Smart Contract Security Audit Report

1. Executive Summary

On May 14, 2020, the SlowMist security team received the ForTube team's security audit application for ForTube2.0_Bond, 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.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module 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 wether there are vulnerabilities in programs suck as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk	Medium vulnerability will affect the operation of DeFi project. It is recommended

vulnerablities	to fix medium-risk vulnerabilities.
Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house 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 attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack

- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background (Context)

3.1 Project Introduction

ForTube is a crypto open financial platform developed by The Force Protocol, relying on blockchain technology to carry out innovative experiments aimed at practicing inclusive finance and providing appropriate and effective financial services to all users of the world.

Project website:

https://www.for.tube/

Audit version code:

https://github.com/thefortube/bond/tree/854527d0ea7ad2ddd3504b4d4ae3fcb57cb6445d

Fixed version code:

https://github.com/thefortube/bond/tree/f405c180c1c56c5b6282d34ee66a1446eec895c1

The documents provided by the ForTube team are as follows:

BondTokens ForTube white paper.pdf:

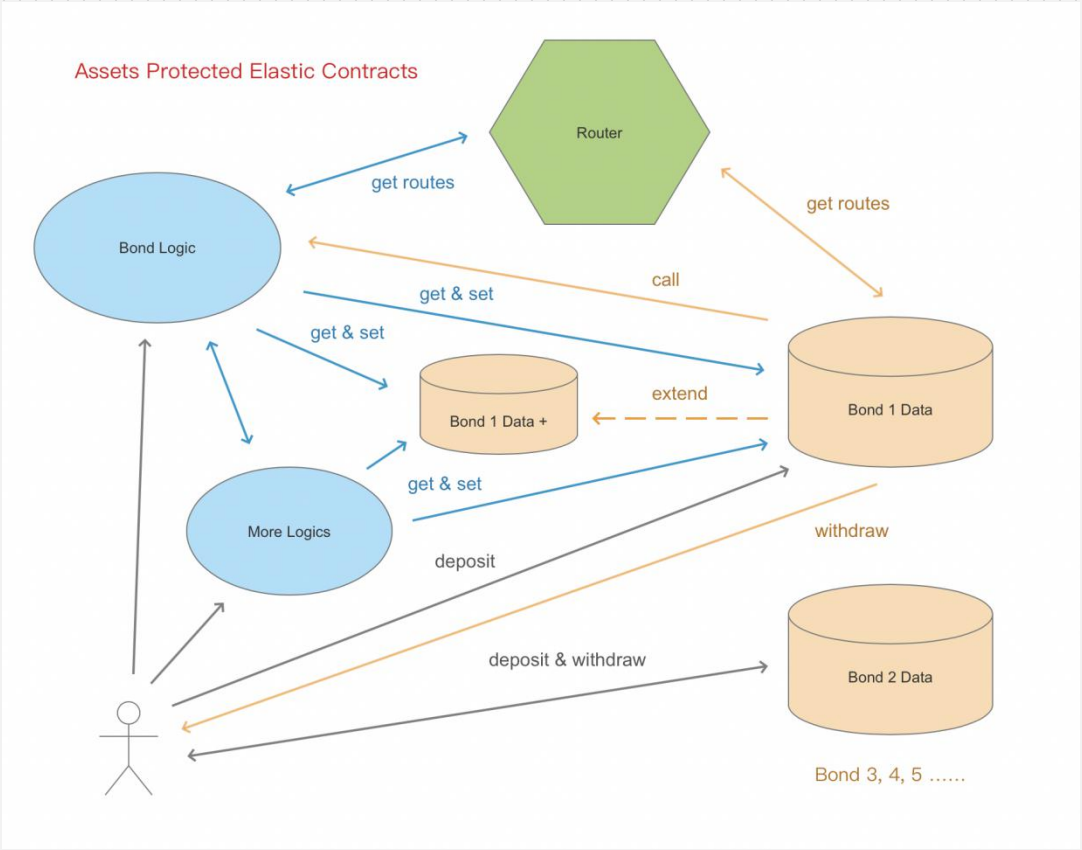
MD5: cd4385b4dd3193a935d69019493fc360

ForTube 2.0(Beta Version)user guide.pdf

MD5: 2ba33577b84895d5bed88e9a0cac1a45

Beta Website: https://beta.for.tube/bond/home

Architecture diagram:



4. Code Overview

4.1 Infrastructure

4.1.1 File Hash

NO.	File Name	SHA-1 Hash
1	bond/contracts/Core.sol	061b58e95aaf9f5bb228d185898d93d90a157323
2	bond/contracts/IRouter.sol	e04ecec4b0a57b2874c3b445a7101188b199077b
3	bond/contracts/StageDefine.sol	32c1b0234d58288919120732549ba12d658351a4
4	bond/contracts/ERC20lib.sol	7b967c3da1d259bf293dad506582a7cd4b67ee10
5	bond/contracts/IBondData.sol	9ab49fbeb7fc05a838dbc8033c6b0b9316b0ff9e
6	bond/contracts/ACL.sol	a03577aa81cf45799911c08fdf47f0e2a65302a0
7	bond/contracts/BondData.sol	20e4021cd19aefdeb1baad1c91c5dc470d8b5211
8	bond/contracts/ReentrancyGuard.sol	37f776802f7f14812d1bb9591d6ae8040ae4356b
9	bond/contracts/BondFactory.sol	b1e9b483a0b6a6d16862cf9ec585491add2f9bb6
10	bond/contracts/Config.sol	acc8fe9ce80ef30c061417bd6804c9570cb9485a
11	bond/contracts/Vote.sol	a38259ab873b2e65795de2f3a097ea7cecbd1dd4
12	bond/contracts/SafeERC20.sol	f3c73010d14659b987660919dda0011898d63387
13	bond/contracts/CoreUtils.sol	bcad885be6ff37e0a3ba73eedcd6537fd7c56131
14	bond/contracts/PRA.sol	37650e50e19a1ceeaf7e8122e5b01e8d26b730f5

15	bond/contracts/Migrations.sol	507804b63af00ee80a92a5d5bacd5c98e3e3dce8
16	bond/contracts/Rating.sol	3f2401fe3c97db2add8a7e4dfed2262967b9f80c
17	bond/contracts/SafeMath.sol	d228dfbe81530eb2399a4eec2987cfdbc8655795
18	bond/contracts/NameGen.sol	c36aec52f0d36c776c4b796a0390d5b1c4e1f93c
19	bond/contracts/Oracle.sol	3b683a4ad911890d6a5d6be48c2be2cf6aaadc99
20	bond/contracts/Router.sol	930927462eda6cb3f815adb1a7529c839559ea70
21	bond/contracts/Verify.sol	97a1965b7e192c87b62ac19412364a6c8df936ec

4.1.2 Contracts Description

Reference: Contracts Description.pdf

4.2 Code Audit

4.2.1 Variables are not checked

_owners_size is not checked, It is recommended to add require(_owners.length>=_owners_size);

- ACL.sol

```
constructor(address[] memory _owners, uint _owners_size) public {
    for (uint256 i = 0; i < _owners.length; ++i) {
        require(_add(_owners[i]), "added address is already an owner");
    }
    admin = msg.sender;
    owners_size = _owners_size;
}
```

4.2.2 Permission control defect

When the sender can pass validation if it meets one of the following four conditions, the access control policy for the other three conditions will be bypassed. Therefore, the authorized address can also be verified by auth modifier and continue to use these function to authorize other users. It is recommended to decouple this part of the code into multiple modifiers for targeted access control.

Other code that uses the auth modifier has a similar risk

- ACL.sol

```
function accessible(address sender, address to, bytes4 sig)
    public
    view
    returns (bool){
    if (msg.sender == admin) return true;
    if (_indexof(sender) != 0) return true;
    if (locked) return false;
    if (cacl[sender][to]) return true;
    if (facl[sender][to][sig]) return true;
    return false;
}
......

function unlock() external auth {
    locked = false;
}
function lock() external auth {
    locked = true;
}

......

function enable(address sender, address to, bytes4 sig) external auth {
    facl[sender][to][sig] = true;
}
function disable(address sender, address to, bytes4 sig) external auth {
    facl[sender][to][sig] = false;
```

```
}
function enableany(address sender, address to) external auth {
    cacl[sender][to] = true;
}
function enableboth(address sender, address to) external auth {
    cacl[sender][to] = true;
    cacl[to][sender] = true;
}
function disableany(address sender, address to) external auth {
    cacl[sender][to] = false;
}
```

4.2.3 Multi-Sign verification defects

multiSigSetACLs, proposeOwner, remove, updateOwnerSize functions are use mulsigauth function for multi-sign verification. But there is a Multi-Sign verification defects. There is no distinction made in the function for the purpose of executing multi-sign verification, It is recommended to add flag to each function and then compute hash.

- ACL.sol

```
function multiSigSetACLs(
        uint8[] memory v,
        bytes32[] memory r,
        bytes32[] memory s,
        address[] memory execTargets,
        address newACL) public {
        bytes32 inputHash = keccak256(abi.encode(newACL, msg.sender, nonce));
        bytes32 totalHash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", inputHash));
        mulsigauth(totalHash, v, r, s, msg.sender);
        nonce += 1;
        for (uint i = 0; i < execTargets.length; ++i) {
            IReplaceACL(execTargets[i]).setACL(newACL);
        }
    }
    function proposeOwner(
        uint8[] calldata v,
        bytes32[] calldata r,
```

```
        bytes32[] calldata s,
        address who
    ) external {
        bytes32 inputHash = keccak256(abi.encode(who, msg.sender, nonce));
        bytes32 totalHash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", inputHash));
        mulsigauth(totalHash, v, r, s, msg.sender);
        pending_owner = who;
        nonce += 1;
    }
    function remove(
        uint8[] calldata v,
        bytes32[] calldata r,
        bytes32[] calldata s,
        address who
    ) external {
        bytes32 inputHash = keccak256(abi.encode(who, msg.sender, nonce));
        bytes32 totalHash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", inputHash));
        mulsigauth(totalHash, v, r, s, msg.sender);
        require(_remove(who), "removed address is not owner");
        require(_size() >= owners_size, "invalid size and weights");
        nonce += 1;
    }
    function updateOwnerSize(
        uint8[] calldata v,
        bytes32[] calldata r,
        bytes32[] calldata s,
        uint256 _owners_size
    ) external {
        bytes32 inputHash = keccak256(abi.encode(_owners_size, msg.sender, nonce));
        bytes32 totalHash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", inputHash));
        mulsigauth(totalHash, v, r, s, msg.sender);
        nonce += 1;
        owners_size = _owners_size;
        require(_size() >= owners_size, "invalid size and weights");
    }
```

4.2.4 Validation can be bypassed

When the contract is constructing the code is null, so soaccountHash == codehash == 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470, If you want to use iscontract to determine that the address is not a contract, isContract can be bypassed.

In this case you can add：require(tx.origin == msg.sender); to fix it.

- ERC20lib.sol, SafeERC20.sol

```
function isContract(address account) internal view returns (bool) {
    // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
    // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
    // for accounts without code, i.e. `keccak256('')`
    bytes32 codehash;
    bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
    // solhint-disable-next-line no-inline-assembly
    assembly { codehash := extcodehash(account) }
    return (codehash != accountHash && codehash != 0x0);
}
```

4.2.5 Reentrancy attack risk

Used external calls before changing this contract variable "deposits[who].amount = line;", If the gov contract has been modified, The gov contract needs to be serurity audited.

- PRA.sol

```
function lock() external {
    address who = msg.sender;
    require(deposits[who].amount == 0, "sender already locked");
    require(
        IERC20(gov).allowance(who, address(this)) >= line,
        "insufficient allowance to lock"
    );
    require(
        IERC20(gov).balanceOf(who) >= line,
```

```
        "insufficient balance to lock"
    );
    deposits[who].amount = line;
    IERC20(gov).safeTransferFrom(who, address(this), line);
    emit MonitorEvent(who, address(0), "lock", abi.encodePacked(line));
}
```

The nonReentrant modifier is not used, The ICore(logic).updatebalance call has Reentrancy attack risk.

- BondData.sol

```
function transfer(address recipient, uint256 bondAmount)
        public override(IERC20, ERC20)
        returns (bool)
    {
        ICore(logic).updateBalance(id, msg.sender, recipient, bondAmount);

        ERC20.transfer(recipient, bondAmount);

        ICore(logic).MonitorEventCallback(msg.sender, address(this), "transfer", abi.encodePacked(
            recipient,
            bondAmount
        ));
        return true;
    }
function transferFrom(address sender, address recipient, uint256 bondAmount)
        public override(IERC20, ERC20)
        returns (bool)
    {
        ICore(logic).updateBalance(id, sender, recipient, bondAmount);

        ERC20.transferFrom(sender, recipient, bondAmount);

        ICore(logic).MonitorEventCallback(sender, address(this), "transferFrom", abi.encodePacked(
            recipient,
            bondAmount
        ));
```

The nonReentrant modifier is not used, The External contract call has Reentrancy attack risk.

- Vote.sol

```
function prcast(uint256 id, address proposal, uint256 reason) external {
    IBondData data = IBondData(IRouter(router).defaultDataContract(id));
    require(data.voteExpired() > now, "vote is expired");
    require(
        IPRA(PRA).raters(msg.sender),
        "sender is not a professional rater"
    );
    IBondData.prwhat memory pr = data.pr();
    require(pr.proposal == address(0), "already professional rating");
    IBondData.what memory _what = data.votes(msg.sender);
    require(_what.proposal == address(0), "already community rating");
    require(data.issuer() != msg.sender, "issuer can't vote for self bond");
    require(
        IConfig(config).ratingCandidates(proposal),
        "proposal is not permissive"
    );
    data.setPr(msg.sender, proposal, reason);
    emit MonitorEvent(
        msg.sender,
        address(data),
        "prcast",
        abi.encodePacked(proposal)
    );
}
```

4.2.6 Redundant code

Arguments in the Verify method is of type uint256[8], but the actual code uses only the values 0-4,

It is recommended to change uint256[8] to uint256[5].

- Verify.sol

```
function verify(address[2] calldata tokens, uint256[8] calldata arguments)
        external
        view
        returns (bool)
    {
        address depositToken = tokens[0];
```

```
        address issueToken = tokens[1];

        uint256 totalIssueAmount = arguments[0];
        uint256 interestRate = arguments[1];
        uint256 maturity = arguments[2];
        uint256 issueFee = arguments[3];
        uint256 minIssueRatio = arguments[4];

        IConfig _config = IConfig(config);

        return
            _config.depositTokenCandidates(depositToken) &&
            _config.issueTokenCandidates(issueToken) &&
            totalIssueAmount <= _config.maxIssueAmount(depositToken, issueToken) &&
            totalIssueAmount >= _config.minIssueAmount(depositToken, issueToken) &&
            _config.interestRateCandidates(interestRate) &&
            _config.maturityCandidates(maturity) &&
            _config.issueFeeCandidates(issueFee) &&
            _config.minIssueRatioCandidates(minIssueRatio);
    }
```

4.2.7 Event function permission control defect

This function without authentication, if the attacker keeps calling "MonitorEventCallback" function in "core.sol" and "vote. sol" to generate malicious events, the web server will check the accounts and perform global shutdown, lead to DoS.

- Core.sol, Vote.sol

```
function MonitorEventCallback(address who, address bond, bytes32 funcName, bytes calldata payload)
external {
    emit MonitorEvent(who, bond, funcName, payload);
}
```

4.2.8 Event and return values are missing

The event and return values are missing, It is recommended that you add a return value and use

event to log the execute result.

- BondData.sol

```
function setBondParam(bytes32 k, uint256 v) external auth {
    if (k == bytes32("discount")) {
        discount = v;
    }

    if (k == bytes32("liquidateLine")) {
        liquidateLine = v;
    }

    if (k == bytes32("depositMultiple")) {
        depositMultiple = v;
    }

    if (k == bytes32("gracePeriod")) {
        gracePeriod = v;
    }

    if (k == bytes32("voteExpired")) {
        voteExpired = v;
    }

    if (k == bytes32("investExpired")) {
        investExpired = v;
    }

    if (k == bytes32("bondExpired")) {
        bondExpired = v;
    }

    if (k == bytes32("partialLiquidateAmount")) {
        partialLiquidateAmount = v;
    }

    if (k == bytes32("fee")) {
        fee = v;
    }

    if (k == bytes32("sysProfit")) {
```

```
        sysProfit = v;
    }

    if (k == bytes32("originLiability")) {
        originLiability = v;
    }

    if (k == bytes32("liability")) {
        liability = v;
    }

    if (k == bytes32("totalWeights")) {
        totalWeights = v;
    }

    if (k == bytes32("totalProfits")) {
        totalProfits = v;
    }

    if (k == bytes32("borrowAmountGive")) {
        issuerBalanceGive = v;
    }

    if (k == bytes32("bondStage")) {
        bondStage = v;
    }

    if (k == bytes32("issuerStage")) {
        issuerStage = v;
    }
}
function setBondParamAddress(bytes32 k, address v) external auth {
    if (k == bytes32("gov")) {
        gov = v;
    }

    if (k == bytes32("top")) {
        top = v;
    }

}
function setBondParamMapping(bytes32 name, address k, uint256 v) external auth {
```

```
    if (name == bytes32("weights")) {
        weights[k] = v;
    }

    if (name == bytes32("profits")) {
        profits[k] = v;
    }
}
```

4.2.9 Code logic error

The event log is outside the if code, so either depositCb function return "true" or "false" will execute MonitorEventCallback, It is recommended to change "if" to "require".

- BondData.sol

```
function deposit(uint256 amount) external nonReentrant {
    if (ICore(logic).depositCb(msg.sender, id, amount)) {
        depositLedger[msg.sender] = depositLedger[msg.sender].add(amount);
        safeTransferFrom(
            collateralToken,
            msg.sender,
            address(this),
            address(this),
            amount
        );
    }

    ICore(logic).MonitorEventCallback(msg.sender, address(this), "deposit", abi.encodePacked(
        amount,
        IERC20(collateralToken).balanceOf(address(this))
    ));
}
```

4.2.10 Possible compatibility issues

BondFactory generates bond contracts based on the tokens address parameter, note the

compatibility of the ERC777.

Reference: https://mp.weixin.qq.com/s/tps3EvxyWWTLHYzxsa9ffw

- BondFactory.sol

```
function issue(
       address[2] calldata tokens,
       uint256 _minCollateralAmount,
       uint256[8] calldata info,
       bool[2] calldata _redeemPutback
   ) external returns (uint256) {
       require(IVerify(verify).verify(tokens, info), "verify error");

       uint256 nr = IRouter(router).bondNr();
       string memory bondName = INameGen(nameGen).gen(IERC20Detailed(tokens[0]).symbol(), nr);
       BondData b = new BondData(
           ACL,
           nr,
           bondName,
           msg.sender,
           tokens[0],
           tokens[1],
           info,
           _redeemPutback
       );
       IRouter(router).setDefaultContract(nr, address(b));
       IRouter(router).setBondNr(nr + 1);

       IACL(ACL).enableany(address(this), address(b));
       IACL(ACL).enableboth(core, address(b));
       IACL(ACL).enableboth(vote, address(b));

       b.setLogics(core, vote);
       IERC20(tokens[0]).safeTransferFrom(msg.sender, address(this), _minCollateralAmount);
       IERC20(tokens[0]).safeApprove(address(b), _minCollateralAmount);
       b.initialDeposit(msg.sender, _minCollateralAmount);

       return nr;
   }
```

4.2.11 Excessive auditing authority

Address which passes the auth validation can unlimited execute the "burnBond" and "mintBond" functions.

```
function burnBond(address who, uint256 amount) external auth {
        _burn(who, amount);
        actualBondIssuance = actualBondIssuance.sub(amount);
    }

    function mintBond(address who, uint256 amount) external auth {
        _mint(who, amount);
        mintCnt = mintCnt.add(amount);
        actualBondIssuance = actualBondIssuance.add(amount);
    }
```

4.2.12 Multiple Rating

Every address can change the rating of the vote by rating multiple times, and the result of the rating is subject to the last time.

Attackers can take advantage of this issues to maliciously change the rating and maliciously manipulate the operation of the project.

5. Audit Result

5.1 High-risk vulnerabilities

- Permission control defect

- Multi-Sign verification defects

  Note: It has been fixed in the fixed version code.

- Reentrancy attack risk

  Note: It has been fixed in the fixed version code.

5.2 Medium-risk Vulnerability

- Event function permission control defect

  Note: It has been fixed in the fixed version code.

- Event and return values are missing

  Note: It has been fixed in the fixed version code.

- Code logic error

  Note: It has been fixed in the fixed version code.

- Multiple Rating

- Excessive auditing authority

5.3 Low-risk Vulnerability

- Possible compatibility issues

5.4 Enhancement Suggestions

- Variables are not checked

  Note: It has been fixed in the fixed version code.

- Validation can be bypassed
- Redundant code

5.5 Conclusion

Audit Result : Some vulnerabilitys has be fixed in the fixed version code

Fixed Commit : f405c180c1c56c5b6282d34ee66a1446eec895c1

Audit Number : 0X002005310001

Audit Date : May 31, 2020

Audit Team : SlowMist Security Team

Summary conclusion: After communication and feedback with the ForTube team, The following vulnerabilities have been fixed in the fixed version code.

- Multi-Sign verification defects
- Reentrancy attack risk
- Event function permission control defect
- Event and return values are missing
- Code logic error
- Variables are not checked

After communication and feedback, the actual risk of the following issues is limited, these issues will not be fixed.

- Permission control defect

The permission management in "ACL.sol" is that the administrator and "owner" are together. Even if two "modifiers" are used separately in "ACL.sol", both modifiers need to be used all the time where they are called, so this issues will not be fixed.

- Multiple Rating

  This is in line with the original design, and users can modify the previous voting options.

- Excessive auditing authority

  If this permission is subdivided, it will increase the complexity of the system. Under the current design, even if the "Auth" private key is lost, the lost "Auth" private key can still be disabled by Multi-Sign, and this permission has no right to use the user's funds, so the user's funds are still safe.

- Possible compatibility issues

  This is currently not found to have an impact on the contract.

- Validation can be bypassed

  This should be an ID issues. When the system is affected by this kind of problem, you can disable the corresponding "validation" by Multi-Sign and restore the original logic.

- Redundant code

  The unverified parameters are unimportant parameters and have little impact on the contract. So this issue will not be 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 base 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 this 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

WeChat Official Account