OMNISCIA

May 17, 2023

# SMART CONTRACT AUDIT REPORT

Gravita



omniscia.io


info@omniscia.io



Online report: gravita-protocol

# Core Protocol Security Audit

## Audit Revisions

| Commit Hash | Date | Revision Hash |
| --- | --- | --- |
| 5e45123d16 | May 17th 2023 | 1f8d3c80e7 |

## Audit Overview

We were tasked with performing an audit of the Gravita Protocol codebase and in particular their core Liquity-based borrowing protocol.

Over the course of the audit, we identified multiple significant vulnerabilities that arise by the dynamic-collateral features introduced in the new Gravita Protocol implementation.

We advise the Gravita Protocol 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 Gravita Protocol 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 Gravita Protocol and identified that certain exhibits had not been adequately dealt with.

We followed up with the Gravita Protocol team and have concluded that they wish to acknowledge them given that the exhibits that have not been directly remediated do not pose a threat to the protocol.

As such, we consider all outputs of the audit report properly consumed by the Gravita Finance team.

## Contracts Assessed

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| ActivePool.sol (APL) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| AdminContract.sol (ACT) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| BaseMath.sol (BMH) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| BorrowerOperations.sol (BOS) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| CollSurplusPool.sol (CSP) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| DebtToken.sol (DTN) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| DefaultPool.sol (DPL) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| ERC20Permit.sol (ERC) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| ERC20Decimals.sol (ERD) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| FeeCollector.sol (FCR) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| GasPool.sol (GPL) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| GravitaBase.sol (GBE) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| GravitaMath.sol (GMH) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |

| Files in Scope | Repository | Commit(s) |
| --- | --- | --- |
| GravitaSafeMath128.sol (GSM) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| PoolBase.sol (PBE) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| PriceFeed.sol (PFD) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| ReentrancyGuardUpgradeable.sol (RGU) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| SafeMath.sol (SMH) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| SortedVessels.sol (SVS) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| StabilityPool.sol (SPL) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| SafetyTransfer.sol (STR) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| Timelock.sol (TKC) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| VesselManager.sol (VMR) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |
| VesselManagerOperations.sol (VMO) | Gravita-SmartContracts | bfa97cb37d, 5e45123d16 |

## Audit Synopsis

| Severity | Identified | Alleviated | Partially Alleviated | Acknowledged |
|---|---|---|---|---|
| Unknown | 3 | 3 | 0 | 0 |
| Informational | 73 | 60 | 10 | 3 |
| Minor | 16 | 4 | 0 | 12 |
| Medium | 4 | 4 | 0 | 0 |
| Major | 3 | 3 | 0 | 0 |

During the audit, we filtered and validated a total of **22 findings utilizing static analysis** tools as well as identified a total of **77 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.

# Compilation

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

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 `0.8.17` based on the version specified within the `hardhat.config.js` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.10`).

We advise them to be locked to `0.8.17` (`=0.8.17`), 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 **457 potential issues** within the codebase of which **370 were ruled out to be false positives** or negligible findings.

The remaining **87 issues** were validated and grouped and formalized into the **22 exhibits** that follow:

| ID | Severity | Addressed | Title |
|---|---|---|---|
| APL-01S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| ACT-01S | Informational | Yes | Data Location Optimization |
| ACT-02S | Informational | Yes | Illegible Numeric Value Representations |
| ACT-03S | Informational | Yes | Inexistent Visibility Specifier |
| ACT-04S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| BOS-01S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| CSP-01S | Informational | Yes | Inexistent Visibility Specifier |
| CSP-02S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| DTN-01S | Informational | Yes | Inexistent Event Emissions |
| DTN-02S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| DPL-01S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| FCR-01S | Informational | Yes | Data Location Optimizations |
| FCR-02S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |

| ID | Severity | Addressed | Title |
| --- | --- | --- | --- |
| FCR-03S | Medium | Yes | Improper Invocations of EIP-20 `transfer` |
| GMH-01S | Informational | Yes | Illegible Numeric Value Representation |
| PFD-01S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| SVS-01S | Informational | Nullified | Inexistent Visibility Specifier |
| SPL-01S | Informational | Nullified | Inexistent Visibility Specifier |
| SPL-02S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| VMR-01S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |
| VMO-01S | Informational | Acknowledged | Illegible Numeric Value Representations |
| VMO-02S | Minor | Acknowledged | Inexistent Sanitization of Input Addresses |

# Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Gravita Protocol's novel borrowing implementation.

As the project at hand implements a Liquity-based borrowing protocol backed by multiple collateral types, 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 several dynamic collateral-related vulnerabilities** within the system which could have had **severe ramifications** to its overall operation.

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

A total of **77 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 |
|---|---|---|---|
| **ACT-01M** | Unknown | Yes | Improper Reset Functionality |
| **ACT-02M** | Medium | Yes | Improper Permission of Collateral Activation |
| **ACT-03M** | Major | Yes | Improper Capability of Gas Compensation Adjustment |
| **ERC-01M** | Minor | Acknowledged | Insecure EIP-2612 Implementation |
| **ERC-02M** | Medium | Yes | Insecure Elliptic Curve Recovery Mechanism |
| **GSM-01M** | Informational | Yes | Improper Application of Safe Arithmetics |

| ID | Severity | Addressed | Title |
|---|---|---|---|
| PFD-01M | Unknown | Yes | Significant Centralization of Sensitive Functionality |
| PFD-02M | Minor | Yes | Incorrect Error Handling |
| PFD-03M | Minor | Yes | Inexistent Initialization of Price |
| PFD-04M | Medium | Yes | Incorrect Lido Staked ETH Value Assumption |
| PFD-05M | Major | Yes | Incorrect Lido Staked ETH Price Usage |
| SMH-01M | Informational | Yes | Improper Application of Safe Arithmetics |
| STR-01M | Minor | Yes | Incorrect Decimal Assumption |
| STR-02M | Major | Yes | Insecure Conversion of Amount |
| SVS-01M | Informational | Acknowledged | Insecure Data List Size Enforcement |
| SPL-01M | Unknown | Yes | Inexistent Normalization of Asset |
| TKC-01M | Minor | Yes | Inexistent Prevention of Duplicate Invocations |

# Code Style

During the manual portion of the audit, we identified **60 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 |
|---|---|---|---|
| **APL-01C** | Informational | Nullified | Inefficient Renunciation of Ownership |
| **APL-02C** | Informational | Partial | Inexplicable Ownable Pattern |
| **APL-03C** | Informational | Yes | Redundant Initialization Paradigm |
| **ACT-01C** | Informational | Yes | Inefficient `mapping` Lookups |
| **ACT-02C** | Informational | Yes | Inexistent Error Message |
| **ACT-03C** | Informational | Partial | Loop Iterator Optimizations |
| **ACT-04C** | Informational | Yes | Misleading Variable Name |
| **BOS-01C** | Informational | Yes | Ineffectual Native Value Check |
| **BOS-02C** | Informational | Yes | Redundant Native Value Check |
| **BOS-03C** | Informational | Yes | Suboptimal Struct Declaration Styles |
| **CSP-01C** | Informational | Nullified | Inefficient Renunciation of Ownership |

| ID | Severity | Addressed | Title |
|---|---|---|---|
| CSP-02C | Informational | Yes | Inefficient `mapping` Lookups |
| CSP-03C | Informational | Partial | Inexplicable Ownable Pattern |
| CSP-04C | Informational | Yes | Redundant Initialization Paradigm |
| DTN-01C | Informational | Yes | Variable Mutability Specifier (Immutable) |
| DPL-01C | Informational | Nullified | Inefficient Renunciation of Ownership |
| DPL-02C | Informational | Yes | Inefficient `mapping` Lookups |
| DPL-03C | Informational | Partial | Inexplicable Ownable Pattern |
| DPL-04C | Informational | Yes | Redundant Initialization Paradigm |
| ERD-01C | Informational | Yes | Non-Standard Interface Name |
| ERC-01C | Informational | Yes | Inefficient `mapping` Lookups |
| ERC-02C | Informational | Yes | Multiple Top-Level Declarations |
| ERC-03C | Informational | Yes | Redundant Low-Level Assembly Blocks |
| ERC-04C | Informational | Yes | Variable Mutability Specifier (Immutable) |
| FCR-01C | Informational | Yes | Inefficient `mapping` Lookups |
| FCR-02C | Informational | Acknowledged | Inexistent Error Messages |
| FCR-03C | Informational | Yes | Leftover Test Code |

