenRange memory tokenRange
    ) external virtual onlyOwner logs lock notPaused {
            ...
            emit PoolTokenInit(msg.sender, address(this),_initialTokens[0],
_initialBalances[0], initialSupply);
            emit SizeChanged(msg.sender, "UPPER", oldCap, etfStatus.upperCap);
            emit SizeChanged(msg.sender, "FLOOR", oldFloor, etfStatus.floorCap);
            ...
    }
```

Solution

It is recommended to use expected parameters for event logging.

Status

Fixed

[N3] [High] Incorrect WETH address

Category: Others

Content

The WETH address constant is hard-coded in the Actions contract, but this address is an EOA address on the Ethereum mainnet and is not the correct WETH address.

Code location: contracts/deploy/Actions.sol#L210

```
    address constant WETH = 0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6;
```

Solution

It is recommended to replace this address with a real WETH address.

Status

Fixed

[N4] [Medium] Incorrect poolTokens check

Category: Design Logic Audit

Content

In the autoJoinSmartPool function of the Actions contract, it checks the tokens deposited by the user through `poolTokens[i] == handleToken`, but handleToken has been replaced by the issueToken parameter during the native token checking phase. Therefore, the `poolTokens[i] == handleToken` check is not accurate. If handleToken is WETH, it will cause an error in the `_makeSwap` operation.

Code location: contracts/deploy/Actions.sol#L385

```
    function autoJoinSmartPool(
        ...
    ) external payable lock {
        ...
        if (handleToken == NATIVE_TOKEN) {
            require(msg.value > 0 && msg.value == issueAmount, 'ERROR_ETH');
            IWETH(WETH).deposit{value: msg.value}();

            issueToken = IERC20(WETH);
        } else {
            issueToken = IERC20(handleToken);
            issueToken.safeTransferFrom(user, address(this), issueAmount);
        }
        ...
            IAggregator.SwapData memory swapData = swapDatas[i];

            poolTokens[i] == handleToken
                ? maxAmountsIn[i] = swapData.quantity
                : maxAmountsIn[i] = _makeSwap(swapBase, swapData,
IERC20(poolTokens[i]));
            ...
        }
        ...
    }
```

Solution

It is recommended to replace handleToken with issueToken.

Status

Fixed

[N5] [Suggestion] Redundant STBT token check

Category: Others

Content

In the autoJoinSmartPool and _exit functions of the Actions contract, when making a token transfer, it will be checked whether the transferred token is an STBT token. But in fact, the protocol does not allow deposits of STBT tokens, and STBT tokens will also be converted into stablecoins after the ETF expires. Therefore, theoretically, there will be no STBT tokens in the contract, so the STBT token check is redundant. .

Code location:

contracts/deploy/Actions.sol#L402

contracts/deploy/Actions.sol#L542

```
    function autoJoinSmartPool(
        ...
    ) external payable lock {
        ...
        for (uint i; i < poolTokens.length; i++) {
            IERC20 token = IERC20(poolTokens[i]);
            if (token.balanceOf(address(this)) > 0 && token != STBT)
token.safeTransfer(user, token.balanceOf(address(this)));
        }
        ...
    }

    function _exit(
        ...
    ) internal {
        ...
        if(!isSmartMode){
            for (uint i = 0; i < tokens.length; i++) {
                IERC20 token = IERC20(tokens[i]);
                if (token.balanceOf(address(this)) > 0 && token != STBT) {
                    token.safeTransfer(msg.sender, token.balanceOf(address(this)));
                }
            }
        }
    }
```

Solution

It is recommended to remove easy checks to save gas.

Status

Fixed; Due to STBT's rebase mechanism, it may be necessary to refund the user during the Join operation, but the user is not in the whitelist, and the transfer operation of STBT tokens will fail. Therefore the project team keeps a check on the STBT token in the Join operation. It was modified in commit 3fe087c9ef5b9830306f6f8551918b1389add9c7.

[N6] [Suggestion] Potential risk of slot conflict

Category: Variable Coverage Vulnerability

Content

In this protocol iteration, both SmartPoolManager and Actions contracts have changed their storage structures. If these contracts use an upgradeable model, and upgrading the contract directly on the original basis may lead to contract storage slot conflicts. In fact, the Actions contract is an upgradeable contract, so special attention should be paid to such risks.

Solution

For contracts whose storage slots have changed, it should be noted that they cannot be used to upgrade old contracts.

Status

Acknowledged; After communicating with the project team, the project team stated that they would not use an upgradable model.

[N7] [Low] Risk of event forgery

Category: Malicious Event Log Audit

Content

In the CRP contract, the exitPool function adds a user parameter to record the real caller when the user exits through the Actions contract. However, users can also exit by calling the exitPool function of the CRP contract. They can pass in any user parameter to make the LogExit event record an incorrect value.

Code location: contracts/base/ConfigurableRightsPool.sol#L389

