Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.07, the SlowMist security team received the Brevis Network team's security audit application for Brevis Contracts AVS, 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 is the AVS version of Brevis Contracts, including App, Proof, and Request parts.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Refund status not checked	Design Logic Audit	Critical	Fixed
N2	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

NO	Title	Category	Level	Status
N3	Missing zero address check	Others	Suggestion	Acknowledged
N4	Submission status of _requestIds not checked	Replay Vulnerability	High	Fixed
N5	Request status cannot be updated	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/brevis-network/brevis-contracts-avs

Initial audit commit: aeee8abc69bede97ee8a282b7bef5144055de164

Final audit commit: c6c07f8198cb0903bfd96e45cece45848811a57d

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:

BrevisRequest			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	FeeVault
setBrevisEigen	External	Can Modify State	onlyOwner
sendRequest	External	Payable	-
fulfillRequest	External	Can Modify State	-
refund	Public	Can Modify State	-

BrevisRequest			
setRequestTimeout	External	Can Modify State	onlyOwner
setChallengeWindow	External	Can Modify State	onlyOwner
setResponseTimeout	External	Can Modify State	onlyOwner
queryRequestStatus	External	-	-
fulfillOpRequests	External	Can Modify State	-
askForQueryData	External	Payable	-
postQueryData	External	-	-
challengeQueryData	External	-	-
askForProof	External	Payable	-
postProof	External	-	-

BrevisApp			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
brevisCallback	External	Can Modify State	-
handleProofResult	Internal	Can Modify State	-
validateOpRequest	Public	-	-

BrevisProof			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
submitProof	External	Can Modify State	-
submitOpResult	External	Can Modify State	onlyBrevisRequest

BrevisProof			
validateOpRequest	External	-	-
hasProof	External	-	-
getProofAppData	External	-	-
verifyRaw	Private	-	-
unpackProofData	Internal	-	-
updateVerifierAddress	Public	Can Modify State	onlyOwner
updateSmtContract	Public	Can Modify State	onlyOwner
updateBrevisRequest	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Critical] Refund status not checked

Category: Design Logic Audit

Content

In the BrevisRequest contract, the `refund` function does not check the refund status of the specified `_requestId`.

Users can call this function repeatedly to perform refund operations until the balance in the contract reaches 0.

- BrevisRequest.sol#L120-L134

```
    function refund(bytes32 _requestId) public {
        require(block.timestamp > requests[_requestId].deadline);
        require(
            !IBrevisProof(brevisProof).hasProof(_requestId),
            "proof already generated"
        );
        requests[_requestId].deadline = 0; //reset deadline, then user is able to
sent request again
        (bool sent, ) = requests[_requestId].refundee.call{
            value: requests[_requestId].fee,
            gas: 50000
        }("");
        require(sent, "send native failed");
        requests[_requestId].status = RequestStatus.Refunded;
```

```
        emit RequestRefunded(_requestId);
    }
```

Solution

It is recommended to check whether the RequestStatus of the specified requestId is Refunded, and report an error if so.

Status

Fixed

[N2] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the BrevisRequest contract, the `owner` role can set important parameters in the contract.

- BrevisRequest.sol

```
function setBrevisEigen
function setRequestTimeout
function setChallengeWindow
function setResponseTimeout
```

2.In the BrevisProof contract, the `owner` role can set important parameters in the contract.

- BrevisProof.sol

```
function updateVerifierAddress
function updateSmtContract
function updateBrevisRequest
```

3.In the FeeVault contract, the `owner` role can set the `feeCollector` role in the contract through the `setFeeCollector` function. The `feeCollector` role can withdraw the balance in the contract through the `collectFee` function.

- FeeVault.sol#L28-L34

```
    function collectFee(
        uint256 _amount,
```

```
        address _to
    ) external onlyFeeCollector {
        (bool sent, ) = _to.call{value: _amount, gas: 50000}("");
        require(sent, "send native failed");
    }

    function setFeeCollector(address _feeCollector) external onlyOwner {
        address oldFeeCollector = feeCollector;
        feeCollector = _feeCollector;
        emit FeeCollectorUpdated(oldFeeCollector, _feeCollector);
    }
```

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. 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

[N3] [Suggestion] Missing zero address check

Category: Others

Content

1.In the BrevisRequest contract, the `constructor` function and `setBrevisEigen` function lack zero address check.

- BrevisRequest.sol#L52-L59,L61-L63

```
    constructor(
        address _feeCollector,
        IBrevisProof _brevisProof,
        IBrevisEigen _brevisEigen
    ) FeeVault(_feeCollector) {
        brevisProof = _brevisProof;
        brevisEigen = _brevisEigen;
    }

    function setBrevisEigen(IBrevisEigen _brevisEigen) external onlyOwner {
```

```
        brevisEigen = _brevisEigen;
    }
```

2.In the BrevisProof contract, the `updateSmtContract` function and `updateBrevisRequest` function lack zero address check.

- BrevisProof.sol#L192-L195,L197-L200

```
    function updateSmtContract(ISMT _smtContract) public onlyOwner {
        smtContract = _smtContract;
        emit SmtContractUpdated(smtContract);
    }

    function updateBrevisRequest(address _brevisRequest) public onlyOwner {
        brevisRequest = _brevisRequest;
        emit BrevisRequestUpdated(_brevisRequest);
    }
```

3.In the BrevisApp abstract contract, the `constructor` function lack zero address check.

- BrevisApp.sol#L9-L11

```
    constructor(IBrevisProof _brevisProof) {
        brevisProof = _brevisProof;
    }
```

4.In the FeeVault contract, the `constructor` function , `collectFee` function and `setFeeCollector` function lack zero address check.

- FeeVault.sol#L19-L21,L28-L34,L36-L40

```
    constructor(address _feeCollector) {
        feeCollector = _feeCollector;
    }

    function collectFee(
        uint256 _amount,
        address _to
    ) external onlyFeeCollector {
        (bool sent, ) = _to.call{value: _amount, gas: 50000}("");
        require(sent, "send native failed");
    }
```

```
function setFeeCollector(address _feeCollector) external onlyOwner {
    address oldFeeCollector = feeCollector;
    feeCollector = _feeCollector;
    emit FeeCollectorUpdated(oldFeeCollector, _feeCollector);
}
```

Solution

It is recommended to add zero address check.

Status

Acknowledged

[N4] [High] Submission status of _requestIds not checked

Category: Replay Vulnerability

Content

In the BrevisRequest contract, the `fulfillOpRequests` function does not check whether the `_requestIds` parameter has been submitted. An attacker can pass in `_requestIds` that have been verified by the `mustVerified` function of the brevisEigen contract, and call this function repeatedly to reset the challenge time of the `_requestIds`, so that it remains in the `challengeWindow` and cannot be finally confirmed. It is also possible to pass in the same `_requestIds` and different `_queryURLs`, causing errors to be logged by the `OpRequestsFulfilled` event.

- BrevisRequest.sol#L199-L218

```
function fulfillOpRequests(
    bytes32[] calldata _requestIds,
    bytes[] calldata _queryURLs
) external {
    require(_requestIds.length > 0, "invalid requestIds");
    require(_requestIds.length == _queryURLs.length);

    // must already verified in brevisEigen
    brevisEigen.mustVerified(_requestIds);

    for (uint i = 0; i < _requestIds.length; i++) {
        brevisProof.submitOpResult(_requestIds[i]);
        requests[_requestIds[i]].status = RequestStatus.OpSubmitted;
        requestExts[_requestIds[i]].canChallengeBefore =
            block.timestamp +
```

```
                challengeWindow;
        }

        emit OpRequestsFulfilled(_requestIds, _queryURLs);
    }
```

Solution

It is recommended to check whether the _requestIds parameter has been submitted, and if it has been submitted, roll back the function.

Status

Fixed

[N5] [Low] Request status cannot be updated

Category: Design Logic Audit

Content

When the user does not call the `fulfillRequest` function of the BrevisRequest contract, but first calls the `submitProof` function of the BrevisProof contract to submit and verify the proof, the `fulfillRequest` function cannot be called again, resulting in `requests[_requestId].status` unable to be updated.

- BrevisRequest.sol#L85-L118

```
    function fulfillRequest(
        bytes32 _requestId,
        uint64 _chainId,
        bytes calldata _proof,
        bytes calldata _appCircuitOutput
    ) external {
        require(
            !IBrevisProof(brevisProof).hasProof(_requestId),
            "proof already generated"
        );

        bytes32 reqIdFromProof = IBrevisProof(brevisProof).submitProof(
            _chainId,
            _proof
        ); // will be reverted when proof is not valid
        require(_requestId == reqIdFromProof, "requestId and proof not match");
        requests[_requestId].status = RequestStatus.ZkAttested;

        emit RequestFulfilled(_requestId);
```

```
        address app = address(requests[_requestId].callback);
        if (app != address(0)) {
            // No matter if the call is success or not. The relayer should set correct
gas limit.
            // If the call exceeds the gasleft(), as the proof data is saved ahead,
            // anyone can still call the app.callback directly to proceed
            app.call(
                abi.encodeWithSelector(
                    IBrevisApp.brevisCallback.selector,
                    _requestId,
                    _appCircuitOutput
                )
            );
        }
    }
```

- BrevisProof.sol#L29-L42

```
    function submitProof(
        uint64 _chainId,
        bytes calldata _proofWithPubInputs
    ) external returns (bytes32 _requestId) {
        require(verifyRaw(_chainId, _proofWithPubInputs), "proof not valid");
        Brevis.ProofData memory data = unpackProofData(_proofWithPubInputs);
        _requestId = data.commitHash;
        require(
            smtContract.isSmtRootValid(_chainId, data.smtRoot),
            "smt root not valid"
        );
        proofs[_requestId].appCommitHash = data.appCommitHash; // save necessary
fields only, to save gas
        proofs[_requestId].appVkHash = data.appVkHash;
    }
```

Solution

It is recommended to add the onlyBrevisRequest modifier to the submitProof function of the BrevisProof contract, so that this function can only be called by the BrevisRequest contract.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404090001	SlowMist Security Team	2024.04.07 - 2024.04.09	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, 1 high risk, 1 medium risk,1 low risk, 1 suggestion. All findings have been fixed or acknowledged. The code was not deployed to the mainnet.

6 Statement

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

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

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist