Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.02.19, the SlowMist security team received the bitlayer team's security audit application for bitlayer-contracts, 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

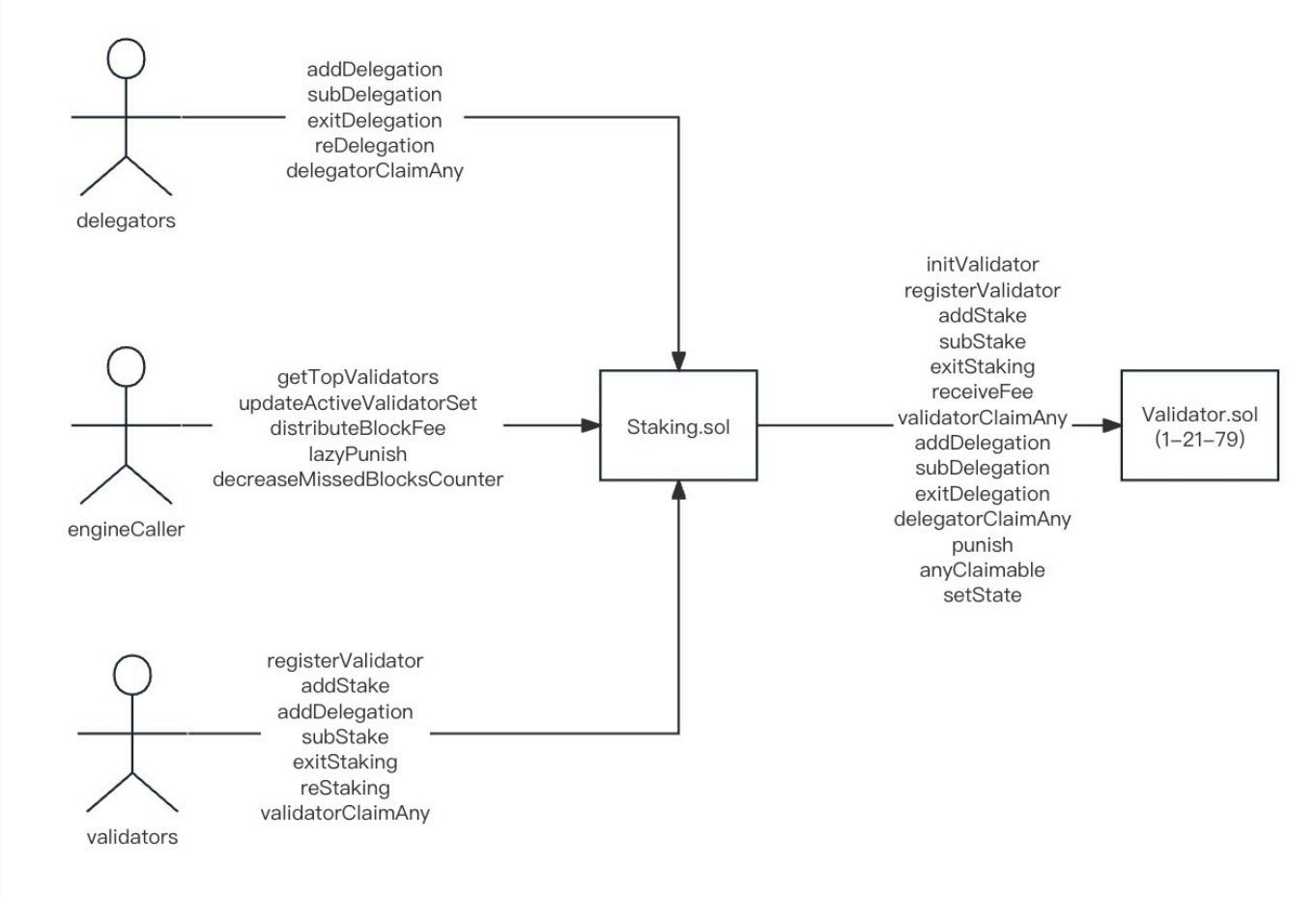
Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
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

This project contains the key contracts of BitLayer, including the built-in contracts and the project token contracts.

