Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.02.21, the SlowMist security team received the LayerZero team's security audit application for LayerZero, 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.

Level	Description
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:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit Version

https://github.com/ryanzarick/LayerZero

commit: e4f481c18bc33e51800dd8e95110a8e71cff5c96

Fixed Version

https://github.com/LayerZero-Labs/LayerZero

commit: a5b266aa54714035314cbe5e451b2ec0db81b552

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Permission check is missing	Authority Control Vulnerability	Low	Confirmed
N2	Missing event record	Others	Suggestion	Fixed
N3	Excessive authority issues	Authority Control Vulnerability	Low	Confirmed
N4	Event capture recommendations	Malicious Event Log Audit	Suggestion	Confirmed
N5	Lack of isContract judgment	Unsafe External Call Audit	Suggestion	Confirmed
N6	Race condition issue	Race Conditions Vulnerability	Low	Confirmed

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:

ChainlinkOracleClient			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateHash	External	Can Modify State	-

ChainlinkOracleClient			
notifyOracle	External	Can Modify State	onlyULN
approveToken	External	Can Modify State	onlyOwner
withdraw	External	Can Modify State	onlyOwner nonReentrant
setUln	External	Can Modify State	onlyOwner
withdrawTokens	Public	Can Modify State	onlyOwner
withdrawOracleQuotedFee	External	Can Modify State	onlyOwner
setJob	Public	Can Modify State	onlyOwner
setPrice	External	Can Modify State	onlyOwner
setApprovedAddress	External	Can Modify State	onlyOwner
fulfillNotificationOfBlock	Public	Can Modify State	recordChainlinkFulfillment
check_utf_uint8	Internal	-	-
byte_2_utf_ints	Internal	-	-
addr2str	Public	-	-
getPrice	External	-	-
isApproved	Public	-	-
<Fallback>	External	Payable	-
<Receive Ether>	External	Payable	-

Endpoint			
Function Name	Visibility	Mutability	Modifiers

Endpoint			
<Constructor>	Public	Can Modify State	-
send	External	Payable	nonReentrant
receivePayload	External	Can Modify State	nonReentrant
retryPayload	External	Can Modify State	nonReentrant
newVersion	External	Can Modify State	onlyOwner
setDefaultSendVersion	External	Can Modify State	onlyOwner validVersion
setDefaultReceiveVersion	External	Can Modify State	onlyOwner validVersion
setConfig	External	Can Modify State	validVersion
setSendVersion	External	Can Modify State	validVersion
setReceiveVersion	External	Can Modify State	validVersion
forceResumeReceive	External	Can Modify State	-
estimateFees	External	-	-
_getSendLibrary	Internal	-	-
isValidSendLibrary	External	-	-
isValidReceiveLibrary	External	-	-
getInboundNonce	External	-	-
getOutboundNonce	External	-	-
getEndpointId	External	-	-
getSendVersion	External	-	-
getReceiveVersion	External	-	-

Endpoint

getConfig	External	-	validVersion
hasStoredPayload	External	-	-

Relayer

Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	proxied initializer
validateTransactionProofV2	External	Payable	onlyApproved nonReentrant
validateTransactionProofV1	External	Can Modify State	onlyApproved nonReentrant
setDstPrice	External	Can Modify State	onlyApproved
setDstConfig	External	Can Modify State	onlyApproved
withdrawQuotedFromULN	External	Can Modify State	onlyApproved
setApprovedAddress	Public	Can Modify State	onlyOwner
_getPrices	Internal	-	-
notifyRelayer	External	Can Modify State	-
getPrice	External	-	-
isApproved	Public	-	-

RelayerWithdraw

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
withdrawZROFromULN	External	Can Modify State	onlyApproved

RelayerWithdraw			
withdrawNativeFromULN	External	Can Modify State	onlyApproved

UltraLightNode			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
validateTransactionProof	External	Can Modify State	-
send	External	Payable	onlyEndpoint
updateHash	External	Can Modify State	-
getAppConfig	Public	-	-
setConfig	External	Can Modify State	onlyEndpoint
getConfig	External	-	-
estimateFees	External	-	-
withdrawZRO	External	Can Modify State	nonReentrant
withdrawNative	External	Can Modify State	nonReentrant
setLayerZeroToken	External	Can Modify State	onlyOwner
setTreasury	External	Can Modify State	onlyOwner
setRelayerFeeContract	External	Can Modify State	onlyOwner
addInboundProofLibraryForChain	External	Can Modify State	onlyOwner
enableSupportedOutboundProof	External	Can Modify State	onlyOwner
setDefaultConfigForChainId	External	Can Modify State	onlyOwner

UltraLightNode			
setDefaultAdapterParamsForChainId	External	Can Modify State	onlyOwner
setRemoteUln	External	Can Modify State	onlyOwner
setChainAddressSize	External	Can Modify State	onlyOwner
getBlockHeaderData	External	-	-
oracleQuotedAmount	External	-	-
relayerQuotedAmount	External	-	-
relayerProtocolNativeAmount	External	-	-
relayerProtocolZroAmount	External	-	-

Treasury			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getFees	External	-	-
setNativeFee	External	Can Modify State	onlyOwner
setZROFee	External	Can Modify State	onlyOwner
setNativeBP	External	Can Modify State	onlyOwner
setZROBP	External	Can Modify State	onlyOwner
withdrawZROFromULN	External	Can Modify State	onlyOwner
withdrawNativeFromULN	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Low] Permission check is missing

Category: Authority Control Vulnerability

Content

UltraLightNode contract's updateHash function has no permission check, But the updateHash function of the ChainlinkOracleClient contract has permission checks.

This issue won't have an impact on the other Oracle contract, but it's better to have a permission check.

- contracts/UltraLightNode.sol#L237-L247

```
    function updateHash(uint16 _srcChainId, bytes32 _lookupHash, uint _confirmations,
bytes32 _data) external override {
        // this function may revert with a default message if the oracle address is
not an ILayerZeroOracle
        BlockData storage bd = hashLookup[msg.sender][_srcChainId][_lookupHash];
        require(bd.data.length == 0 || bd.confirmations < _confirmations, "LayerZero:
oracle data can only update if it has more confirmations");

        // set the new information into storage
        bd.confirmations = _confirmations;
        bd.data = _data;

        emit HashReceived(_srcChainId, msg.sender, _confirmations, _lookupHash);
    }
```

- contracts/chainlink/ChainlinkOracleClient.sol#L53-L56

```
    function updateHash(uint16 _remoteChainId, bytes32 _blockHash, uint
_confirmations, bytes32 _data) external {
        require(approvedAddresses[msg.sender], "Oracle: caller must be approved");
        uln.updateHash(_remoteChainId, _blockHash, _confirmations, _data);
    }
```

Solution

It is recommended to add a check for permissions for the updateHash function of the UltraLightNode contract.

Status

Confirmed; The Project team response:

(1) The Oracle contract will decide its own permission control. It is external to LayerZero's design.

(2) In the proof verification process, the Ultra Light Node will lookup block data by contract's address. The submission of a rogue contract will simply not be used by any user application. Therefore Ultra Light Node does not need to authenticate the submission.

[N2] [Suggestion] Missing event record

Category: Others

Content

Owner can modify the configuration of the contract, but there is no event record, which is not conducive to community review.

- contracts/UltraLightNode.sol#L416-L462

```
    function setLayerZeroToken(address _layerZeroToken) external onlyOwner {
        require(_layerZeroToken != address(0x0), "LayerZero: _layerZeroToken cannot
be zero address");
        layerZeroToken = IERC20(_layerZeroToken);
    }

    function setTreasury(address _treasury) external onlyOwner {
        require(_treasury != address(0x0), "LayerZero: treasury cannot be zero
address");
        treasuryContract = ILayerZeroTreasury(_treasury);
    }

    function setRelayerFeeContract(address _relayerFeeContract) external onlyOwner {
        require(_relayerFeeContract != address(0x0), "LayerZero: relayerFee contract
cannot be zero address");
        relayerWithdrawContractAddress = _relayerFeeContract;
    }

    function addInboundProofLibraryForChain(uint16 _chainId, address _library)
external onlyOwner {
        require(_library != address(0x0), "LayerZero: library cannot be zero
address");
        maxInboundProofLibrary[_chainId]++;
        inboundProofLibrary[_chainId][maxInboundProofLibrary[_chainId]] = _library;
    }

    function enableSupportedOutboundProof(uint16 _chainId, uint16 _proofType)
external onlyOwner {
        supportedOutboundProof[_chainId][_proofType] = true;
    }

    function setDefaultConfigForChainId(uint16 _chainId, uint16
_inboundProofLibraryVersion, uint64 _inboundBlockConfirmations, address _relayer,
uint16 _outboundProofType, uint16 _outboundBlockConfirmations, address _oracle)
external onlyOwner {
        require(_inboundProofLibraryVersion <= maxInboundProofLibrary[_chainId] &&
_inboundProofLibraryVersion > 0, "LayerZero: invalid inbound proof library version");
        require(_inboundBlockConfirmations > 0, "LayerZero: invalid inbound block
confirmation");
        require(_relayer != address(0x0), "LayerZero: invalid relayer address");
        require(supportedOutboundProof[_chainId][_outboundProofType], "LayerZero:
invalid outbound proof type");
        require(_outboundBlockConfirmations > 0, "LayerZero: invalid outbound block
confirmation");
        require(_oracle != address(0x0), "LayerZero: invalid oracle address");
        defaultAppConfig[_chainId] =
ApplicationConfiguration(_inboundProofLibraryVersion, _inboundBlockConfirmations,
_relayer, _outboundProofType, _outboundBlockConfirmations, _oracle);
    }

    function setDefaultAdapterParamsForChainId(uint16 _chainId, uint16 _proofType,
bytes calldata _adapterParams) external onlyOwner {
        defaultAdapterParams[_chainId][_proofType] = _adapterParams;
    }

    function setRemoteUln(uint16 _remoteChainId, bytes32 _remoteUln) external
onlyOwner {
        require(ulnLookup[_remoteChainId] == bytes32(uint(uint8(0))), "LayerZero:
remote uln already set");
        ulnLookup[_remoteChainId] = _remoteUln;
    }

    function setChainAddressSize(uint16 _chainId, uint _size) external onlyOwner {
        chainAddressSizeMap[_chainId] = _size;
    }
```

- contracts/chainlink/ChainlinkOracleClient.sol#L103-L105

```
    function setUln(address ulnAddress) external onlyOwner {
        uln = ILayerZeroUltraLightNodeV1(ulnAddress);
    }
```

- contracts/chainlink/ChainlinkOracleClient.sol#L120-L131

```
    function setJob(uint16 _chain, address _oracle, bytes32 _id, uint _fee) public
onlyOwner {
        jobs[_chain] = Job(_oracle, _id, _fee, block.number - 1);
    }

    function setPrice(uint16 _destinationChainId, uint16 _outboundProofType, uint
_price) external onlyOwner {
        chainPriceLookup[_outboundProofType][_destinationChainId] = _price;
    }

    function setApprovedAddress(address _oracleAddress, bool _approve) external
onlyOwner {
        approvedAddresses[_oracleAddress] = _approve;
    }
```

Solution

It is recommended to add a record of the event to facilitate review by the community.

Status

Fixed; setApprovedAddress, setPrice, setJob, setUln have not set event record yet.Other functions have added event logging.

[N3] [Low] Excessive authority issues

Category: Authority Control Vulnerability

Content

The Owner can modify the configuration of the contract, and can modify the contract address of external calls. If the new contract has not undergone security audit, there may be potential risks.

Solution

It is recommended to set Owner address to timelock contract, governance contract, or multi-sign contract to reduce the risk of private key loss.

Status

Confirmed; The project team response: Acknowledged. LayerZero will manage it using Multi-sig and proper governance.

[N4] [Suggestion] Event capture recommendations

Category: Malicious Event Log Audit

Content

When capturing an event, pay attention to determining the attribution address of the event to prevent attackers from publishing false events through malicious contracts, resulting in false events being captured off-chain and thus suffering losses.

Solution

It is recommended to judge that the captured event is true and credible when capturing the event, rather than falsely maliciously constructed.

Status

Confirmed; The project team response: Acknowledged.

[N5] [Suggestion] Lack of isContract judgment

Category: Unsafe External Call Audit

Content

Before executing `ILayerZeroReceiver(_dstAddress).lzReceive` the code should determine that _dstAddress is a contract address, not an EOA address.

- contracts/Endpoint.sol#L82-L119

```
function receivePayload(uint16 _srcChainId, bytes calldata _srcAddress, address
_dstAddress, uint64 _nonce, uint _gasLimit, bytes calldata _payload) external
override nonReentrant {
        // authentication to prevent cross-version message validation
        // protects against a malicious library from passing arbitrary data
        LibraryConfig storage uaConfig = uaConfigLookup[_dstAddress];
        if (uaConfig.receiveVersion == DEFAULT_VERSION) {
            require(defaultReceiveLibraryAddress == msg.sender, "LayerZero: invalid
default library");
        } else {
            require(uaConfig.receiveLibraryAddress == msg.sender, "LayerZero: invalid
library");
        }

        // assert and increment the nonce. no message shuffling
        require(_nonce == ++inboundNonce[_srcChainId][_srcAddress], "LayerZero: wrong
nonce");

        try ILayerZeroReceiver(_dstAddress).lzReceive{gas: _gasLimit}(_srcChainId,
_srcAddress, _nonce, _payload) {
            // success, do nothing, end of the message delivery
        } catch {
            // revert nonce if any uncaught errors/exceptions. the nonce will block
the queue
            inboundNonce[_srcChainId][_srcAddress]--;

            storedPayload[_srcChainId][_srcAddress][_dstAddress] =
StoredPayload(_payload.length, keccak256(_payload));
            emit PayloadStored(_srcChainId, _srcAddress, _dstAddress, _payload);
        }
    }

    function retryPayload(uint16 _srcChainId, bytes calldata _srcAddress, address
_dstAddress, bytes calldata _payload) external override nonReentrant {
        StoredPayload storage sp = storedPayload[_srcChainId][_srcAddress]
[_dstAddress];
        require(sp.payloadHash != bytes32(0), "LayerZero: no stored payload");
        require(_payload.length == sp.payloadLength && keccak256(_payload) ==
sp.payloadHash, "LayerZero: invalid payload");

        // empty the storedPayload
        sp.payloadLength = 0;
        sp.payloadHash = bytes32(0);

        // call the external lzReceiver with the following nonce
        uint64 nonce = ++inboundNonce[_srcChainId][_srcAddress];
        ILayerZeroReceiver(_dstAddress).lzReceive(_srcChainId, _srcAddress, nonce,
_payload);
        emit PayloadCleared(_srcChainId, _srcAddress, _dstAddress);
    }
```

Solution

It is recommended to check whether _dstAddress is a contract address, not an EOA address.

Status

Confirmed; The project team response: It is a design tradeoff. LayerZero assumes proper configuration of user applications.

[N6] [Low] Race condition issue

Category: Race Conditions Vulnerability

Content

There is a conditional competition issue in setting the fee. Owner and onlyApproved can use transaction ordering to modify dstPriceLookup and chainPriceLookup before executing send function, which will affect the results of protocolFee and totalNativeFee calculated by send function during execution.

This may cause the actual fee paid by the _zroPaymentAddress to be inconsistent with the _zroPaymentAddress's expectations.

onlyApproved can modify dstPriceLookup.

- contracts/Relayer.sol#L73-L75

```
    function setDstPrice(uint16 _chainId, uint128 _dstPriceRatio, uint128
_dstGasPriceInWei) external onlyApproved {
        dstPriceLookup[_chainId] = DstPrice(_dstPriceRatio, _dstGasPriceInWei);
    }
```

Owner can modify chainPriceLookup.

- contracts/chainlink/ChainlinkOracleClient.sol#L123-L125

```
    function setPrice(uint16 _destinationChainId, uint16 _outboundProofType, uint
_price) external onlyOwner {
        chainPriceLookup[_outboundProofType][_destinationChainId] = _price;
    }
```

Set the price to a larger value through transaction sorting and then execute the send function, the

_zroPaymentAddress/msg.sender will pay unexpected protocolFee/totalNativeFee.

