Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.11.11, the SlowMist security team received the Brevis Network team's security audit application for Brevis Prover Network 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

Smart contracts for Brevis Prover Network.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	There is no reasonable range for parameters without limitations.	Others	Low	Acknowledged

NO	Title	Category	Level	Status
N2	StakingController reinitialize risk	Design Logic Audit	Suggestion	Acknowledged
N3	retireProver lacks permission control	Authority Control Vulnerability Audit	Medium	Fixed
N4	retireProver commission loss risk	Design Logic Audit	Critical	Fixed
N5	Structural data residue risk	Variable Coverage Vulnerability	High	Fixed
N6	Dust assets cause retireProver to be permanently blocked.	Integer Overflow and Underflow Vulnerability	Medium	Fixed
N7	ERC4626 share inflation risk	Design Logic Audit	Medium	Fixed
N8	addRewards does not align with the actual amount credited.	Design Logic Audit	Low	Acknowledged
N9	Instantaneous manipulation of commission rates	Design Logic Audit	Low	Acknowledged
N10	Incorrect asset balance logic	Design Logic Audit	Medium	Acknowledged
N11	Unchecked return value	Others	Low	Fixed
N12	Unprotected Initializer	Authority Control Vulnerability Audit	Low	Acknowledged
N13	Prover status verification and management are incorrect.	Design Logic Audit	Low	Acknowledged
N14	_getEffectiveProver intent and implementation discrepancy	Design Logic Audit	Low	Acknowledged
N15	ReqId Unbound Requester Caused Request Hijacking and Rushing	Design Logic Audit	Low	Acknowledged

NO	Title	Category	Level	Status
N16	Anti-reentry coding recommendation	Reentrancy Vulnerability	Suggestion	Acknowledged
N17	Residual Infinite Authorization	Authority Control Vulnerability Audit	Suggestion	Acknowledged
N18	Unnecessary state variables causing storage expansion	Others	Suggestion	Acknowledged
N19	Bid fee exposure risk	Others	Low	Acknowledged
N20	Excessive Authority Risk	Authority Control Vulnerability Audit	Medium	Acknowledged
N21	ZK proof reuse risk	Replay Vulnerability	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

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

Initial audit commit: b5aa4a96bd5acdad387ae4a17620ddba147f9a7d

Final audit commit: 73d1637447e7f90ed42665747a0455f1ee450dae

Audit Scope:

```
./src
├── market
│   ├── BrevisMarket.sol
│   ├── ProverSubmitters.sol
│   └── interfaces
│       └── IBrevisMarket.sol
└── staking
    ├── controller
    │   ├── PendingUnstakes.sol
    │   └── StakingController.sol
    ├── interfaces
    │   ├── IProverVault.sol
    │   ├── IStakingController.sol
    │   └── IVaultFactory.sol
```

```
└── vault
    ├── ProverVault.sol
    └── VaultFactory.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:

BrevisMarket			
Function Name	Visibility	Mutability	Modifiers
constructor	public	-	-
init	external	-	-
_init	internal	-	-
requestProof	external	-	-
bid	external	-	-
reveal	external	-	-
submitProof	external	-	nonReentrant
refund	public	-	nonReentrant
batchRefund	external	-	-
slash	external	-	-
setPicoVerifier	external	-	onlyOwner
setBiddingPhaseDuration	external	-	onlyOwner
setRevealPhaseDuration	external	-	onlyOwner
setMinMaxFee	external	-	onlyOwner
setMaxMaxFee	external	-	onlyOwner

BrevisMarket			
setSlashBps	external	-	onlyOwner
setSlashWindow	external	-	onlyOwner
setProtocolFeeBps	external	-	onlyOwner
setOvercommitBps	external	-	onlyOwner
withdrawProtocolFee	external	-	onlyOwner
getRequest	external	view	-
getBidders	external	view	-
getProof	external	view	-
getBidHash	external	view	-
getProtocolFeeInfo	external	view	-
getProverPendingRequests	external	view	-
getSenderPendingRequests	external	view	-
_updateBidders	internal	-	-
_requireProverEligible	internal	view	-
_reassignObligation	internal	-	-
_releaseObligation	internal	-	-
scheduleStatsEpoch	external	-	onlyOwner
popStatsEpoch	external	-	onlyOwner
getProverStatsTotal	external	view	-
getProverRecentStats	external	view	-
getRecentStatsInfo	external	view	-
getProverStatsForStatsEpoch	external	view	-

BrevisMarket			
getGlobalStatsTotal	external	view	-
getGlobalRecentStats	external	view	-
getGlobalStatsForStatsEpoch	external	view	-
statsEpochsLength	external	view	-
_syncStatsEpochId	internal	-	-
_currentCumulativeStats	internal	-	-
_zeroStats	internal	pure	-
_diffStats	internal	pure	-
_currentCumulativeGlobalStats	internal	-	-
_zeroGlobalStats	internal	pure	-
_diffGlobalStats	internal	pure	-

ProverSubmitters			
Function Name	Visibility	Mutability	Modifiers
setSubmitterConsent(address)	external	-	-
registerSubmitter(address)	public	-	-
registerSubmitters(address[])	external	-	-
unregisterSubmitter(address)	public	-	-
unregisterSubmitters(address[])	external	-	-
unregisterSubmitter()	external	-	-
_removeSubmitter(address,address)	internal	-	-
getSubmittersForProver(address)	external	view	-
_getEffectiveProver(address)	internal	view	-

PendingUnstakes			
Function Name	Visibility	Mutability	Modifiers
_receiveUnstake(address,address,uint256)	internal	-	(none)
_completeUnstake(address)	internal	-	(none)
_slashUnstaking(address,uint256)	internal	-	(none)
_calculateEffectiveAmount(uint256,uint256,uint256)	internal	pure	(none)
getPendingUnstakes(address,address)	external	view	(none)
getUnstakingInfo(address,address)	external	view	(none)
getProverTotalUnstaking(address)	external	view	(none)
getProverSlashingScale(address)	external	view	(none)
getStakersWithPendingUnstakes(address)	external	view	(none)
getStakersWithPendingUnstakesCount(address)	external	view	(none)
stakerHasPendingUnstakes(address,address)	external	view	(none)

StakingController			
Function Name	Visibility	Mutability	Modifiers
constructor(address,address,uint256,uint256,uint256)	public	-	-
init(address,address,uint256,uint256,uint256)	external	-	-
_init(address,address,uint256,uint256,uint256)	internal	-	-
initializeProver(uint64)	external	-	onlyAuthorized whenNotPaused
deactivateProver(address)	external	-	onlyOwner
deactivateProvers(address[])	external	-	onlyOwner
reactivateProver(address)	public	-	-

StakingController			
reactivateProvers(address[])	external	-	-
jailProver(address)	external	-	onlyOwner
jailProvers(address[])	external	-	onlyOwner
retireProver(address)	public	-	-
retireProvers(address[])	external	-	onlyOwner
setProverProfile(string,string)	external	-	-
setProverProfileByAdmin(address,string,string)	external	-	onlyOwner
_setProverProfile(address,string,string)	internal	-	-
stake(address,uint256)	public	-	whenNotPaused nonReentrant
requestUnstake(address,uint256)	external	-	nonReentrant
completeUnstake(address)	external	-	whenNotPaused nonReentrant
addRewards(address,uint256)	external	-	whenNotPaused nonReentrant
claimCommission()	external	-	whenNotPaused nonReentrant
setCommissionRate(address,uint64)	external	-	-
resetCommissionRate(address)	external	-	-
slash(address,uint256)	external	-	whenNotPaused nonReentrant onlyRole(SLASHER_ROLE)
slashByAmount(address,uint256)	external	-	whenNotPaused nonReentrant onlyRole(SLASHER_ROLE)
_executeSlash(address,uint256)	internal	-	-
maxWithdraw(address,address)	external	view	-
maxRedeem(address,address)	external	view	-

StakingController			
maxDeposit(address,address)	external	view	-
maxMint(address,address)	external	view	-
beforeShareTransfer(address,address,uint256)	external	view	-
onShareTransfer(address,address,address,uint256)	external	-	-
getProverInfo(address)	external	view	-
getProverStakers(address)	external	view	override
getStakeInfo(address,address)	external	view	override
isProverEligible(address,uint256)	external	view	override
getProverState(address)	external	view	override
getProverVault(address)	public	view	override
getProverProfile(address)	external	view	override
getAllProvers()	external	view	-
getActiveProvers()	external	view	-
getProverCount()	external	view	-
getActiveProverCount()	external	view	-
getTotalVaultAssets()	external	view	-
getTotalActiveProverVaultAssets()	external	view	-
getCommissionRate(address,address)	public	view	override
getCommissionRates(address)	external	view	override
getProverTotalAssets(address)	public	view	override
_changeProverState(address,ProverState)	internal	-	-

StakingController			
_isActive(address)	internal	view	-
setMinSelfStake(uint256)	external	-	onlyOwner
setMaxSlashBps(uint256)	external	-	onlyOwner
setUnstakeDelay(uint256)	external	-	onlyOwner
setRequireAuthorization(bool)	external	-	onlyOwner
withdrawTreasury(address,uint256)	external	-	whenNotPaused onlyOwner
emergencyRecover(address,uint256)	external	-	whenPaused onlyOwner

ProverVault			
Function Name	Visibility	Mutability	Modifiers
constructor(IERC20,string,string,address,address)	public	-	-
maxWithdraw(address)	public	view	override
maxRedeem(address)	public	view	override
maxDeposit(address)	public	view	override
maxMint(address)	public	view	override
deposit(uint256,address)	public	-	override onlyController nonReentrant
mint(uint256,address)	public	-	override onlyController nonReentrant
withdraw(uint256,address,address)	public	-	override onlyController nonReentrant
redeem(uint256,address,address)	public	-	override onlyController nonReentrant
_update(address,address,uint256)	internal	-	override
controllerSlash(uint256,address)	external	-	onlyController

ProverVault			
getTransferableShares(address)	public	view	override
onlyController (modifier)	-	-	-

VaultFactory			
Function Name	Visibility	Mutability	Modifiers
getVault(address)	public	view	override
init(address)	external	-	-
createVault(address,address,address)	external	-	onlyController override
predictVaultAddress(address,address,address)	external	view	override
isVaultDeployed(address)	external	view	override
getVaultCount()	external	view	override
getVaultAtIndex(uint256)	external	view	override
_generateSalt(address,address)	internal	pure	-
_getCreationCode(address,address,address)	internal	pure	-
_addressToString(address)	internal	pure	-
onlyController (modifier)	-	-	-

4.3 Vulnerability Summary

[N1] [Low] There is no reasonable range for parameters without limitations.

Category: Others

Content

The following function does not limit the range of the parameters when passed, and the operator's incorrect operation may lead to serious abnormalities in the values, resulting in asset loss or contract failure.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
function init(
    address _stakingToken, //SlowMist//
    address _vaultFactory, //SlowMist//
    uint256 _unstakeDelay,
    uint256 _minSelfStake,
    uint256 _maxSlashBps
) external {

function _init(
    address _stakingToken,
    address _vaultFactory,
    uint256 _unstakeDelay, //SlowMist//
    uint256 _minSelfStake, //SlowMist//
    uint256 _maxSlashBps //SlowMist//
) internal {

function setUnstakeDelay(uint256 newDelay) external override onlyOwner {
```

- prover-network-contracts/src/market/BrevisMarket.sol

```
function init(
    IPicoVerifier _picoVerifier, //SlowMist//
    IStakingController _stakingController, //SlowMist//
    uint64 _biddingPhaseDuration,
    uint64 _revealPhaseDuration,
    uint256 _minMaxFee
) external

function _init(
    IPicoVerifier _picoVerifier,
    IStakingController _stakingController,
    uint64 _biddingPhaseDuration, //SlowMist//
    uint64 _revealPhaseDuration, //SlowMist//
    uint256 _minMaxFee //SlowMist//
) internal

function setBiddingPhaseDuration(uint64 newDuration) external override onlyOwner {

function setRevealPhaseDuration(uint64 newDuration) external override onlyOwner {

function setMinMaxFee(uint256 newMinFee) external override onlyOwner {

function setMaxMaxFee(uint256 newMaxFee) external override onlyOwner {
```

```
    function setSlashBps(uint256 newBps) external override onlyOwner {

    function setSlashWindow(uint256 newWindow) external override onlyOwner {
```

- prover-network-contracts/src/staking/vault/ProverVault.sol

```
    constructor(IERC20 asset_, string memory name_, string memory symbol_, address
prover_, address controller_)
        ERC20(name_, symbol_)
        ERC4626(asset_)
    {
        prover = prover_;
        controller = controller_;
    }
```

Solution

The address type needs to be checked to see if it is a zero address; The numerical parameter needs to be restricted to the maximum or minimum value.

Status

Acknowledged

[N2] [Suggestion] StakingController reinitialize risk

Category: Design Logic Audit

Content

When the owner in the contract storage is equal to `address(0)`, any external address can call `init(...)` again to reset the critical state and owner. It is not reliable to judge solely based on `owner == address(0)`: the storage can be reset by other logic or upgrade accidents.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    function init(
        address _stakingToken,
        address _vaultFactory,
        uint256 _unstakeDelay,
        uint256 _minSelfStake,
        uint256 _maxSlashBps
    ) external {
        _init(_stakingToken, _vaultFactory, _unstakeDelay, _minSelfStake,
```

```
_maxSlashBps);
        initOwner();
```

- prover-network-contracts/src/market/BrevisMarket.sol

```
    function init(
        IPicoVerifier _picoVerifier,
        IStakingController _stakingController,
        uint64 _biddingPhaseDuration,
        uint64 _revealPhaseDuration,
        uint256 _minMaxFee
    ) external {
        _init(_picoVerifier, _stakingController, _biddingPhaseDuration,
_revealPhaseDuration, _minMaxFee);
        initOwner();
```

Solution

Use a one-time initialization protection flag (such as the OpenZeppelin initializer / reinitializer modifier)

Status

Acknowledged

[N3] [Medium] retireProver lacks permission control

Category: Authority Control Vulnerability Audit

Content

Anyone can call `retireProver` to clean and delete the information of the prover when meeting the asset and queue conditions.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
function retireProver(address prover) public override {
        // Validate prover exists
        ProverInfo storage proverInfo = _proverInfo[prover];
        if (proverInfo.state == ProverState.Null) revert
ControllerProverNotInitialized();
//...
```

External individuals can remove the prover that meets the conditions at any time, thereby disrupting the predictable lifecycle and governance process.

Solution

Only administrators or the prover themselves can call it.

Status

Fixed

[N4] [Critical] retireProver commission loss risk

Category: Design Logic Audit

Content

`pendingCommission` is deleted by `delete _proverInfo[prover]` without being settled or verified, resulting in the disappearance of the commission account payable to the prover, with the funds still remaining in the contract balance and unable to be withdrawn by the prover.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    function retireProver(address prover) public override {
         //...
        delete _proverInfo[prover];
```

Solution

When `pendingCommission` >0, delete prover is prohibited.

Status

Fixed

[N5] [High] Structural data residue risk

Category: Variable Coverage Vulnerability

Content

In retireProver, executing `delete _proverInfo[prover];` attempts to clear all information of the prover.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    // Core data structures
    mapping(address => ProverInfo) private _proverInfo;

    function retireProver(address prover) public override {
//...
```

```
        delete _proverInfo[prover];
        delete pendingUnstakes[prover];
```

`ProverInfo` internally contains `mapping(address => uint256) shares;` as well as `EnumerableSet.AddressSet stakers`, `EnumerableMap.AddressToUintMap commissionRates`. The delete semantics of Solidity for structures will only reset the length markers of value types and fixed-length references, but will not enumerate and erase the storage slots that have been written to the internal mapping.

- prover-network-contracts/src/staking/interfaces/IStakingController.sol

```
    struct ProverInfo {
        ProverState state;
        address vault; // Dedicated ERC4626 vault address
        mapping(address => uint256) shares; // Individual stake shares per staker
        EnumerableSet.AddressSet stakers; // Set of stakers with active stakes
        uint256 pendingCommission; // Accumulated commission waiting to be claimed
        // Map of reward source to commission rate in basis points (0-10000).
address(0) = default rate
        EnumerableMap.AddressToUintMap commissionRates;
        uint64 joinedAt; // Timestamp when the prover joined (initialized)
        // Profile fields (display-only, editable by prover/admin)
        string name; // <= 128 bytes
        string iconUrl; // <= 512 bytes
    }
```

`shares[staker]` entries remain unchanged, and `getStakeInfo(prover, staker)` can still return non-zero old data after `retire`, forming "ghost shares."

Solution

Manually clear the map values in the structure.

Status

Fixed

[N6] [Medium] Dust assets cause retireProver to be permanently blocked.

Category: Integer Overflow and Underflow Vulnerability

Content

`retireProver` uses `vault.totalAssets() == 0` as a necessary condition for retirement, but ERC4626 vaults have an inevitable source of "dust assets":

Rounding: `convertToAssets/convertToShares` rounds down, and redemption/exchange may leave an extremely small balance in the vault contract.

Donation attack: Anyone can directly `ERC20.transfer` a small amount of underlying assets to the vault address (ERC20 cannot prevent this), and `totalAssets` immediately becomes non-zero.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
function retireProver(address prover) public override {
    //...
    address vault = proverInfo.vault;
    uint256 vaultAssets = IProverVault(vault).totalAssets();
    if (vaultAssets > 0) revert ControllerCannotRetireProverWithAssets();
```

Any address can permanently block `retireProver` by simply sending 1 wei of asset to the vault.

Solution

Set up `dustThreshold`; if `vaultAssets <= dustThreshold`, allow direct retirement and sweep.

Status

Fixed

[N7] [Medium] ERC4626 share inflation risk

Category: Design Logic Audit

Content

The stake function allows any delegator to call with any small amount > 0 at any time when the prover is active, and the first or priority stakers occupy all or most of the shares when the total share supply is extremely low.

After one or more large reward donations (`addRewards` or third parties transferring underlying assets directly to the vault), since no new shares are issued, existing holders enjoy the full growth of assets with a very small share.

When new stakers call stake later, `convertToShares`, based on the expanded `totalAssets/totalSupply`, will receive very few shares, effectively transferring most of the new value purchased to early holders.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    function stake(address prover, uint256 amount)
        public
        override
        whenNotPaused
        nonReentrant
        returns (uint256 shares)
    {
        if (amount == 0) revert ControllerZeroAmount();
        //...
        shares = IProverVault(proverInfo.vault).deposit(amount, staker);
```

One solution is to pre-deposit a portion of liquidity `minSelfStake` when initializing the vault, but there is no reasonable minimum limit for `minSelfStake`.

Solution

Limit `minSelfStake` to a larger value.

Status

Fixed

[N8] [Low] addRewards does not align with the actual amount credited.

Category: Design Logic Audit

Content

The token is either "transaction fee/deflationary" or non-standard ERC20, causing the actual receipt of `safeTransferFrom` to be `received < amount`.

The code calculates and transfers `toStakers = amount - commission` based on the nominal amount, without aligning the received amount.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    function addRewards(address prover, uint256 amount)
        external
        override
```

```
        whenNotPaused
        nonReentrant
        returns (uint256 commission, uint256 toStakers)
    {
        // Validate prover is active
        ProverInfo storage proverInfo = _proverInfo[prover];
        if (proverInfo.state == ProverState.Null) revert
ControllerProverNotInitialized();
        if (!_isActive(prover)) revert ControllerProverNotActive();


        // Transfer assets from caller
        stakingToken.safeTransferFrom(msg.sender, address(this), amount);


        // Get commission rate for this source (msg.sender)
        uint256 commissionRate = getCommissionRate(prover, msg.sender);


        // Calculate commission split
        commission = (amount * commissionRate) / BPS_DENOMINATOR;
        toStakers = amount - commission;
```

Solution

Calculate the actual number of tokens received by using the difference in balance before and after the transfer.

Status

Acknowledged

[N9] [Low] Instantaneous manipulation of commission rates

Category: Design Logic Audit

Content

Prover can increase the commission fee rate of a certain source address by preemptively boosting it when observing large reward transactions about to call `addRewards`, thereby capturing a higher proportion of rewards and reducing the returns that should be allocated to ordinary stakers. The current implementation achieves rate updates with no delay, no limit on the scale, and no cooling-off period.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    function setCommissionRate(address source, uint64 newRate) external override {
        //...
```

```
    }
```

Attack path example:

(1). Large reward transaction `addRewards(prover, amount)` enters the memory pool, with the source address as S.

(2). The Prover discovers this transaction (or colludes with the reward giver), and submits `setCommissionRate(S, 9000)` prematurely.

(3). The original `addRewards` is packaged and executed by the miner/sorter, calculating `commission = amount * 9000 / 10000`, with the ordinary staker only receiving 10%.

(4). The Prover subsequently withdraws the large commission with `claimCommission`, distorting the economic distribution.

Solution

It should be assumed that Prover is controlled by an untrusted third party, with delayed effective rate configuration or slippage control at the UX layer when claiming rewards.

Status

Acknowledged

[N10] [Medium] Incorrect asset balance logic

Category: Design Logic Audit

Content

When `ProverInfo.state == ProverState.Null`, return 0 directly, but this does not reflect the actual asset situation of the prover. Monitoring/scheduling/risk control scripts may treat the error address as a normal zero-asset prover.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    function getProverTotalAssets(address prover) public view override returns
(uint256 totalAssets) {
        // Validate prover exists
        ProverInfo storage proverInfo = _proverInfo[prover];
        if (proverInfo.state == ProverState.Null) return 0;
```

```
        //...
    }
```

Solution

If prover state is `Null` then `revert ControllerProverNotInitialized();`

Status

Acknowledged

[N11] [Low] Unchecked return value

Category: Others

Content

Some tokens return false without reverting (due to old versions or malicious implementations), and the current logic still continues to execute concurrent events, leading to a discrepancy between the event and the actual asset status.

- prover-network-contracts/src/staking/controller/StakingController.sol

```
    function emergencyRecover(address to, uint256 amount) external override whenPaused
onlyOwner {
        stakingToken.transfer(to, amount);
        emit EmergencyRecovered(to, amount);
    }
```

Solution

Use `safeTransfer` instead.

Status

Fixed

[N12] [Low] Unprotected Initializer

Category: Authority Control Vulnerability Audit

Content

After the contract is deployed, before any trusted account is initialized, any address can first call `init` to set `stakingController`, permanently seize the `onlyController` permission, and then control the execution of `createVault` and the control of the downstream `ProverVault`.

- prover-network-contracts/src/staking/vault/VaultFactory.sol

```
    function init(address _controller) external {
        if (stakingController != address(0)) {
            revert VaultFactoryAlreadyInitialized();
        }
        if (_controller == address(0)) {
            revert VaultFactoryZeroAddress();
        }
        stakingController = _controller;
    }
```

Solution

Remove init, change to constructor initialization.

Status

Acknowledged

[N13] [Low] Prover status verification and management are incorrect.

Category: Design Logic Audit

Content

In the `registerSubmitter` function, only provers with the state `ProverState.Null` are rejected, allowing provers in the `Deactivated/Jailed` state to still register as submitters.

This directly undermines the intent of the state machine and governance control, potentially allowing penalized entities to continue participating in the critical path of the market; it can be exploited sustainably under the lack of thorough verification of the Active implementation.

Similarly, the `submitterState` state validation also has a similar issue.

- prover-network-contracts/src/market/ProverSubmitters.sol

```
    function registerSubmitter(address submitter) public {
        //...
        IStakingController.ProverState state =
stakingController.getProverState(prover);
        if (state == IStakingController.ProverState.Null) {
            revert MarketProverNotRegistered();
        }
        //...
```

```
        if (submitterState != IStakingController.ProverState.Null) {
            revert MarketCannotRegisterProverAsSubmitter(submitter);
        }
```

Solution

The reasonable expectation should be that only `ProverState.Active` should be allowed to register submitters.

Status

Acknowledged; Intend to allow inactive provers to register as submitters.

[N14] [Low] _getEffectiveProver intent and implementation discrepancy

Category: Design Logic Audit

Content

In the `_getEffectiveProver` function, if the caller is a registered submitter, it returns the mapped prover; otherwise, it directly returns the caller itself. If subsequent market logic (bidding/submitting proof, etc.) only uses the output of `_getEffectiveProver` without performing state verification, any address can pretend to be a prover.

- prover-network-contracts/src/market/ProverSubmitters.sol

```
    function _getEffectiveProver(address caller) internal view returns (address
prover) {
        // First check if caller is a registered submitter
        address registeredProver = submitterToProver[caller];
        if (registeredProver != address(0)) {
            return registeredProver;
        }


        // Otherwise, caller is acting as themselves (direct prover)
        return caller;
    }
```

According to the context, it should be an intention and implementation discrepancy.

Solution

If the caller is not registered, an error is thrown.

Status

Acknowledged

[N15] [Low] ReqId Unbound Requester Caused Request Hijacking and Rushing

Category: Design Logic Audit

Content

The current reqid in `requestProof` is only calculated from `(nonce, vk, publicValuesDigest)` and is not bound to the identity of the requestor `(msg.sender)`. Malicious actors can listen to the parameters of others' initiated requests in the memory pool, construct the same triplet, and execute `requestProof` first with higher gas, leading to:

The original requestor's transaction is reverted due to `MarketRequestAlreadyExists(reqid)` and cannot submit the same request;

- prover-network-contracts/src/market/BrevisMarket.sol

```
   function requestProof(ProofRequest calldata req) external override {
       //...
       bytes32 reqid = keccak256(abi.encodePacked(req.nonce, req.vk,
req.publicValuesDigest));


       ReqState storage reqState = requests[reqid];
       if (reqState.timestamp != 0) revert MarketRequestAlreadyExists(reqid);
//...
   }
```

Attack Path/Reproduction Steps

(1). The victim prepares `req = {nonce, vk, publicValuesDigest, fee(…)}` off-chain and sends the requestProof transaction.

(2). The attacker listens to the memory pool, reads the victim's `(nonce, vk, publicValuesDigest)`, constructs the same triplet but customizes the fee parameter (e.g., a shorter deadline or an extreme minStake).

(3). The attacker calls `requestProof` first with higher gas, and the contract generates the same reqid and records the transaction.

(4). The original transaction of the victim reverts due to `MarketRequestAlreadyExists(reqid)`, and the victim cannot submit the request; subsequent bidding/revealing/submitting proof related to the reqid are all bound to the parameters set by the attacker.

Solution

Change the calculation logic of reqid, bind it to the chain, contract, and requestor; and add a view function for consistent calculation on the frontend.

Status

Acknowledged

[N16] [Suggestion] Anti-reentry coding recommendation

Category: Reentrancy Vulnerability

Content

The following function has external calls, and it is recommended to implement anti-reentrancy protection for the function to avoid external risks.

- prover-network-contracts/src/market/BrevisMarket.sol

```
function slash(bytes32 reqid) external override {
```

- prover-network-contracts/src/staking/controller/StakingController.sol

```
function initializeProver(uint64 defaultCommissionRate)
    external
    override
    onlyAuthorized
    whenNotPaused
    returns (address vault)
{
```

Solution

Although there is no direct risk, it is recommended to add the nonReentrant decorator to the code to avoid security risks caused by changes in external code.

Status

Acknowledged

[N17] [Suggestion] Residual Infinite Authorization

Category: Authority Control Vulnerability Audit

Content

In `BrevisMarket._init`, `feeToken.approve(controller, type(uint256).max)` was executed on `stakingController`, but when reinitializing/migrating the controller, the authorization from the old controller was not revoked first, resulting in the old controller retaining unlimited authorization over the `feeToken` of this contract.

- prover-network-contracts/src/market/BrevisMarket.sol

```
function _init(
    IPicoVerifier _picoVerifier,
    IStakingController _stakingController,
    uint64 _biddingPhaseDuration,
    uint64 _revealPhaseDuration,
    uint256 _minMaxFee
) internal {
    //...
    stakingController = _stakingController;
    //...
    feeToken.approve(address(stakingController), type(uint256).max);
```

Solution

Revoke old authorization, then set new authorization.

Status

Acknowledged; No code path to reinitialize or migrate stakingController.

[N18] [Suggestion] Unnecessary state variables causing storage expansion

Category: Others

Content

Each request writes three variable-length fields on the chain `(string imgURL, bytes inputData, string inputURL)`; frequent use will significantly increase the contract state volume and interaction cost.

- prover-network-contracts/src/market/interfaces/IBrevisMarket.sol

```
struct ProofRequest {
    uint64 nonce; // allow re-submit same data
    bytes32 vk; // verify key for binary
    bytes32 publicValuesDigest; // sha256(publicValues) & bytes32(uint256((1 <<
```

```
253) - 1)))
        string imgURL; // URL to ELF binary, can be empty if vk is already known to the
prover network
        bytes inputData; // input data for the binary, can be empty if inputURL is
provided
        string inputURL; // URL to input data, if inputData is not provided
        FeeParams fee;
    }
```

Solution

Remove unused fields.

Status

Acknowledged

[N19] [Low] Bid fee exposure risk

Category: Others

Content

The protocol treats nonce as a salt but does not enforce randomness or minimum entropy; users can use small integers (0, 1, 2...). When the fee range is narrow (limited by `minMaxFee` and `maxMaxFee`), observers can offline enumerate `(fee, nonce)` combinations, match the bidHash that has been chained, and thus know the real bid in advance.

- prover-network-contracts/src/market/BrevisMarket.sol

```
    function reveal(bytes32 reqid, uint256 fee, uint256 nonce) external override {
         //...
        bytes32 expectedHash = keccak256(abi.encodePacked(reqid, prover, fee, nonce));
```

Attackers know the approximate distribution of the lowest bids before the revelation stage, and can: precisely lower their own bids, occupying the "lowest" position.

Solution

Upgrade the nonce semantics to a strong random "salt" or "blindingFactor", requiring high entropy.

Status

Acknowledged

[N20] [Medium] Excessive Authority Risk

Category: Authority Control Vulnerability Audit

Content

- prover-network-contracts/src/market/BrevisMarket.sol

  (1). Owner can call:

  `setPicoVerifier、setBiddingPhaseDuration、setRevealPhaseDuration、setMinMaxFee、setMaxMaxFee、setSlashBps、setSlashWindow、setProtocolFeeBps、setOvercommitBps、withdrawProtocolFee、scheduleStatsEpoch、popStatsEpoch`

  Owner can replace the proof verifier to a malicious contract, modify auction and penalty/fee parameters, directly withdraw the protocol fee, affecting the fairness and the boundary of capital security.

- prover-network-contracts/src/staking/controller/StakingController.sol

  (1). Owner can call:

  `deactivateProver/deactivateProvers, jailProver/jailProvers, retireProvers, parameter settings (setMinSelfStake, setMaxSlashBps, setUnstakeDelay, setRequireAuthorization), fund operations (withdrawTreasury, emergencyRecover)`

  (2). SLASHER_ROLE can call:

  `slash/slashByAmount`

Owner/Slasher has strong control and fund handling capabilities `(freeze/withdraw/penalty/withdrawal)`, which have a significant impact on availability and fund security.

Solution

In the short term, transferring privileged roles to a multi-signature wallet can effectively mitigate the single point of failure risk. In the long term, transferring privileged roles to DAO governance can effectively address the risk of excessive privilege. During the transition period, managing through multi-signature with delayed transaction execution via timelock can effectively mitigate the risk of excessive privilege.

Status

Acknowledged

[N21] [Information] ZK proof reuse risk

Category: Replay Vulnerability

Content

The current verification logic only calls `picoVerifier.verifyPicoProof(req.vk, req.publicValuesDigest, proof)`. As long as the same `(vk, publicValuesDigest)` is shared among different requests, the same proof can be repeatedly submitted and rewards can be claimed in multiple requests:

(1). The reqid of `requestProof` is generated by `(nonce, vk, publicValuesDigest)`, and changing the nonce can create multiple "equivalent" requests.

(2). `submitProof` does not detect whether the proof has been used in other requests before.

(3). The proof is not bound to reqid / unique request context, leading to the same work being monetized multiple times.

(4). If multiple requests are initiated by the same or different addresses, attackers can create requests in bulk.

- prover-network-contracts/src/market/BrevisMarket.sol

```
   function requestProof(ProofRequest calldata req) external override {
        //...
       bytes32 reqid = keccak256(abi.encodePacked(req.nonce, req.vk,
req.publicValuesDigest));
//...
    }


   function submitProof(bytes32 reqid, uint256[8] calldata proof) external override
nonReentrant {
//...
       picoVerifier.verifyPicoProof(req.vk, req.publicValuesDigest, proof);
//...
    }
```

Solution

Change the calculation logic of reqid, bind it to the chain, contract, and requestor; and add a view function for consistent calculation on the frontend.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002511190004	SlowMist Security Team	2025.11.11 - 2025.11.19	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, 5 medium risk, 9 low risk, 4 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