| ID | Severity | Addressed | Title |
|---|---|---|---|
| FCR-04C | Informational | Partial | Loop Iterator Optimization |
| FCR-05C | Informational | Yes | Redundant Initialization Paradigm |
| GBE-01C | Informational | Yes | Unused Function Declaration |
| GMH-01C | Informational | Yes | Illegible Representation of Value Literal |
| GMH-02C | Informational | Yes | Repetitive Value Literal |
| PBE-01C | Informational | Partial | Significantly Inefficient Merging of Pending Gains / Distributed Funds |
| PBE-02C | Informational | Yes | Unused Error Declaration |
| PFD-01C | Informational | Nullified | Inexistent Error Message |
| PFD-02C | Informational | Yes | Redundant External Self-Calls |
| PFD-03C | Informational | Yes | Redundant Function Implementation |
| PFD-04C | Informational | Yes | Redundant Initialization Paradigm |
| PFD-05C | Informational | Yes | Suboptimal Struct Declaration Styles |
| RGU-01C | Informational | Yes | Inefficient Reentrancy Guard Implementation |
| SVS-01C | Informational | Nullified | Inefficient Renunciation of Ownership |
| SVS-02C | Informational | Yes | Inefficient `mapping` Lookups |
| SVS-03C | Informational | Partial | Inexplicable Ownable Pattern |

| ID | Severity | Addressed | Title |
|---|---|---|---|
| SVS-04C | Informational | Yes | Redundant Initialization Paradigm |
| SPL-01C | Informational | Nullified | Inefficient Renunciation of Ownership |
| SPL-02C | Informational | Partial | Inefficient `mapping` Lookups |
| SPL-03C | Informational | Yes | Inexplicable Contract Member |
| SPL-04C | Informational | Yes | Inexplicable Ownable Pattern |
| SPL-05C | Informational | Partial | Loop Iterator Optimizations |
| SPL-06C | Informational | Yes | Redundant Initialization Paradigm |
| SPL-07C | Informational | Nullified | Suboptimal Struct Declaration Style |
| TKC-01C | Informational | Yes | Inefficient Application of Access Control |
| TKC-02C | Informational | Yes | Redundant Function Implementation |
| VMR-01C | Informational | Yes | Inefficient `mapping` Lookups |
| VMR-02C | Informational | Yes | Redundant Data Point |
| VMR-03C | Informational | Yes | Redundant External Self-Call |
| VMR-04C | Informational | Yes | Redundant Initialization Paradigm |
| VMO-01C | Informational | Partial | Loop Iterator Optimizations |
| VMO-02C | Informational | Yes | Redundant Initialization Paradigm |

| ID | Severity | Addressed | Title |
| --- | --- | --- | --- |
| VMO-03C | Informational | Yes | Suboptimal Struct Declaration Styles |

# ActivePool Static Analysis Findings

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

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | ActivePool.sol:L84-L106 |

### 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/ActivePool.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _collSurplusPoolAddress,
    address _defaultPoolAddress,
    address _stabilityPoolAddress,
    address _vesselManagerAddress,
    address _vesselManagerOperationsAddress
) external initializer {
    require(!isInitialized, "Already initialized");
    isInitialized = true;

    __Ownable_init();
    __ReentrancyGuard_init();

    borrowerOperationsAddress = _borrowerOperationsAddress;
    collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
    defaultPool = IDefaultPool(_defaultPoolAddress);
    stabilityPoolAddress = _stabilityPoolAddress;
    vesselManagerAddress = _vesselManagerAddress;
    vesselManagerOperationsAddress = _vesselManagerOperationsAddress;

    renounceOwnership();
}
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# AdminContract Static Analysis Findings

## ACT-01S: Data Location Optimization

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | AdminContract.sol:L211 |

### Description:

The linked input argument is set as `memory` in an `external` function.

### Example:

contracts/AdminContract.sol

SOL

```sol
210 function isWrappedMany(
211     address[] memory _collaterals
212 ) external view returns (bool[] memory wrapped) {
```

### Recommendation:

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

### Alleviation:

The argument's data location has been properly updated from `memory` to `calldata`, optimizing its read-access gas cost.

# ACT-02S: Illegible Numeric Value Representations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | AdminContract.sol:L44-L45, L51, L321, L336, L366, L369, L415 |

### Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

### Example:

contracts/AdminContract.sol

SOL

```sol
44  uint256 public constant MCR_DEFAULT = 1100000000000000000; // 110%
45  uint256 public constant CCR_DEFAULT = 1500000000000000000; // 150%
```

### Recommendation:

To properly illustrate each 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:

All numeric denominations of the contract have been updated to either utilize the `ether` representation or the underscore-separated paradigm outlined in the exhibit. As such, we consider this exhibit fully alleviated.

# ACT-03S: Inexistent Visibility Specifier

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | AdminContract.sol:L69 |

### Description:

The linked variable has no visibility specifier explicitly set.

### Example:

contracts/AdminContract.sol

SOL

```sol
69  mapping(address => CollateralParams) collateralParams;
```

### 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:

An `internal` visibility specifier has been introduced to the `collateralParams` contract member, ensuring that no inconsistencies can arise between compiler versions.

## ACT-04S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | AdminContract.sol:L134-L153 |

### 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/AdminContract.sol

SOL

```sol
134 function setAddresses(
135     address _communityIssuanceAddress,
136     address _activePoolAddress,
137     address _defaultPoolAddress,
138     address _stabilityPoolAddress,
139     address _collSurplusPoolAddress,
140     address _priceFeedAddress,
141     address _shortTimelock,
142     address _longTimelock
143 ) external onlyOwner {
144     require(!isInitialized);
145     communityIssuance = ICommunityIssuance(_communityIssuanceAddress);
146     activePool = IActivePool(_activePoolAddress);
147     defaultPool = IDefaultPool(_defaultPoolAddress);
148     stabilityPool = IStabilityPool(_stabilityPoolAddress);
149     collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
150     priceFeed = IPriceFeed(_priceFeedAddress);
151     shortTimelock = _shortTimelock;
152     longTimelock = _longTimelock;
153 }
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# BorrowerOperations Static Analysis Findings

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

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | BorrowerOperations.sol:L91-L111 |

### 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/BorrowerOperations.sol

SOL

```sol
function setAddresses(
    address _vesselManagerAddress,
    address _stabilityPoolAddress,
    address _gasPoolAddress,
    address _collSurplusPoolAddress,
    address _sortedVesselsAddress,
    address _debtTokenAddress,
    address _feeCollectorAddress,
    address _adminContractAddress
) external override {
    require(!isInitialized, "Already initialized");
    isInitialized = true;
    vesselManager = IVesselManager(_vesselManagerAddress);
    stabilityPool = IStabilityPool(_stabilityPoolAddress);
    gasPoolAddress = _gasPoolAddress;
    collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
    sortedVessels = ISortedVessels(_sortedVesselsAddress);
    debtToken = IDebtToken(_debtTokenAddress);
    feeCollector = IFeeCollector(_feeCollectorAddress);
    adminContract = IAdminContract(_adminContractAddress);
}
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# CollSurplusPool Static Analysis Findings

## CSP-01S: Inexistent Visibility Specifier

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | CollSurplusPool.sol:L26 |

### Description:

The linked variable has no visibility specifier explicitly set.

### Example:

contracts/CollSurplusPool.sol

SOL

```
26  mapping(address => uint256) balances;
```

### 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:

An `internal` visibility specifier has been introduced to the `balances` contract member, ensuring that no inconsistencies can arise between compiler versions.

