Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.10.25, the SlowMist security team received the Celer Network team's security audit application for SGN, 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

Contracts for the Celer State Guardian Network (SGN) V2.

Audit Version:

https://github.com/celer-network/sgn-v2-contracts

commit: b06ee02a3487f58f7361431386b84b499e4cb334

Fixed Verson:

https://github.com/celer-network/sgn-v2-contracts

commit: a4f213f98e7f925eec80cc004aef016b9152bf45

November 30, 2021 Review Version:

https://github.com/celer-network/sgn-v2-contracts

commit: 80d6556d3a5aac105e0bed27eaaac48a6cbc4b34

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Compatibility issues	Design Logic Audit	Suggestion	Confirmed
N2	Risk of faked events	Others	Suggestion	Ignored
N3	Single sign risk	Authority Control Vulnerability	Low	Confirmed
N4	VoteOption check issue	Design Logic Audit	Suggestion	Fixed
N5	Missing event record	Others	Suggestion	Fixed
N6	TODO label issue	Others	Suggestion	Fixed
N7	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N8	bondValidator logic defect	Design Logic Audit	Medium	Fixed
N9	Gas optimization	Others	Suggestion	Ignored
N10	Lack of access control	Authority Control Vulnerability	Low	Confirmed
N11	List creation issue	Design Logic Audit	Medium	Fixed
N12	Risk of re-entrancy	Reentrancy Vulnerability	Critical	Fixed
N13	Low-level external call issue	Others	Low	Fixed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

Bridge			
Function Name	Visibility	Mutability	Modifiers
send	External	Can Modify State	nonReentrant
relay	External	Can Modify State	-
setMinSend	External	Can Modify State	onlyOwner
setMinSlippage	External	Can Modify State	onlyOwner
setWrap	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-

FarmingRewards			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
claimRewards	External	Can Modify State	whenNotPaused
contributeToRewardPool	External	Can Modify State	whenNotPaused

FarmingRewards			
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
drainToken	External	Can Modify State	whenPaused onlyOwner

Govern			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getParamProposalVote	Public	-	-
createParamProposal	External	Can Modify State	-
voteParam	External	Can Modify State	-
confirmParamProposal	External	Can Modify State	-
collectForfeiture	External	Can Modify State	-

Pool			
Function Name	Visibility	Mutability	Modifiers
addLiquidity	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	-

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

SGN			
updateSgnAddr	External	Can Modify State	-
deposit	External	Can Modify State	whenNotPaused
withdraw	External	Can Modify State	whenNotPaused
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
drainToken	External	Can Modify State	whenPaused onlyOwner

Signers			
Function Name	Visibility	Mutability	Modifiers
verifySigs	Public	-	-
updateSigners	External	Can Modify State	-
resetSigners	External	Can Modify State	onlyOwner
notifyResetSigners	External	Can Modify State	onlyOwner
increaseNoticePeriod	External	Can Modify State	onlyOwner
_verifySignedPowers	Private	-	-
_updateSigners	Private	Can Modify State	-

Staking			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-

Staking			
initializeValidator	External	Can Modify State	whenNotPaused onlyWhitelisted
updateValidatorSigner	External	Can Modify State	-
bondValidator	External	Can Modify State	-
confirmUnbondedValidator	External	Can Modify State	-
delegate	Public	Can Modify State	whenNotPaused
undelegateShares	External	Can Modify State	-
undelegateTokens	External	Can Modify State	-
completeUndelegate	External	Can Modify State	-
updateCommissionRate	External	Can Modify State	-
updateMinSelfDelegation	External	Can Modify State	-
slash	External	Can Modify State	whenNotPaused
collectForfeiture	External	Can Modify State	-
validatorNotice	External	Can Modify State	-
setParamValue	External	Can Modify State	-
setGovContract	External	Can Modify State	onlyOwner
setRewardContract	External	Can Modify State	onlyOwner
setWhitelistEnabled	External	Can Modify State	onlyOwner
addWhitelisted	External	Can Modify State	onlyOwner
removeWhitelisted	External	Can Modify State	onlyOwner
setMaxSlashFactor	External	Can Modify State	onlyOwner