Core logic flowchart:



3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N3	Reentrancy risks in Staking.doReStake	Reentrancy Vulnerability	High	Fixed
N5	Missing zero address validation	Design Logic Audit	Low	Fixed
N6	Missing events arithmetic	Malicious Event Log Audit	Suggestion	Fixed
N7	False top-up risks	Unsafe External Call Audit	Low	Fixed
N8	Missing check return value	Unsafe External Call Audit	Medium	Fixed
N9	Using obsolete features to transfer native token	Denial of Service Vulnerability	Low	Fixed

NO	Title	Category	Level	Status
N10	Amount should larger than zero in slashFromUnbound function	Unsafe External Call Audit	Low	Acknowledged
N11	Redundant code	Others	Information	Acknowledged
N12	Risk of excessive authority	Design Logic Audit	Medium	Acknowledged
N13	Deletion on `unboundRecords` will not delete the mapping	Scoping and Declarations Audit	Low	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/bitlayer-org/bitlayer-contracts/tree/develop/contracts

Initial audit commit: 751c34b1072b7539c1a063b38522cf7606c9268a

Final audit commit: da9e618fc0358b626800fe4cba1ea7f64c7fe206

```
contracts/builtin/Staking.sol
contracts/builtin/Validator.sol
contracts/basic/BRC.sol
contracts/basic/LockingContract.sol
contracts/basic/Vault.sol
contracts/basic/TokenFactory.sol
contracts/basic/MultiSigWallet.sol
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

Staking			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer onlyValidAddress onlyValidAddress
initValidator	External	Can Modify State	onlyInitialized onlyNotExists
removePermission	External	Can Modify State	onlyAdmin
changeFoundationPool	External	Can Modify State	onlyAdmin
getTopValidators	External	-	-
updateActiveValidatorSet	External	Can Modify State	onlyEngine onlyOperateOnce onlyBlockEpoch
distributeBlockFee	External	Payable	onlyEngine onlyOperateOnce
distributeFeeToVals	Internal	Can Modify State	-
getActiveValidators	External	-	-
getBackupValidators	External	-	-
lazyPunish	External	Can Modify State	onlyEngine onlyExists onlyOperateOnce
decreaseMissedBlocksCounter	External	Can Modify State	onlyEngine onlyBlockEpoch onlyOperateOnce
doSlash	Private	Can Modify State	-
registerValidator	External	Can Modify State	onlyNotExists
takeStakedToken	Internal	Can Modify State	-
addStake	External	Can Modify State	onlyExistsAndByManager
addDelegation	External	Can Modify State	onlyExists
addStakeOrDelegation	Private	Can Modify State	-

Staking			
subStake	External	Can Modify State	onlyExistsAndByManager
subDelegation	External	Can Modify State	onlyExists
subStakeOrDelegation	Private	Can Modify State	-
exitStaking	External	Can Modify State	onlyExistsAndByManager
exitDelegation	External	Can Modify State	onlyExists
doExit	Private	Can Modify State	-
reStaking	External	Can Modify State	onlyExistsAndByManager onlyExists
reDelegation	External	Can Modify State	onlyExists onlyExists
doReStake	Private	Can Modify State	-
validatorClaimAny	External	Can Modify State	onlyExistsAndByManager nonReentrant
delegatorClaimAny	External	Can Modify State	onlyExists nonReentrant
doClaimAny	Private	Can Modify State	-
mustConvertStake	Private	-	-
afterLessStake	Private	Can Modify State	-
updateRanking	Private	Can Modify State	-
anyClaimable	Public	-	-
getAllValidatorsLength	External	-	-
getPunishValidatorsLen	Public	-	-
getPunishRecord	Public	-	-

Vault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
transferOwnership	External	Can Modify State	onlyRole
addWhitelist	External	Can Modify State	onlyRole
removeWhitelist	External	Can Modify State	onlyRole
releaseTreasure	External	Can Modify State	onlyRole
releaseERC20	External	Can Modify State	onlyRole
getWhitelists	External	-	-

Validator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	onlyValidAddress onlyValidAddress onlyValidRate
manager	External	-	-