## CSP-02S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | CollSurplusPool.sol:L32-L49 |

### 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/CollSurplusPool.sol

SOL

```sol
function setAddresses(
    address _activePoolAddress,
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _vesselManagerOperationsAddress
) external override initializer {
    require(!isInitialized, "Already initialized");
    isInitialized = true;

    __Ownable_init();

    activePoolAddress = _activePoolAddress;
    borrowerOperationsAddress = _borrowerOperationsAddress;
    vesselManagerAddress = _vesselManagerAddress;
    vesselManagerOperationsAddress = _vesselManagerOperationsAddress;

    renounceOwnership();
}
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# DebtToken Static Analysis Findings

## DTN-01S: Inexistent Event Emissions

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Informational | DebtToken.sol:L88-L90, L92-L94 |

### Description:

The linked functions adjust sensitive contract variables yet do not emit an event for it.

### Example:

contracts/DebtToken.sol

SOL

```sol
88  function addWhitelist(address _address) external override onlyTimelock {
89      whitelistedContracts[_address] = true;
90  }
```

### Recommendation:

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

### Alleviation:

A `WhitelistChanged` event has been introduced to the `DebtToken` contract and is now correspondingly emitted in both referenced functions, alleviating this exhibit in full.

# DTN-02S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | DebtToken.sol:L42-L52 |

### 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/DebtToken.sol

SOL

```sol
42  constructor(
43      address _vesselManagerAddress,
44      address _stabilityPoolAddress,
45      address _borrowerOperationsAddress,
46      address _timelockAddress
47  ) ERC20("GRAI", "GRAI") {
48      vesselManagerAddress = _vesselManagerAddress;
49      timelockAddress = _timelockAddress;
50      stabilityPool = IStabilityPool(_stabilityPoolAddress);
51      borrowerOperationsAddress = _borrowerOperationsAddress;
52  }
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# DefaultPool Static Analysis Findings

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

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | DefaultPool.sol:L36-L49 |

### 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/DefaultPool.sol

SOL

```sol
36  function setAddresses(address _vesselManagerAddress, address _activePoolAddress)
37      external
38      initializer
39  {
40      require(!isInitialized, "Already initialized");
41      isInitialized = true;
42
43      __Ownable_init();
44
45      vesselManagerAddress = _vesselManagerAddress;
46      activePoolAddress = _activePoolAddress;
47
48      renounceOwnership();
49  }
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# FeeCollector Static Analysis Findings

## FCR-01S: Data Location Optimizations

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | FeeCollector.sol:L136 |

### Description:

The linked input arguments are set as `memory` in `external` function(s).

### Example:

contracts/FeeCollector.sol

SOL

```sol
136 function collectFees(address[] memory _borrowers, address[] memory _assets) external c
```

### Recommendation:

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

### Alleviation:

All input arguments of the **`FeeCollector::collectFees`** function have been adjusted to `calldata`, optimizing their read-access gas cost significantly.

# FCR-02S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | FeeCollector.sol:L42-L63 |

### 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/FeeCollector.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _grvtStakingAddress,
    address _debtTokenAddress,
    address _treasuryAddress,
    bool _routeToGRVTStaking
) external initializer {
    require(!isInitialized);
    require(_treasuryAddress != address(0));
    borrowerOperationsAddress = _borrowerOperationsAddress;
    vesselManagerAddress = _vesselManagerAddress;
    grvtStaking = IGRVTStaking(_grvtStakingAddress);
    debtTokenAddress = _debtTokenAddress;
    treasuryAddress = _treasuryAddress;
    routeToGRVTStaking = _routeToGRVTStaking;
    if (_routeToGRVTStaking && address(grvtStaking) == address(0)) {
        revert FeeCollector__InvalidGRVTStakingAddress();
    }
    __Ownable_init();
    isInitialized = true;
}
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# FCR-03S: Improper Invocations of EIP-20 `transfer`

| Type | Severity | Location |
| --- | --- | --- |
| **Standard Conformity** | Medium | FeeCollector.sol:L343, L358 |

### Description:

The linked statement do not properly validate the returned `bool` of the **EIP-20** standard `transfer` function. As the **standard dictates**, callers **must not** assume that `false` is never returned.

### Impact:

If the code mandates that the returned `bool` is `true`, this will cause incompatibility with tokens such as USDT / Tether as no such `bool` is returned to be evaluated causing the check to fail at all times. On the other hand, if the token utilized can return a `false` value under certain conditions but the code does not validate it, the contract itself can be compromised as having received / sent funds that it never did.

### Example:

contracts/FeeCollector.sol

SOL

```
343 IDebtToken(debtTokenAddress).transfer(collector, _feeAmount);
```

### Recommendation:

Since not all standardized tokens are **EIP-20** compliant (such as Tether / USDT), we advise a safe wrapper library to be utilized instead such as `SafeERC20` by OpenZeppelin to opportunistically validate the returned `bool` only if it exists in each instance.

### Alleviation:

Both **EIP-20** transfer instances now utilize their `safe`-prefixed counterparts, ensuring that they are performed safely regardless of the underlying **EIP-20** implementation.

# GravitaMath Static Analysis Findings

## GMH-01S: Illegible Numeric Value Representation

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | GravitaMath.sol:L62, L63 |

### Description:

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

### Example:

contracts/Dependencies/GravitaMath.sol

SOL

```
62  if (_minutes > 525600000) {
```

### 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:

The value, now relocated to its dedicated `EXPONENT_CAP` declaration, has had the underscore separator introduced in the correct locations thus alleviating this exhibit.

# PriceFeed Static Analysis Findings

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

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | PriceFeed.sol:L54-L67 |

### 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/PriceFeed.sol

SOL

```sol
54  function setAddresses(
55      address _adminContract,
56      address _rethToken,
57      address _stethToken,
58      address _wstethToken
59  ) external initializer {
60      require(!isInitialized);
61      isInitialized = true;
62      __Ownable_init();
63      adminContract = _adminContract;
64      rethToken = _rethToken;
65      stethToken = _stethToken;
66      wstethToken = _wstethToken;
67  }
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# SortedVessels Static Analysis Findings

## SVS-01S: Inexistent Visibility Specifier

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | SortedVessels.sol:L49 |

### Description:

The linked variable has no visibility specifier explicitly set.

### Example:

contracts/SortedVessels.sol

SOL

```sol
49  uint256 constant MAX_UINT256 = type(uint256).max;
```

### 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:

The referenced variable is no longer present in the codebase rendering this exhibit no longer applicable.

# StabilityPool Static Analysis Findings

## SPL-01S: Inexistent Visibility Specifier

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | StabilityPool.sol:L174 |

### Description:

The linked variable has no visibility specifier explicitly set.

### Example:

contracts/StabilityPool.sol

SOL

```sol
174 mapping(address => Colls) pendingCollGains;
```

### 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:

The `pendingCollGains` variable is no longer present in the codebase rendering this exhibit no longer applicable.