`layerZeroToken.safeTransferFrom(_zroPaymentAddress, address(this), protocolFee);`

- contracts/UltraLightNode.sol#L205-L225

```
 function send(address _ua, uint64 _nonce, uint16 _chainId, bytes calldata
_destination, bytes calldata _payload, address payable _refundAddress, address
_zroPaymentAddress, bytes calldata _adapterParams) external payable override
onlyEndpoint {
        ApplicationConfiguration memory uaConfig = getAppConfig(_chainId, _ua);
        address ua = _ua;
        uint64 nonce = _nonce;
        uint16 chainId = _chainId;
        require(ulnLookup[chainId] != bytes32(0), "LayerZero: chainId does not
exist");

        uint totalNativeFee;
        {
            uint oracleFee;
            // (a - 1), pay the oracle
            {
                oracleFee = ILayerZeroOracle(uaConfig.oracle).getPrice(chainId,
uaConfig.outboundProofType);
                oracleQuotedFees[uaConfig.oracle] =
oracleQuotedFees[uaConfig.oracle].add(oracleFee);
            }

            // (a - 2), pay the relayer
            {
                uint payloadSize = _payload.length;
                ILayerZeroRelayer relayer = ILayerZeroRelayer(uaConfig.relayer);
                if (_adapterParams.length == 0) {
                    bytes memory defaultAdaptorParam = defaultAdapterParams[chainId]
[uaConfig.outboundProofType];
                    totalNativeFee = relayer.getPrice(chainId,
uaConfig.outboundProofType, ua, payloadSize, defaultAdaptorParam);
                    relayer.notifyRelayer(chainId, uaConfig.outboundProofType,
defaultAdaptorParam);
                } else {
                    totalNativeFee = relayer.getPrice(chainId,
uaConfig.outboundProofType, ua, payloadSize, _adapterParams);
                    relayer.notifyRelayer(chainId, uaConfig.outboundProofType,
_adapterParams);
                }
                relayerQuotedFees[uaConfig.relayer] =
relayerQuotedFees[uaConfig.relayer].add(totalNativeFee); // totalNativeFee ==
relayerFee here

                // emit the param events
                emit RelayerParams(chainId, nonce, uaConfig.outboundProofType,
_adapterParams);
            }

            // (a - 3), pay the protocol
            {
                // if no ZRO token or not specifying a payment address, pay in native
token
                bool payInNative = _zroPaymentAddress == address(0x0) ||
address(layerZeroToken) == address(0x0);
                uint protocolFee = treasuryContract.getFees(!payInNative,
totalNativeFee, oracleFee); // totalNativeFee == relayerFee here

                if (protocolFee > 0) {
                    if (payInNative) {
                        treasuryNativeFees = treasuryNativeFees.add(protocolFee);
                        totalNativeFee = totalNativeFee.add(protocolFee);
                    } else {
                        // zro payment address must equal the _ua or the tx.origin
otherwise the transaction reverts
                        require(_zroPaymentAddress == ua || _zroPaymentAddress ==
tx.origin, "LayerZero: must be paid by sender or origin");

                        // transfer the LayerZero token to this contract from the
payee
                        layerZeroToken.safeTransferFrom(_zroPaymentAddress,
address(this), protocolFee);

                        treasuryZROFees = treasuryZROFees.add(protocolFee);
                    }
                }
            }

            totalNativeFee = totalNativeFee.add(oracleFee);
        }

        // (b) emit payload and the adapterParams if any
        {
            bytes memory encodedPayload = abi.encodePacked(nonce, ua, _destination,
_payload);
            emit Packet(chainId, encodedPayload);
            // (c) notify the oracle
            ILayerZeroOracle(uaConfig.oracle).notifyOracle(chainId,
uaConfig.outboundProofType, uaConfig.outboundBlockConfirmations);
        }

        require(totalNativeFee <= msg.value, "LayerZero: not enough native for
fees");
        // refund if they send too much
        uint amount = msg.value.sub(totalNativeFee);
        if (amount > 0) {
            (bool success, ) = _refundAddress.call{value: amount}("");
            require(success, "LayerZero: failed to refund");
        }
    }
```

Solution

It is recommended to add expected price management to chainPrice and dstPrice. When executing the sender function, you need to pass in the relevant price parameters to ensure that the actual price set by Owner/onlyApproved is consistent with what the user expects.

Status

Confirmed; The project team response: User Applications should make an informed decision to choose LayerZero's ecosystem partners (relayers/ oracles). We will provide analytics tools to find any misbehaviour of oracles/ relayers.

Adding an expected price would cost write/read in every single message, so we decided not to included that.

we will regulate these behaviours through markets.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002202280001	SlowMist Security Team	2022.02.21 - 2022.02.28	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 3 low risk, 3 suggestion vulnerabilities. A suggestion issue has been fixed, the other issues are confirmed. 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.