OMNISCIA

December 28, 2024

# SMART CONTRACT AUDIT REPORT

Perpetual Airdrop
Token One

omniscia.io
info@omniscia.io
Online report: perpetual-airdrop-token-one

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: perpetual-airdrop-token-one

# Token One Security Audit

## Audit Report Revisions

| Commit Hash | Date | Audit Report Hash |
| --- | --- | --- |
| 1bb58ccd22 | December 19th 2024 | 3c44649f80 |
| a5e8ff5ca3 | December 20th 2024 | 03f464c931 |
| 9fe8a4a3fc | December 28th 2024 | ae1ade8a7c |

# Audit Overview

We were tasked with performing an audit of the Perpetual Airdrop codebase and in particular their Token One airdrop infrastructure.

The system implements a novel token implementation meant to support a perpetual airdrop scheme via usage of Chainlink's v2.5 VRF system.

Over the course of the audit, we identified a significant access control flaw in the way randomness requests are fulfilled as well as a governance manipulation attack due to improper accounting of votes.

To note, the contract operates close to block gas limitations and should be configured securely by the Perpetual Airdrop team to ensure that it cannot ever breach the block gas limit of the network the system is deployed to.

We advise the Perpetual Airdrop 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 Perpetual Airdrop 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 Perpetual Airdrop and have identified that a single exhibit has not been adequately dealt with. We advise the Perpetual Airdrop team to revisit the following exhibit: **TOG-04M**

Additionally, the following `informational` findings remain unaddressed and should be revisited: **TOG-03S**, **TOG-01C**, **TOE-01C**

Finally, we observed the omission of a `routine` call in the `ChainlinkAutomationProvider` that we advise be re-incorporated to the codebase.

# Post-Audit Conclusion (9fe8a4a3fc)

The Perpetual Airdrop team evaluated our follow-up recommendations and have addressed all remaining exhibits in the audit report except for `TOE-01C` which was safely acknowledged.

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

# Audit Synopsis

| Severity | Identified | Alleviated | Partially Alleviated | Acknowledged |
|---|---|---|---|---|
| Unknown | 1 | 1 | 0 | 0 |
| Informational | 18 | 17 | 0 | 1 |
| Minor | 4 | 3 | 0 | 1 |
| Medium | 3 | 3 | 0 | 0 |
| Major | 3 | 3 | 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 **20 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/perpetual-airdrop/contracts**
- Commit: 1bb58ccd224b90e7fb2fee22aff4d981ddfb9a52
- Language: Solidity
- Network: Ethereum
- Revisions: **1bb58ccd22**, **a5e8ff5ca3**, **9fe8a4a3fc**

## Contracts Assessed

| File | Total Finding(s) |
| --- | --- |
| **contracts/CustomGovernorCounting.sol (CGC)** | 0 |
| **contracts/ChainlinkAutomationProvider.sol (CAP)** | 3 |
| **contracts/ChainlinkRandomnessProvider.sol (CRP)** | 5 |
| **contracts/RandomnessRouter.sol (RRR)** | 1 |
| **contracts/TokenOne.sol (TOE)** | 9 |
| **contracts/types/TokenOneTypes.sol (TOT)** | 0 |
| **contracts/TokenOneGovernor.sol (TOG)** | 11 |

# Compilation

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

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.20` based on the version specified within the `hardhat.config.ts` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.20`).

We advise them to be locked to `0.8.20` (`=0.8.20`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` 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 **67 potential issues** within the codebase of which **54 were ruled out to be false positives** or negligible findings.

The remaining **13 issues** were validated and grouped and formalized into the **9 exhibits** that follow:

| ID | Severity | Addressed | Title |
|---|---|---|---|
| CAP-01S | Informational | Yes | Multiple Top-Level Declarations |
| CRP-01S | Informational | Yes | Inexistent Sanitization of Input Addresses |
| CRP-02S | Informational | Yes | Inexistent Visibility Specifier |
| TOE-01S | Informational | Yes | Inexistent Event Emission |
| TOE-02S | Informational | Yes | Inexistent Sanitization of Input Address |
| TOG-01S | Informational | Nullified | Illegible Numeric Value Representation |
| TOG-02S | Informational | Yes | Inexistent Sanitization of Input Address |
| TOG-03S | Informational | Yes | Inexistent Visibility Specifier |
| TOG-04S | Informational | Yes | Multiple Top-Level Declarations |

# Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Perpetual Airdrop's Token One implementation.

As the project at hand implements a perpetual airdrop EIP-20, 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 **severe ramifications** to its overall operation; we urge the Perpetual Airdrop team to evaluate and remediate the 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 an exemplary extent in the form of a whitepaper, and certain discrepancies were identified and outlined as exhibits within the audit report.

A total of **20 findings** were identified over the course of the manual review of which **11 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 |
|---|---|---|---|
| **CRP-01M** | Unknown | Yes | No-Op Function Implementation |
| **CRP-02M** | Medium | Yes | Inexistent Provision of Timepoint |
| **RRR-01M** | Major | Yes | Inexistent Access Control |
| **TOE-01M** | Minor | Yes | Inexistent Handling of No Reward |
| **TOE-02M** | Minor | Acknowledged | Inexistent Restriction of Initial Airdrop Configuration |

| | | | |
|---|---|---|---|
| **TOE-03M** | Medium | Yes | Inexistent Support of Past Regular Airdrop Computations |
| **TOE-04M** | Major | Yes | Insecure Usage of Current Balance |
| **TOG-01M** | Minor | Yes | Inexistent Access Control |
| **TOG-02M** | Minor | Yes | Inexistent Validation of Active Threshold |
| **TOG-03M** | Medium | Yes | Incorrect Voting Proposal Threshold |
| **TOG-04M** | Major | Yes | Potential Quorum Manipulation |

# Code Style

During the manual portion of the audit, we identified **9 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 |
|---|---|---|---|
| **CAP-01C** | Informational | Nullified | Redundant Return Statement |
| **CAP-02C** | Informational | Nullified | Variable Mutability Specifiers (Immutable) |
| **CRP-01C** | Informational | Yes | Variable Mutability Specifiers (Immutable) |
| **TOE-01C** | Informational | Acknowledged | Confusing Terminology |
| **TOE-02C** | Informational | Yes | Inefficient Initialization of Indices |
| **TOE-03C** | Informational | Yes | Variable Mutability Specifier (Immutable) |
| **TOG-01C** | Informational | Yes | Redundant Parenthesis Statements |
| **TOG-02C** | Informational | Yes | Redundant Visibility Specifier |
| **TOG-03C** | Informational | Yes | Variable Mutability Specifiers (Immutable) |

# ChainlinkAutomationProvider Static Analysis Findings

## CAP-01S: Multiple Top-Level Declarations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | ChainlinkAutomationProvider.sol:L9, L15, L42 |

### Description:

The referenced file contains multiple top-level declarations that decrease the legibility of the codebase.

### Example:

contracts/ChainlinkAutomationProvider.sol

SOL

```sol
interface IAutomationRegistrar {
    function registerUpkeep(
        UpkeepRegistrationParams calldata requestParams
    ) external returns (uint256);
}

interface IAutomationRegistry {
```

### Recommendation:

We advise all highlighted top-level declarations to be split into their respective code files, avoiding unnecessary imports as well as increasing the legibility of the codebase.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

All top-level declarations have been properly relocated to their dedicated files, addressing this exhibit.

# ChainlinkRandomnessProvider Static Analysis Findings

## CRP-01S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| Input Sanitization | Informational | ChainlinkRandomnessProvider.sol:L32-L43 |

### 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:

contracts/ChainlinkRandomnessProvider.sol

SOL

```sol
32  constructor(
33      IRandomnessConsumer _consumer,
34      VrfConfig memory _vrfConfig,
35      address _linkTokenAddress
36  ) VRFConsumerBaseV2Plus(_vrfConfig.coordinatorAddress) {
37      consumer = IRandomnessConsumer(_consumer);
38      vrfConfig = _vrfConfig;
39      LINKTOKEN = LinkTokenInterface(_linkTokenAddress);
40
41      subscriptionId = s_vrfCoordinator.createSubscription();
```

## Example (Cont.):

```sol
        s_vrfCoordinator.addConsumer(subscriptionId, address(this));
    }
```

### Recommendation:

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

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

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

# CRP-02S: Inexistent Visibility Specifier

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | ChainlinkRandomnessProvider.sol:L18 |

## Description:

The linked variable has no visibility specifier explicitly set.

## Example:

contracts/ChainlinkRandomnessProvider.sol

SOL

```
18  LinkTokenInterface immutable LINKTOKEN;
```

### 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 (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

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

# TokenOne Static Analysis Findings

## TOE-01S: Inexistent Event Emission

| Type | Severity | Location |
| --- | --- | --- |
| Language Specific | Informational | TokenOne.sol:L83-L85 |

### Description:

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

### Example:

contracts/TokenOne.sol

SOL

```sol
83  function setRandomnessProvider(address _randomnessProviderAddress) external
onlyOwner {
84      randomnessProvider = IRandomnessProvider(_randomnessProviderAddress);
85  }
```

### Recommendation:

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

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The `RandomnessProviderSet` event was introduced to the codebase and is correspondingly emitted in the **`TokenOne::setRandomnessProvider`** function, addressing this exhibit in full.

# TOE-02S: Inexistent Sanitization of Input Address

| Type | Severity | Location |
| --- | --- | --- |
| Input Sanitization | Informational | TokenOne.sol:L83-L85 |

### 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:

contracts/TokenOne.sol

SOL

```sol
83  function setRandomnessProvider(address _randomnessProviderAddress) external
onlyOwner {
84      randomnessProvider = IRandomnessProvider(_randomnessProviderAddress);
85  }
```

### Recommendation:

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

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

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

# TokenOneGovernor Static Analysis Findings

## TOG-01S: Illegible Numeric Value Representation

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | TokenOneGovernor.sol:L45 |

### Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

### Example:

contracts/TokenOneGovernor.sol

SOL

```sol
45  return 100000;
```

### Recommendation:

To properly illustrate the value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The referenced literal is no longer present in the codebase rendering this exhibit inapplicable.

# TOG-02S: Inexistent Sanitization of Input Address

| Type | Severity | Location |
| --- | --- | --- |
| Input Sanitization | Informational | TokenOneGovernor.sol:L24-L33 |

### 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:

contracts/TokenOneGovernor.sol

SOL

```sol
24  constructor(
25      IVotes _tokenOne,
26      string memory governorName,
27      uint256 _votingPeriod,
28      uint256 _activeThreshold
29  ) Governor(governorName) GovernorVotes(_tokenOne) {
30      votingPeriodInput = _votingPeriod;
31      activeThreshold = _activeThreshold;
32      tokenOne = ITokenOne(address(_tokenOne));
33  }
```

### Recommendation:

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

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

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

# TOG-03S: Inexistent Visibility Specifier

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | TokenOneGovernor.sol:L19 |

## Description:

The linked variable has no visibility specifier explicitly set.

## Example:

contracts/TokenOneGovernor.sol

SOL

```
19  ITokenOne tokenOne;
```

### 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 (a5e8ff5ca3):

While the variable has been set as immutable, no visibility specifier has been introduced to it rendering this exhibit acknowledged.

### Alleviation (9fe8a4a3fc):

A `public` visibility specifier was introduced to the `tokenOne` variable addressing this exhibit.

# TOG-04S: Multiple Top-Level Declarations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | TokenOneGovernor.sol:L11, L15 |

### Description:

The referenced file contains multiple top-level declarations that decrease the legibility of the codebase.

### Example:

contracts/TokenOneGovernor.sol

SOL

```sol
11  interface OwnableWithAcceptInterface {
12      function acceptOwnership() external;
13  }
14
15  contract TokenOneGovernor is Governor, GovernorVotes, CustomGovernorCounting {
```

### Recommendation:

We advise all highlighted top-level declarations to be split into their respective code files, avoiding unnecessary imports as well as increasing the legibility of the codebase.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The extra top-level declaration has been properly relocated to its dedicated file, addressing this exhibit.

# ChainlinkRandomnessProvider Manual Review Findings

## CRP-01M: No-Op Function Implementation

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Unknown | ChainlinkRandomnessProvider.sol:L97 |

### Description:

The `ChainlinkRandomnessProvider::logTransaction` function will not perform any action with the input arguments it is supplied with.

### Impact:

The purpose of the function is presently unclear and does not permit a severity to be accurately assessed.

### Example:

contracts/ChainlinkRandomnessProvider.sol

SOL

```sol
97  function logTransaction(address from, address to, uint256 amount, address token)
external { }
```

### Recommendation:

We advise either an `event` to be emitted, or the function arguments to become unnamed so as to indicate that the function exists for compatibility purposes.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The Perpetual Airdrop team evaluated this exhibit and clarified that the `ChainlinkRandomnessProvider::logTransaction` function selector is expected to be invoke-able on the contract as part of an out-of-scope governance mechanism.

As such, we consider this exhibit to be addressed via the commenting out of redundant arguments as the function itself should remain invoke-able.

# CRP-02M: Inexistent Provision of Timepoint

| Type | Severity | Location |
| --- | --- | --- |
| Language Specific | Medium | ChainlinkRandomnessProvider.sol:L71 |

### Description:

The `ChainlinkRandomnessProvider::fulfillRandomWords` will eventually invoke the `TokenOne::fulfillRegularAirdrop` function with a `timepoint` of `0` which is insecure as the contract will resort to current-balance measurements.

### Impact:

The regular airdrop mechanism can be presently manipulated via spot balance fluctuations, such as by acquiring a flash-loan of Token One balance via an AMM contract.

### Example:

contracts/ChainlinkRandomnessProvider.sol

SOL

```sol
67  /**
68   * @dev Callback function used by VRF Coordinator to return the random words.
69   */
70  function fulfillRandomWords(uint256 requestId, uint256[] calldata randomWords)
internal override {
71      consumer.fulfillRandomWords(requestId, randomWords, 0);
72  }
```

### Recommendation:

We advise the code to always supply a proper timepoint, potentially by utilizing the timepoint recorded when the airdrop period began.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The code of the `TokenOne::fulfillRegularAirdrop` function was updated to default the value of the `timepoint` to the `state.requestTimestamp` of the `RegularAirdropState`, ensuring that active balances are not utilized during airdrops and thus addressing this exhibit.

# RandomnessRouter Manual Review Findings

## RRR-01M: Inexistent Access Control

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Major | RandomnessRouter.sol:L36-L42 |

### Description:

The `RandomnessRouter::fulfillRandomWords` function lacks access control permitting anyone to fulfil a randomness request and thus bypass critical security assumptions about the `randomWords` yielded by the contract.

### Impact:

A randomness request can be fulfilled with deterministic data by anyone, compromising a critical security assumption in the system.

### Example:

contracts/RandomnessRouter.sol

SOL

```sol
36  function fulfillRandomWords(uint256 requestId, uint256[] calldata randomWords,
uint256 timepoint) external {
37      requestIdToConsumer[requestId].fulfillRandomWords(
38          requestId,
39          randomWords,
40          timepoint
41      );
42  }
```

### Recommendation:

We advise proper access control to be imposed ensuring that the function is solely callable by the currently-active `randomnessProvider`.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

Access control was properly introduced ensuring the function's caller is the `randomnessProvider`, addressing this exhibit in full.

# TokenOne Manual Review Findings

## TOE-01M: Inexistent Handling of No Reward

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | TokenOne.sol:L284, L294, L296, L298, L300-L305 |

### Description:

The `TokenOne::fulfillRegularAirdrop` function will misbehave if it is unable to find an eligible winner based on the `timepoint` of the airdrop as it will increase the `totalBalance` of the `RegularAirdropState` incorrectly and will assign winner balances to the zero-address.

### Impact:

An airdrop might be won by the zero-address which is incorrect and contradicts the code's intentions.

### Example:

contracts/TokenOne.sol

SOL

```
275 for (uint256 i = 0; i < numRequested; i++) {
276     uint256 randomIndex = randomWords[i] % numParticipants;
277     address winner = eligibilityList.at(randomIndex);
278
279     if (timepoint != 0) {
280         uint256 attempts = 0;
281         while (getPastVotes(winner, timepoint) <
regularAirdropConfig.eligibilityThreshold) {
282             // Prevent infinite loop
283             if (attempts >= 10) {
284                 winner = address(0);
```

## Example (Cont.):

```sol
                break;
            }

            // Get new random index by hashing the previous one
            randomIndex = uint256(keccak256(abi.encode(randomIndex))) % numParticipants;
            winner = eligibilityList.at(randomIndex);
            attempts++;
        }
    }
    uint256 balance = timepoint == 0 ? balanceOf(winner) : getPastVotes(winner, timepoint);

    state.totalBalance += balance;

    EnumerableMap.AddressToUintMap storage winnerBalances = regularAirdropWinnerBalances[index];

    if (winnerBalances.contains(winner)) {
        uint256 currentBalance = winnerBalances.get(winner);
        winnerBalances.set(winner, currentBalance + balance);
    } else {
        winnerBalances.set(winner, balance);
    }
}
```

### Recommendation:

We advise the loop to `continue` if the `winner` is the zero-address after a timepoint lookup, ensuring improper reward entries are not created.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The code was updated to skip an airdrop draw if no winner was found within the allocated attempts, alleviating this exhibit.

# TOE-02M: Inexistent Restriction of Initial Airdrop Configuration

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | TokenOne.sol:L69, L70 |

### Description:

The Token One whitepaper denotes that the initial airdrop is composed of 10 winners and a total of `100_000_000e18` tokens distributed, however, this configuration is not observed in the contract.

### Impact:

The configuration of the `TokenOne` contract is arbitrary and might not match its whitepaper specification.

### Example:

contracts/TokenOne.sol

SOL

```sol
62  constructor(
63      string memory tokenName,
64      string memory tokenSymbol,
65      InitialAirdropConfig memory _initialAirdropConfig,
66      RegularAirdropConfig memory _regularAirdropConfig,
67      uint256 _governanceThreshold
68  ) ERC20(tokenName, tokenSymbol) ERC20Permit(tokenName) Ownable(msg.sender) {
69      require(_initialAirdropConfig.numWinners > 0, "Invalid winners count");
70      require(_initialAirdropConfig.amount > 0, "Invalid airdrop amount");
71      require(_initialAirdropConfig.participants.length > 0, "No participants");
```

### Recommendation:

We advise the configuration to be made strict, either by necessitating the configuration via equality case `require` clauses or by configuring it directly, ensuring that the `numWinners` are `10` and that the `amount` of each winner is `10_000_000e18` units.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The Perpetual Airdrop team evaluated this exhibit and opted to not enforced a fixed configuration as they wish their deployment to remain flexible.

# TOE-03M: Inexistent Support of Past Regular Airdrop Computations

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Medium | TokenOne.sol:L317 |

## Description:

The `TokenOne::computeRegularAirdropEarnings` function does not support the computation of elapsed regular airdrops, resulting in computed airdrops that have not yet been distributed to not be activate-able.

## Impact:

The system does not permit any airdrop index to be computed, causing airdrops which have been requested and properly fulfilled to potentially not be distributed.

## Example:

contracts/TokenOne.sol

SOL

```sol
317 function computeRegularAirdropEarnings(uint16 batchSize) external {
318     uint32 index = regularAirdropCurrentIndex();
```

### Recommendation:

We advise the code to introduce a secondary `TokenOne::computeRegularAirdropEarnings` function that permits the `index` of the airdrop to be supplied as an argument, ensuring past regular airdrops that were potentially requested close to their index period's conclusion to be properly processed.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The code was updated to allow for an additional `index` argument, ensuring that elapsed regular airdrops can still be computed and thus claimed.

# TOE-04M: Insecure Usage of Current Balance

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Major | TokenOne.sol:L294 |

### Description:

A user is able to manipulate the fulfilment of a regular airdrop with a timepoint of `0` (regular functionality of the system) by acquiring a substantial portion of funds in the same block the randomness request is fulfilled and refunding them right after the request is fulfilled.

### Impact:

A regular airdrop can be manipulated via MEV by sandwiching the airdrop's fulfillment with a transaction that acquires a substantial portion of Token One tokens before the request is fulfilled and refunds them right after.

### Example:

contracts/TokenOne.sol

SOL

```sol
279 if (timepoint != 0) {
280     uint256 attempts = 0;
281     while (getPastVotes(winner, timepoint) <
regularAirdropConfig.eligibilityThreshold) {
282         // Prevent infinite loop
283         if (attempts >= 10) {
284             winner = address(0);
285             break;
286         }
287
288         // Get new random index by hashing the previous one
```

## Example (Cont.):

```sol
289         randomIndex = uint256(keccak256(abi.encode(randomIndex))) %
numParticipants;
290         winner = eligibilityList.at(randomIndex);
291         attempts++;
292     }
293 }
294 uint256 balance = timepoint == 0 ? balanceOf(winner) : getPastVotes(winner,
timepoint);
```

### Recommendation:

We advise the code to prevent a `timepoint` of `0` from being valid, and to always utilize a `timepoint` in the past that increases the airdrop mechanism's resilience to manipulation attacks.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

Spot balance evaluations have been omitted from the codebase ensuring a non-zero timepoint is defined or, in the case it has not been defined, that the `requestTimestamp` of the `RegularAirdropStates` entry is utilized thereby addressing this exhibit.

# TokenOneGovernor Manual Review Findings

## TOG-01M: Inexistent Access Control

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | TokenOneGovernor.sol:L74-L76 |

### Description:

The `TokenOneGovernor::acceptOwnership` function permits anyone to accept ownership of another contract on behalf of the `TokenOneGovernor` which we consider ill-advised as the function signature might clash with another and generally represents a call to an arbitrary untrusted address.

### Impact:

The `TokenOneGovernor::acceptOwnership` function does not impose any access control, causing potential proposals that execute it to fail if ownership is accepted prior to the proposal's execution.

### Example:

contracts/TokenOneGovernor.sol

SOL

```sol
74  function acceptOwnership(OwnableWithAcceptInterface ownableContract) external {
75      ownableContract.acceptOwnership();
76  }
```

### Recommendation:

We advise ownership acceptance to be restricted via the `Governor::onlyGovernance` modifier, ensuring that ownership acceptances are properly authorized.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The function was updated to apply the `Ownable::onlyOwner` modifier instead, ensuring some form of access control is imposed and thus addressing this exhibit.

# TOG-02M: Inexistent Validation of Active Threshold

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | TokenOneGovernor.sol:L31 |

### Description:

The configured active threshold should be validated as being a value that is equal to or lower than the `TokenOneGovernor::proposalThreshold` so as to ensure the proposal threshold is properly upheld.

### Impact:

Should the `activeThreshold` be configured to a value higher than the `TokenOneGovernor::proposalThreshold`, a proposal will not be able to be created with the threshold defined which we consider incorrect.

### Example:

contracts/TokenOneGovernor.sol

SOL

```sol
24  constructor(
25      IVotes _tokenOne,
26      string memory governorName,
27      uint256 _votingPeriod,
28      uint256 _activeThreshold
29  ) Governor(governorName) GovernorVotes(_tokenOne) {
30      votingPeriodInput = _votingPeriod;
31      activeThreshold = _activeThreshold;
32      tokenOne = ITokenOne(address(_tokenOne));
33  }
```

## Example (Cont.):

```sol

34
35
36  function votingPeriod() public view virtual override returns (uint256) {
37      return votingPeriodInput;
38  }
39
40  function votingDelay() public pure virtual override returns (uint256) {
41    return 0;
42  }
43
44  function proposalThreshold() public pure override returns (uint256) {
45      return 100000;
46  }
47
48  function _getVotes(address account, uint256 timepoint, bytes memory params)
internal view override(Governor, GovernorVotes) returns (uint256) {
49      uint256 votes = super._getVotes(account, timepoint, params);
50      return (votes >= activeThreshold) ? votes : 0;
51  }
```

### Recommendation:

We advise this restriction to be imposed, ensuring that proposals can accurately be created with the threshold balance outlined by the contract.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The thresholds in the system are now validated as advised, alleviating this exhibit.

# TOG-03M: Incorrect Voting Proposal Threshold

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Medium | TokenOneGovernor.sol:L45 |

### Description:

The `TokenOneGovernor::proposalThreshold` does not match the one that the Perpetual Airdrop team desires as it represents `100_000` units rather than `100_000e18` units of the Token One token which possesses `18` decimal places.

### Impact:

The current proposal threshold does not match the one defined in the Token One whitepaper, permitting the governor to be spammed with incorrect requests.

### Example:

contracts/TokenOneGovernor.sol

SOL

```sol
44  function proposalThreshold() public pure override returns (uint256) {
45      return 100000;
46  }
```

### Recommendation:

We advise the threshold to be increased accordingly, ensuring that the governance's proposal list cannot be polluted trivially with malicious proposals.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The proposal threshold is now defined by the contract's deployer which we consider that the Perpetual Airdrop team will appropriately configure, rendering this exhibit addressed.

# TOG-04M: Potential Quorum Manipulation

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Major | TokenOneGovernor.sol:L65 |

### Description:

The `TokenOneGovernor::propose` function contains a flaw in the way it tracks proposal quorums. Specifically, it will utilize the currently-active `ITokenOne::totalVotes` value which represents the actively vote-able balance **at the time it is queried**.

The `Governor` implementation will permit proposals to be voted on with **a balance right before the time the proposal is created**, causing an inconsistency between the sum of the total vote-able balances and the actual quorum recorded in the contract.

This quorum can be manipulated downward maliciously by transferring one's balance right before creating a proposal thereby reducing it by the amount the user would vote on the proposal.

### Impact:

The quorum of a particular `TokenOneGovernor` proposal can be manipulated downward effectively compromising the governance process.

### Example:

contracts/TokenOneGovernor.sol

SOL

```sol
53  /**
54   * @dev Override to capture the total supply at the time of the proposal
creation.
55   */
56  function propose(
57      address[] memory targets,
58      uint256[] memory values,
59      bytes[] memory calldatas,
60      string memory description
61  ) public override(Governor) returns (uint256) {
62      uint256 proposalId = super.propose(targets, values, calldatas, description);
```

## Example (Cont.):

```sol

        // Store 50% of the total voting power at the time of proposal creation
        proposalQuorums[proposalId] = (tokenOne.totalVotes()) / 2;

        return proposalId;
    }
```

### Recommendation:

We advise the overall approach to be refactored, utilizing a historical `ITokenOne::totalVotes` accounting system that accurately depicts the total votes **right before the current block**.

To achieve this, a simple approach would be to finalize any `ITokenOne::totalVotes` changes **in the next block** by updating the function to be mutable and introducing two new variables to track pending changes to the sum.

### Alleviation (a5e8ff5ca3):

While the system was updated to utilize a `previousBlockTotalVotes` mechanism, it is still insufficient as the value will only be updated on the first transaction of a new block. As such, it can still contain an inaccurate value that can be manipulated depending on whether it satisfies the proposer's intentions.

We advise our original recommendation to be applied, utilizing a historical total supply tracking mechanism.

### Alleviation (9fe8a4a3fc):

The code of the `TokenOne` implementation was updated to support a historical accounting system of past total votes, permitting the `TokenOneGovernor` to integrate it and thus alleviate this exhibit in full.

# ChainlinkAutomationProvider Code Style Findings

## CAP-01C: Redundant Return Statement

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | ChainlinkAutomationProvider.sol:L167 |

### Description:

The referenced `return` statement is redundant as the function will automatically conclude at that point.

### Example:

contracts/ChainlinkAutomationProvider.sol

SOL

```sol
/**
 * @dev Performs the upkeep actions based on performData.
 */
function performUpkeep(bytes calldata /* performData */) external override {
    fundUpkeep();
    tokenContract.randomnessProvider().routine();

    // Compute initial airdrop earnings
    if (tokenContract.isReadyToComputeInitial()) {
        tokenContract.computeInitialAirdropEarnings();
```

## Example (Cont.):

```sol
        return;
    }

    // Request regular airdrop earnings
    if (tokenContract.isReadyToRequestRegular()) {
        tokenContract.requestRegularAirdrop();
        return;
    }

    // Compute regular airdrop earnings
    if (tokenContract.isReadyToComputeRegular()) {
        tokenContract.computeRegularAirdropEarnings(computeBatchSize);
        return;
    }

    // Distribute earnings
    if (tokenContract.hasPendingEarnings()) {
        tokenContract.distributeEarnings(distributionBatchSize);
        return;
    }
}
```

### Recommendation:

We advise it to be omitted, reducing the syntactic sugar of the codebase.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The code was refactored to accommodate for multiple `airdropCoordinators` and thus operates a loop that requires the `return` statement to be present, rendering this exhibit nullified.

# CAP-02C: Variable Mutability Specifiers (Immutable)

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | **ChainlinkAutomationProvider.sol**:<br>• I-1: **L53**<br>• I-2: **L55**<br>• I-3: **L56** |

## Description:

The linked variables are assigned to only once during the contract's `constructor`.

## Example:

contracts/ChainlinkAutomationProvider.sol

SOL

```sol
constructor(
  address _tokenContract,
  address _linkTokenAddress,
  AutomationConfig memory _automationConfig,
  uint16 _computeBatchSize,
  uint16 _distributionBatchSize
) {
    LINKTOKEN = LinkTokenInterface(_linkTokenAddress);

    automationConfig = _automationConfig;
```

## Example (Cont.):

```sol
        automationRegistrar = IAutomationRegistrar(_automationConfig.registrarAddress);
        automationRegistry = IAutomationRegistry(_automationConfig.registryAddress);

        tokenContract = ITokenOne(_tokenContract);

        computeBatchSize = _computeBatchSize;
        distributionBatchSize = _distributionBatchSize;
    }
```

### Recommendation:

We advise them to be set as `immutable` greatly optimizing their read-access gas cost.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The referenced arguments were all updated to dynamic arrays assigned during the contract's **`ChainlinkAutomationProvider::constructor`**, rendering the optimization no longer applicable.

# ChainlinkRandomnessProvider Code Style Findings

## CRP-01C: Variable Mutability Specifiers (Immutable)

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | **ChainlinkRandomnessProvider.sol**:<br>• I-1: **L24**<br>• I-2: **L27** |

### Description:

The linked variables are assigned to only once during the contract's `constructor`.

### Example:

contracts/ChainlinkRandomnessProvider.sol

SOL

```sol
23  // VRF subscription ID for the contract
24  uint256 public subscriptionId;
25
26  // Consumer
27  IRandomnessConsumer public consumer;
28
29  /**
30   * @dev Constructor initializes the contract with the necessary configurations
and sets up the VRF subscription.
31   */
32  constructor(
```

## Example (Cont.):

```sol
        IRandomnessConsumer _consumer,
        VrfConfig memory _vrfConfig,
        address _linkTokenAddress
    ) VRFConsumerBaseV2Plus(_vrfConfig.coordinatorAddress) {
        consumer = IRandomnessConsumer(_consumer);
        vrfConfig = _vrfConfig;
        LINKTOKEN = LinkTokenInterface(_linkTokenAddress);

        subscriptionId = s_vrfCoordinator.createSubscription();
        s_vrfCoordinator.addConsumer(subscriptionId, address(this));
    }
```

### Recommendation:

We advise them to be set as `immutable` greatly optimizing their read-access gas cost.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The `consumer`, and `subscriptionId` contract-level variables of the contract have been set as `immutable`, optimizing their read-access gas cost significantly.

# TokenOne Code Style Findings

## TOE-01C: Confusing Terminology

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | TokenOne.sol:L40 |

### Description:

The `IRandomnessProvider` is actually a `RandomnessRouter` implementation that exposes functions to interact with randomness providers.

### Example:

contracts/TokenOne.sol

SOL

```sol
40  IRandomnessProvider public randomnessProvider;
```

### Recommendation:

We advise this ambiguity in the codebase to be addressed by renaming relevant contracts to more concise names (i.e. `IRandomnessProvider` -> `IRandomnessRouter`).

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

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

# TOE-02C: Inefficient Initialization of Indices

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | TokenOne.sol:L198-L201 |

### Description:

The referenced loop is inefficient as it will redundantly iterate through all participants of the initial airdrop to initialize their indices.

### Example:

contracts/TokenOne.sol

SOL

```sol
196 uint256[] memory available = new uint256[](numParticipants);
197
198 // Initialize available indices
199 for (uint256 i = 0; i < numParticipants; i++) {
200     available[i] = i;
201 }
```

### Recommendation:

We advise the code to simply assume that a zero-value `available` entry indicates a participant whose position has not been swapped, and a non-zero entry indicates the next index that should be used for disbursing rewards.

This process will significantly optimize the gas cost of the function.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The array initialized was updated to `participants` and is optimally initialized to the `participants` of the `initialAirdropConfig`, addressing this exhibit.

# TOE-03C: Variable Mutability Specifier (Immutable)

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | TokenOne.sol:L44 |

## Description:

The linked variable is assigned to only once during the contract's `constructor`.

## Example:

contracts/TokenOne.sol

SOL

```sol
constructor(
    string memory tokenName,
    string memory tokenSymbol,
    InitialAirdropConfig memory _initialAirdropConfig,
    RegularAirdropConfig memory _regularAirdropConfig,
    uint256 _governanceThreshold
) ERC20(tokenName, tokenSymbol) ERC20Permit(tokenName) Ownable(msg.sender) {
    require(_initialAirdropConfig.numWinners > 0, "Invalid winners count");
    require(_initialAirdropConfig.amount > 0, "Invalid airdrop amount");
    require(_initialAirdropConfig.participants.length > 0, "No participants");
```

## Example (Cont.):

```sol
        require(_regularAirdropConfig.amount > 0, "Invalid regular airdrop amount");
        require(_regularAirdropConfig.numWinners > 0, "Invalid regular winners count");
        require(_regularAirdropConfig.airdropTimeInterval > 0, "Invalid airdrop interval");
        require(_regularAirdropConfig.windowDuration <= _regularAirdropConfig.airdropTimeInterval, "Invalid window duration");

        initialAirdropConfig = _initialAirdropConfig;
        regularAirdropConfig = _regularAirdropConfig;

        governanceThreshold = _governanceThreshold;
    }
```

### Recommendation:

We advise it to be set as `immutable` greatly optimizing its read-access gas cost.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The `governanceThreshold` contract-level variable of the contract has been set as `immutable`, optimizing its read-access gas cost significantly.

# TokenOneGovernor Code Style Findings

## TOG-01C: Redundant Parenthesis Statements

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | TokenOneGovernor.sol:L50, L65 |

### Description:

The referenced statements are redundantly wrapped in parenthesis' ( `()` ).

### Example:

contracts/TokenOneGovernor.sol

SOL

```sol
50  return (votes >= activeThreshold) ? votes : 0;
```

### Recommendation:

We advise them to be safely omitted, increasing the legibility of the codebase.

### Alleviation (a5e8ff5ca3):

The redundant parenthesis remain in the codebase, rendering this exhibit acknowledged.

### Alleviation (9fe8a4a3fc):

Both redundant parenthesis statements have been omitted, increasing the code's legibility.

# TOG-02C: Redundant Visibility Specifier

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | TokenOneGovernor.sol:L16 |

## Description:

The referenced variable is set as `public` yet is exposed via the **`TokenOneGovernor::votingPeriod`** function.

## Example:

contracts/TokenOneGovernor.sol

SOL

```sol
16  uint256 public votingPeriodInput;
17  uint256 public activeThreshold;
18
19  ITokenOne tokenOne;
20
21  // Mapping to store the quorum threshold at the time of the proposal
22  mapping(uint256 => uint256) private proposalQuorums;
23
24  constructor(
25      IVotes _tokenOne,
```

## Example (Cont.):

```sol
    string memory governorName,
    uint256 _votingPeriod,
    uint256 _activeThreshold
) Governor(governorName) GovernorVotes(_tokenOne) {
    votingPeriodInput = _votingPeriod;
    activeThreshold = _activeThreshold;
    tokenOne = ITokenOne(address(_tokenOne));
}


function votingPeriod() public view virtual override returns (uint256) {
    return votingPeriodInput;
}
```

### Recommendation:

We advise its `public` visibility specifier to be replaced by `internal` or `private`, ensuring a getter function is not redundantly generated for it.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The variable has been set to `private` as advised, addressing this exhibit.

# TOG-03C: Variable Mutability Specifiers (Immutable)

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | **TokenOneGovernor.sol**:<br>• I-1: **L16**<br>• I-2: **L17**<br>• I-3: **L19** |

## Description:

The linked variables are assigned to only once during the contract's `constructor`.

## Example:

contracts/TokenOneGovernor.sol

SOL

```sol
16  uint256 public votingPeriodInput;
17  uint256 public activeThreshold;
18
19  ITokenOne tokenOne;
20
21  // Mapping to store the quorum threshold at the time of the proposal
22  mapping(uint256 => uint256) private proposalQuorums;
23
24  constructor(
25      IVotes _tokenOne,
```

## Example (Cont.):

```sol
    string memory governorName,
    uint256 _votingPeriod,
    uint256 _activeThreshold
) Governor(governorName) GovernorVotes(_tokenOne) {
    votingPeriodInput = _votingPeriod;
    activeThreshold = _activeThreshold;
    tokenOne = ITokenOne(address(_tokenOne));
}
```

### Recommendation:

We advise them to be set as `immutable` greatly optimizing their read-access gas cost.

### Alleviation (a5e8ff5ca31fe8c5fc57732866142ce01ab9a49c):

The `votingPeriodInput`, `voteThreshold`, and `tokenOne` contract-level variables of the contract have been set as `immutable`, optimizing their read-access gas cost significantly.

# 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.