# SPL-02S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | StabilityPool.sol:L242-L268 |

### 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/StabilityPool.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _activePoolAddress,
    address _debtTokenAddress,
    address _sortedVesselsAddress,
    address _communityIssuanceAddress,
    address _adminContractAddress
) external initializer override {
    require(!isInitialized, "StabilityPool: Already initialized");

    isInitialized = true;
    __Ownable_init();
    __ReentrancyGuard_init();

    borrowerOperations = IBorrowerOperations(_borrowerOperationsAddress);
    vesselManager = IVesselManager(_vesselManagerAddress);
    activePool = IActivePool(_activePoolAddress);
    debtToken = IDebtToken(_debtTokenAddress);
    sortedVessels = ISortedVessels(_sortedVesselsAddress);
    communityIssuance = ICommunityIssuance(_communityIssuanceAddress);
    adminContract = IAdminContract(_adminContractAddress);

    P = DECIMAL_PRECISION;

    renounceOwnership();
}
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# VesselManager Static Analysis Findings

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

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | VesselManager.sol:L124-L147 |

### 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/VesselManager.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _stabilityPoolAddress,
    address _gasPoolAddress,
    address _collSurplusPoolAddress,
    address _debtTokenAddress,
    address _feeCollectorAddress,
    address _sortedVesselsAddress,
    address _vesselManagerOperationsAddress,
    address _adminContractAddress
) external override initializer {
    require(!isInitialized, "Already initialized");
    isInitialized = true;
    __Ownable_init();
    borrowerOperations = _borrowerOperationsAddress;
    stabilityPool = IStabilityPool(_stabilityPoolAddress);
    gasPoolAddress = _gasPoolAddress;
    collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
    debtToken = IDebtToken(_debtTokenAddress);
    feeCollector = IFeeCollector(_feeCollectorAddress);
    sortedVessels = ISortedVessels(_sortedVesselsAddress);
    vesselManagerOperations = IVesselManagerOperations(_vesselManagerOperationsAddress
    adminContract = IAdminContract(_adminContractAddress);
}
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# VesselManagerOperations Static Analysis Findings

## VMO-01S: Illegible Numeric Value Representations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | VesselManagerOperations.sol:L15, L389, L960 |

### Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

### Example:

contracts/VesselManagerOperations.sol

SOL

```sol
15  uint256 public constant REDEMPTION_SOFTENING_PARAM = 970; // 97%
```

### Recommendation:

To properly illustrate each 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:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

## VMO-02S: Inexistent Sanitization of Input Addresses

| Type | Severity | Location |
| --- | --- | --- |
| **Input Sanitization** | Minor | VesselManagerOperations.sol:L59-L76 |

### 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/VesselManagerOperations.sol

SOL

```sol
59  function setAddresses(
60      address _vesselManagerAddress,
61      address _sortedVesselsAddress,
62      address _stabilityPoolAddress,
63      address _collSurplusPoolAddress,
64      address _debtTokenAddress,
65      address _adminContractAddress
66  ) external initializer {
67      require(!isInitialized, "Already initialized");
68      __Ownable_init();
69      vesselManager = IVesselManager(_vesselManagerAddress);
70      sortedVessels = ISortedVessels(_sortedVesselsAddress);
71      stabilityPool = IStabilityPool(_stabilityPoolAddress);
72      collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
73      debtToken = IDebtToken(_debtTokenAddress);
74      adminContract = IAdminContract(_adminContractAddress);
75      isInitialized = true;
76  }
```

### Recommendation:

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

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# AdminContract Manual Review Findings

## ACT-01M: Improper Reset Functionality

| Type | Severity | Location |
| --- | --- | --- |
| **Centralization Concern** | Unknown | AdminContract.sol:L279-L281 |

### Description:

The `AdminContract::setAsDefault` function permits the configuration of a collateral to be re-set to its default values, a trait that should not be accessible to a centralized party.

### Example:

contracts/AdminContract.sol

SOL

```sol
279 function setAsDefault(address _collateral) external onlyOwner {
280     _setAsDefault(_collateral);
281 }
```

### Recommendation:

We advise this function to either be omitted from the codebase or locked behind the long timelock to avoid improper resets of collateral configurations.

### Alleviation:

The default values of a collateral parameterization have been relocated to the `AdminContract::addNewCollateral` function instead, ensuring that these default values cannot be adjusted and that they are applied in a trustless fashion to each new collateral rather than being set by a centralized entity. As such, this exhibit has been alleviated as no `AdminContract::setAsDefault` or similar mechanism is present in the codebase.

# ACT-02M: Improper Permission of Collateral Activation

| Type | Severity | Location |
| --- | --- | --- |
| **Logical Fault** | Medium | AdminContract.sol:L273-L277 |

### Description:

The `AdminContract::sanitizeParameters` function permits any **EIP-20** asset to be configured within the Gravita Protocol, a trait that is highly undesirable.

### Impact:

While a collateral would still need an oracle to be configured for it to behave properly, the ability to arbitrarily configure a collateral to its default values is an ill-advised trait that can be exploited under ideal conditions, such as an oracle being initialized prior to the collateral being configured by a timelock vote.

### Example:

contracts/AdminContract.sol

SOL

```sol
273 function sanitizeParameters(address _collateral) external {
274     if (!collateralParams[_collateral].hasCollateralConfigured) {
275         _setAsDefault(_collateral);
276     }
277 }
```

### Recommendation:

We advise the code to disallow such an initialization, instead ensuring that the collateral has already been configured wherever it is invoked (i.e. `BorrowerOperations::openVessel`).

### Alleviation:

The `AdminContract::sanitizeParameters` function has been omitted from the codebase entirely and the `BorrowerOperations::openVessel` function that was utilizing it now ensures that the `_asset` is active at the `AdminContract` instance, rendering this exhibit fully alleviated.

# ACT-03M: Improper Capability of Gas Compensation Adjustment

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Major | AdminContract.sol:L376-L388 |

### Description:

The gas compensation that is provided for Vessels is an integral part of the protocol and must not change throughout an asset's lifetime as it will retroactively affect existing vessels, potentially causing them to acquire higher / smaller collateral values than expected.

### Impact:

All debt-related functions (i.e. `GravitaBase::_getCompositeDebt`, `VesselManagerOperations::_liquidateNormalMode`, etc.) will be significantly affected by a downward / upward movement in the gas compensation to a point whereby the system's accounting will become inaccurate and over-track / under-track the debt of existing vessels.

### Example:

contracts/AdminContract.sol

SOL

```sol
376 function setDebtTokenGasCompensation(
377     address _collateral,
378     uint256 gasCompensation
379 )
380     public
381     override
382     longTimelockOnly
383     safeCheck("Gas Compensation", _collateral, gasCompensation, 1 ether, 400 ether)
384 {
385     uint256 oldGasComp = collateralParams[_collateral].debtTokenGasCompensation;
386     collateralParams[_collateral].debtTokenGasCompensation = gasCompensation;
387     emit GasCompensationChanged(oldGasComp, gasCompensation);
388 }
```

### Recommendation:

We advise this function to be omitted and configuration of the `debtTokenGasCompensation` to solely be permitted during an asset's initialization in the system.

### Alleviation:

Our recommended course of action has been applied fully, removing the `AdminContract::setDebtTokenGasCompensation` function from the system entirely and permitting configuration of this value solely during a collateral's inclusion to the system via `AdminContract::addNewCollateral`.

# ERC20Permit Manual Review Findings

## ERC-01M: Insecure EIP-2612 Implementation

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | ERC20Permit.sol:L61-L78 |

### Description:

The `ERC20Permit` contract will calculate the `DOMAIN_SEPARATOR` only once during its lifetime within its `constructor`.

### Impact:

While the likelihood of a blockchain fork resulting in a viable chain is very low, the attack vector is trivially exploitable should this happen and would cause fund loss.

### Example:

contracts/Dependencies/ERC20Permit.sol

SOL

```
61  constructor() {
62      uint256 chainID;
63      assembly {
64          chainID := chainid()
65      }
66
67      DOMAIN_SEPARATOR = keccak256(
68          abi.encode(
69              keccak256(
70                  "EIP712Domain(string name,string version,uint256 chainId,address verif
71              ),
72              keccak256(bytes(name())),
73              keccak256(bytes("1")), // Version
74              chainID,
75              address(this)
76          )
77      );
78  }
```

### Recommendation:

We strongly advise a paradigm similar to **OpenZeppelin's** `draft-ERC20Permit` to be applied, re-calculating the `DOMAIN_SEPARATOR` with the current `chainid` on a need-to basis as the contract is currently susceptible to cross-chain replay attacks should the blockchain it is deployed into be forked.

### Alleviation:

The Gravita Protocol team has opted to not apply a remediation for this exhibit thus rendering it acknowledged.

# ERC-02M: Insecure Elliptic Curve Recovery Mechanism

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Medium | ERC20Permit.sol:L101 |

### Description:

The `ecrecover` function is a low-level cryptographic function that should be utilized after appropriate sanitizations have been enforced on its arguments, namely on the `s` and `v` values. This is due to the inherent trait of the curve to be symmetrical on the x-axis and thus permitting signatures to be replayed with the same `x` value (`r`) but a different `y` value (`s`).

### Impact:

Should the payload being verified by the signature rely on differentiation based on the `s` or `v` arguments, it will be possible to replay the signature for the same data validly and acquire authorization twice.

### Example:

contracts/Dependencies/ERC20Permit.sol

SOL

```sol
function permit(
    address owner,
    address spender,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) public virtual override {
    require(block.timestamp <= deadline, "Permit: expired deadline");

    bytes32 hashStruct = keccak256(
        abi.encode(PERMIT_TYPEHASH, owner, spender, amount, _nonces[owner].current(),
    );

    bytes32 _hash = keccak256(abi.encodePacked(uint16(0x1901), DOMAIN_SEPARATOR, hashS

    address signer = ecrecover(_hash, v, r, s);
    require(signer != address(0) && signer == owner, "ERC20Permit: Invalid signature")

    _nonces[owner].increment();
    _approve(owner, spender, amount);
}
```

### Recommendation:

We advise them to be sanitized by ensuring that `v` is equal to either `27` or `28` (`v ∈ {27, 28}`) and to ensure that `s` is existent in the lower half order of the elliptic curve (`0 < s < secp256k1n ÷ 2 + 1`) by ensuring it is less than `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A1`. A reference implementation of those checks can be observed in the **ECDSA** library of OpenZeppelin and the rationale behind those restrictions exists within **Appendix F of the Yellow Paper**.

### Alleviation:

The `ECDSA` library of OpenZeppelin is now in use by the codebase that applies the relevant security checks, alleviating this exhibit.

# GravitaSafeMath128 Manual Review Findings

## GSM-01M: Improper Application of Safe Arithmetics

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Informational | GravitaSafeMath128.sol:L9, L17 |

### Description:

The `GravitaSafeMath128` contract improperly applies "safety" in the **`GravitaSafeMath128::add`** function by evaluating a `require` conditional after the unsafe operation has been performed. Additionally, the **`GravitaSafeMath128::sub`** function will apply a `require` check that guarantees the safety of the ensuing subtraction, executing it inefficiently.

### Example:

contracts/Dependencies/GravitaSafeMath128.sol

SOL

```sol
7   library GravitaSafeMath128 {
8       function add(uint128 a, uint128 b) internal pure returns (uint128) {
9           uint128 c = a + b;
10          require(c >= a, "GravitaSafeMath128: addition overflow");
11
12          return c;
13      }
14
15      function sub(uint128 a, uint128 b) internal pure returns (uint128) {
16          require(b <= a, "GravitaSafeMath128: subtraction overflow");
17          uint128 c = a - b;
18
19          return c;
20      }
21  }
```

### Recommendation:

We advise both code blocks to be wrapped in `unchecked` code blocks due to Solidity's built-in safe arithmetics in versions `0.8.X` and up. In the present code, an overflow in **`GravitaSafeMath128::add`** will **never yield the error message of the `require` check** as the overflow would fail immediately during the addition. As such, the code presently has unreachable statements as well as inefficient code in both of its functions.

### Alleviation:

The `GravitaSafeMath128` contract has been omitted from the codebase entirely as a result of this finding. As a result, we consider this exhibit alleviated as its described issue is no longer present in the codebase.

# PriceFeed Manual Review Findings

## PFD-01M: Significant Centralization of Sensitive Functionality

| Type | Severity | Location |
| --- | --- | --- |
| Centralization Concern | Unknown | PriceFeed.sol:L71-L85, L87-L89, L91-L93 |

### Description:

The `PriceFeed` oracle system can be adjusted by the owner and / or `adminContract` of the Gravita Protocol system at will.

### Example:

contracts/PriceFeed.sol

SOL

```sol
function addOracle(
    address _token,
    address _chainlinkOracle,
    bool _isEthIndexed
) external override isController {
    AggregatorV3Interface newOracle = AggregatorV3Interface(_chainlinkOracle);
    _validateFeedResponse(newOracle);
    if (registeredOracles[_token].exists) {
        uint256 timelockRelease = block.timestamp.add(_getOracleUpdateTimelock());
        queuedOracles[_token] = OracleRecord(newOracle, timelockRelease, true, true, _
    } else {
        registeredOracles[_token] = OracleRecord(newOracle, block.timestamp, true, tru
        emit NewOracleRegistered(_token, _chainlinkOracle, _isEthIndexed);
    }
}

function deleteOracle(address _token) external override isController {
    delete registeredOracles[_token];
}

function deleteQueuedOracle(address _token) external override isController {
    delete queuedOracles[_token];
}
```

### Recommendation:

We advise these functions to be invoke-able solely by governance mechanisms as they present a significant centralization threat to the protocol. To note, the oracle update timelock can be bypassed entirely by invoking `PriceFeed::deleteOracle` followed by `PriceFeed::addOracle`, a trait that should also be addressed in the system.

### Alleviation:

The code was revised to instead ensure that the `PriceFeed::addOracle` (now labelled `PriceFeed::setOracle`) function can be solely invoked by a timelock instead of a centralized entity. As such, we consider this exhibit alleviated provided that the timelock is in use by a multi-signature wallet, DAO, or similar multi-party collective.

# PFD-02M: Incorrect Error Handling

| Type | Severity | Location |
| --- | --- | --- |
| **Logical Fault** | Minor | PriceFeed.sol:L281 |

### Description:

The `catch` clause of the first `try-catch` construct in **`PriceFeed::_fetchCurrentFeedResponse`** is incorrect as it will continue execution of the function. As such, if the `_priceAggregator` does not implement the `decimals` function but implements the `latestRoundData` function it will be accepted by the contract as correct with a decimal accuracy of `0` incorrectly.

### Impact:

The potential of an aggregator supporting the `latestRoundData` function but not the `decimals` one is inexistent, however, custom oracle implementations may fall into this category and would cause the system to misbehave greatly.

### Example:

contracts/PriceFeed.sol

SOL

```sol
function _fetchCurrentFeedResponse(AggregatorV3Interface _priceAggregator)
    internal
    view
    returns (FeedResponse memory response)
{
    try _priceAggregator.decimals() returns (uint8 decimals) {
        response.decimals = decimals;
    } catch {}
    try _priceAggregator.latestRoundData() returns (
        uint80 roundId,
        int256 answer,
        uint256, /* startedAt */
        uint256 timestamp,
        uint80 /* answeredInRound */
    ) {
        response.roundId = roundId;
        response.answer = answer;
        response.timestamp = timestamp;
        response.success = true;
    } catch {}
}
```

### Recommendation:

We advise the code to instead yield `response` directly in the first `catch` clause, ensuring that the Chainlink response is treated as invalid if the feed does not support the `decimals` function similarly to the Liquity implementation.

### Alleviation:

The code was updated to yield the empty `response` akin to the `catch` block of the `latestRoundData` invocation, ensuring that the code properly fails if the `decimals` function is not supported by the Chainlink oracle being added.

# PFD-03M: Inexistent Initialization of Price

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | PriceFeed.sol:L71-L85 |

### Description:

The registration of an oracle to the system via `PriceFeed::addOracle` does not set an initial price for the asset in contrast to the Liquity implementation. As such, if `PriceFeed::fetchPrice` is invoked when the Chainlink oracle stops behaving properly the yielded `lastTokenGoodPrice` will be `0` incorrectly.

### Impact:

If an oracle is added to the system and immediately stops behaving properly, the `PriceFeed::fetchPrice` function will yield an incorrect price of `0` that will be consumed by its callers.

### Example:

contracts/PriceFeed.sol

SOL

```sol
71  function addOracle(
72      address _token,
73      address _chainlinkOracle,
74      bool _isEthIndexed
75  ) external override isController {
76      AggregatorV3Interface newOracle = AggregatorV3Interface(_chainlinkOracle);
77      _validateFeedResponse(newOracle);
78      if (registeredOracles[_token].exists) {
79          uint256 timelockRelease = block.timestamp.add(_getOracleUpdateTimelock());
80          queuedOracles[_token] = OracleRecord(newOracle, timelockRelease, true, true, _
81      } else {
82          registeredOracles[_token] = OracleRecord(newOracle, block.timestamp, true, tru
83          emit NewOracleRegistered(_token, _chainlinkOracle, _isEthIndexed);
84      }
85  }
```

### Recommendation:

We advise the `PriceFeed::addOracle` function to set the latest good price as well, ensuring that the system will behave properly under all circumstances.

### Alleviation:

The `PriceFeed::addOracle` (now labelled `PriceFeed::setOracle`) function now properly extracts and consumes the most recent responses of the Chainlink oracle being added, rendering the behaviour outlined in the exhibit impossible in the latest iteration of the codebase.

# PFD-04M: Incorrect Lido Staked ETH Value Assumption

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Medium | PriceFeed.sol:L171 |

### Description:

The referenced statement will attempt to query the `stETH` price using a USD oracle and if it does not exist, it will treat the `stETH` equivalent of the `wstETH` as one-to-one interchangeable with `ETH` thus using the price of `ETH` to calculate the price of the `wstETH` value.

### Impact:

The arbitrage opportunities introduced can lead to the creation of bad debt in the system and can be exaggerated via flash-loans.

### Example:

contracts/PriceFeed.sol

SOL

```
168 function _fetchNativeWstETHPrice() internal returns (uint256 price) {
169     uint256 wstEthToStEthValue = _getWstETH_StETHValue();
170     OracleRecord storage stEth_UsdOracle = registeredOracles[stethToken];
171     price = stEth_UsdOracle.exists ? this.fetchPrice(stethToken) : _calcEthPrice(wstEt
172     _storePrice(wstethToken, price);
173 }
```

### Recommendation:

We advise this fallback mechanism to be omitted as staked counterparts of `ETH` always trade at either a premium or a loss in comparison to the actual `ETH` asset, causing `PriceFeed::_fetchNativeWstETHPrice` to introduce arbitrage opportunities.

### Alleviation:

Assets that relate to `ETH2.0` wrapped `ETH` are no longer treated as a special case by the oracle, instead utilizing the traditional Chainlink-related methodology to assess their price. As such, we consider this exhibit fully alleviated.

# PFD-05M: Incorrect Lido Staked ETH Price Usage

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Major | PriceFeed.sol:L171 |

### Description:

The referenced statement will fetch the price of the `stETH` token and store it as the price of the `wstETH` token which is incorrect.

### Impact:

The price reported per unit of `wstETH` will always be incorrect if a USD oracle has been defined for `stETH` as it will yield the price of `stETH` and not `wstETH`.

### Example:

contracts/PriceFeed.sol

SOL

```
168 function _fetchNativeWstETHPrice() internal returns (uint256 price) {
169     uint256 wstEthToStEthValue = _getWstETH_StETHValue();
170     OracleRecord storage stEth_UsdOracle = registeredOracles[stethToken];
171     price = stEth_UsdOracle.exists ? this.fetchPrice(stethToken) : _calcEthPrice(wstEt
172     _storePrice(wstethToken, price);
173 }
```

### Recommendation:

We advise the `price` of the `stETH` token fetched to be multiplied by the `wstEthToStEthValue` as it represents the exchange rate between `wstETH` and `stETH`, the former's price being what we are interested in.

### Alleviation:

Assets that relate to `ETH2.0` wrapped `ETH` are no longer treated as a special case by the oracle, instead utilizing the traditional Chainlink-related methodology to assess their price. As such, we consider this exhibit fully alleviated.

# SafeMath Manual Review Findings

## SMH-01M: Improper Application of Safe Arithmetics

| Type | Severity | Location |
| --- | --- | --- |
| **Language Specific** | Informational | SafeMath.sol:L32, L68, L90 |

### Description:

The `SafeMath` contract improperly applies "safety" in the **`SafeMath::add`** and **`SafeMath::mul`** functions by evaluating a `require` conditional after each unsafe operation has been performed. Additionally, the **`SafeMath::sub`** function will apply a `require` check that guarantees the safety of the ensuing subtraction, executing it inefficiently.

### Example:

contracts/Dependencies/SafeMath.sol

SOL

```sol
/**
 * @dev Returns the subtraction of two unsigned integers, reverting with custom messag
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 * - Subtraction cannot overflow.
 *
 * _Available since v2.4.0._
 */
function sub(
    uint256 a,
    uint256 b,
    string memory errorMessage
) internal pure returns (uint256) {
    require(b <= a, errorMessage);
    uint256 c = a - b;

    return c;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `*` operator.
 *
 * Requirements:
 * - Multiplication cannot overflow.
 */
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
    if (a == 0) {
        return 0;
    }

    uint256 c = a * b;
    require(c / a == b, "mul overflow");

    return c;
}
```

### Recommendation:

We advise both code blocks to be wrapped in `unchecked` code blocks due to Solidity's built-in safe arithmetics in versions `0.8.X` and up. In the present code, an overflow in **`SafeMath::add`** / **`SafeMath::mul`** will **never yield the error message of the `require` check** as the overflow would fail immediately during the addition / multiplication. As such, the code presently has unreachable statements as well as inefficient code in all of its functions.

### Alleviation:

The `SafeMath` contract has been omitted from the codebase entirely as a result of this finding. As a result, we consider this exhibit alleviated as its described issue is no longer present in the codebase.

# SafetyTransfer Manual Review Findings

## STR-01M: Incorrect Decimal Assumption

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Minor | SafetyTransfer.sol:L21 |

### Description:

The `SafetyTransfer::decimalsCorrection` function will misbehave if the `_token` has decimals that are greater than the value of `18`.

### Impact:

The decimal correction mechanism will be incorrect in tokens with abnormal decimals, yielding significantly less values than expected.

### Example:

contracts/Dependencies/SafetyTransfer.sol

SOL

```sol
//_amount is in ether (1e18) and we want to convert it to the token decimal
function decimalsCorrection(address _token, uint256 _amount)
    internal
    view
    returns (uint256)
{
    if (_token == address(0)) return _amount;
    if (_amount == 0) return 0;

    uint8 decimals = ERC20Decimals(_token).decimals();
    if (decimals < 18) {
        return _amount.div(10**(18 - decimals));
    }

    return _amount;
}
```

### Recommendation:

We advise the code to introduce an `else` branch that evaluates whether `decimals` is greater-than (`>`) the value of `18`, in which case it should offset the `_amount` via a multiplication rather than division.

### Alleviation:

The `decimals` of a token are properly handled by the **`SafetyTransfer::decimalsCorrection`** function as they are normalized in either an upwards or downwards trajectory depending on whether the decimals exceed the default value of `18` or subceed it.

## STR-02M: Insecure Conversion of Amount

| Type | Severity | Location |
| --- | --- | --- |
| **Mathematical Operations** | Major | SafetyTransfer.sol:L22 |

### Description:

The `SafetyTransfer::decimalsCorrection` function is utilized to assess **how much funds should be transferred**. As such, it is possible to specify a value that will truncate to `0` when "normalized" to the token's decimal accuracy, permitting deposits of zero funds to acquire a non-zero effective value in the protocol.

### Impact:

It is currently possible to trick functions such as `BorrowerOperations::_activePoolAddColl` to perform a zero-value transfer yet credit a non-zero deposit value to the caller, significantly compromising the operational integrity of the protocol.

### Example:

contracts/Dependencies/SafetyTransfer.sol

SOL

```sol
function decimalsCorrection(address _token, uint256 _amount)
    internal
    view
    returns (uint256)
{
    if (_token == address(0)) return _amount;
    if (_amount == 0) return 0;

    uint8 decimals = ERC20Decimals(_token).decimals();
    if (decimals < 18) {
        return _amount.div(10**(18 - decimals));
    }

    return _amount;
}
```

### Recommendation:

We advise the code to mandate that `_amount` modulo (`%`) the divisor (i.e. `10**(18 - decimals)`) equals zero, preventing impossible deposit values from being specified.

### Alleviation:

The **`SafetyTransfer::decimalsCorrection`** function will now validate that the amount being converted is fully divisible via a modulo (`%`) operator, ensuring that the code will never yield assets that are less than the expected amount.

# SortedVessels Manual Review Findings

## SVS-01M: Insecure Data List Size Enforcement

| Type | Severity | Location |
| --- | --- | --- |
| Logical Fault | Informational | SortedVessels.sol:L121-L123 |

### Description:

The `SortedVessels` function will set the `maxSize` of an asset's linked list to the maximum of `uint256` insecurely, enabling Denial-of-Service attacks to manifest.

### Impact:

The overall list is utilized by off-chain components as per the Gravita Finance team. As such, the impact of this exhibit is negligible and has been downgraded to `informational` severity.

### Example:

contracts/SortedVessels.sol

SOL

```sol
121 if (data[_asset].maxSize == 0) {
122     data[_asset].maxSize = MAX_UINT256;
123 }
```

### Recommendation:

While the blockchain that the Gravita Protocol will be deployed in may have significantly less gas costs than its Liquity counterpart, it still needs to apply an upper bound as regardless of the cost of executing a transaction, **there is an inherent block gas limit** that needs to be respected. As such, we advise a higher than Liquity but still sensible bound to be applied to avoid Denial-of-Service attacks.

### Alleviation:

While the `MAX_UINT256` "unlimited" limit is no longer set for the `maxSize` variable of the list, no max size is set and the **`SortedVessels::isFull`** function is no longer utilized by the code. The Gravita Finance team has opted to acknowledge this exhibit as the function is purely utilized for off-chain purposes.

# StabilityPool Manual Review Findings

## SPL-01M: Inexistent Normalization of Asset

| Type | Severity | Location |
| --- | --- | --- |
| **Logical Fault** | Unknown | StabilityPool.sol:L801 |

### Description:

The `StabilityPool::_sendGainsToDepositor` function will not attempt to normalize the `amount` value when transferring the `asset` in contrast to the rest of the codebase.

### Impact:

Presently, the code will misbehave if non-18 decimal assets are introduced to `AdminContract` which is permitted and actually expected by some of the contracts in the system. If it is a business requirement to support unwrapped non-18 decimal assets, this finding will be upgraded in severity to "major".

### Example:

contracts/StabilityPool.sol

SOL

```sol
function _sendGainsToDepositor(
    address _to,
    address[] memory assets,
    uint256[] memory amounts
) internal {
    uint256 assetsLen = assets.length;
    require(assetsLen == amounts.length, "StabilityPool: Length mismatch");
    for (uint256 i = 0; i < assetsLen; ++i) {
        uint256 amount = amounts[i];
        if (amount == 0) {
            continue;
        }
        address asset = assets[i];
        // Assumes we're internally working only with the wrapped version of ERC20 tok
        IERC20Upgradeable(asset).safeTransferFrom(address(this), _to, amount);
    }
    totalColl.amounts = _leftSubColls(totalColl, assets, amounts);

    // Reset pendingCollGains since those were all sent to the borrower
    Colls memory tempPendingCollGains;
    pendingCollGains[_to] = tempPendingCollGains;
}
```

### Recommendation:

We advise the code to be streamlined, either normalizing the amount in `StabilityPool::_sendGainsToDepositor` or ensuring that only wrapped assets are introduced to the `AdminContract::addNewCollateral` function by evaluating their decimals.

### Alleviation:

The decimals of newly introduced assets via `AdminContract::addNewCollateral` are now mandated to be equal to `DEFAULT_DECIMALS`, streamlining the codebase and thus alleviating this exhibit as a result.

# Timelock Manual Review Findings

## TKC-01M: Inexistent Prevention of Duplicate Invocations

| Type | Severity | Location |
| --- | --- | --- |
| **Logical Fault** | Minor | Timelock.sol:L106-L125, L127-L138 |

### Description:

Based on the implementation of the Gravita Protocol codebase, the `Timelock` contract is expected to be managed by an EOA / multi-signature wallet rather than an on-chain decentralized smart contract. As such, calls to it aren't restricted similarly to how DAOs prevent the same payload to be queued again.

### Impact:

It is presently possible to emit events that do not correspond to the real state of the `Timelock`, cancelling a transaction that has already been executed thus breaking the guarantee that a `CancelTransaction` event is meant to indicate the transaction has not been executed and has been cancelled.

### Example:

contracts/Timelock.sol

SOL

```sol
function cancelTransaction(
    address target,
    uint value,
    string memory signature,
    bytes memory data,
    uint eta
) public adminOnly {
    bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
    queuedTransactions[txHash] = false;

    emit CancelTransaction(txHash, target, value, signature, data, eta);
}

function executeTransaction(
    address target,
    uint value,
    string memory signature,
    bytes memory data,
    uint eta
) public payable adminOnly returns (bytes memory) {
    bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
    if (!queuedTransactions[txHash]) {
        revert Timelock__TxNoQueued();
    }
    if (getBlockTimestamp() < eta) {
        revert Timelock__TxStillLocked();
    }
    if (getBlockTimestamp() > eta + GRACE_PERIOD) {
        revert Timelock__TxExpired();
    }

    queuedTransactions[txHash] = false;

    bytes memory callData;

    if (bytes(signature).length == 0) {
        callData = data;
    } else {
        callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
    }

    // Execute the call
    (bool success, bytes memory returnData) = target.call{ value: value }(callData);
    if (!success) {
        revert Timelock__TxReverted();
    }
```

```
172         }
173
174         emit ExecuteTransaction(txHash, target, value, signature, data, eta);
175
176         return returnData;
177     }
```

### Recommendation:

We advise the code of transaction queueing and transaction cancelling to prevent execution if the transaction is already queued or already cancelled respectively. This will prevent misleading `QueueTransaction` and `CancelTransaction` events from being emitted, such as a transaction actually being executed by **`Timelock::executeTransaction`** and then "cancelled" by **`Timelock::cancelTransaction`** even though it has already been executed.

### Alleviation:

The queue status of a transaction is now sanitized in all statements that adjust it, ensuring that it solely transitions from an unqueued to a queued state and vice versa.

# ActivePool Code Style Findings

## APL-01C: Inefficient Renunciation of Ownership

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | ActivePool.sol:L105 |

### Description:

The `ActivePool::setAddresses` function will invoke the `OwnableUpgradeable::renounceOwnership` function which in turn will apply the `onlyOwner` modifier redundantly.

### Example:

contracts/ActivePool.sol

SOL

```sol
84  function setAddresses(
85      address _borrowerOperationsAddress,
86      address _collSurplusPoolAddress,
87      address _defaultPoolAddress,
88      address _stabilityPoolAddress,
89      address _vesselManagerAddress,
90      address _vesselManagerOperationsAddress
91  ) external initializer {
92      require(!isInitialized, "Already initialized");
93      isInitialized = true;
94
95      __Ownable_init();
96      __ReentrancyGuard_init();
97
98      borrowerOperationsAddress = _borrowerOperationsAddress;
99      collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
100     defaultPool = IDefaultPool(_defaultPoolAddress);
101     stabilityPoolAddress = _stabilityPoolAddress;
102     vesselManagerAddress = _vesselManagerAddress;
103     vesselManagerOperationsAddress = _vesselManagerOperationsAddress;
104
105     renounceOwnership();
106 }
```

### Recommendation:

We advise the `OwnableUpgradeable::_transferOwnership` function to be utilized directly, transferring ownership to the zero address.

### Alleviation:

Ownership of the contract is no longer renounced in the latest iteration of the codebase rendering this exhibit inapplicable.

