OMNISCIA

May 6, 2025

SMART CONTRACT AUDIT REPORT

Suzaku Network
Validator Manager


omniscia.io


info@omniscia.io


Online report: suzaku-network-validator-manager

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.


omniscia.io


info@omniscia.io

Online report: suzaku-network-validator-manager

Validator Manager Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
c24eb0d919	April 4th 2025	4445460feb
911849f546	May 6th 2025	ebc5be49b2

Audit Overview

We were tasked with performing an audit of the Suzaku Network codebase and in particular their Validator Manager module meant for the Avalanche ecosystem.

The implementation introduces a specialized `ValidatorManager` that guards access control to security modules with a maximum weight allowance, permitting different access control approaches to lead to different weight totals being managed through the `BalancerValidatorManager`.

Over the course of the audit, we identified that the removal of a validator due to expiry is not adequately handled in the weight calculations and would result in incorrect weight totals being maintained.

Additionally, we observed that the contract is meant to be an upgrade of an existing implementation which was not denoted in the audit scope and should be explicitly outlined to ensure the new implementation is compatible with the storage structure of the previous one.

We advise the Suzaku Network team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Suzaku Network team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Suzaku Network and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	8	8	0	0
Minor	1	1	0	0
Medium	2	2	0	0
Major	1	1	0	0

During the audit, we filtered and validated a total of **1 findings utilizing static analysis** tools as well as identified a total of **11 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

- **Scope**
- **Compilation**
- **Static Analysis**
- **Manual Review**
- **Code Style**

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/suzaku-network/suzaku-contracts-library**
- Commit: c24eb0d919e685c4e3c3b1605bd2a0fdb4cb8eb7
- Language: Solidity
- Network: Avalanche
- Revisions: **c24eb0d919**, **911849f546**

Contracts Assessed

File	Fin
src/contracts/ValidatorManager/BalancerValidatorManager.sol (BVM)	
src/contracts/ValidatorManager/SecurityModule/PoASecurityModule.sol (PAS)	

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.25` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used, however, they are solely contained in external dependencies and can thus be safely ignored.

The `pragma` statements have been locked to `0.8.25` (`=0.8.25`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **9 potential issues** within the codebase of which **8 were ruled out to be false positives** or negligible findings.

The remaining **1 issues** were validated and grouped and formalized into the **1 exhibits** that follow:

ID	Severity	Addressed	Title
PAS-01S	Informational	Yes	Inexistent Sanitization of Input Address

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Suzaku Network's specialized Validator Manager.

As the project at hand implements an Avalanche compatible validator manager, intricate care was put into ensuring that the **flow of validator states within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed an unhandled validator state transition** within the system which could have had **moderate-to-severe ramifications** to its overall operation; for more information, kindly consult the relevant major-severity exhibit within the audit report.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be.

A total of **11 findings** were identified over the course of the manual review of which **7 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
BVM-01M	Informational	Yes	Inexistent Validation of Initial Security Module Weight
BVM-02M	Informational	Nullified	Outdated Validator Manager Dependency
BVM-03M	Minor	Nullified	Incorrect Handling of Validation Weight Updates
BVM-04M	Medium	Yes	Inexistent Evaluation of Duplicates
BVM-05M	Major	Yes	Incorrect Handling of Validation Ends

PAS-01M	Informational	Yes	Inexplicable Access Control of Functions
PAS-02M	Medium	Nullified	Insufficient Integration of `BalancerValidatorManager`

Code Style

During the manual portion of the audit, we identified **4 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
BVM-01C	Informational	Yes	Inefficient Data Pointer Types
BVM-02C	Informational	Yes	Inefficient Dynamic Query of Security Module
BVM-03C	Informational	Yes	Inefficient Re-Query of Validator
PAS-01C	Informational	Yes	Generic Typographic Mistake

PoASecurityModule Static Analysis Findings

PAS-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	PoASecurityModule.sol:L25-L27

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/contracts/ValidatorManager/SecurityModule/PoASecurityModule.sol

SOL

```sol
25  constructor(address balancerValidatorManager_, address initialOwner)
Ownable(initialOwner) {
26      balancerValidatorManager =
IBalancerValidatorManager(balancerValidatorManager_);
27  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:

The input `balancerValidatorManager_` address argument of the **`PoASecurityModule::constructor`** function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

BalancerValidatorManager Manual Review Findings

BVM-01M: Inexistent Validation of Initial Security Module Weight

Type	Severity	Location
Input Sanitization	Informational	BalancerValidatorManager.sol:L104, L366

Description:

The initial security module configured for the `BalancerValidatorManager` should have an upper weight bound that is at least equal to or higher than the `vms._churnTracker.totalWeight` value; a relation that is presently not enforced.

Impact:

A misconfigured upper bound for the security module would lead to a failure of the contract's initialization in any case, rendering this exhibit to be of informational nature.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```sol
98  // solhint-disable-next-line no-empty-blocks
99  function __BalancerValidatorManager_init_unchained(
100     address initialSecurityModule,
101     uint64 initialSecurityModuleMaxWeight,
102     bytes[] calldata migratedValidators
103 ) internal onlyInitializing {
104     _setUpSecurityModule(initialSecurityModule, initialSecurityModuleMaxWeight);
105     _migrateValidators(migratedValidators);
106 }
```

Recommendation:

We advise such validation to be introduced, increasing the code's maintainability as well as the capabilities of off-chain code to handle errors gracefully.

To note, this can also result in an optimization by having the `BalancerValidatorManager::_migrateValidators` function update the `$.securityModuleWeight` data entry directly without validation of its maximum weight.

Alleviation:

The code the `BalancerValidatorManager` initialization code was updated to ensure that the configured initial security module maximum weight is at most equal to the current total weight of the churn tracker, alleviating this exhibit.

BVM-02M: Outdated Validator Manager Dependency

Type	Severity	Location
Standard Conformity	Informational	BalancerValidatorManager.sol:L10

Description:

The `ValidatorManager` dependency of the project via the `@avalabs/icm-contracts` library entry is outdated and does not point to the latest commit hash of the `pos-validator-manager-external-library` branch.

Impact:

As the Ava Labs dependency has not introduced any changes to the actual dependencies of the `BalancerValidatorManager`, we consider this to be an informational notice rather than an important issue in the codebase.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```
10  import {ValidatorManager} from "@avalabs/icm-contracts/validator-
manager/ValidatorManager.sol";
```

Recommendation:

We advise the latest version of the dependency to be utilized, ensuring the code is up-to-date with the latest Ava Labs implementation.

Alleviation:

The Suzaku Network team evaluated this exhibit and clarified that they wish to point to an audited version of the `@avalabs/icm-contracts` dependency rather than its latest implementation.

As the actual dependency has no code delta between versions, we consider the approach of the Suzaku Network team to be correct rendering this exhibit to be nullified.

BVM-03M: Incorrect Handling of Validation Weight Updates

Type	Severity	Location
Logical Fault	Minor	BalancerValidatorManager.sol:L162-L195, L197-L222

Description:

The `BalancerValidatorManager::initializeValidatorWeightUpdate` will cause the total weight of the security module to be immediately updated which is an incorrect approach as the effects of an initialized action should occur in its completion hook (i.e. `BalancerValidatorManager::completeValidatorWeightUpdate`).

Impact:

From a security perspective, the security module should eagerly consume its allowance by marking it as "utilized" when initializing the registration of a validator and marking it as "available" when finalizing the removal of a validator.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```sol
162 /// @inheritdoc IBalancerValidatorManager
163 function initializeValidatorWeightUpdate(
164     bytes32 validationID,
165     uint64 newWeight
166 ) external onlySecurityModule returns (Validator memory validator) {
167     BalancerValidatorManagerStorage storage $ =
_getBalancerValidatorManagerStorage();
168
169     // Check that the newWeight is greater than zero
170     if (newWeight == 0) {
171         revert BalancerValidatorManager__NewWeightIsZero();
```

Example (Cont.):

```sol
    }

    // Ensure the validation period is active and that the validator is not already being updated
    // The initial validator set must have been set already to have active validators.
    validator = getValidator(validationID);
    if (validator.status != ValidatorStatus.Active) {
        revert InvalidValidatorStatus(validator.status);
    }
    if ($.validatorPendingWeightUpdate[validationID] != 0) {
        revert BalancerValidatorManager__PendingWeightUpdate(validationID);
    }

    _checkValidatorSecurityModule(validationID, msg.sender);
    uint64 oldWeight = getValidator(validationID).weight;
    (, bytes32 messageID) = _setValidatorWeight(validationID, newWeight);

    // Update the security module weight
    uint64 newSecurityModuleWeight = $.securityModuleWeight[msg.sender] + newWeight - oldWeight;
    _updateSecurityModuleWeight(msg.sender, newSecurityModuleWeight);

    $.validatorPendingWeightUpdate[validationID] = messageID;

    return validator;
}

/// @inheritdoc IBalancerValidatorManager
function completeValidatorWeightUpdate(bytes32 validationID, uint32 messageIndex) external {
    BalancerValidatorManagerStorage storage $ = _getBalancerValidatorManagerStorage();
```

Example (Cont.):

```sol
    Validator memory validator = getValidator(validationID);

    // Check that the validator is active and being updated
    if (validator.status != ValidatorStatus.Active) {
        revert InvalidValidatorStatus(validator.status);
    }
    if ($.validatorPendingWeightUpdate[validationID] == 0) {
        revert BalancerValidatorManager__NoPendingWeightUpdate(validationID);
    }

    // Unpack the Warp message
    (bytes32 messageValidationID, uint64 nonce,) = ValidatorMessages
.unpackL1ValidatorWeightMessage(_getPChainWarpMessage(messageIndex).payload);

    if (validationID != messageValidationID) {
        revert InvalidValidationID(validationID);
    }
    if (validator.messageNonce < nonce) {
        revert BalancerValidatorManager__InvalidNonce(nonce);
    }

    delete $.validatorPendingWeightUpdate[validationID];
}
```

Recommendation:

We advise the update of the validator's weight in the security module to be handled after it has been finalized, ensuring that the security module's weight allowance is updated after actions have been finalized rather than when they are initiated.

Alleviation:

The Suzaku Network team evaluated this exhibit and stated that this behaviour is expected as the total weight of a security module should be updated at the same time as the validator's weight.

We consider the current approach to be sound and thus this exhibit to be nullified.

BVM-04M: Inexistent Evaluation of Duplicates

Type	Severity	Location
Logical Fault	Medium	BalancerValidatorManager.sol:L358-L363

Description:

The `BalancerValidatorManager::_migrateValidators` function does not prevent duplicate migrated validators to be specified which would result in an inflated `migratedValidatorsTotalWeight` and thus an inflated security module total weight.

Impact:

Presently, it is possible to skip the migration of certain validators by substituting them with others that are of equal weight as long as the `vms._churnTracker.totalWeight` is satisfied.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```sol
350 function _migrateValidators(
351     bytes[] calldata migratedValidators
352 ) internal {
353     BalancerValidatorManagerStorage storage $ =
_getBalancerValidatorManagerStorage();
354     ValidatorManager.ValidatorManagerStorage storage vms =
_getValidatorManagerStorage();
355
356     // Add the migrated validators to the initial security module
357     uint64 migratedValidatorsTotalWeight = 0;
358     for (uint256 i = 0; i < migratedValidators.length; i++) {
359         bytes32 validationID = registeredValidators(migratedValidators[i]);
```

Example (Cont.):

```sol
        Validator memory validator = getValidator(validationID);
        $.validatorSecurityModule[validationID] = $.securityModules.keys()[0];
        migratedValidatorsTotalWeight += validator.weight;
    }

    // Check that the migrated validators total weight equals the current L1 total weight
    if (migratedValidatorsTotalWeight != vms._churnTracker.totalWeight) {
        revert BalancerValidatorManager__MigratedValidatorsTotalWeightMismatch(
            migratedValidatorsTotalWeight, vms._churnTracker.totalWeight
        );
    }

    // Update the initial security module weight
    _updateSecurityModuleWeight($.securityModules.keys()[0], migratedValidatorsTotalWeight);
}
```

Recommendation:

We advise the code to prevent duplicates, either by confirming their `$.validatorSecurityModule` data entry is empty or by ensuring that the `migratedValidators` have been set in a strictly ascending order, the latter of which we recommend and would be the most gas-efficient solution.

Alleviation:

The Suzaku Network team opted to implement the former of the two recommended mitigations, ensuring that any migrated validator does not have a non-zero `validatorSecurityModule` data entry.

BVM-05M: Incorrect Handling of Validation Ends

Type	Severity	Location
Logical Fault	Major	BalancerValidatorManager.sol:L133-L153, L156-L160

Description:

The `BalancerValidatorManager` will incorrectly handle weight updates for a particular security module during the removal of a validator due to not handling the removal of **an expired validator**.

Specifically, the `BalancerValidatorManager::completeEndValidation` might be invoked after an explicit call to `BalancerValidatorManager::initializeEndValidation` or due to an expired validator in the `PendingAdded` state.

Additionally, the `BalancerValidatorManager::initializeEndValidation` will render the weight removed available for another validator immediately rather than waiting for the validator's removal to be finalized.

Impact:

The end of a validator due to their expiry is presently unhandled, resulting in incorrect weights per security module and thus incorrect enforcements of maximum weights per module; a core feature of the contract implementation.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```sol
function initializeEndValidation(
    bytes32 validationID
) external onlySecurityModule returns (Validator memory) {
    BalancerValidatorManagerStorage storage $ =
_getBalancerValidatorManagerStorage();
    Validator memory validator = getValidator(validationID);

    // Ensure the validator weight is not being updated
    if ($.validatorPendingWeightUpdate[validationID] != 0) {
        revert BalancerValidatorManager__PendingWeightUpdate(validationID);
    }
```

Example (Cont.):

SOL

```sol

    _checkValidatorSecurityModule(validationID, msg.sender);
    validator = _initializeEndValidation(validationID);

    // Update the security module weight
    uint64 newSecurityModuleWeight = $.securityModuleWeight[msg.sender] -
validator.weight;
    _updateSecurityModuleWeight(msg.sender, newSecurityModuleWeight);

    return validator;
}

/// @inheritdoc IValidatorManager
function completeEndValidation(
    uint32 messageIndex
) external {
    _completeEndValidation(messageIndex);
}
```

Recommendation:

We advise the code to update the available total weight of a validator after the `BalancerValidatorManager::completeEndValidation` function has been invoked, ensuring that expired inclusions are correctly handled and that the available weight of a security module is updated after the removals have been finalized.

Alleviation:

The code was instead updated to handle the invalidation status of a validator which is configured solely when the end of a validator's validation has not been initialized.

In such a case, the `BalancerValidatorManager::completeEndValidation` function will update the security module weight properly thereby ensuring that security module weights are correctly maintained in the system.

PoASecurityModule Manual Review Findings

PAS-01M: Inexplicable Access Control of Functions

Type	Severity	Location
Logical Fault	Informational	**PoASecurityModule.sol**: • I-1: **L30-L35** • I-2: **L38-L42** • I-3: **L53-L57** • I-4: **L67-L71**

Description:

The referenced `PoASecurityModule` functions are meant to protect interactions with the `BalancerValidatorManager` through the `Ownable::onlyOwner` modifier, however, the actual underlying functions in the parent of the `BalancerValidatorManager` (the `ValidatorManager` of `ava-labs/icm-contracts` at `046ca6d8`) do not impose any access control and thus can be invoked directly.

Example:

src/contracts/ValidatorManager/SecurityModule/PoASecurityModule.sol

SOL

```sol
29  /// @inheritdoc IValidatorManager
30  function initializeValidatorSet(
31      ConversionData calldata conversionData,
32      uint32 messageIndex
33  ) external onlyOwner {
34      balancerValidatorManager.initializeValidatorSet(conversionData, messageIndex);
35  }
```

Recommendation:

We advise either the `BalancerValidatorManager` to `override` those functions and apply proper security-module based access control or the `PoASecurityModule` to not impose the `Ownable::onlyOwner` modifier, either of which we consider an adequate resolution to this exhibit.

Alleviation:

The Suzaku Network team proceeded with removing access control from the four referenced functions after assessing that this is the correct approach, addressing this inconsistency.

PAS-02M: Insufficient Integration of `BalancerValidatorManager`

Type	Severity	Location
Logical Fault	Medium	PoASecurityModule.sol:L29-L78

Description:

The `PoASecurityModule` lacks several functions around the new weight update related functionality of the `BalancerValidatorManager`; namely, the `BalancerValidatorManager::initializeValidatorWeightUpdate`, the `BalancerValidatorManager::completeValidatorWeightUpdate`, and the `BalancerValidatorManager::resendValidatorWeightUpdate` functions.

Impact:

Critical weight-related functionality that the `BalancerValidatorManager` exposes is inaccessible via the `PoASecurityModule` which we consider invalid.

Example:

src/contracts/ValidatorManager/SecurityModule/PoASecurityModule.sol

SOL

```sol
22 contract PoASecurityModule is IPoAValidatorManager, Ownable {
23     IBalancerValidatorManager public immutable balancerValidatorManager;
```

Recommendation:

We advise support for those functions to be introduced as they can solely be invoked via security modules, ensuring that the underlying `BalancerValidatorManager` interfaced by the `PoASecurityModule` remains usable to its fullest extent.

Alleviation:

The Suzaku Network team clarified that the `PoASecurityModule` variant is meant to be compatible with the `IPoAValidatorManager` interface for integration within Ava Labs tooling and thus does not need to support the weight-related functionality exposed by the `BalancerValidatorManager` implementation.

As such, we consider this exhibit to be nullified as the implementation is correct for the deployment purpose it is meant for.

BalancerValidatorManager Code Style Findings

BVM-01C: Inefficient Data Pointer Types

Type	Severity	Location
Gas Optimization	Informational	**BalancerValidatorManager.sol**: • I-1: **L138** • I-2: **L200** • I-3: **L228** • I-4: **L360**

Description:

The referenced data pointers have been specified as `memory` even though they read a subset of the validator's data entries.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```sol
358 for (uint256 i = 0; i < migratedValidators.length; i++) {
359     bytes32 validationID = registeredValidators(migratedValidators[i]);
360     Validator memory validator = getValidator(validationID);
361     $.validatorSecurityModule[validationID] = $.securityModules.keys()[0];
362     migratedValidatorsTotalWeight += validator.weight;
363 }
```

Recommendation:

We advise the `ValidatorManager::_getValidatorManagerStorage` function to be invoked and the `_validationPeriods` data entry to be accessed, permitting the data pointers to be set to `storage` and thus greatly optimize each code segment's gas cost.

Alleviation:

The first referenced instance was optimized by removing the memory read altogether whereas the latter three were optimized per our recommendation by accessing `storage` directly.

BVM-02C: Inefficient Dynamic Query of Security Module

Type	Severity	Location
Gas Optimization	Informational	BalancerValidatorManager.sol:L104, L105, L361, L373

Description:

THe `BalancerValidatorManager::_migrateValidators` function will dynamically query the initial security module even though it could have been passed in as an argument in the `BalancerValidatorManager::__BalancerValidatorManager_init_unchained` function.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```
350 function _migrateValidators(
351     bytes[] calldata migratedValidators
352 ) internal {
353     BalancerValidatorManagerStorage storage $ =
_getBalancerValidatorManagerStorage();
354     ValidatorManager.ValidatorManagerStorage storage vms =
_getValidatorManagerStorage();
355
356     // Add the migrated validators to the initial security module
357     uint64 migratedValidatorsTotalWeight = 0;
358     for (uint256 i = 0; i < migratedValidators.length; i++) {
359         bytes32 validationID = registeredValidators(migratedValidators[i]);
```

Example (Cont.):

```sol
        Validator memory validator = getValidator(validationID);
        $.validatorSecurityModule[validationID] = $.securityModules.keys()[0];
        migratedValidatorsTotalWeight += validator.weight;
    }

    // Check that the migrated validators total weight equals the current L1 total weight
    if (migratedValidatorsTotalWeight != vms._churnTracker.totalWeight) {
        revert BalancerValidatorManager__MigratedValidatorsTotalWeightMismatch(
            migratedValidatorsTotalWeight, vms._churnTracker.totalWeight
        );
    }

    // Update the initial security module weight
    _updateSecurityModuleWeight($.securityModules.keys()[0], migratedValidatorsTotalWeight);
}
```

Recommendation:

We advise it to be passed in as an argument, greatly optimizing the code's gas cost.

Alleviation:

The `initialSecurityModule` is properly relayed to the inner `BalancerValidatorManager::_migrateValidators` function from the top-level initialization function, addressing this exhibit.

BVM-03C: Inefficient Re-Query of Validator

Type	Severity	Location
Gas Optimization	Informational	BalancerValidatorManager.sol:L176, L185

Description:

The referenced statements both invoke the `ValidatorManager::getValidator` function even though no change can be observed in the underlying data source.

Example:

src/contracts/ValidatorManager/BalancerValidatorManager.sol

SOL

```sol
176 validator = getValidator(validationID);
177 if (validator.status != ValidatorStatus.Active) {
178     revert InvalidValidatorStatus(validator.status);
179 }
180 if ($.validatorPendingWeightUpdate[validationID] != 0) {
181     revert BalancerValidatorManager__PendingWeightUpdate(validationID);
182 }
183
184 _checkValidatorSecurityModule(validationID, msg.sender);
185 uint64 oldWeight = getValidator(validationID).weight;
```

Recommendation:

We advise the `validator.weight` to be used for the `oldWeight` assignment rather than the dynamic evaluation that is currently present, optimizing the code's gas cost.

Alleviation:

The inefficient re-query of the validator is avoided by caching the initial lookup to a `Validator storage` pointer, optimizing the codebase.

PoASecurityModule Code Style Findings

PAS-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	PoASecurityModule.sol:L20

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix, a non-`snake_case` module) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

src/contracts/ValidatorManager/SecurityModule/PoASecurityModule.sol

SOL

```
20  * @custom:security-contact https://github.com/ava-
labs/teleporter/blob/main/SECURITY.md
```

Recommendation:

We advise this to be done so to enhance the legibility of the codebase.

Alleviation:

The security contact of the contract has been properly updated to point to the Suzaku Network team.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

- Impact (High): A core invariant of the protocol can be broken for an extended duration.
- Impact (Moderate): A non-core invariant of the protocol can be broken for an extended duration or at scale, or an otherwise major-severity issue is reduced due to hypotheticals or external factors affecting likelihood.
- Impact (Low): A non-core invariant of the protocol can be broken with reduced likelihood or impact.
- Impact (None): A code or documentation flaw whose impact does not achieve low severity, or an issue without theoretical impact; a valuable best-practice
- Likelihood (High): A flaw in the code that can be exploited trivially and is ever-present.
- Likelihood (Moderate): A flaw in the code that requires some external factors to be exploited that are likely to manifest in practice.
- Likelihood (Low): A flaw in the code that requires multiple external factors to be exploited that may manifest in practice but would be unlikely to do so.
- Likelihood (None): A flaw in the code that requires external factors proven to be impossible in a production environment, either due to mathematical constraints, operational constraints, or system-related factors (i.e. EIP-20 tokens not being re-entrant).

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.