addStake	External	Can Modify State	onlyOwner onlyCanDoStaking
subStake	External	Can Modify State	onlyOwner onlyCanDoStaking
exitStaking	External	Can Modify State	onlyOwner
receiveFee	External	Payable	onlyOwner
validatorClaimAny	External	Can Modify State	onlyOwner
addDelegation	External	Can Modify State	onlyOwner onlyCanDoStaking
subDelegation	External	Can Modify State	onlyOwner onlyCanDoStaking

Validator			
exitDelegation	External	Can Modify State	onlyOwner onlyCanDoStaking
innerSubDelegation	Private	Can Modify State	-
delegatorClaimAny	External	Can Modify State	onlyOwner
handleDelegatorPunishment	Private	Can Modify State	-
calcDelegatorPunishment	Private	-	-
canDoStaking	Private	-	-
addUnboundRecord	Private	Can Modify State	-
processClaimableUnbound	Private	Can Modify State	-
slashFromUnbound	Private	Can Modify State	-
addTotalStake	Private	Can Modify State	-
subTotalStake	Private	Can Modify State	-
punish	External	Can Modify State	onlyOwner
anyClaimable	External	-	onlyOwner
validatorClaimable	Private	-	-
delegatorClaimable	Private	-	-
getClaimableUnbound	Private	-	-
getPendingUnboundRecord	Public	-	-
getAllDelegatorsLength	Public	-	-
getSelfDebt	Public	-	-
getSelfSettledRewards	Public	-	-

Validator			
setState	External	Can Modify State	onlyOwner
testCalcDelegatorPunishment	Public	-	-
testGetClaimableUnbound	Public	-	-
testSlashFromUnbound	Public	Can Modify State	-

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

LockingContract			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
changeBeneficiary	External	Can Modify State	-
getVestingAmount	Public	-	-
getCurrentPeriod	Internal	-	-
claim	External	Can Modify State	-

MultiSigWallet			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
startTx	External	Can Modify State	onlySigner
signTx	External	Can Modify State	onlySigner

MultiSigWallet			
execute	External	Can Modify State	-
addSigner	External	Can Modify State	onlySelf
removeSigner	External	Can Modify State	onlySelf
changeThreshold	External	Can Modify State	onlySelf
getTxHash	Public	-	-
getTxSignedCount	Public	-	-
getPendingTx	Public	-	-
getTxSigners	Public	-	-
getSigners	Public	-	-

TokenFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
transferOwnership	External	Can Modify State	onlyRole
createErc20Token	External	Can Modify State	onlyRole
mintTo	External	Can Modify State	onlyRole
getTokenAddress	External	-	-
getTotalDeployed	External	-	-
getTokenSalt	External	-	-
getDeployedTokens	External	-	-

StandardToken			
Function Name	Visibility	Mutability	Modifiers

StandardToken			
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
balanceOf	Public	-	-
approve	Public	Can Modify State	-
allowance	Public	-	-

CustomERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
decimals	Public	-	-
mint	External	Can Modify State	onlyFactory

4.3 Vulnerability Summary

[N3] [High] Reentrancy risks in Staking.doReStake

Category: Reentrancy Vulnerability

Content

- bitlayer-contracts/contracts/builtin/Staking.sol

```
/**
 * @dev reStaking is used for a validator to move it's self stake.
 * @param _oldVal, the validitor moved from.
 * @param _newVal, the validitor moved to.
 **/

function reStaking(
    address _oldVal,
    address _newVal,
    uint256 _amount
) external onlyExistsAndByManager(_oldVal) onlyExists(_newVal) {
    doReStake(_oldVal, _newVal, _amount, true);
```

```
}

/**
 * @dev reDelegation is used for a user to move it's self stake.
 * @param _oldVal, the validitor moved from.
 * @param _newVal, the validitor moved to.
 **/
function reDelegation(
    address _oldVal,
    address _newVal,
    uint256 _amount
) external onlyExists(_oldVal) onlyExists(_newVal) {
    doReStake(_oldVal, _newVal, _amount, false);
}

function doReStake(address _oldVal, address _newVal, uint256 _amount, bool
_byValidator) private {
    require(_amount > 0, "E23");

    IValidator oldVal = valMaps[_oldVal];
    RankingOp op = RankingOp.Noop;

    if (_byValidator) {
        doClaimAny(_oldVal, true);
        op = oldVal.subStake(_amount, false);
    } else {
        doClaimAny(_oldVal, false);
        op = oldVal.subDelegation(_amount, msg.sender, false);
    }
    afterLessStake(_oldVal, oldVal, _amount, op);
    addStakeOrDelegation(_newVal, msg.sender, _amount, false);
}
```

There are external calls and state variables written after the calls.

Solution

Use openzeppelin `ReentrancyGuard.nonReentrant` modifier in `reStaking` and `reDelegation`.

Status

Fixed

[N5] [Low] Missing zero address validation

Category: Design Logic Audit

Content

- bitlayer-contracts/contracts/builtin/Staking.sol

```
Staking.initialize(address,address,uint256,address)._foundationPool
(contracts/builtin/Staking.sol#120) lacks a zero-check on :
- foundationPool = _foundationPool (contracts/builtin/Staking.sol#126)

Staking.changeFoundationPool(address)._foundationPool
(contracts/builtin/Staking.sol#160) lacks a zero-check on :
- foundationPool = _foundationPool (contracts/builtin/Staking.sol#161)
```

- bitlayer-contracts/contracts/basic/LockingContract.sol

```
LockingContract.constructor(address[],uint256[],uint256[],uint256[],uint256,address)._
lockingToken (contracts/basic/LockingContract.sol#41) lacks a zero-check on :
- LockingToken = _lockingToken (contracts/basic/LockingContract.sol#66)
```

- bitlayer-contracts/contracts/basic/TokenFactory.sol

```
CustomERC20.constructor(string,string,uint8,address)._factory
(contracts/basic/TokenFactory.sol#23) lacks a zero-check on :
- factory = _factory (contracts/basic/TokenFactory.sol#26)
```

Solution

Check that the address is not zero.

Status

Fixed

[N6] [Suggestion] Missing events arithmetic

Category: Malicious Event Log Audit

Content

- bitlayer-contracts/contracts/builtin/Validator.sol

```
function addStake(uint256 _amount)
function subStake(
        uint256 _amount,
        bool _isUnbound
```

```
    )
function receiveFee()
```

Solution

Emit an event for critical parameter changes.

Status

Fixed

[N7] [Low] False top-up risks

Category: Unsafe External Call Audit

Content

- bitlayer-contracts/contracts/basic/BRC.sol

```
function transfer(address _to, uint256 _value) public override returns (bool ) {
    if (balances[msg.sender] >= _value && _value > 0) {
        balances[msg.sender] -= _value;
        balances[_to] += _value;
        emit Transfer(msg.sender, _to, _value);
        return true;
    } else { return false; } //SlowMist//
}

function transferFrom(address _from, address _to, uint256 _value) public override
returns (bool ) {
    if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && _value >
0) {
        balances[_to] += _value;
        balances[_from] -= _value;
        allowed[_from][msg.sender] -= _value;
        emit Transfer(_from, _to, _value);
        return true;
    } else { return false; } //SlowMist//
}
```

The transfer/transferFrom functions return false when an error occurs on the chain, the transaction would be successful, if the receiver is an exchange, he may top-up for the user without checking the status of the calls, which will cause the asset loss.

Solution

Revert the transaction instead of return false;

Status

Fixed

[N8] [Medium] Missing check return value

Category: Unsafe External Call Audit

Content

The transfer function will return false if error occurs, but the claim function does not deal with it.

- bitlayer-contracts/contracts/basic/LockingContract.sol

```
function claim() external {
    //...
    IERC20(LockingToken).transfer(msg.sender, tokensToRelease);
    //...
}
```

- bitlayer-contracts/contracts/basic/Vault.sol

```
function releaseERC20(
    address erc20Token,
    address to,
    uint256 amount
)
    external
    onlyRole(AdminRole)
{
    //...
    token.transfer(to, amount);
    //...
}
```

Solution

Revert the transaction if the return value is false, or use openzeppelin `safeTransfer` function.

Status

Fixed

[N9] [Low] Using obsolete features to transfer native token

Category: Denial of Service Vulnerability

Content

Any smart contract that uses transfer() is taking a hard dependency on gas costs by forwarding a fixed amount of gas: 2300.

Gas costs are not constant. Smart contracts should be robust to this fact.

- bitlayer-contracts/contracts/basic/Vault.sol

```
function releaseTreasure(
    address receiver,
    uint256 amount
)
    external
    onlyRole(AdminRole)
{
    //...
    payable(receiver).transfer(amount);
    //...
}
```

Reference:

https://consensys.net/diligence/blog/2019/09/stop-using-soliditys-transfer-now/

Solution

Stop using transfer() in your code and switch to using call() instead. This carries a risk regarding reentrancy. Be sure to use one of the robust methods available for preventing reentrancy vulnerabilities.

Status

Fixed

[N10] [Low] Amount should larger than zero in slashFromUnbound function

Category: Unsafe External Call Audit

Content

- bitlayer-contracts/contracts/builtin/Validator.sol

```
function slashFromUnbound(address _owner, uint _amount) private {
    uint restAmount = _amount; //SlowMist//
```

```
    UnboundRecord storage rec = unboundRecords[_owner];
    // require there's enough pendingAmount
    require(rec.pendingAmount >= _amount, "E30");
    //...
}
```

It is meaning less if `_amount` is zero, which will spend more gas.

Solution

require(_amount > 0, "message");

Status

Acknowledged

[N11] [Information] Redundant code

Category: Others

Content

- bitlayer-contracts/contracts/builtin/Staking.sol

```
modifier onlyExistsAndByManager(address _val) {
    IValidator val = valMaps[_val];
    require(val != EMPTY_ADDRESS, "E08"); //SlowMist//
    require(val.manager() == msg.sender, "E02");
    _;
}
```

Solution

Remove these codes

Status

Acknowledged

[N12] [Medium] Risk of excessive authority

Category: Design Logic Audit

Content

- bitlayer-contracts/contracts/builtin/Staking.sol

  In Staking contract:

admin can `registerValidator` before `removePermission`.

- bitlayer-contracts/contracts/basic/Vault.sol

  In Vault contract:

  admin can `addWhitelist` / `removeWhitelist` / `releaseTreasure` / `releaseERC20`

- bitlayer-contracts/contracts/basic/TokenFactory.sol

  In TokenFactory contract:

  admin can `createErc20Token` / `mintTo`

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. And the authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged

[N13] [Low] Deletion on `unboundRecords` will not delete the mapping

Category: Scoping and Declarations Audit

Content

- bitlayer-contracts/contracts/builtin/Validator.sol

```
function processClaimableUnbound(address _owner) private returns (uint) {
        //...
        if (rec.startIdx == rec.count) {
            // all cleaned
            delete unboundRecords[_owner]; //SlowMist//
        } else {
            if (amount > 0) {
                rec.pendingAmount -= amount;
            }
        }
        //...
}
```

```
function slashFromUnbound(address _owner, uint _amount) private {
    //...
    if (rec.startIdx == rec.count) {
        // all cleaned
        delete unboundRecords[_owner]; //SlowMist//
    } else {
        rec.pendingAmount -= _amount;
    }
}
```

Looking into `unboundRecords` definition:

```
struct UnboundRecord {
    uint count; // total pending unbound number;
    uint startIdx; // start index of the first pending record. unless the count is zero, otherwise the startIdx will only just increase.
    uint pendingAmount; // total pending stakes
    mapping(uint => PendingUnbound) pending;
}
mapping(address => UnboundRecord) public unboundRecords;
```

Deletions are made in the structure containing the mapping `mapping(uint => PendingUnbound) pending;`, they will not be deleted.

Solution

Delete each `rec.pending` before `delete unboundRecords`.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002402270002	SlowMist Security Team	2024.02.19 - 2024.02.27	Medium Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 2 medium risk, 5 low risk, 1 suggestion vulnerabilities. And 1 medium risk vulnerabilities were confirmed and being fixed;

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