```
    function exitPool(uint poolAmountIn, uint[] calldata minAmountsOut, address user)
external logs lock needsBPool notPaused {
        ...
        uint[] memory redeemFeesReceived = new uint[](poolTokens.length);
```

```
        for (uint i = 0; i < poolTokens.length; i++) {
            ...
            emit LogExit(user, poolTokens[i], finalAmountOut);
        }
        ...
    }
```

Solution

To log the correct user parameter, you should ensure that msg.sender is a trusted Actions contract. Change the parameters recorded by LogExit by determining whether msg.sender is a valid Actions contract.

Status

Acknowledged; After communicating with the project team, the project team stated that this is only for recording purposes, and they will optimize it in future versions.

[N8] [Suggestion] Potential risk of denial of service due to large CRPFactory array

Category: Denial of Service Vulnerability

Content

In the CRPFactory contract, when the Blabs role performs addCRPFactory and removeCRPFactory operations, it will use a for loop to traverse the entire CRPFactorys array. If the array length is too large, it will lead to DoS risks.

Code location: contracts/deploy/CRPFactory.sol#L168-L193

```
    function addCRPFactory(address _crpFactory) external onlyBlabs {
        uint len = CRPFactorys.length;

        for(uint i=0; i<len; i++){
            require(CRPFactorys[i] != _crpFactory, "ERR_HAS_BEEN_ADDED");
        }
        CRPFactorys.push(_crpFactory);
        emit AddCRPFactory(msg.sender, _crpFactory);
    }

    function removeCRPFactory(address _crpFactory) external onlyBlabs  {
        uint len = CRPFactorys.length;
        for(uint i=0; i<len; i++){
            if(CRPFactorys[i] == _crpFactory){
                CRPFactorys[i] = CRPFactorys[len-1];
                CRPFactorys.pop();
                emit RemoveCRPFactory(msg.sender, _crpFactory);
                break;
```

```
            }
        }
    }
```

Solution

If you need to add a large number of CRPFactory addresses, it is recommended to use OpenZeppelin's EnumerableMap library to avoid this risk.

Status

Acknowledged; After communicating with the project team, the project team stated that this function is controlled by the admin role, which will control the length of the CRPFactorys array.

[N9] [Suggestion] Potential risk of endless loop

Category: Denial of Service Vulnerability

Content

In the CRPFactory contract, the isCrp function is used to check whether the address passed by the user is a CRP contract. If not, the CRPFactorys array will be circulated and the isCrp function of other CRPFactory will be called to check. However, it should be noted that if the Blabs role adds this contract address to the CRPFactorys array, the user will fall into an infinite loop error when querying a CRP address that is not recorded in this contract through the isCrp function.

Code location: contracts/deploy/CRPFactory.sol#L159

```
    function isCrp(address addr) external view returns (bool) {
        if(_isCrp[addr]) return _isCrp[addr];

        for(uint i=0; i<CRPFactorys.length; i++){
            if(ICRPFactory(CRPFactorys[i]).isCrp(addr)) return true;
        }
    }

    function addCRPFactory(address _crpFactory) external onlyBlabs {
        uint len = CRPFactorys.length;

        for(uint i=0; i<len; i++){
            require(CRPFactorys[i] != _crpFactory, "ERR_HAS_BEEN_ADDED");
        }
        CRPFactorys.push(_crpFactory);
```

```
        emit AddCRPFactory(msg.sender, _crpFactory);
    }
```

Solution

When performing the addCRPFactory operation, you should be careful not to add this contract address to the CRPFactorys array. It is recommended to check that `_crpFactory` is not equal to `address(this)` in the addCRPFactory function.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002309270001	SlowMist Security Team	2023.09.26 - 2023.09.27	Low 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 high risk, 2 medium risk, 1 low risk, and 5 suggestions. All the findings were fixed or acknowledged. The code was not deployed to the mainnet.

6 Statement

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

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

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist