OMNISCIA

May 15, 2025

SMART CONTRACT AUDIT REPORT

Suzaku Network
Core Contracts


omniscia.io


info@omniscia.io


Online report: suzaku-network-core-contracts

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-core-contracts

Core Contracts Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
955e016386	April 14th 2025	43f3fb635a
98eef72bcf	May 3rd 2025	872048764d
0e858f0f4f	May 15th 2025	19415099fc

Audit Overview

We were tasked with performing an audit of the Suzaku Network codebase and in particular their core implementation.

The system implements an Avalanche L1 staking implementation permitting users to gain stake access to several node operators.

Over the course of the audit, we identified several issues stemming from improper access control, maliciously induced overflows resulting in Denial-of-Service attacks, and other implementation-specific vulnerabilities.

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 a single exhibit has not been adequately dealt with as its alleviation is incorrect. We advise the Suzaku Network team to revisit the following exhibit: **LRY-01M**

Additionally, the following `informational` findings remain unaddressed and should be revisited: **ACR-03C**, **ALM-03C**, **BDR-01C**, **OLO-02C**, **OVO-02C**, **VTD-03C**, **VTD-04C**

Post-Audit Conclusion (0e858f0f4f)

All remaining exhibits have either been adequately addressed or safely acknowledged in the latest commit hash provided to us by the Suzaku Network team.

We consider all outputs of the audit report properly consumed by the Suzaku Network team with no outstanding remediative actions remaining.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	1	0	0
Informational	55	40	0	15
Minor	6	3	0	3
Medium	5	4	0	1
Major	2	2	0	0

During the audit, we filtered and validated a total of **9 findings utilizing static analysis** tools as well as identified a total of **60 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-core**
- Commit: 955e0163860f860edf38eaeb01b2f3e8dab5506e
- Language: Solidity
- Network: Avalanche
- Revisions: **955e016386**, **98eef72bcf**, **0e858f0f4f**

Contracts Assessed

File	Total Finding(s)
src/contracts/middleware/AssetClassRegistry.sol (ACR)	5
src/contracts/middleware/AvalancheL1Middleware.sol (ALM)	13
src/contracts/delegator/BaseDelegator.sol (BDR)	5
src/contracts/libraries/Checkpoints.sol (CST)	5
src/contracts/DelegatorFactory.sol (DFY)	2
src/contracts/common/Entity.sol (EYT)	2
src/contracts/libraries/ERC4626Math.sol (ERC)	2
src/contracts/L1Registry.sol (LRY)	4
src/contracts/delegator/L1RestakeDelegator.sol (LRD)	1
src/contracts/middleware/libraries/MapWithTimeData.sol (MWT)	3

src/contracts/common/MigratableEntityProxy.sol (MEP)	0
src/contracts/middleware/MiddlewareVaultManager.sol (MVM)	4
src/contracts/OperatorRegistry.sol (ORY)	1
src/contracts/service/OperatorL1OptInService.sol (OLO)	4
src/contracts/service/OperatorVaultOptInService.sol (OVO)	4
src/contracts/SlasherFactory.sol (SFY)	1
src/contracts/middleware/libraries/StakeConversion.sol (SCN)	2
src/contracts/common/StaticDelegateCallable.sol (SDC)	1
src/contracts/VaultFactory.sol (VFY)	2
src/contracts/vault/VaultTokenized.sol (VTD)	8

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 as the `pragma` statements of the contracts are open-ended (`^0.8.0`).

We advise them to be 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 **296 potential issues** within the codebase of which **277 were ruled out to be false positives** or negligible findings.

The remaining **19 issues** were validated and grouped and formalized into the **9 exhibits** that follow:

ID	Severity	Addressed	Title
ALM-01S	Informational	Yes	Inexistent Event Emission
ALM-02S	Informational	Yes	Inexistent Sanitization of Input Addresses
ALM-03S	Informational	Yes	Inexistent Visibility Specifier
BDR-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
EYT-01S	Informational	Yes	Inexistent Sanitization of Input Address
MVM-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
OLO-01S	Informational	Acknowledged	Inexistent Sanitization of Input Addresses
OVO-01S	Informational	Acknowledged	Inexistent Sanitization of Input Addresses
VTD-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 restaking system.

As the project at hand implements a multi-operator restaking system, intricate care was put into ensuring that the **flow of funds 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 multiple significant vulnerabilities** within the system which could have had **moderate-to-severe ramifications** to its overall operation; for more information, kindly consult the relevant medium and major severity exhibits 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 a certain extent, however, we strongly recommend it to be expanded at certain complex points such as the intertwinement of the `MiddelwareVaultManager` and the balancer implementation.

A total of **60 findings** were identified over the course of the manual review of which **17 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
ACR-01M	Minor	Yes	Inexistent Validation of Validator Stake Relations
ACR-02M	Major	Yes	Inexistent Application of Access Control
ALM-01M	Unknown	Yes	Incorrect Erasure of Operator Locked Stake
ALM-02M	Informational	Yes	Inexistent Validation of Existence
ALM-03M	Minor	Yes	Improper Order of Conditionals

ALM-04M	Medium	Yes	Potential Underflow of Security Module Weight
ALM-05M	Major	Yes	Incorrect Function Overrides
BDR-01M	Informational	Yes	Inexistent Invocation of Access Control Initialization
BDR-02M	Medium	Yes	Insufficient Validation of Middleware
CST-01M	Minor	Acknowledged	Inexplicable & Improper Zero Management
ERC-01M	Minor	Acknowledged	Restrictive Decimal Offset Configuration
LRY-01M	Minor	Yes	Inexistent Access Control
MVM-01M	Medium	Yes	Insecure Casting of Asset Class ID
SDC-01M	Informational	Acknowledged	Inexistent Access Control of Read-Only Delegate Calls

VFY-01M	Medium	Acknowledged	Inexplicable Dynamicity of Deployments
VTD-01M	Minor	Acknowledged	Inconsistent Rounding Protection
VTD-02M	Medium	Yes	Inexistent Access Control of Delegator / Slasher Configurations

Code Style

During the manual portion of the audit, we identified **43 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
ACR-01C	Informational	Yes	Inefficient Asset Class Initialization
ACR-02C	Informational	Yes	Inefficient `mapping` Lookups
ACR-03C	Informational	Yes	Redundant Evaluations of `EnumerableSet::contains`
ALM-01C	Informational	Yes	Duplicate Imposition of Modifier
ALM-02C	Informational	Acknowledged	Inefficient Array Copy
ALM-03C	Informational	Acknowledged	Inefficient Integration of `EnumerableSet`
ALM-04C	Informational	Acknowledged	Inefficient Loop Limit Evaluations
ALM-05C	Informational	Acknowledged	Inefficient Vault Queries
BDR-01C	Informational	Acknowledged	Inefficient `mapping` Lookups
BDR-02C	Informational	Yes	Redundant Function Bodies

CST-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
CST-02C	Informational	Acknowledged	Inefficient Data Entry
CST-03C	Informational	Acknowledged	Optimization of Hint-Based Lookups
CST-04C	Informational	Acknowledged	Redundant Hint Encoding
DFY-01C	Informational	Acknowledged	Duplicate Implementation
DFY-02C	Informational	Yes	Unused Function Implementation
ERC-01C	Informational	Yes	Inefficient Function Implementation
EYT-01C	Informational	Yes	Redundant Function Body
LRY-01C	Informational	Yes	Duplicate Application of Sanitization
LRY-02C	Informational	Yes	Misleading Modifier Name
LRY-03C	Informational	Yes	Redundant Registration Check
LRD-01C	Informational	Yes	Inefficient `mapping` Lookups
MWT-01C	Informational	Yes	Misleading Error Message
MWT-02C	Informational	Acknowledged	Repetitive Value Literal

MWT-03C	Informational	Yes	Unused & Incorrect Bit Masks
MVM-01C	Informational	Acknowledged	Inefficient Integration of `EnumerableSet`
MVM-02C	Informational	Nullified	Inefficient Loop Limit Evaluation
OLO-01C	Informational	Yes	Inefficient Nonce Increases
OLO-02C	Informational	Yes	Inefficient `mapping` Lookups
OLO-03C	Informational	Yes	Repetitive Value Literals
ORY-01C	Informational	Yes	Inefficient Evaluation of Registration Status
OVO-01C	Informational	Yes	Inefficient Nonce Increases
OVO-02C	Informational	Yes	Inefficient `mapping` Lookups
OVO-03C	Informational	Yes	Repetitive Value Literals
SFY-01C	Informational	Yes	Unused Function Implementation
SCN-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
SCN-02C	Informational	Yes	Inefficient Loop Limit Evaluation
VFY-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics

VTD-01C	Informational	Yes	Deprecated Revert Pattern
VTD-02C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
VTD-03C	Informational	Yes	Inefficient Storage Entries
VTD-04C	Informational	Yes	Inefficient `mapping` Lookups
VTD-05C	Informational	Yes	Misleading Errors

AvalancheL1Middleware Static Analysis Findings

ALM-01S: Inexistent Event Emission

Type	Severity	Location
Language Specific	Informational	AvalancheL1Middleware.sol:L174-L178

Description:

The linked function adjusts a sensitive contract variable yet does not emit an event for it.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
174 function setVaultManager(
175     address vaultManager_
176 ) external onlyOwner {
177     vaultManager = MiddlewareVaultManager(vaultManager_);
178 }
```

Recommendation:

We advise an `event` to be declared and correspondingly emitted to ensure off-chain processes can properly react to this system adjustment.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The `VaultManagerUpdated` event was introduced to the codebase and is correspondingly emitted in the **`AvalancheL1Middleware::setVaultManager`** function, addressing this exhibit in full.

ALM-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	**AvalancheL1Middleware.sol**: • I-1: **L101-L126** • I-2: **L174-L178**

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

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/middleware/AvalancheL1Middleware.sol

SOL

```sol
174 function setVaultManager(
175     address vaultManager_
176 ) external onlyOwner {
177     vaultManager = MiddlewareVaultManager(vaultManager_);
178 }
```

Recommendation:

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

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

All input argument(s) of the **`AvalancheL1Middleware::constructor`**, and **`AvalancheL1Middleware::setVaultManager`** functions are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

ALM-03S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	AvalancheL1Middleware.sol:L68

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```
68  MiddlewareVaultManager vaultManager;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

BaseDelegator Static Analysis Findings

BDR-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	BaseDelegator.sol:L85-L100

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

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/delegator/BaseDelegator.sol

SOL

```sol
85  constructor(
86      address l1Registry,
87      address vaultFactory,
88      address operatorVaultOptInService,
89      address operatorL1OptInService,
90      address delegatorFactory,
91      uint64 entityType
92  ) {
93      _disableInitializers();
94      L1_REGISTRY = l1Registry;
```

Example (Cont.):

```sol
        VAULT_FACTORY = vaultFactory;
        OPERATOR_VAULT_OPT_IN_SERVICE = operatorVaultOptInService;
        OPERATOR_L1_OPT_IN_SERVICE = operatorL1OptInService;
        FACTORY = delegatorFactory;
        TYPE = entityType;
    }
```

Recommendation:

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

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

All input arguments of the `BaseDelegator::constructor` function are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

Entity Static Analysis Findings

EYT-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	**Entity.sol:L24-L29**

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/common/Entity.sol

SOL

```sol
24  constructor(address factory, uint64 type_) {
25      _disableInitializers();
26
27      FACTORY = factory;
28      TYPE = type_;
29  }
```

Recommendation:

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

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

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

MiddlewareVaultManager Static Analysis Findings

MVM-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	MiddlewareVaultManager.sol:L35-L38

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

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/middleware/MiddlewareVaultManager.sol

SOL

```sol
35  constructor(address vaultRegistry, address owner, address middlewareAddress)
Ownable(owner) {
36      VAULT_REGISTRY = vaultRegistry;
37      middleware = AvalancheL1Middleware(middlewareAddress);
38  }
```

Recommendation:

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

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

All input arguments of the `MiddlewareVaultManager::constructor` function are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

OperatorL1OptInService Static Analysis Findings

OLO-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	OperatorL1OptInService.sol:L52-L55

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

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/service/OperatorL1OptInService.sol

SOL

```sol
52  constructor(address whoRegistry, address whereRegistry, string memory name)
EIP712(name, "1") {
53      WHO_REGISTRY = whoRegistry;
54      WHERE_REGISTRY = whereRegistry;
55  }
```

Recommendation:

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

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

OperatorVaultOptInService Static Analysis Findings

OVO-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	OperatorVaultOptInService.sol:L51-L54

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

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/service/OperatorVaultOptInService.sol

SOL

```sol
51  constructor(address operatorRegistry, address vaultFactory, string memory name)
EIP712(name, "1") {
52      WHO_REGISTRY = operatorRegistry;
53      WHERE_REGISTRY = vaultFactory;
54  }
```

Recommendation:

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

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

VaultTokenized Static Analysis Findings

VTD-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	VaultTokenized.sol:L94-L100

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/vault/VaultTokenized.sol

SOL

```sol
94  constructor(
95      address vaultFactory
96  ) {
97      _disableInitializers();
98
99      FACTORY = vaultFactory;
100 }
```

Recommendation:

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

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

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

AssetClassRegistry Manual Review Findings

ACR-01M: Inexistent Validation of Validator Stake Relations

Type	Severity	Location
Input Sanitization	Minor	AssetClassRegistry.sol:L90, L91

Description:

The referenced variables are meant to configure minimum and maximum validator stakes yet do not enforce the relation that the minimum is lower or equal to the maximum.

Impact:

It is presently possible to configure an inoperable asset class.

Example:

src/contracts/middleware/AssetClassRegistry.sol

SOL

```sol
74  function _addAssetClass(
75      uint256 assetClassId,
76      uint256 minValidatorStake,
77      uint256 maxValidatorStake,
78      address initialAsset
79  ) internal {
80      if (assetClassIds.contains(assetClassId)) {
81          revert AssetClassRegistry__AssetClassAlreadyExists();
82      }
83      if (initialAsset == address(0)) {
```

Example (Cont.):

```sol
            revert AssetClassRegistry__InvalidAsset();
        }

        assetClassIds.add(assetClassId);

        AssetClass storage cls = assetClasses[assetClassId];
        cls.minValidatorStake = minValidatorStake;
        cls.maxValidatorStake = maxValidatorStake;

        emit AssetClassAdded(assetClassId, minValidatorStake, maxValidatorStake);

        _addAssetToClass(assetClassId, initialAsset);
    }
```

Recommendation:

We advise such a limitation to be imposed, ensuring all asset classes are correctly constructed.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced input validation has been applied as advised, restricting its enforcement solely to the `1` asset class ID as other asset classes are not meant to have a maximum validator stake.

ACR-02M: Inexistent Application of Access Control

Type	Severity	Location
Logical Fault	Major	AssetClassRegistry.sol:L38-L40

Description:

The `AssetClassRegistry::addAssetToClass` function, in contrast to its other counterparts, does not impose any access control and thus permits anyone to add assets to a particular class.

Impact:

It is presently possible for anyone to corrupt the asset-to-class relations due to absence of access control.

Example:

src/contracts/middleware/AssetClassRegistry.sol

SOL

```sol
38  function addAssetToClass(uint256 assetClassId, address asset) external {
39      _addAssetToClass(assetClassId, asset);
40  }
41
42  /// @inheritdoc IAssetClassRegistry
43  function removeAssetFromClass(uint256 assetClassId, address asset) external virtual
onlyOwner {
44      _removeAssetFromClass(assetClassId, asset);
45  }
```

Recommendation:

We advise proper access control to be imposed via the `Ownable::onlyOwner` modifier, preventing unauthorized modifications of the asset-to-class relations.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The `Ownable::onlyOwner` modifier has been properly applied to the **`AssetClassRegistry::addAssetToClass`** function, alleviating this exhibit.

AvalancheL1Middleware Manual Review Findings

ALM-01M: Incorrect Erasure of Operator Locked Stake

Type	Severity	Location
Logical Fault	Unknown	AvalancheL1Middleware.sol:L600, L715, L716

Description:

The `AvalancheL1Middleware::_calcAndCacheNodeWeightsForOperatorAtEpoch` function will reset the `operatorLockedStake` of an `operator` as long as a single `nodePendingUpdate` validation ID has been processed which contradicts the `operatorLockedStake` variable and its increment rather than overwrite per `AvalancheL1Middleware::_initializeValidatorWeightUpdateAndLock` function invocation.

Impact:

Although the assignment is incorrectly placed, it might ultimately function as expected given that all nodes pending an update are processed in the `AvalancheL1Middleware::_calcAndCacheNodeWeightsForOperatorAtEpoch` function. As such, this exhibit's severity will be re-assessed after the Suzaku Network team reviews it.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
function _calcAndCacheNodeWeightsForOperatorAtEpoch(address operator, uint48 epoch) internal {
    uint48 prevEpoch = (epoch == 0) ? 0 : epoch - 1;
    bytes32[] storage nodeArray = operatorNodesArray[operator];
    for (uint256 i = nodeArray.length; i > 0;) {
        i--;
        bytes32 nodeId = nodeArray[i];
        bytes32 valID = balancerValidatorManager.registeredValidators(abi.encodePacked(uint160(uint256(nodeId))));

        // If no removal/update, just carry over from prevEpoch (only if we haven't set it yet)
        if (!nodePendingRemoval[valID] && !nodePendingUpdate[valID]) {
```

Example (Cont.):

```sol
            if (nodeWeightCache[epoch][valID] == 0) {
                nodeWeightCache[epoch][valID] = nodeWeightCache[prevEpoch]
[valID];
            }
            continue;
        }

        if (
            nodePendingRemoval[valID] && nodeWeightCache[epoch][valID] == 0
                && nodeWeightCache[prevEpoch][valID] != 0
        ) {
            _removeNodeFromArray(operator, nodeId);
            nodePendingRemoval[valID] = false;
        }

        // If there was a pending update, finalize and clear the pending markers
        if (nodePendingUpdate[valID]) {
            nodePendingUpdate[valID] = false;
            operatorLockedStake[operator] = 0;
        }
    }
}
```

Recommendation:

We advise the `operatorLockedStake` reset to be per node rather than global or the assignment to be performed once after the `for` loop indicating that all nodes pending an update have been processed.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and clarified that the reset operation is intentional as the Balancer integration will apply the new weight right away, making the stake-weight effectively "live" and synchronized.

Nevertheless, the Suzaku Network team treated this as a potential optimization and relocated the erasure outside the `for` loop optimizing the code's gas cost.

ALM-02M: Inexistent Validation of Existence

Type	Severity	Location
Input Sanitization	Informational	AvalancheL1Middleware.sol:L193

Description:

The `AvalancheL1Middleware::activateSecondaryAssetClass` function will not validate whether the provided asset class ID already exists in the secondary asset classes.

Impact:

Although inconsequential, the current validation pattern of the `AvalancheL1Middleware::activateSecondaryAssetClass` function does not align with other implementations in the codebase.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
183 function activateSecondaryAssetClass(
184     uint256 assetClassId
185 ) external onlyOwner updateGlobalNodeWeightsOncePerEpoch {
186     if (!assetClassIds.contains(assetClassId)) {
187         revert AssetClassRegistry__AssetClassNotFound();
188     }
189     if (assetClassId == PRIMARY_ASSET_CLASS) {
190         revert AssetClassRegistry__AssetClassAlreadyExists();
191     }
192
```

Example (Cont.):

```sol
    secondaryAssetClasses.add(assetClassId);
}
```

Recommendation:

We advise such validation to be imposed via the return value of the `EnumerableSet::add` function, preventing redundant secondary asset class inclusions.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The activation of the secondary asset class has been streamlined in line with the rest of the codebase, mandating that the inclusion of the asset class ID is uniquely performed.

ALM-03M: Improper Order of Conditionals

Type	Severity	Location
Logical Fault	Minor	AvalancheL1Middleware.sol:L322-L325

Description:

The `AvalancheL1Middleware::addNode` function will validate the `newWeight` of the introduced node before restricting it based on the `maxStake` which is incorrect.

Impact:

Transactions that would end up below the `available` balance of the user after the `maxStake` limitation has been imposed would presently fail incorrectly.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
322 if (newWeight < minStake || newWeight > available) {
323     revert AvalancheL1Middleware__NotEnoughFreeStake(newWeight);
324 }
325 newWeight = (newWeight > maxStake) ? maxStake : newWeight;
```

Recommendation:

We advise the code's statements to be re-ordered, restricting the `newWeight` prior to validating it.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The code was updated to enforce the `maxStake` limitation prior to validating the `newStake` against the `available` one, alleviating this exhibit.

ALM-04M: Potential Underflow of Security Module Weight

Type	Severity	Location
Logical Fault	Medium	AvalancheL1Middleware.sol:L379, L996-L1004

Description:

The `AvalancheL1Middleware::_getOperatorAvailableStake` function may underflow if the security module maximum weight has been adjusted, resulting in several crucial functions such as the `AvalancheL1Middleware::forceUpdateNodes` function becoming inaccessible.

Impact:

A malicious operator might detect a security module maximum weight adjustment and force their `operatorLockedStake` to be high enough to cause revert errors after the update is applied.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
/**
 * @notice Returns the available stake for an operator
 * @param operator The operator address
 * @return The available stake
 */
function _getOperatorAvailableStake(
    address operator
) internal view returns (uint256) {
    uint48 epoch = getCurrentEpoch();
    uint256 totalStake = getOperatorStake(operator, epoch, PRIMARY_ASSET_CLASS);
```

Example (Cont.):

```sol
994
995         // Enforce max security module weight
996         (, uint64 securityModuleMaxWeight) =
balancerValidatorManager.getSecurityModuleWeights(address(this));
997         uint256 convertedSecurityModuleMaxWeight =
998             StakeConversion.weightToStake(securityModuleMaxWeight,
WEIGHT_SCALE_FACTOR);
999         if (totalStake > convertedSecurityModuleMaxWeight) {
100
0               totalStake = convertedSecurityModuleMaxWeight;
100
1           }
100
2
100
3           // Subtract locked stake
100
4           return totalStake - operatorLockedStake[operator];
100
5       }
```

Recommendation:

We advise the subtraction performed to be flooring, yielding `0` if an underflow would otherwise occur.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The code was updated to perform a saturating subtraction between the `operatorLockedStake` and the `totalStake`, ensuring that the subtraction will never underflow and will result `0` in case it would have.

ALM-05M: Incorrect Function Overrides

Type	Severity	Location
Logical Fault	Major	**AvalancheL1Middleware.sol**: • I-1: **L218-L231** • I-2: **L237-L245**

Description:

The referenced function overrides will re-implement asset class removal related functions from the `AssetClassRegistry` yet will bypass the `Ownable::onlyOwner` modifier imposed on it, permitting any user to remove any asset and asset class.

Impact:

Any user is able to remove an asset and asset class as long as it is not utilized which is invalid.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
213 /**
214  * @notice Removes an asset from an asset class, except primary asset
215  * @param assetClassId The ID of the asset class
216  * @param asset The address of the asset to remove
217  */
218 function removeAssetFromClass(
219     uint256 assetClassId,
220     address asset
221 ) external override updateGlobalNodeWeightsOncePerEpoch {
222     if (assetClassId == 1 && asset == PRIMARY_ASSET) {
```

Example (Cont.):

```sol
        revert AssetClassRegistry__AssetIsPrimaryAssetClass(assetClassId);
    }

    if (_isUsedAsset(assetClassId, asset)) {
        revert AvalancheL1Middleware__AssetStillInUse(assetClassId);
    }

    _removeAssetFromClass(assetClassId, asset);
}

/**
 * @notice Removes an asset class
 * @param assetClassId The asset class ID
 */
function removeAssetClass(
    uint256 assetClassId
) external override updateGlobalNodeWeightsOncePerEpoch {
    if (secondaryAssetClasses.contains(assetClassId)) {
        revert AvalancheL1Middleware__ActiveSecondaryAssetCLass(assetClassId);
    }

    _removeAssetClass(assetClassId);
}
```

Recommendation:

We advise the `AvalancheL1Middleware` overridden function implementations to invoke the `AssetClassRegistry` function variants via the `super` keyword, ensuring access control is properly imposed.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

Both instances were updated to invoke the `super` implementation of the relevant removal functions, however, the first instance also introduced the `Ownable::onlyOwner` modifier redundantly which we consider suboptimal.

In any case, the actual vulnerability has been addressed rendering this exhibit to be considered resolved.

BaseDelegator Manual Review Findings

BDR-01M: Inexistent Invocation of Access Control Initialization

Type	Severity	Location
Standard Conformity	Informational	BaseDelegator.sol:L240

Description:

While the present `AccessControlUpgradeable` implementation contains an empty initializer, it remains uninvoked in the **`BaseDelegator::_initialize`** function.

Example:

src/contracts/delegator/BaseDelegator.sol

SOL

```sol
231 function _initialize(
232     bytes calldata data
233 ) internal {
234     (address vault_, bytes memory data_) = abi.decode(data, (address, bytes));
235
236     if (!IRegistry(VAULT_FACTORY).isEntity(vault_)) {
237         revert BaseDelegator__NotVault();
238     }
239
240     __ReentrancyGuard_init();
```

Example (Cont.):

```sol

    vault = vault_;

    BaseParams memory baseParams = __initialize(vault_, data_);

    hook = baseParams.hook;

    if (baseParams.defaultAdminRoleHolder != address(0)) {
        _grantRole(DEFAULT_ADMIN_ROLE, baseParams.defaultAdminRoleHolder);
    }
    if (baseParams.hookSetRoleHolder != address(0)) {
        _grantRole(HOOK_SET_ROLE, baseParams.hookSetRoleHolder);
    }
}
```

Recommendation:

We advise it to be invoked so as to ensure any dependency updates do not break the contract's functionality as initialization code might be introduced at any point in the OpenZeppelin `AccessControlUpgradeable` implementation.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The `AccessControlUpgradeable::__AccessControl_init` initializer is properly invoked in the revised codebase, alleviating this exhibit.

BDR-02M: Insufficient Validation of Middleware

Type	Severity	Location
Logical Fault	Medium	BaseDelegator.sol:L162

Description:

The `BaseDelegator::setMaxL1Limit` function is meant to ensure that an L1 has been registered with a middleware, however, it fails to do so properly as it does not utilize the return value of the `L1Registry::isRegisteredWithMiddleware` function.

As the function will solely `revert` when the middleware is present and does not match the vault expected, it is possible for an `l1` to have a limit imposed by a vault that is not authorized if its middleware has not been defined yet.

Impact:

As long as an L1 has been registered and has not defined a middleware explicitly, it is possible to configure its `maxL1Limit` arbitrarily which will be enforced once a middleware has been registered for the L1.

This enables race conditions for newly introduced L1s who might have incorrect maximum L1 limits imposed.

Example:

src/contracts/delegator/BaseDelegator.sol

SOL

```sol
157 function setMaxL1Limit(address l1, uint96 assetClass, uint256 amount) external
nonReentrant {
158     if (!IL1Registry(L1_REGISTRY).isRegistered(l1)) {
159         revert BaseDelegator__NotL1();
160     }
161
162     IL1Registry(L1_REGISTRY).isRegisteredWithMiddleware(l1, msg.sender);
163
164     if (maxL1Limit[l1][assetClass] == amount) {
165         revert BaseDelegator__AlreadySet();
166     }
```

Example (Cont.):

```sol

    maxL1Limit[l1][assetClass] = amount;

    _setMaxL1Limit(l1, assetClass, amount);

    emit SetMaxL1Limit(l1, assetClass, amount);
}
```

Recommendation:

We advise the code to ensure the value yielded by the `L1Registry::isRegisteredWithMiddleware` function is `true`, preventing unauthorized impositions of `maxL1Limit` configurations.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The `IL1Registry::isRegisteredWithMiddleware` invocation result is now properly validated as `true` per our recommendation, addressing this exhibit.

Checkpoints Manual Review Findings

CST-01M: Inexplicable & Improper Zero Management

Type	Severity	Location
Logical Fault	Minor	Checkpoints.sol:L192-L194

Description:

The `ExtendedCheckpoints::push` variant for the `Trace256` structure will push an empty value to the `self._values` data entry without any justification as to this modification.

This will result in inconsistencies within the codebase as no such entry is introduced for the `self._trace` data entry, meaning that the first entry in the trace will incorrectly be considered "empty". Additionally, binary searches as well as hint based access that result in the index will result in misleading entries.

Impact:

The first data entry introduced to the `_trace` will be considered zero regardless of its underlying value for the `Trace256` data structure.

Example:

src/contracts/libraries/Checkpoints.sol

SOL

```sol
186 /**
187  * @dev Pushes a (`key`, `value`) pair into a Trace256 so that it is stored as the checkpoint.
188  *
189  * Returns previous value and new value.
190  */
191 function push(Trace256 storage self, uint48 key, uint256 value) internal returns (uint256, uint256) {
192     if (self._values.length == 0) {
193         self._values.push(0);
194     }
195
```

Example (Cont.):

```sol
    uint256 len = self._values.length;
    self._trace.push(key, uint208(len));
    self._values.push(value);

    return (self._values[len - 1], value);
}

/**
 * @dev Returns the value in the last (most recent) checkpoint with a key lower or equal than the search key, or zero
 * if there is none.
 */
function upperLookupRecent(Trace256 storage self, uint48 key) internal view returns (uint256) {
    uint208 idx = self._trace.upperLookupRecent(key);
    return idx > 0 ? self._values[idx] : 0;
}

/**
 * @dev Returns the value in the last (most recent) checkpoint with a key lower or equal than the search key, or zero
 * if there is none.
 *
 * NOTE: This is a variant of {upperLookupRecent} that can be optimized by getting the hint
 * (index of the checkpoint with a key lower or equal than the search key).
 */
function upperLookupRecent(Trace256 storage self, uint48 key, bytes memory hint_) internal view returns (uint256) {
    if (hint_.length == 0) {
        return upperLookupRecent(self, key);
    }

```

Example (Cont.):

SOL

```
224     uint32 hint = abi.decode(hint_, (uint32));
225     Checkpoint256 memory checkpoint = at(self, hint);
226     if (checkpoint._key == key) {
227         return checkpoint._value;
228     }
229
230     if (checkpoint._key < key && (hint == length(self) - 1 || at(self, hint +
1)._key > key)) {
231         return checkpoint._value;
232     }
233
234     return upperLookupRecent(self, key);
235 }
```

Recommendation:

We advise a proper entry to be introduced in the trace as well, ensuring consistency between the trace and value data entries.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and after consideration opted not to apply any changes to the `Checkpoints` codebase.

ERC4626Math Manual Review Findings

ERC-01M: Restrictive Decimal Offset Configuration

Type	Severity	Location
Logical Fault	Minor	ERC4626Math.sol:L58-L60

Description:

In contrast to the typical **EIP-4626** implementation by OpenZeppelin, the `ERC4626Math` library restricts the decimal offset to `0`.

This infers that first-deposit inflation attacks incur an expense equal to the lost value for the attacker, and thus can still constitute valid griefing pathways.

Impact:

The `ERC4626Math` implementation is inflexible in contrast to its OpenZeppelin implementation counterpart, potentially resulting in viable griefing attack paths.

Example:

src/contracts/libraries/ERC4626Math.sol

SOL

```sol
58  function _decimalsOffset() private pure returns (uint8) {
59      return 0;
60  }
```

Recommendation:

We advise either a non-zero decimal offset to be utilized or a decimal offset set as a function pointer / value argument for the relevant conversion functions, either of which we consider an acceptable alleviation of this exhibit.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

After consideration, the Suzaku Network team opted to acknowledge this issue as they believe that vaults can mitigate it by a deposit during their whitelist enforcement phase.

L1Registry Manual Review Findings

LRY-01M: Inexistent Access Control

Type	Severity	Location
Logical Fault	Minor	L1Registry.sol:L53-L71

Description:

The `L1Registry::registerL1` function permits any address to be registered as an L1 even though it might not be so.

Impact:

The registration of an L1 will solely incur gas cost for a malicious user, permitting them to broadcast several L1 registrations and thus to render the discovery of actual L1s by Suzaku operators difficult.

Example:

src/contracts/L1Registry.sol

SOL

```sol
52  /// @inheritdoc IL1Registry
53  function registerL1(
54      address l1,
55      address l1Middleware_,
56      string calldata metadataURL
57  ) external isZeroAddress(l1) onlyValidatorManagerOwner(l1) {
58      if (isRegistered(l1)) {
59          revert L1Registry__L1AlreadyRegistered();
60      }
61      if (l1 == address(0)) {
```

Example (Cont.):

```sol
            revert L1Registry__InvalidValidatorManager(l1);
        }
        l1s.add(l1);
        l1Middleware[l1] = l1Middleware_;
        l1MetadataURL[l1] = metadataURL;

        emit RegisterL1(l1);
        emit SetL1Middleware(l1, l1Middleware_);
        emit SetMetadataURL(l1, metadataURL);
    }
```

Recommendation:

We advise the code to impose some form of access control, for example via off-chain signature validation, to prevent the discovery of L1s by Suzaku operators from being polluted with invalid or malicious entries.

Alleviation (98eef72bcf):

The Suzaku Network team evaluated this exhibit and opted to impose a registration fee instead of restricting access.

We consider the native payment required for the registration of an L1 to be an adequate mitigation to the spamming attack scenario described.

Evaluation of the payment distribution code indicates that fund loss would occur if a `registerFee` has been configured with no `feeCollector`, rendering this exhibit to remain open.

We advise a disbursement of pending fees whenever the fee collector is set, preventing funds from being locked within the contract.

Alleviation (0e858f0f4f):

The code was refactored to ensure the configured fee collector is non-zero under all cases.

Additionally, an `unclaimedFees` variable was introduced to the contract that is consistently maintained in case any native fee transfer fails toward the configured fee recipient.

While the overall adjustments have been correctly applied, we would like to note that the state change at L231 is redundant as the ensuing `revert` would revert the contract's state to its original pre-transaction state in any case.

MiddlewareVaultManager Manual Review Findings

MVM-01M: Insecure Casting of Asset Class ID

Type	Severity	Location
Mathematical Operations	Medium	MiddlewareVaultManager.sol:L193

Description:

The `MiddlewareVaultManager::slashVault` function will insecurely cast the `assetClassId` to the `uint8` data type, permitting different vault stakes to be read and different asset classes to be ultimately slashed.

Impact:

Although a particular vault containing more than `255` asset classes is unrealistic, it could potentially be maliciously crafted into such a state redirecting slashing operations toward incorrect assets.

Example:

src/contracts/middleware/MiddlewareVaultManager.sol

SOL

```sol
168 function slashVault(
169     uint256 totalOperatorStake,
170     uint256 amount,
171     uint96 assetClassId,
172     address operator,
173     uint48 epochStartTs
174 ) external {
175     // Simple pro-rata slash
176     for (uint256 i; i < vaults.length(); ++i) {
177         (address vault, uint48 enabledTime, uint48 disabledTime) =
vaults.atWithTimes(i);
```

Example (Cont.):

```sol
        if (!_wasActiveAt(enabledTime, disabledTime, epochStartTs)) {
            continue;
        }

        if (vaultToAssetClass[vault] != assetClassId) {
            continue;
        }

        uint256 vaultStake =
BaseDelegator(IVaultTokenized(vault).delegator()).stakeAt(
            middleware.L1_VALIDATOR_MANAGER(), assetClassId, operator,
epochStartTs, new bytes(0)
        );

        if (vaultStake == 0) continue;

        uint256 slashAmt = (amount * vaultStake) / totalOperatorStake;
        _slashVault(epochStartTs, vault, uint8(assetClassId), operator, slashAmt);
    }
}
```

Recommendation:

We advise a safe cast to be performed either via explicit bound checking or the usage of a utility library.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

Slashing capability has been eliminated from the codebase, rendering this exhibit to have been alleviated indirectly.

StaticDelegateCallable Manual Review Findings

SDC-01M: Inexistent Access Control of Read-Only Delegate Calls

Type	Severity	Location
Logical Fault	Informational	StaticDelegateCallable.sol:L12-L18

Description:

The `StaticDelegateCallable::staticDelegateCall` function is meant to permit `delegatecall` operations that will not result in any state changes by reverting with the return data, however, certain unique instructions such as `selfdestruct` can halt execution with state changes without permitting a `revert`.

Impact:

While in most chains the current implementation should not result in a vulnerability, we still consider it best practice to impose access control.

Example:

src/contracts/common/StaticDelegateCallable.sol

SOL

```sol
function staticDelegateCall(address target, bytes calldata data) external {
    (bool success, bytes memory returndata) = target.delegatecall(data);
    bytes memory revertData = abi.encode(success, returndata);
    assembly {
        revert(add(32, revertData), mload(revertData))
    }
}
```

Recommendation:

While these interactions are no longer possible due to the updated `selfdestruct` operation code, we still advise access control to be imposed on the function to the callers that are expected to invoke it.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and opted to acknowledge it due to the inclusion of **EIP-6780** in Avalanche C-Chain's ACP-131 proposal.

VaultFactory Manual Review Findings

VFY-01M: Inexplicable Dynamicity of Deployments

Type	Severity	Location
Logical Fault	Medium	VaultFactory.sol:L120-L125

Description:

The deployment of a `VaultTokenized` permits arbitrary configurations to be specified and thus arbitrary delegator and slasher factories.

As such, a user might create a seemingly valid vault with a malicious slasher factory that permits them to slash stakes unfairly.

Impact:

It is presently possible to configure arbitrary slasher and delegator implementations in `VaultTokenized` instances which we consider insecure.

Example:

src/contracts/VaultFactory.sol

SOL

```sol
107 function create(
108     uint64 version,
109     address owner_,
110     bytes calldata data,
111     address delegatorFactory,
112     address slasherFactory
113 ) external returns (address entity_) {
114     // Ensure the version is not blacklisted
115     if (blacklisted[version]) {
116         revert MigratableFactory__VersionBlacklisted();
```

Example (Cont.):

```sol
    }

    // Validate factory addresses using ERC165.
    if (!delegatorFactory.supportsInterface(INTERFACE_ID_IDELEGATOR_FACTORY)) {
        revert MigratableFactory__InvalidImplementation();
    }
    if (!slasherFactory.supportsInterface(INTERFACE_ID_ISLASHER_FACTORY)) {
        revert MigratableFactory__InvalidImplementation();
    }

    // Deploy a new MigratableEntityProxy using CREATE2 for deterministic address
    entity_ = address(
        new MigratableEntityProxy{
            salt: keccak256(abi.encode(totalEntities(), version, owner_, data, delegatorFactory, slasherFactory))
        }(
            implementation(version),
            abi.encodeCall(IVaultTokenized.initialize, (version, owner_, data, delegatorFactory, slasherFactory))
        )
    );

    _addEntity(entity_);
}
```

Recommendation:

We advise the code to utilize protocol-provided factories so as to avoid any obfuscation attacks that result in fund loss via rug-pull tactics.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and opted to acknowledge it as prospective users should be able to determine whether they use the correct factories on their own.

VaultTokenized Manual Review Findings

VTD-01M: Inconsistent Rounding Protection

Type	Severity	Location
Mathematical Operations	Minor	VaultTokenized.sol:L631, L645-L648

Description:

The slashing mechanism implemented in the `VaultTokenized::onSlash` function will protect against an upward rounding error for the withdrawals being slashed, however, this rounding error accommodation will not be applied between the active stake slashed and the next withdrawal's amount slashed.

Impact:

It is presently possible for a slashing operation to fail incorrectly due to rounding the active stake slashed downwards and thus the withdrawal stakes upwards.

Example:

src/contracts/vault/VaultTokenized.sol

SOL

```
626 if (captureEpoch == currentEpoch_) {
627     uint256 slashableStake = activeStake_ + nextWithdrawals;
628     slashedAmount = Math.min(amount, slashableStake);
629     if (slashedAmount > 0) {
630         uint256 activeSlashed = slashedAmount.mulDiv(activeStake_, slashableStake);
631         uint256 nextWithdrawalsSlashed = slashedAmount - activeSlashed;
632
633         vs._activeStake.push(Time.timestamp(), activeStake_ - activeSlashed);
634         vs.withdrawals[captureEpoch + 1] = nextWithdrawals -
nextWithdrawalsSlashed;
635     }
```

Example (Cont.):

```sol
} else {
    uint256 withdrawals_ = vs.withdrawals[currentEpoch_];
    uint256 slashableStake = activeStake_ + withdrawals_ + nextWithdrawals;
    slashedAmount = Math.min(amount, slashableStake);
    if (slashedAmount > 0) {
        uint256 activeSlashed = slashedAmount.mulDiv(activeStake_, slashableStake);
        uint256 nextWithdrawalsSlashed = slashedAmount.mulDiv(nextWithdrawals, slashableStake);
        uint256 withdrawalsSlashed = slashedAmount - activeSlashed - nextWithdrawalsSlashed;

        if (withdrawals_ < withdrawalsSlashed) {
            nextWithdrawalsSlashed += withdrawalsSlashed - withdrawals_;
            withdrawalsSlashed = withdrawals_;
        }

        vs._activeStake.push(Time.timestamp(), activeStake_ - activeSlashed);
        vs.withdrawals[currentEpoch_ + 1] = nextWithdrawals - nextWithdrawalsSlashed;
        vs.withdrawals[currentEpoch_] = withdrawals_ - withdrawalsSlashed;
    }
}
```

Recommendation:

We advise rounding errors to be accommodated for in both of the referenced instances between the `nextWithdrawalsSlashed` and the `activeSlashed`, ensuring that slashing operations always succeed regardless of rounding errors.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and opted to acknowledge it as they do not intend to utilize slashing in the current implementation.

VTD-02M: Inexistent Access Control of Delegator / Slasher Configurations

Type	Severity	Location
Logical Fault	Medium	**VaultTokenized.sol**: • I-1: **L736-L759** • I-2: **L761-L786**

Description:

The `VaultTokenized::setDelegator` and `VaultTokenized::setSlasher` functions do not impose any access control, permitting any user to invoke them as long as the delegator configured has been deployed via the relevant factory with no restrictions whatsoever in the slasher configuration.

Impact:

It is possible to corrupt the delegator and slasher entries of a `VaultTokenized` that has just been deployed.

Example:

src/contracts/vault/VaultTokenized.sol

SOL

```sol
736 function setDelegator(
737     address delegator_
738 ) external nonReentrant {
739     VaultStorageStruct storage vs = _vaultStorage();
740
741     if (vs.isDelegatorInitialized) {
742         revert Vault__DelegatorAlreadyInitialized();
743     }
744
745     // replace by IDelegatorFactory
```

Example (Cont.):

```sol
    if (!IDelegatorFactory(vs.DELEGATOR_FACTORY).isEntity(delegator_)) {
        revert Vault__NotDelegator();
    }

    if (IBaseDelegator(delegator_).vault() != address(this)) {
        revert Vault__InvalidDelegator();
    }

    vs.delegator = delegator_;

    vs.isDelegatorInitialized = true;

    emit SetDelegator(delegator_);
}

function setSlasher(
    address slasher_
) external nonReentrant {
    VaultStorageStruct storage vs = _vaultStorage();

    if (vs.isSlasherInitialized) {
        revert Vault__SlasherAlreadyInitialized();
    }

    // replace by ISlasherFactory
    if (slasher_ != address(0)) {
        if (!IRegistry(vs.SLASHER_FACTORY).isEntity(slasher_)) {
            revert Vault__NotSlasher();
```

Example (Cont.):

```sol
        }

        if (IBaseSlasher(slasher_).vault() != address(this)) {
            revert Vault__InvalidSlasher();
        }

        vs.slasher = slasher_;
    }

    vs.isSlasherInitialized = true;

    emit SetSlasher(slasher_);
}
```

Recommendation:

We advise the code to impose proper access control as it is presently possible to configure arbitrary delegators/slashers and to prevent the configuration of a valid slasher by setting a zero-address slasher.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The delegator and slasher configurations have been properly restricted via the `DEFAULT_ADMIN_ROLE`, alleviating this exhibit in full.

AssetClassRegistry Code Style Findings

ACR-01C: Inefficient Asset Class Initialization

Type	Severity	Location
Gas Optimization	Informational	AssetClassRegistry.sol:L83-L85, L95, L99-L104, L107-

Description:

The initialization of an asset class will introduce an initial asset which will redundantly validate that the asset class ID exists, the initial asset is non-zero, and that the initial asset has not already been added to the newly created asset class.

Example:

src/contracts/middleware/AssetClassRegistry.sol

SOL

```sol
function _addAssetClass(
    uint256 assetClassId,
    uint256 minValidatorStake,
    uint256 maxValidatorStake,
    address initialAsset
) internal {
    if (assetClassIds.contains(assetClassId)) {
        revert AssetClassRegistry__AssetClassAlreadyExists();
    }
    if (initialAsset == address(0)) {
```

Example (Cont.):

```sol
        revert AssetClassRegistry__InvalidAsset();
    }

    assetClassIds.add(assetClassId);

    AssetClass storage cls = assetClasses[assetClassId];
    cls.minValidatorStake = minValidatorStake;
    cls.maxValidatorStake = maxValidatorStake;

    emit AssetClassAdded(assetClassId, minValidatorStake, maxValidatorStake);

    _addAssetToClass(assetClassId, initialAsset);
}

function _addAssetToClass(uint256 assetClassId, address asset) internal {
    if (!assetClassIds.contains(assetClassId)) {
        revert AssetClassRegistry__AssetClassNotFound();
    }
    if (asset == address(0)) {
        revert AssetClassRegistry__InvalidAsset();
    }

    AssetClass storage cls = assetClasses[assetClassId];
    if (cls.assets.contains(asset)) {
        revert AssetClassRegistry__AssetAlreadyRegistered();
    }
    cls.assets.add(asset);

```

Example (Cont.):

```sol
    emit AssetAdded(assetClassId, asset);
}
```

Recommendation:

We advise the `AssetClassRegistry::_addAssetClass` function to manually add the asset to the asset class without any validation as all conditions are effectively implied due to the asset class being new.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The code was revised as advised, optimizing the new asset class inclusion flow without affecting the asset inclusion to an existing class.

ACR-02C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	AssetClassRegistry.sol:L71

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

src/contracts/middleware/AssetClassRegistry.sol

SOL

```sol
71  return (assetClasses[assetClassId].minValidatorStake,
assetClasses[assetClassId].maxValidatorStake);
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

All referenced inefficient `mapping` lookups have been optimized to the greatest extent possible, significantly reducing the gas cost of the functions the statements were located in.

ACR-03C: Redundant Evaluations of `EnumerableSet::contains`

Type	Severity	Location
Gas Optimization	Informational	**AssetClassRegistry.sol**: • I-1: **L80**, **L87** • I-2: **L107**, **L110** • I-3: **L121**, **L124** • I-4: **L136**, **L144**

Description:

The referenced function invocation pairs are inefficient as the `EnumerableSet::add` and `EnumerableSet::remove` function calls will yield a boolean indicating whether the operation was successful, permitting it to be utilized in place of an existence / inexistence check.

Example:

src/contracts/middleware/AssetClassRegistry.sol

SOL

```
136 if (!assetClassIds.contains(assetClassId)) {
137     revert AssetClassRegistry__AssetClassNotFound();
138 }
139
140 if (assetClasses[assetClassId].assets.length() != 0) {
141     revert AssetClassRegistry__AssetsStillExist();
142 }
143
144 assetClassIds.remove(assetClassId);
```

Recommendation:

We advise these optimizations to be applied, significantly reducing the gas cost of all referenced functions.

Alleviation (98eef72bcf):

This particular optimization was marked as applied in its dedicated GitHub thread yet it cannot be observed as fixed in the codebase, rendering it to be partially addressed.

Alleviation (0e858f0f4f):

The optimization has been applied in full in the revised codebase, addressing this exhibit.

AvalancheL1Middleware Code Style Findings

ALM-01C: Duplicate Imposition of Modifier

Type	Severity	Location
Gas Optimization	Informational	AvalancheL1Middleware.sol:L352, L609

Description:

The referenced applications of the `AvalancheL1Middleware::onlyRegisteredOperatorNode` modifier overlap.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
350 function removeNode(
351     bytes32 nodeId
352 ) external updateGlobalNodeWeightsOncePerEpoch
onlyRegisteredOperatorNode(msg.sender, nodeId) {
353     _removeNode(msg.sender, nodeId);
354 }
```

Recommendation:

We advise one of the two referenced instances to be omitted, optimizing the code's gas cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The duplicate instance of the `modifier` in the `internal` function has been omitted, optimizing the code's gas cost.

ALM-02C: Inefficient Array Copy

Type	Severity	Location
Gas Optimization	Informational	AvalancheL1Middleware.sol:L915, L924, L928-L931

Description:

The `AvalancheL1Middleware::getActiveNodesForEpoch` function will copy values between arrays in an attempt to yield a properly-sized array to its caller.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
905 function getActiveNodesForEpoch(
906     address operator,
907     uint48 epoch
908 ) external view returns (bytes32[] memory activeNodeIds) {
909     uint48 epochStartTs = getEpochStartTs(epoch);
910
911     // Gather all nodes from the never-removed set
912     bytes32[] memory allNodeIds = operatorNodes[operator].values();
913
914     bytes32[] memory temp = new bytes32[](allNodeIds.length);
```

Example (Cont.):

```sol
    uint256 activeCount;

    for (uint256 i = 0; i < allNodeIds.length; i++) {
        bytes32 nodeId = allNodeIds[i];
        bytes32 validationID =
balancerValidatorManager.registeredValidators(abi.encodePacked(uint160(uint256(nodeId
))));
        Validator memory validator =
balancerValidatorManager.getValidator(validationID);

        if (_wasActiveAt(uint48(validator.startedAt), uint48(validator.endedAt),
epochStartTs)) {
            temp[activeCount++] = nodeId;
        }
    }

    activeNodeIds = new bytes32[](activeCount);
    for (uint256 j = 0; j < activeCount; j++) {
        activeNodeIds[j] = temp[j];
    }
}
```

Recommendation:

We advise the array length of the `temp` variable to be mutated in-memory using `assembly`; an approach that is considered secure as it reduces the memory consumption of the function.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

ALM-03C: Inefficient Integration of `EnumerableSet`

Type	Severity	Location
Gas Optimization	Informational	**AvalancheL1Middleware.sol**: • I-1: **L202**, **L210** • I-2: **L253**, **L263**

Description:

The referenced statement combinations will evaluate the existence or absence of an entry via an `EnumerableSet::contains` call explicitly instead of utilizing the return arguments of the inclusion / removal functions.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
199 function deactivateSecondaryAssetClass(
200     uint256 assetClassId
201 ) external onlyOwner updateGlobalNodeWeightsOncePerEpoch {
202     if (!secondaryAssetClasses.contains(assetClassId)) {
203         revert AssetClassRegistry__AssetClassNotFound();
204     }
205
206     if (_isUsedAssetClass(assetClassId)) {
207         revert AvalancheL1Middleware__AssetStillInUse(assetClassId);
208     }
```

Example (Cont.):

```sol

    secondaryAssetClasses.remove(assetClassId);
}
```

Recommendation:

We advise the return arguments to be utilized, optimizing the code's gas cost.

Alleviation (98eef72bcf):

Only the first of the two instances has been properly optimized, rendering the exhibit partially addressed.

Alleviation (0e858f0f4f):

The Suzaku Network team has opted to acknowledge this exhibit's remaining instance.

ALM-04C: Inefficient Loop Limit Evaluations

Type	Severity	Location
Gas Optimization	Informational	**AvalancheL1Middleware.sol**: • I-1: **L544** • I-2: **L555** • I-3: **L558** • I-4: **L576** • I-5: **L762** • I-6: **L779**

Description:

The linked `for` loops evaluate their limit inefficiently on each iteration.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
544 for (uint256 i = 0; i < operators.length(); i++) {
```

Recommendation:

We advise the statements within the `for` loop limits to be relocated outside to a local variable declaration that is consequently utilized for the evaluations to significantly reduce the codebase's gas cost. We should note the same optimization is applicable for storage reads present in those limits as they are newly read on each iteration (i.e. `length` members of arrays in storage).

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

ALM-05C: Inefficient Vault Queries

Type	Severity	Location
Gas Optimization	Informational	**AvalancheL1Middleware.sol**: • I-1: **L763** • I-2: **L780**

Description:

The referenced statements are meant to solely extract the `vault` entry at the specified index yet read several redundant variables.

Example:

src/contracts/middleware/AvalancheL1Middleware.sol

SOL

```sol
761 function _isUsedAsset(uint256 assetClassId, address asset) internal view returns
(bool) {
762     for (uint256 i; i < vaultManager.getVaultCount(); ++i) {
763         (address vault,,) = vaultManager.getVaultAtWithTimes(i);
764         if (vaultManager.vaultToAssetClass(vault) == assetClassId &&
IVaultTokenized(vault).collateral() == asset) {
765             return true;
766         }
767     }
768     return false;
769 }
```

Recommendation:

We advise a dedicated getter function to be introduced in the `MiddlewareVaultManager` implementation, optimizing the code's gas cost significantly.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

BaseDelegator Code Style Findings

BDR-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	BaseDelegator.sol:L164, L168

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

src/contracts/delegator/BaseDelegator.sol

SOL

```sol
164 if (maxL1Limit[l1][assetClass] == amount) {
165     revert BaseDelegator__AlreadySet();
166 }
167
168 maxL1Limit[l1][assetClass] = amount;
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (98eef72bcf):

The inefficient `mapping` lookup has not been optimized as the interim `maxL1Limit[l1]` lookup is not cached in memory, rendering this exhibit partially addressed.

Alleviation (0e858f0f4f):

The Suzaku Network team has opted to acknowledge this optimization.

BDR-02C: Redundant Function Bodies

Type	Severity	Location
Standard Conformity	Informational	**BaseDelegator.sol**: • I-1: **L262** • I-2: **L264** • I-3: **L266** • I-4: **L268**

Description:

The referenced function declarations are meant to be overridden in derivative implementations of the abstract `BaseDelegator` contract yet contain empty function bodies.

Example:

src/contracts/delegator/BaseDelegator.sol

SOL

```sol
256 function _stakeAt(
257     address l1,
258     uint96 assetClass,
259     address operator,
260     uint48 timestamp,
261     bytes memory hints
262 ) internal view virtual returns (uint256, bytes memory) {}
263
264 function _stake(address l1, uint96 assetClass, address operator) internal view
virtual returns (uint256) {}
265
```

Example (Cont.):

```sol
function _setMaxL1Limit(address l1, uint96 assetClass, uint256 amount) internal virtual {}

function __initialize(address vault_, bytes memory data) internal virtual returns (BaseParams memory) {}
```

Recommendation:

We advise the function bodies to be omitted, ensuring that derivative implementations `override` all functions expected.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

All referenced function bodies have been omitted as advised, standardizing the contract's code.

Checkpoints Code Style Findings

CST-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**Checkpoints.sol**: • I-1: **L72** • I-2: **L133** • I-3: **L230** • I-4: **L291**

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

src/contracts/libraries/Checkpoints.sol

SOL

```
291 if (checkpoint._key < key && (hint == length(self) - 1 || at(self, hint + 1)._key > key)) {
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and after consideration opted not to apply any changes to the `Checkpoints` codebase.

CST-02C: Inefficient Data Entry

Type	Severity	Location
Gas Optimization	Informational	Checkpoints.sol:L33

Description:

The `Checkpoint256` data entry contains a `uint48` key which will occupy a full 256-bit slot.

Example:

src/contracts/libraries/Checkpoints.sol

SOL

```sol
32  struct Checkpoint256 {
33      uint48 _key;
34      uint256 _value;
35  }
```

Recommendation:

We advise it to be upcast to the 256-bit data type, optimizing the code's gas cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and after consideration opted not to apply any changes to the `Checkpoints` codebase.

CST-03C: Optimization of Hint-Based Lookups

Type	Severity	Location
Gas Optimization	Informational	**Checkpoints.sol**: • I-1: **L61-L77** • I-2: **L118-L138** • I-3: **L219-L235** • I-4: **L276-L296**

Description:

The referenced functions expand the OpenZeppelin checkpointing system to introduce a hint-based lookups, avoiding a binary search if the hint provided correlates to the entry being searched.

These implementations will presently evaluate the key and if it does not match, they will commence a full binary search for the relevant entry.

Example:

src/contracts/libraries/Checkpoints.sol

SOL

```sol
276 function upperLookupRecentCheckpoint(
277     Trace256 storage self,
278     uint48 key,
279     bytes memory hint_
280 ) internal view returns (bool, uint48, uint256, uint32) {
281     if (hint_.length == 0) {
282         return upperLookupRecentCheckpoint(self, key);
283     }
284
285     uint32 hint = abi.decode(hint_, (uint32));
```

Example (Cont.):

```sol
    Checkpoint256 memory checkpoint = at(self, hint);
    if (checkpoint._key == key) {
        return (true, checkpoint._key, self._values[checkpoint._value], hint);
    }

    if (checkpoint._key < key && (hint == length(self) - 1 || at(self, hint + 1)._key > key)) {
        return (true, checkpoint._key, self._values[checkpoint._value], hint);
    }

    return upperLookupRecentCheckpoint(self, key);
}
```

Recommendation:

The lookups can be further optimized by initiating the binary search with different upper or lower bounds based on the hint, narrowing down the binary search area and thus significantly reducing the gas cost involved in lookups.

As the hint will most likely be very close to the desired entry, using it as either the upper or lower bound will result in a reduction in the total iterations required to find the relevant data entry.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and after consideration opted not to apply any changes to the `Checkpoints` codebase.

CST-04C: Redundant Hint Encoding

Type	Severity	Location
Gas Optimization	Informational	**Checkpoints.sol**: • I-1: **L61** • I-2: **L121** • I-3: **L219** • I-4: **L279**

Description:

The referenced statements highlight that the `hint_` meant for a particular lookup is passed in as a `bytes memory` argument inefficiently, ultimately decoded to the `uint32` index of the checkpoint list.

Example:

src/contracts/libraries/Checkpoints.sol

SOL

```sol
276 function upperLookupRecentCheckpoint(
277     Trace256 storage self,
278     uint48 key,
279     bytes memory hint_
280 ) internal view returns (bool, uint48, uint256, uint32) {
281     if (hint_.length == 0) {
282         return upperLookupRecentCheckpoint(self, key);
283     }
284
285     uint32 hint = abi.decode(hint_, (uint32));
```

Recommendation:

We advise a `uint32` argument to be utilized instead, optimizing the code's gas cost.

If a flag value is desired to indicate no hint has been provided, the code can utilize the maximum value of a `uint32` (`type(uint32).max`).

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit and after consideration opted not to apply any changes to the `Checkpoints` codebase.

DelegatorFactory Code Style Findings

DFY-01C: Duplicate Implementation

Type	Severity	Location
Standard Conformity	Informational	DelegatorFactory.sol:L15

Description:

The `DelegatorFactory` implementation is identical to the `SlasherFactory` implementation.

Example:

src/contracts/DelegatorFactory.sol

SOL

```sol
15  contract DelegatorFactory is IDelegatorFactory, Ownable, ERC165 {
```

Recommendation:

We advise both factories to inherit from the same base implementation, optimizing the project's structure.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

DFY-02C: Unused Function Implementation

Type	Severity	Location
Gas Optimization	Informational	DelegatorFactory.sol:L137-L143

Description:

The referenced function implementation remains unused within the codebase.

Example:

src/contracts/DelegatorFactory.sol

SOL

```sol
function _checkEntity(
    address account
) internal view {
    if (!isEntity(account)) {
        revert DelegatorFactory__EntityNotExist();
    }
}
```

Recommendation:

We advise it to be omitted, optimizing the contract's bytecode size.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced function has been omitted from the codebase as advised.

ERC4626Math Code Style Findings

ERC-01C: Inefficient Function Implementation

Type	Severity	Location
Gas Optimization	Informational	ERC4626Math.sol:L58-L60

Description:

The referenced function is a `pure` implementation that yields a fixed value and is `private`.

Example:

src/contracts/libraries/ERC4626Math.sol

SOL

```sol
function _decimalsOffset() private pure returns (uint8) {
    return 0;
}
```

Recommendation:

We advise its value to be relocated to a library-level `constant`, optimizing any function that would invoke it by replacing the invocation with the newly introduced `constant`.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The code was updated to utilize a `library` level `constant` instead, optimizing its gas cost.

Entity Code Style Findings

EYT-01C: Redundant Function Body

Type	Severity	Location
Standard Conformity	Informational	Entity.sol:L49-L51

Description:

The referenced function implementation is meant to be overridden by derivative implementations in the abstract `Entity` contract, however, it is declared with an empty function body.

Example:

src/contracts/common/Entity.sol

SOL

```sol
function _initialize(
    bytes calldata /* data */
) internal virtual {}
```

Recommendation:

We advise the function body declaration to be omitted, forcing derivative implementations to explicitly `override` the relevant function.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The function body of the referenced function has been omitted as advised, standardizing the contract's code.

L1Registry Code Style Findings

LRY-01C: Duplicate Application of Sanitization

Type	Severity	Location
Gas Optimization	Informational	L1Registry.sol:L57, L61-L63

Description:

The `L1Registry::registerL1` function will validate that the input `l1` is non-zero twice; once in the `L1Registry::isZeroAddress` modifier and once in the code itself.

Example:

src/contracts/L1Registry.sol

SOL

```sol
52  /// @inheritdoc IL1Registry
53  function registerL1(
54      address l1,
55      address l1Middleware_,
56      string calldata metadataURL
57  ) external isZeroAddress(l1) onlyValidatorManagerOwner(l1) {
58      if (isRegistered(l1)) {
59          revert L1Registry__L1AlreadyRegistered();
60      }
61      if (l1 == address(0)) {
```

Example (Cont.):

SOL

```
62          revert L1Registry__InvalidValidatorManager(l1);
63      }
64      l1s.add(l1);
65      l1Middleware[l1] = l1Middleware_;
66      l1MetadataURL[l1] = metadataURL;
67
68      emit RegisterL1(l1);
69      emit SetL1Middleware(l1, l1Middleware_);
70      emit SetMetadataURL(l1, metadataURL);
71  }
```

Recommendation:

We advise the `if` conditional in the code itself to be omitted, optimizing the code's gas cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The explicit `l1` address validation from the function has been omitted, optimizing its gas cost.

LRY-02C: Misleading Modifier Name

Type	Severity	Location
Code Style	Informational	L1Registry.sol:L43

Description:

The referenced modifier name implies that the argument is validated as zero when it is validated as non-zero.

Example:

src/contracts/L1Registry.sol

SOL

```sol
43  modifier isZeroAddress(
44      address l1
45  ) {
46      if (l1 == address(0)) {
47          revert L1Registry__InvalidValidatorManager(l1);
48      }
49      _;
50  }
```

Recommendation:

We advise it to be renamed to `notZeroAddress`, optimizing the modifier's legibility.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The `modifier` name has been revised per our recommendation, increasing the code's legibility.

LRY-03C: Redundant Registration Check

Type	Severity	Location
Gas Optimization	Informational	L1Registry.sol:L58, L64

Description:

The referenced statements will explicitly check the registration of an `l1` prior to including it to the `EnumerableSet` which is inefficient.

Example:

src/contracts/L1Registry.sol

SOL

```sol
58  if (isRegistered(l1)) {
59      revert L1Registry__L1AlreadyRegistered();
60  }
61  if (l1 == address(0)) {
62      revert L1Registry__InvalidValidatorManager(l1);
63  }
64  l1s.add(l1);
```

Recommendation:

We advise the return value of the `EnumerableSet::add` operation to be utilized instead in the referenced `if` conditional, optimizing the code's gas cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The redundant explicit registration check has been removed, utilizing the `EnumerableSet::add` function result instead.

L1RestakeDelegator Code Style Findings

LRD-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	**L1RestakeDelegator.sol**: • I-1: **L115**, **L118** • I-2: **L191**, **L193**

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

src/contracts/delegator/L1RestakeDelegator.sol

SOL

```sol
115 if (maxL1Limit[l1][assetClass] == 0) {
116     revert L1RestakeDelegator__MaxL1LimitNotSet();
117 }
118 if (amount > maxL1Limit[l1][assetClass]) {
119     revert L1RestakeDelegator__ExceedsMaxL1Limit();
120 }
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

All referenced inefficient `mapping` lookups have been optimized to the greatest extent possible, significantly reducing the gas cost of the functions the statements were located in.

MapWithTimeData Code Style Findings

MWT-01C: Misleading Error Message

Type	Severity	Location
Code Style	Informational	MapWithTimeData.sol:L30

Description:

The referenced error message implies that the specified `addr` is `NotEnabled` yet its condition indicates that it is either not enabled or it has already been disabled.

Example:

src/contracts/middleware/libraries/MapWithTimeData.sol

SOL

```sol
29  if (uint48(value) == 0 || uint48(value >> 48) != 0) {
30      revert NotEnabled();
31  }
```

Recommendation:

We advise the error message to be updated to indicate the actual status of the `addr` entry.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced conditional was broken into two distinct ones with explicit error messages, optimizing the code's legibility.

MWT-02C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	MapWithTimeData.sol:L18, L29, L33, L40, L55, L64

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

src/contracts/middleware/libraries/MapWithTimeData.sol

SOL

```
18  uint256 private constant DISABLED_TIME_MASK = 0xFFFFFFFFFFFFFFFFFFFFFFFF << 48;
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

MWT-03C: Unused & Incorrect Bit Masks

Type	Severity	Location
Code Style	Informational	MapWithTimeData.sol:L17-L18

Description:

The referenced bit masks are incorrect and remain unused in the codebase.

Example:

src/contracts/middleware/libraries/MapWithTimeData.sol

SOL

```sol
17  uint256 private constant ENABLED_TIME_MASK = 0xFFFFFFFFFFFFFFFFFFFFFFFF;
18  uint256 private constant DISABLED_TIME_MASK = 0xFFFFFFFFFFFFFFFFFFFFFFFF << 48;
```

Recommendation:

We advise them to be omitted or corrected, either of which we consider an acceptable alleviation of this exhibit.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The incorrect bitmasks have been removed per our recommendation.

MiddlewareVaultManager Code Style Findings

MVM-01C: Inefficient Integration of `EnumerableSet`

Type	Severity	Location
Gas Optimization	Informational	**MiddlewareVaultManager.sol**: • I-1: **L50**, **L66** • I-2: **L100**, **L114**

Description:

The referenced statement combinations will evaluate the existence or absence of an entry via an `EnumerableSet::contains` call explicitly instead of utilizing the return arguments of the inclusion / removal functions.

Example:

src/contracts/middleware/MiddlewareVaultManager.sol

SOL

```sol
50  if (vaults.contains(vault)) {
51      revert AvalancheL1Middleware__VaultAlreadyRegistered();
52  }
53
54  uint48 vaultEpoch = IVaultTokenized(vault).epochDuration();
55  address slasher = IVaultTokenized(vault).slasher();
56  if (slasher != address(0) && IEntity(slasher).TYPE() == VETO_SLASHER_TYPE) {
57      vaultEpoch -= IVetoSlasher(slasher).vetoDuration();
58  }
59  if (vaultEpoch < middleware.SLASHING_WINDOW()) {
```

Example (Cont.):

```sol
    revert AvalancheL1Middleware__VaultEpochTooShort();
}

vaultToAssetClass[vault] = assetClassId;
_setVaultMaxL1Limit(vault, assetClassId, vaultMaxL1Limit);

vaults.add(vault);
```

Recommendation:

We advise the return arguments to be utilized, optimizing the code's gas cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The Suzaku Network team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

MVM-02C: Inefficient Loop Limit Evaluation

Type	Severity	Location
Gas Optimization	Informational	MiddlewareVaultManager.sol:L176

Description:

The linked `for` loop evaluates its limit inefficiently on each iteration.

Example:

src/contracts/middleware/MiddlewareVaultManager.sol

SOL

```sol
176 for (uint256 i; i < vaults.length(); ++i) {
```

Recommendation:

We advise the statements within the `for` loop limit to be relocated outside to a local variable declaration that is consequently utilized for the evaluation to significantly reduce the codebase's gas cost. We should note the same optimization is applicable for storage reads present in such limits as they are newly read on each iteration (i.e. `length` members of arrays in storage).

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The exhibit is no longer applicable as the relevant code has been removed from the codebase.

OperatorL1OptInService Code Style Findings

OLO-01C: Inefficient Nonce Increases

Type	Severity	Location
Gas Optimization	Informational	**OperatorL1OptInService.sol**: • I-1: **L151** • I-2: **L169**

Description:

The `OperatorL1OptInService` will increase the nonce of a particular `who` and `where` combo regardless of whether a signature consumed the nonce or not.

Example:

src/contracts/service/OperatorL1OptInService.sol

SOL

```sol
156 function _optOut(address who, address where) internal {
157     (, uint48 latestTimestamp, uint208 latestValue) = _isOptedIn[who]
[where].latestCheckpoint();
158
159     if (latestValue == 0) {
160         revert OptInService__NotOptedIn();
161     }
162
163     if (latestTimestamp == Time.timestamp()) {
164         revert OptInService__OptOutCooldown();
165     }
```

Example (Cont.):

```sol
166
167     _isOptedIn[who][where].push(Time.timestamp(), 0);
168
169     _increaseNonce(who, where);
170
171     emit OptOut(who, where);
172 }
```

Recommendation:

We advise the nonce to be solely increased whenever a signature is consumed, optimizing the code's gas cost whenever a `who` member invokes the relevant opt functions directly.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The code of opting in and out was updated as advised, ensuring nonces are consumed solely when a signature is processed.

OLO-02C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	OperatorL1OptInService.sol:L157, L167

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

src/contracts/service/OperatorL1OptInService.sol

SOL

```sol
157 (, uint48 latestTimestamp, uint208 latestValue) = _isOptedIn[who]
[where].latestCheckpoint();
158
159 if (latestValue == 0) {
160     revert OptInService__NotOptedIn();
161 }
162
163 if (latestTimestamp == Time.timestamp()) {
164     revert OptInService__OptOutCooldown();
165 }
166
```

Example (Cont.):

```sol
167 _isOptedIn[who][where].push(Time.timestamp(), 0);
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (98eef72bcf):

While the first lookup was stored to a local `trace` variable, the second lookup does not utilize it rendering the optimization partially applied.

Alleviation (0e858f0f4f):

The second lookup has been optimized as well, addressing this exhibit.

OLO-03C: Repetitive Value Literals

Type	Severity	Location
Code Style	Informational	**OperatorL1OptInService.sol**: • I-1: **L66**, **L73**, **L149** • I-2: **L159**, **L167**

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

src/contracts/service/OperatorL1OptInService.sol

```sol
66  return _isOptedIn[who][where].upperLookupRecent(timestamp, hint) == 1;
```

Recommendation:

We advise each to be set to its dedicated `constant` variable instead, optimizing the legibility of the codebase.

In case some of the `constant` declarations have already been introduced, we advise them to be properly re-used across the code.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced value literals `1`, and `0` have all been relocated to contract-level `constant` declarations labelled `OPT_IN_VALUE`, and `OPT_OUT_VALUE` respectively, optimizing the code's legibility.

OperatorRegistry Code Style Findings

ORY-01C: Inefficient Evaluation of Registration Status

Type	Severity	Location
Gas Optimization	Informational	OperatorRegistry.sol:L25, L29

Description:

The `OperatorRegistry::registerOperator` function will evaluate the registration of the caller twice; once explicitly via the `OperatorRegistry::isRegistered` function and once via the return value of the `EnumerableSet::add` operation.

Example:

src/contracts/OperatorRegistry.sol

SOL

```sol
25  if (isRegistered(msg.sender)) {
26      revert OperatorRegistry__OperatorAlreadyRegistered();
27  }
28
29  operators.add(msg.sender);
```

Recommendation:

We advise the return value of the `EnumerableSet::add` function to be utilized instead, optimizing the code's gas cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The redundant explicit registration check has been removed, utilizing the `EnumerableSet::add` function result instead.

OperatorVaultOptInService Code Style Findings

OVO-01C: Inefficient Nonce Increases

Type	Severity	Location
Gas Optimization	Informational	**OperatorVaultOptInService.sol**: • I-1: **L149** • I-2: **L167**

Description:

The `OperatorVaultOptInService` will increase the nonce of a particular `who` and `where` combo regardless of whether a signature consumed the nonce or not.

Example:

src/contracts/service/OperatorVaultOptInService.sol

SOL

```sol
134 function _optIn(address who, address where) internal {
135     if (!IOperatorRegistry(WHO_REGISTRY).isRegistered(who)) {
136         revert OptInService__NotWho();
137     }
138
139     if (!IVaultFactory(WHERE_REGISTRY).isEntity(where)) {
140         revert OptInService__NotWhereEntity();
141     }
142
143     if (isOptedIn(who, where)) {
```

Example (Cont.):

```sol
        revert OptInService__AlreadyOptedIn();
    }

    _isOptedIn[who][where].push(Time.timestamp(), 1);

    _increaseNonce(who, where);

    emit OptIn(who, where);
}
```

Recommendation:

We advise the nonce to be solely increased whenever a signature is consumed, optimizing the code's gas cost whenever a `who` member invokes the relevant opt functions directly.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The code of opting in and out was updated as advised, ensuring nonces are consumed solely when a signature is processed.

OVO-02C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	OperatorVaultOptInService.sol:L155, L165

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

src/contracts/service/OperatorVaultOptInService.sol

SOL

```sol
155 (, uint48 latestTimestamp, uint208 latestValue) = _isOptedIn[who]
[where].latestCheckpoint();
156
157 if (latestValue == 0) {
158     revert OptInService__NotOptedIn();
159 }
160
161 if (latestTimestamp == Time.timestamp()) {
162     revert OptInService__OptOutCooldown();
163 }
164
```

Example (Cont.):

```sol
165 _isOptedIn[who][where].push(Time.timestamp(), 0);
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (98eef72bcf):

While the first lookup was stored to a local `trace` variable, the second lookup does not utilize it rendering the optimization partially applied.

Alleviation (0e858f0f4f):

The second lookup has been optimized as well, addressing this exhibit.

OVO-03C: Repetitive Value Literals

Type	Severity	Location
Code Style	Informational	**OperatorVaultOptInService.sol**: • I-1: **L65**, **L72**, **L147** • I-2: **L157**, **L165**

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

src/contracts/service/OperatorVaultOptInService.sol

SOL

```sol
65  return _isOptedIn[who][where].upperLookupRecent(timestamp, hint) == 1;
```

Recommendation:

We advise each to be set to its dedicated `constant` variable instead, optimizing the legibility of the codebase.

In case some of the `constant` declarations have already been introduced, we advise them to be properly re-used across the code.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced value literals `1`, and `0` have all been relocated to contract-level `constant` declarations labelled `OPT_IN_VALUE`, and `OPT_OUT_VALUE` respectively, optimizing the code's legibility.

SlasherFactory Code Style Findings

SFY-01C: Unused Function Implementation

Type	Severity	Location
Gas Optimization	Informational	SlasherFactory.sol:L137-L143

Description:

The referenced function implementation remains unused within the codebase.

Example:

src/contracts/SlasherFactory.sol

SOL

```sol
function _checkEntity(
    address account
) internal view {
    if (!isEntity(account)) {
        revert SlasherFactory__EntityNotExist();
    }
}
```

Recommendation:

We advise it to be omitted, optimizing the contract's bytecode size.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced function has been omitted from the codebase as advised.

StakeConversion Code Style Findings

SCN-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	StakeConversion.sol:L35

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

src/contracts/middleware/libraries/StakeConversion.sol

SOL

```sol
for (uint256 i = 0; i < arr.length; i++) {
    if (arr[i] == nodeId) {
        uint256 lastIndex = arr.length - 1;
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced subtraction has been wrapped in an `unchecked` code block, optimizing its gas cost.

SCN-02C: Inefficient Loop Limit Evaluation

Type	Severity	Location
Gas Optimization	Informational	StakeConversion.sol:L33

Description:

The linked `for` loop evaluates its limit inefficiently on each iteration.

Example:

src/contracts/middleware/libraries/StakeConversion.sol

SOL

```sol
33  for (uint256 i = 0; i < arr.length; i++) {
```

Recommendation:

We advise the statements within the `for` loop limit to be relocated outside to a local variable declaration that is consequently utilized for the evaluation to significantly reduce the codebase's gas cost. We should note the same optimization is applicable for storage reads present in such limits as they are newly read on each iteration (i.e. `length` members of arrays in storage).

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The loop limit is now cached outside its `for` construct, optimizing the code's gas cost.

VaultFactory Code Style Findings

VFY-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	VaultFactory.sol:L70

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

src/contracts/VaultFactory.sol

SOL

```sol
function implementation(
    uint64 version
) public view checkVersion(version) returns (address) {
    return _whitelistedImplementations.at(version - 1);
}
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced subtraction has been wrapped in an `unchecked` code block, optimizing its gas cost.

VaultTokenized Code Style Findings

VTD-01C: Deprecated Revert Pattern

Type	Severity	Location
Standard Conformity	Informational	VaultTokenized.sol:L226

Description:

The referenced statement will issue a `revert` statement without any error code.

Example:

src/contracts/vault/VaultTokenized.sol

SOL

```sol
224 function _migrate(uint64, /* newVersion */ bytes calldata /* data */ ) internal
virtual onlyInitializing {
225     // Implement migration logic here
226     revert();
227 }
```

Recommendation:

We advise an `error` to be declared signaling that migrations are not permitted in the current implementation.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The code was updated to yield the `Vault__MigrationNotImplemented` error, addressing this exhibit.

VTD-02C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	VaultTokenized.sol:L845

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

src/contracts/vault/VaultTokenized.sol

SOL

```sol
842 if (timestamp < vs.epochDurationInit) {
843     revert Vault__InvalidTimestamp();
844 }
845 return (timestamp - vs.epochDurationInit) / vs.epochDuration;
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced subtraction and division has been safely wrapped in an `unchecked` code block, optimizing its gas cost.

VTD-03C: Inefficient Storage Entries

Type	Severity	Location
Gas Optimization	Informational	VaultTokenized.sol:L118-L121

Description:

The referenced storage entries are assigned to value literals during the contract's initialization.

Example:

src/contracts/vault/VaultTokenized.sol

SOL

```sol
118 vs.DEPOSIT_WHITELIST_SET_ROLE = keccak256("DEPOSIT_WHITELIST_SET_ROLE");
119 vs.DEPOSITOR_WHITELIST_ROLE = keccak256("DEPOSITOR_WHITELIST_ROLE");
120 vs.IS_DEPOSIT_LIMIT_SET_ROLE = keccak256("IS_DEPOSIT_LIMIT_SET_ROLE");
121 vs.DEPOSIT_LIMIT_SET_ROLE = keccak256("DEPOSIT_LIMIT_SET_ROLE");
```

Recommendation:

We advise them to be set as contract-level `constant` declarations, optimizing the code's gas cost.

Alleviation (98eef72bcf):

The Suzaku Network team clarified that they deliberately store those variables in storage for upgradeability.

We would like to note that `constant` variables behave correctly in upgradeable contracts, rendering our recommendation to remain valid.

Alleviation (0e858f0f4f):

The referenced variables have been set as `constant` declarations, greatly optimizing the code's gas cost as originally advised.

VTD-04C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	VaultTokenized.sol:L821, L831

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

src/contracts/vault/VaultTokenized.sol

SOL

```sol
821 if (vs.isWithdrawalsClaimed[epoch][msg.sender]) {
822     revert Vault__AlreadyClaimed();
823 }
824
825 amount = withdrawalsOf(epoch, msg.sender);
826
827 if (amount == 0) {
828     revert Vault__InsufficientClaim();
829 }
830
```

Example (Cont.):

```sol
831 vs.isWithdrawalsClaimed[epoch][msg.sender] = true;
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (98eef72bcf):

The inefficient `mapping` lookup has not been optimized as the interim `isWithdrawalsClaimed[epoch]` lookup is not cached in memory, rendering this exhibit partially addressed.

Alleviation (0e858f0f4f):

The referenced `mapping` lookups have been optimized per our recommendation, utilizing an interim `mapping(address => bool) storage` local variable.

VTD-05C: Misleading Errors

Type	Severity	Location
Code Style	Informational	**VaultTokenized.sol**: • I-1: **L159** • I-2: **L169**

Description:

The referenced `Vault__MissingRoles` errors will be yielded when evaluating entirely different conditions (i.e. a role has been defined when it should not be, or a limit has been imposed without deposit limitations being enabled).

Example:

src/contracts/vault/VaultTokenized.sol

SOL

```sol
152 if (params.defaultAdminRoleHolder == address(0)) {
153     if (params.depositWhitelistSetRoleHolder == address(0)) {
154         if (params.depositWhitelist) {
155             if (params.depositorWhitelistRoleHolder == address(0)) {
156                 revert Vault__MissingRoles();
157             }
158         } else if (params.depositorWhitelistRoleHolder != address(0)) {
159             revert Vault__MissingRoles();
160         }
161     }
```

Example (Cont.):

```sol

    if (params.isDepositLimitSetRoleHolder == address(0)) {
        if (params.isDepositLimit) {
            if (params.depositLimit == 0 && params.depositLimitSetRoleHolder == address(0)) {
                revert Vault__MissingRoles();
            }
        } else if (params.depositLimit != 0 || params.depositLimitSetRoleHolder != address(0)) {
            revert Vault__MissingRoles();
        }
    }
}
```

Recommendation:

We advise proper `error` declarations to be yielded by the referenced statements, optimizing the code's legibility.

Alleviation (98eef72bcf41d8ede9d070fa4f2213f2b2ae67d3):

The referenced error messages have been updated to indicate inconsistent roles rather than missing ones, addressing this exhibit.

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.