Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.12.19, the SlowMist security team received the Brevis Network team's security audit application for Brevis Contracts, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Preemptive Initialization	Race Conditions Vulnerability	Suggestion	Acknowledged
N2	Gas limit reentrancy risk	Reentrancy Vulnerability	Low	Acknowledged
N3	Missing event record	Malicious Event Log Audit	Suggestion	Acknowledged

NO	Title	Category	Level	Status
N4	Information about applyBrevisProof	Others	Information	Acknowledged
N5	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N6	The status is not set correctly	Others	High	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/brevis-network/vip-hook

Commit: 63e0e94363f7cbeb9357048d55ed15281a3c9e9f

Review Commit: 80372023148b037d904f81803d9cda1466d7f5e1

Review Commit: 059ef6c3a36982066e33001bef9b67269bf4c18f

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

Commit: e3f11190a54ca68c248519393cdcf3bba61d03c1

Review Commit: 9f1dd2e86ffd5ceffa5d1e3156484ff11efdd575

```
vip-hook/src
├── BrevisApp.sol
├── BrevisFeeVault.sol
├── Ownable.sol
├── VipDiscountMap.sol
├── pool-bin
│   └── BinVipHook.sol
└── pool-cl
    └── CLVipHook.sol

brevis-contracts/contracts
├── sdk
│   └── core
│       ├── BrevisProof.sol
│       └── BrevisRequest.sol
```

```
└── smt
    └── SMT.sol
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

SMT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getLatestRoot	Public	-	-
isSmtRootValid	Public	-	-
updateRoot	External	Can Modify State	onlyActiveProver
verifyProof	Private	-	-
setAnchorProvider	External	Can Modify State	onlyOwner
setVerifier	External	Can Modify State	onlyOwner
setCircuitDigest	External	Can Modify State	onlyOwner

BrevisProof			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BrevisAggProof
init	External	Can Modify State	-
submitProof	External	Can Modify State	onlyActiveProver
validateProofAppData	External	-	-
updateVerifierAddress	Public	Can Modify State	onlyOwner

BrevisProof			
unpackProofData	Internal	-	-
verifyRaw	Private	-	-

BrevisRequest			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	FeeVault
init	External	Can Modify State	-
sendRequest	External	Payable	-
fulfillRequest	External	Can Modify State	onlyActiveProver
fulfillRequests	External	Can Modify State	onlyActiveProver
increaseGasFee	External	Payable	-
refund	External	Can Modify State	-
fulfillOpRequests	External	Can Modify State	onlyActiveProver
setRequestStatus	External	Can Modify State	-
applyBrevisProof	External	Can Modify State	-
applyBrevisAggProof	External	Can Modify State	-
applyBrevisAggProof	External	Can Modify State	-
setRequestTimeout	External	Can Modify State	onlyOwner
setBaseDataURL	External	Can Modify State	onlyOwner
setBrevisProof	External	Can Modify State	onlyOwner
setBrevisDispute	External	Can Modify State	onlyOwner
setBvnSigsVerifier	External	Can Modify State	onlyOwner
setAvsSigsVerifier	External	Can Modify State	onlyOwner

BrevisRequest			
queryRequestStatus	External	-	-
validateOpAppData	External	-	-
dataURL	External	-	-
_brevisCallback	Private	Can Modify State	-
_submitOpStates	Private	Can Modify State	-
_queryRequestStatus	Private	-	-
_validateOpAppData	Private	-	-
_bitSet	Private	-	-

BrevisApp			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
handleProofResult	Internal	Can Modify State	-
handleOpProofResult	Internal	Can Modify State	-
_setOpChallengeWindow	Internal	Can Modify State	-
_setBrevisRequest	Internal	Can Modify State	-
brevisCallback	External	Can Modify State	onlyBrevisRequest
brevisBatchCallback	External	Can Modify State	onlyBrevisRequest
applyBrevisOpResult	Public	Can Modify State	-
applyBrevisOpResults	External	Can Modify State	-

BrevisFee			
Function Name	Visibility	Mutability	Modifiers

BrevisFee			
fund	External	Payable	-
collect	External	Can Modify State	onlyOwner
collectAll	External	Can Modify State	onlyOwner

Ownable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initOwner	Internal	Can Modify State	-
owner	Public	-	-
transferOwnership	Public	Can Modify State	onlyOwner
_setOwner	Private	Can Modify State	-

VipDiscountMap			
Function Name	Visibility	Mutability	Modifiers
updateBatch	Internal	Can Modify State	-
getFee	Public	-	-

BinVipHook			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BinBaseHook BrevisApp
init	External	Can Modify State	-
getHooksRegistrationBitmap	External	-	-
beforeSwap	External	Can Modify State	poolManagerOnly
handleProofResult	Internal	Can Modify State	-

BinVipHook			
setFee	External	Can Modify State	onlyOwner
setVkHash	External	Can Modify State	onlyOwner
setBrevisRequest	External	Can Modify State	onlyOwner

CLVipHook			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	CLBaseHook BrevisApp
init	External	Can Modify State	-
getHooksRegistrationBitmap	External	-	-
beforeSwap	External	Can Modify State	poolManagerOnly
handleProofResult	Internal	Can Modify State	-
setFee	External	Can Modify State	onlyOwner
setVkHash	External	Can Modify State	onlyOwner
setBrevisRequest	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Suggestion] Preemptive Initialization

Category: Race Conditions Vulnerability

Content

By calling the init function to initialize the contract, there is a potential issue that malicious attackers preemptively call the init function to initialize.

- contracts/sdk/core/BrevisProof.sol

```
function init(ISMT _smtContract) external {
    initOwner();
```

```
    smtContract = _smtContract;
}
```

- contracts/sdk/core/BrevisRequest.sol

```
function init(
    address _feeCollector,
    IBrevisProof _brevisProof,
    IBvnSigsVerifier _bvnSigsVerifier,
    uint256 _requestTimeout
) external {
    initOwner();
    feeCollector = _feeCollector;
    brevisProof = _brevisProof;
    bvnSigsVerifier = _bvnSigsVerifier;
    requestTimeout = _requestTimeout;
}
```

- src/pool-cl/CLVipHook.sol

```
function init(
    uint24 _origFee,
    address _brevisRequest,
    bytes32 _vkHash
) external {
    initOwner();
    _setBrevisRequest(_brevisRequest);
    origFee = _origFee;
    vkHash = _vkHash;
}
```

- src/pool-bin/BinVipHook.sol

```
function init(
    uint24 _origFee,
    address _brevisRequest,
    bytes32 _vkHash
) external {
    initOwner();
    _setBrevisRequest(_brevisRequest);
    origFee = _origFee;
    vkHash = _vkHash;
}
```

Solution

It is suggested that the initialization operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Acknowledged

[N2] [Low] Gas limit reentrancy risk

Category: Reentrancy Vulnerability

Content

Preventing reentrancy attacks by merely setting a gas limit of 50000 during refund operations is not a robust solution. The gas consumption patterns may change due to future smart contract upgrades or Ethereum protocol modifications, potentially invalidating the preset gas limit and reintroducing reentrancy vulnerabilities.

- contracts/sdk/core/BrevisRequest.sol

```
function refund(
    bytes32 _proofId,
    uint64 _nonce,
    uint256 _amount,
    address _refundee
) external {
    bytes32 requestKey = keccak256(abi.encodePacked(_proofId, _nonce));
    Request memory request = requests[requestKey];
    RequestStatus status = request.status;
    require(status == RequestStatus.ZkPending || status == RequestStatus.OpPending,
"invalid status");
    require(block.timestamp > request.timestamp + requestTimeout);

    bytes32 feeHash = onchainRequests[requestKey].feeHash;
    require(feeHash == keccak256(abi.encodePacked(_amount, _refundee)), "invalid
input");
    //@SlowMist
    (bool sent, ) = _refundee.call{value: _amount, gas: 50000}("");
    require(sent, "send native failed");
    requests[requestKey].status = RequestStatus.Refunded;
    emit RequestRefunded(_proofId, _nonce);
}
```

Solution

It is recommended to use a reentrancy guard to prevent reentrancy attacks.

Status

Acknowledged

[N3] [Suggestion] Missing event record

Category: Malicious Event Log Audit

Content

The changes to the following key parameters have not been logged with corresponding events.

- src/BrevisApp.sol

```
function _setOpChallengeWindow(uint256 _challangeWindow) internal {
    opChallengeWindow = _challangeWindow;
}

function _setBrevisRequest(address _brevisRequest) internal {
    brevisRequest = _brevisRequest;
}
```

Solution

Record the corresponding event.

Status

Acknowledged

[N4] [Information] Information about applyBrevisProof

Category: Others

Content

The applyBrevisProof function is designed to be publicly callable with an externally specified _callbackTarget. This open design aligns with its purpose, where BrevisApp as the callback target can implement its own verification logic to decide whether to accept the proof.

- contracts/sdk/core/BrevisRequest.sol

```
function applyBrevisProof(
    bytes32 _proofId,
    bytes32 _appVkHash,
    bytes32 _appCommitHash,
    bytes calldata _appCircuitOutput,
    address _callbackTarget
) external {

    brevisProof.validateProofAppData(_proofId, _appCommitHash, _appVkHash);

    require(_appCommitHash == keccak256(_appCircuitOutput), "invalid circuit
output");
    IBrevisApp(_callbackTarget).brevisCallback(_appVkHash, _appCircuitOutput);
}
```

Solution

Status

Acknowledged

[N5] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

Owner of the contract can set some key parameters. If the owner private key is leaked, it will have a serious impact on the functionality of the entire contract.

- contracts/smt/SMT.sol

```
Owner can setAnchorProvider
Owner can setVerifier
Owner can setCircuitDigest
ActiveProver can updateRoot
```

- contracts/sdk/core/BrevisProof.sol

```
ActiveProver can submitProof
Owner can updateVerifierAddress
```

- contracts/sdk/core/BrevisRequest.sol

```
ActiveProver can fulfillRequest
ActiveProver can fulfillRequests
ActiveProver can fulfillOpRequests
Owner can setRequestTimeout
Owner can setBaseDataURL
Owner can setBrevisProof
Owner can setBrevisDispute
Owner can setBvnSigsVerifier
Owner can setAvsSigsVerifier
brevisDispute can setRequestStatus
```

- src/pool-bin/BinVipHook.sol

```
Owner can setFee
Owner can setVkHash
Owner can setBrevisRequest
```

- src/pool-cl/CLVipHook.sol

```
Owner can setFee
Owner can setVkHash
Owner can setBrevisRequest
```

Solution

In the short term, during the early stages of the project, the protocol may need to frequently set various parameters to ensure the stable operation of the protocol. Therefore, transferring the ownership of core roles to a multisig management can effectively solve the single-point risk, but it cannot mitigate the excessive privilege risk. In the long run, after the protocol stabilizes, transferring the owner ownership to community governance and executing through a timelock can effectively mitigate the excessive privilege risk and increase the community users' trust in the protocol.

Status

Acknowledged

[N6] [High] The status is not set correctly

Category: Others

Content

When _option == 0, status is not set to RequestStatus.ZkPending, which will cause the status to be incorrect.

- contracts/sdk/core/BrevisRequest.sol

```
function sendRequest(
        bytes32 _proofId,
        uint64 _nonce,
        address _refundee,
        Callback calldata _callback,
        uint8 _option
    ) external payable {

        ...

        if (_option == 0) {
            status == RequestStatus.ZkPending;
        } else {
            status = RequestStatus.OpPending;
        }
        ...
    }
```

Solution

Set the status correctly.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002412250001	SlowMist Security Team	2024.12.19 - 2024.12.25	Medium Risk

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

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