Staking			
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
drainToken	External	Can Modify State	whenPaused onlyOwner
verifySignatures	Public	-	-
verifySigs	Public	-	-
getQuorumTokens	Public	-	-
getValidatorTokens	Public	-	-
getValidatorStatus	Public	-	-
isBondedValidator	Public	-	-
getValidatorNum	Public	-	-
getBondedValidatorNum	Public	-	-
getBondedValidatorsTokens	Public	-	-
hasMinRequiredTokens	Public	-	-
getDelegatorInfo	Public	-	-
getParamValue	Public	-	-
_undelegate	Private	Can Modify State	-
_setBondedValidator	Private	Can Modify State	-
_setUnbondingValidator	Private	Can Modify State	-
_bondValidator	Private	Can Modify State	-
_replaceBondedValidator	Private	Can Modify State	-

Staking			
_unbondValidator	Private	Can Modify State	-
_decentralizationCheck	Private	-	-
_tokenToShare	Private	-	-
_shareToToken	Private	-	-

StakingReward			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
claimReward	External	Can Modify State	whenNotPaused
contributeToRewardPool	External	Can Modify State	whenNotPaused
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
drainToken	External	Can Modify State	whenPaused onlyOwner

Viewer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getValidatorInfos	Public	-	-
getBondedValidatorInfos	Public	-	-
getValidatorInfo	Public	-	-
getDelegatorInfos	Public	-	-

Viewer			
getDelegatorTokens	Public	-	-
getMinValidatorTokens	Public	-	-
shouldBondValidator	Public	-	-

Whitelist			
Function Name	Visibility	Mutability	Modifiers
isWhitelisted	Public	-	-
_setWhitelistEnabled	Internal	Can Modify State	-
_addWhitelisted	Internal	Can Modify State	-
_removeWhitelisted	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Suggestion] Compatibility issues

Category: Design Logic Audit

Content

In the Bridge contract, the user can transfer the tokens that need to be cross-chain into the Bridge contract through the send function, and the Bridge contract will directly transfer the amount of tokens (_amount) specified by the user through the safeTransferFrom function, and finally record the event. If the user needs the cross-chain tokens to be deflationary tokens, then the tokens actually received by the Bridge contract do not match the transfer amount recorded in the event.

The same is true for the addLiquidity function of the Pool contract.

The same is true for the deposit function of the SGN contract.

Code location: contracts/Bridge.sol

```
    function send(
        address _receiver,
        address _token,
        uint256 _amount,
        uint64 _dstChainId,
        uint64 _nonce,
        uint32 _maxSlippage // slippage * 1M, eg. 0.5% -> 5000
    ) external nonReentrant {
        require(_amount > minSend[_token], "amount too small");
        require(_maxSlippage > mams, "max slippage too small");
        bytes32 transferId = keccak256(
            // uint64(block.chainid) for consistency as entire system uses uint64 for
chain id
            abi.encodePacked(msg.sender, _receiver, _token, _amount, _dstChainId,
_nonce, uint64(block.chainid))
        );
        require(transfers[transferId] == false, "transfer exists");
        transfers[transferId] = true;
        IERC20(_token).safeTransferFrom(msg.sender, address(this), _amount);

        emit Send(transferId, msg.sender, _receiver, _token, _amount, _dstChainId,
_nonce, _maxSlippage);
    }
```

Solution

It is recommended to record according to the difference between the contract balance before and after the user transfer.

Status

Confirmed; After communicating with the project party, the project stated that it will not support deflationary tokens.

[N2] [Suggestion] Risk of faked events

Category: Others

Content

In the brideg contract, any user can transfer any token to the brideg contract through the send function and trigger

the corresponding event. If the token transferred by the user maliciously implements the same false event as the send function, and the relay is not strictly checked, it will lead to the risk of false top-up.

The same is true for the addLiquidity function in the Pool contract.

The same is true for the deposit function in the SGN contract.

Refer to the event of pNetwork being exploited:

https://medium.com/pnetwork/pnetwork-post-mortem-pbtc-on-bsc-exploit-170890c58d5f

Solution

None.

Status

Ignored

[N3] [Low] Single sign risk

Category: Authority Control Vulnerability

Content

The verification of signatures in Bridge, FarmingRewards and Pool contracts depends on the Signers contract, but the Signers contract does not limit the power of each signer. Therefore, as long as the power of a single signer is sufficient, the signer only needs to sign the operation by itself to withdraw the funds in the contract.

Code location: contracts/Signers.sol

```
    function _verifySignedPowers(
        bytes32 _hash,
        bytes[] calldata _sigs,
        address[] calldata _signers,
        uint256[] calldata _powers
    ) private pure {
        require(_signers.length == _powers.length, "signers and powers length not
match");
        uint256 totalPower; // sum of all signer.power
        for (uint256 i = 0; i < _signers.length; i++) {
            totalPower += _powers[i];
```

```
        }
        uint256 quorum = (totalPower * 2) / 3 + 1;

        uint256 signedPower; // sum of signer powers who are in sigs
        address prev = address(0);
        uint256 index = 0;
        for (uint256 i = 0; i < _sigs.length; i++) {
            address signer = _hash.recover(_sigs[i]);
            require(signer > prev, "signers not in ascending order");
            prev = signer;
            // now find match signer add its power
            while (signer > _signers[index]) {
                index += 1;
                require(index < _signers.length, "signer not found");
            }
            if (signer == _signers[index]) {
                signedPower += _powers[index];
            }
            if (signedPower >= quorum) {
                // return early to save gas
                return;
            }
        }
        revert("quorum not reached");
    }

    function _updateSigners(address[] calldata _signers, uint256[] calldata _powers)
private {
        require(_signers.length == _powers.length, "signers and powers length not
match");
        address prev = address(0);
        for (uint256 i = 0; i < _signers.length; i++) {
            require(_signers[i] > prev, "New signers not in ascending order");
            prev = _signers[i];
        }
        ssHash = keccak256(abi.encodePacked(_signers, _powers));
        emit SignersUpdated(_signers, _powers);
    }
```

Solution

It is recommended to limit the power of a single signer or require a minimum number of signers.

Status

Confirmed; After communicating with the project party, the project party stated that the signers are updated by SGN validators to reflect the validator signer addresses and voting powers. The staking contract has decentralization checks that require no single validator to have more than 1/3 voting power.

[N4] [Suggestion] VoteOption check issue

Category: Design Logic Audit

Content

In the Govern contract, the user can vote on the proposal through the voteParam function, but it does not check whether the user's VoteOption is null.

Code location: contracts/Govern.sol

```
    function voteParam(uint256 _proposalId, VoteOption _vote) external {
        address valAddr = msg.sender;
        require(staking.getValidatorStatus(valAddr) == dt.ValidatorStatus.Bonded,
"Voter is not a bonded validator");
        ParamProposal storage p = paramProposals[_proposalId];
        require(p.status == ProposalStatus.Voting, "Invalid proposal status");
        require(block.number < p.voteDeadline, "Vote deadline passed");
        require(p.votes[valAddr] == VoteOption.Null, "Voter has voted");

        p.votes[valAddr] = _vote;

        emit VoteParam(_proposalId, valAddr, _vote);
    }
```

Solution

It is recommended that the VoteOption passed in by the user is not null.

Status

Fixed

[N5] [Suggestion] Missing event record

Category: Others

Content

In the Bridge contract, the owner can use the setMinSend, setMinSlippage and setWrap functions to set minSend, mams and nativeWrap parameters, but no event recording is performed.

In the staking contract, the owner can set the govContract, rewardContract, and MaxSlashFactor parameters through the setGovContract, setRewardContract, and setMaxSlashFactor functions, respectively, but the event record is not performed.

Code location:

contracts/Bridge.sol

```
    function setMinSend(address[] calldata tokens, uint256[] calldata minsend)
external onlyOwner {
        require(tokens.length == minsend.length, "length mismatch");
        for (uint256 i = 0; i < tokens.length; i++) {
            minSend[tokens[i]] = minsend[i];
        }
    }

    function setMinSlippage(uint32 minslip) external onlyOwner {
        mams = minslip;
    }

    function setWrap(address _weth) external onlyOwner {
        nativeWrap = _weth;
    }
```

contracts/Staking.sol

```
    function setGovContract(address _addr) external onlyOwner {
        require(govContract == address(0), "gov contract already set");
        govContract = _addr;
    }

    function setRewardContract(address _addr) external onlyOwner {
        require(rewardContract == address(0), "reward contract already set");
```

```
        rewardContract = _addr;
    }

    function setMaxSlashFactor(uint256 _maxSlashFactor) external onlyOwner {
        params[dt.ParamName.MaxSlashFactor] = _maxSlashFactor;
    }
```

Solution

It is recommended to record incidents when modifying sensitive parameters of the contract for follow-up self-examination or community review.

Status

Fixed

[N6] [Suggestion] TODO label issue

Category: Others

Content

The comment of the confirmParamProposal function in the Govern contract has a TODO tag that has not been resolved.

Code location: contracts/Govern.sol

```
    /**
     * @notice Confirm a parameter proposal
     * @param _proposalId the id of the parameter proposal
     * TODO: add latest confirm time
     */
```

Solution

It is recommended to confirm whether the implementation of the confirmParamProposal function meets the requirements.

Status

Fixed

[N7] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the Signers contract, the owner can reset the signer and power through the resetSigners function, which will lead to the risk of excessive owner authority.

Code location: contracts/Signers.sol

```
    function resetSigners(address[] calldata _signers, uint256[] calldata _powers)
external onlyOwner {
        require(block.timestamp > resetTime, "not reach reset time");
        resetTime = MAX_INT;
        _updateSigners(_signers, _powers);
    }
```

Solution

It is recommended to transfer the ownership of the owner to community governance.

Status

Confirmed; After communicating with the project party, the project party stated that they plan to start with a guarded launch mode, for which we have a hardware wallet owner account to handle emergencies if there are any. The owner role will be revoked or transferred to a governance contract when the mainnet is proven secure and stable.
However, the current contract has not been deployed to the mainnet, and the ownership of the contract has not yet been transferred to the governance contract. Therefore, there is still a risk of excessive authority.

[N8] [Medium] bondValidator logic defect

Category: Design Logic Audit

Content

In the staking contract, the bondValidator function is used for bonded validator. Among them, when

`bondedValAddrs.length <maxBondedValidators`, _bondValidator is returned directly, but the power check is not performed through the _decentralizationCheck function.

Code location: contracts/Staking.sol

```
    function bondValidator() external {
        address valAddr = msg.sender;
        if (signerVals[msg.sender] != address(0)) {
            valAddr = signerVals[msg.sender];
        }
        dt.Validator storage validator = validators[valAddr];
        require(
            validator.status == dt.ValidatorStatus.Unbonded || validator.status ==
dt.ValidatorStatus.Unbonding,
            "Invalid validator status"
        );
        require(block.number >= validator.bondBlock, "Bond block not reached");
        require(block.number >= nextBondBlock, "Too frequent validator bond");
        nextBondBlock = block.number + params[dt.ParamName.ValidatorBondInterval];
        require(hasMinRequiredTokens(valAddr, true), "Not have min tokens");

        uint256 maxBondedValidators = params[dt.ParamName.MaxBondedValidators];
        // if the number of validators has not reached the max_validator_num,
        // add validator directly
        if (bondedValAddrs.length < maxBondedValidators) {
            return _bondValidator(valAddr);
        }
        // if the number of validators has already reached the max_validator_num,
        // add validator only if its tokens is more than the current least bonded
validator tokens
        uint256 minTokens = dt.MAX_INT;
        uint256 minTokensIndex;
        for (uint256 i = 0; i < maxBondedValidators; i++) {
            if (validators[bondedValAddrs[i]].tokens < minTokens) {
                minTokensIndex = i;
                minTokens = validators[bondedValAddrs[i]].tokens;
                if (minTokens == 0) {
                    break;
                }
            }
        }
        require(validator.tokens > minTokens, "Insufficient tokens");
```

```
        _replaceBondedValidator(valAddr, minTokensIndex);
        _decentralizationCheck(validator.tokens);
    }
```

Solution

It is recommended to execute _decentralizationCheck after the _bondValidator operation.

Status

Fixed

[N9] [Suggestion] Gas optimization

Category: Others

Content

In the staking contract, the user can cancel the delegation through the undelegateShares function and the undelegateTokens function, but it does not check whether the delegate has been delegated, but relies on the delegate.shares operation of the _undelegate function to check.

Code location: contracts/Staking.sol

```
    function undelegateShares(address _valAddr, uint256 _shares) external {
        require(_shares >= dt.CELR_DECIMAL, "Minimal amount is 1 share");
        dt.Validator storage validator = validators[_valAddr];
        require(validator.status != dt.ValidatorStatus.Null, "Validator is not
initialized");
        uint256 tokens = _shareToToken(_shares, validator.tokens, validator.shares);
        _undelegate(validator, _valAddr, tokens, _shares);
    }

    function undelegateTokens(address _valAddr, uint256 _tokens) external {
        require(_tokens >= dt.CELR_DECIMAL, "Minimal amount is 1 CELR");
        dt.Validator storage validator = validators[_valAddr];
        require(validator.status != dt.ValidatorStatus.Null, "Validator is not
initialized");
        uint256 shares = _tokenToShare(_tokens, validator.tokens, validator.shares);
        _undelegate(validator, _valAddr, _tokens, shares);
    }
```

Solution

It is recommended to check whether the delegate has ever been delegated before revoking the delegation.

Status

Ignored

[N10] [Low] Lack of access control

Category: Authority Control Vulnerability

Content

In the staking contract, any user can trigger the ValidatorNotice event through the validatorNotice function, but this function annotation explains the `Validator send notice event`.

Code location: contracts/Staking.sol

```
    function validatorNotice(
        address _valAddr,
        string calldata _key,
        bytes calldata _data
    ) external {
        dt.Validator storage validator = validators[_valAddr];
        require(validator.status != dt.ValidatorStatus.Null, "Validator is not
initialized");
        emit ValidatorNotice(_valAddr, _key, _data, msg.sender);
    }
```

Solution

It is recommended to clarify the design requirements.

Status

Confirmed; After communicating with the project party, the project party stated that event listener should check the from address field.

[N11] [Medium] List creation issue

Category: Design Logic Audit

Content

In the staking contract, users can obtain ValidatorTokens information through the getBondedValidatorsTokens function, but the length of valAddrs is incorrectly used when creating the ValidatorTokens return value list.

Code location: contracts/Staking.sol

```
    function getBondedValidatorsTokens() public view returns (dt.ValidatorTokens[]
memory) {
        dt.ValidatorTokens[] memory infos = new dt.ValidatorTokens[]
(valAddrs.length);
        for (uint256 i = 0; i < bondedValAddrs.length; i++) {
            address valAddr = bondedValAddrs[i];
            infos[i] = dt.ValidatorTokens(valAddr, validators[valAddr].tokens);
        }
        return infos;
    }
```

Solution

It is recommended to use the bondedValAddrs length to create the list.

Status

Fixed

[N12] [Critical] Risk of re-entrancy

Category: Reentrancy Vulnerability

Content

In the Pool contract, executeTransfer is used to execute the transfer operation in the delayedTransfer queue. When the user's withdrawn token is a native token and `withdraws[id]` is false, the Pool contract will transfer the native

token to the target user through a low-level call. However, the Pool contract does not limit the gas usage of this low-level call. If the target user is a malicious contract address, there will be a risk of reentry.

Code location: contracts/Pool.sol

```
    function executeTransfer(bytes32 id) external whenNotPaused {
        delayedTransfer memory transfer = delayedTransfers[id];
        require(transfer.timestamp > 0, "transfer not exist");
        require(block.timestamp > transfer.timestamp + delayPeriod, "transfer still
locked");
        if (transfer.token == nativeWrap && withdraws[id] == false) {
            // withdraw then transfer native to receiver
            IWETH(nativeWrap).withdraw(transfer.amount);
            (bool sent, ) = transfer.receiver.call{value: transfer.amount}("");
            require(sent, "failed to relay native token");
        } else {
            IERC20(transfer.token).safeTransfer(transfer.receiver, transfer.amount);
        }
        delete delayedTransfers[id];
        emit TransferExecuted(id, transfer.receiver, transfer.token,
transfer.amount);
    }
```

Solution

It is recommended to follow the Checks-Effects-Interactions principle, first modify the delayedTransfers state before performing transfer operations, and limit the gas usage of low-level calls. If executeTransfer only needs to realize the transfer of tokens, then we recommend using the transfer interface instead of low-level calls.

Status

Fixed

[N13] [Low] Low-level external call issue

Category: Others

Content

In the Pool and Bridge contracts, when users withdraw tokens through the executeTransfer function and the relay function, the contract uses low-level calls and does not limit the amount of gas used to transfer tokens to the user.

Code location:

contracts/Pool.sol

```
        if (transfer.token == nativeWrap && withdraws[id] == false) {
            // withdraw then transfer native to receiver
            IWETH(nativeWrap).withdraw(transfer.amount);
            (bool sent, ) = transfer.receiver.call{value: transfer.amount}("");
            require(sent, "failed to relay native token");
        }
```

contracts/Bridge.sol

```
            if (request.token == nativeWrap) {
                // withdraw then transfer native to receiver
                IWETH(nativeWrap).withdraw(request.amount);
                (bool sent, ) = request.receiver.call{value: request.amount}("");
                require(sent, "failed to relay native token");
            }
```

Solution

When using low-level calls, it is recommended to limit the amount of gas used. Or use the transfer interface to replace low-level calls.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002111020001	SlowMist Security Team	2021.10.25 - 2021.11.02	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 critical risk, 3 medium risks, 3 low risks, 6 suggestion vulnerabilities. And 1 medium risk, 2 low risks, 1 suggestion vulnerabilities were confirmed and being fixed; 2 suggestion vulnerabilities were ignored; All other findings were fixed. The current contract has not been deployed to the mainnet, and the ownership of the contract has not yet been transferred to the governance contract. Therefore, there is still a risk of excessive authority.

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