# APL-02C: Inexplicable Ownable Pattern

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | ActivePool.sol:L95, L105 |

### Description:

The `ActivePool` inherits the `OwnableUpgradeable` implementation redundantly as it initializes it within the `ActivePool::setAddresses` function and consequently renounces ownership in the same call.

### Example:

contracts/ActivePool.sol

SOL

```sol
84  function setAddresses(
85      address _borrowerOperationsAddress,
86      address _collSurplusPoolAddress,
87      address _defaultPoolAddress,
88      address _stabilityPoolAddress,
89      address _vesselManagerAddress,
90      address _vesselManagerOperationsAddress
91  ) external initializer {
92      require(!isInitialized, "Already initialized");
93      isInitialized = true;
94
95      __Ownable_init();
96      __ReentrancyGuard_init();
97
98      borrowerOperationsAddress = _borrowerOperationsAddress;
99      collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
100     defaultPool = IDefaultPool(_defaultPoolAddress);
101     stabilityPoolAddress = _stabilityPoolAddress;
102     vesselManagerAddress = _vesselManagerAddress;
103     vesselManagerOperationsAddress = _vesselManagerOperationsAddress;
104
105     renounceOwnership();
106 }
```

### Recommendation:

We advise it to be removed, inheriting the `Initializable` implementation of OpenZeppelin instead which is properly put in use within the contract.

### Alleviation:

While the renunciation has been removed, the `OwnableUpgradeable` contract is still inherited by the `ActivePool`. To properly alleviate this exhibit, we advise the `OwnableUpgradeable` contract to be omitted from the `ActivePool` entirely.

# APL-03C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | ActivePool.sol:L91-L93 |

### Description:

The `ActivePool` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the `ActivePool::setAddresses` function. As such, the manual `isInitialized` flag is redundant.

### Example:

contracts/ActivePool.sol

SOL

```sol
84  function setAddresses(
85      address _borrowerOperationsAddress,
86      address _collSurplusPoolAddress,
87      address _defaultPoolAddress,
88      address _stabilityPoolAddress,
89      address _vesselManagerAddress,
90      address _vesselManagerOperationsAddress
91  ) external initializer {
92      require(!isInitialized, "Already initialized");
93      isInitialized = true;
94
95      __Ownable_init();
96      __ReentrancyGuard_init();
97
98      borrowerOperationsAddress = _borrowerOperationsAddress;
99      collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
100     defaultPool = IDefaultPool(_defaultPoolAddress);
101     stabilityPoolAddress = _stabilityPoolAddress;
102     vesselManagerAddress = _vesselManagerAddress;
103     vesselManagerOperationsAddress = _vesselManagerOperationsAddress;
104
105     renounceOwnership();
106 }
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# AdminContract Code Style Findings

## ACT-01C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | AdminContract.sol:L294, L295, L323-L324, L338-L339, L353-L354, L368, L371, L385-L386, L399-L400, L414, L417, L422, L425 |

### Description:

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

### Example:

contracts/AdminContract.sol

SOL

```sol
283 function setAsDefaultWithRemptionBlock(
284     address _collateral,
285     uint256 blockInDays
286 )
287     external
288     onlyOwner // TODO: Review if should set to controller
289 {
290     if (blockInDays > 14) {
291         blockInDays = REDEMPTION_BLOCK_DAY;
292     }
293
294     if (collateralParams[_collateral].redemptionBlock == 0) {
295         collateralParams[_collateral].redemptionBlock = block.timestamp + (blockInDays
296     }
297
298     _setAsDefault(_collateral);
299 }
```

### 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.

### Alleviation:

All inefficient `mapping` lookups have been significantly optimized per our recommendation, rendering this exhibit fully alleviated.

# ACT-02C: Inexistent Error Message

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | AdminContract.sol:L144 |

### Description:

The linked `require` check has no error message explicitly defined.

### Example:

contracts/AdminContract.sol

SOL

```sol
144 require(!isInitialized);
```

### Recommendation:

We advise one to be set so to increase the legibility of the codebase and aid in validating the `require` check's condition.

### Alleviation:

An error message has been properly introduced to the referenced `require` check as advised.

# ACT-03C: Loop Iterator Optimizations

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | AdminContract.sol:L214, L245 |

### Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

### Example:

contracts/AdminContract.sol

SOL

```sol
214 for (uint256 i = 0; i < _collaterals.length; i++) {
```

### Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

### Alleviation:

The loop iterator increments have been optimized as advised, however, their `i++` counterpart is utilized instead of `++i`. We advise the latter to be set in use as it is more optimal than the present code.

# ACT-04C: Misleading Variable Name

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | AdminContract.sol:L294-L295 |

## Description:

The `redemptionBlock` of the `collateralParams` of a given asset does not represent blocks and instead represents time as evidenced in **`AdminContract::setAsDefaultWithRemptionBlock`** and **`VesselManagerOperations::_validateRedemptionRequirements`**.

## Example:

contracts/AdminContract.sol

SOL

```sol
283 function setAsDefaultWithRemptionBlock(
284     address _collateral,
285     uint256 blockInDays
286 )
287     external
288     onlyOwner // TODO: Review if should set to controller
289 {
290     if (blockInDays > 14) {
291         blockInDays = REDEMPTION_BLOCK_DAY;
292     }
293
294     if (collateralParams[_collateral].redemptionBlock == 0) {
295         collateralParams[_collateral].redemptionBlock = block.timestamp + (blockInDays
296     }
297
298     _setAsDefault(_collateral);
299 }
```

### Recommendation:

We advise the data point to be aptly renamed to illustrate that it represents time rather than blocks, avoiding potential confusion when reading the codebase.

### Alleviation:

The `redemptionBlock` variable was renamed to `redemptionBlockTimestamp`, illustrating the variable's purpose in a clearer way.

# BorrowerOperations Code Style Findings

## BOS-01C: Ineffectual Native Value Check

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | BorrowerOperations.sol:L582 |

### Description:

The `BorrowerOperations::_requireNonZeroAdjustment` function will evaluate whether the `msg.value` is non-zero, however, such a case is impossible in the codebase as the functions it is invoked in are not `payable`.

### Example:

contracts/BorrowerOperations.sol

SOL

```sol
576 function _requireNonZeroAdjustment(
577     uint256 _collWithdrawal,
578     uint256 _debtTokenChange,
579     uint256 _assetSent
580 ) internal view {
581     require(
582         msg.value != 0 || _collWithdrawal != 0 || _debtTokenChange != 0 || _assetSent
583         "BorrowerOps: There must be either a collateral change or a debt change"
584     );
585 }
```

### Recommendation:

We advise this part of the conditional to be safely omitted, optimizing its gas cost.

### Alleviation:

The `msg.value` evaluation was removed from the function, optimizing its gas cost and permitting it to be set to `pure`.

# BOS-02C: Redundant Native Value Check

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | BorrowerOperations.sol:L291 |

### Description:

The `BorrowerOperations::_adjustVessel` function will mandate that the `msg.value` is zero, however, it is impossible to be otherwise due to the function's invocation in non-payable contexts.

### Example:

contracts/BorrowerOperations.sol

SOL

```
291 require(msg.value == 0, "BorrowerOps: msg.value must be zero");
```

### Recommendation:

We advise the referenced `require` check to be safely omitted from the code, optimizing its gas cost.

### Alleviation:

The redundant `require` check has been safely removed from the codebase as advised.

# BOS-03C: Suboptimal Struct Declaration Styles

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | BorrowerOperations.sol:L124, L292 |

### Description:

The linked declaration styles of the referenced structs are using index-based argument initialization.

### Example:

contracts/BorrowerOperations.sol

SOL

```sol
124 ContractsCache memory contractsCache = ContractsCache(vesselManager, adminContract.act
```

### Recommendation:

We advise the key-value declaration format to be utilized instead in each instance, greatly increasing the legibility of the codebase.

### Alleviation:

The key-value declaration style is now in use in both referenced instances of the exhibit, addressing it in full.

# CollSurplusPool Code Style Findings

## CSP-01C: Inefficient Renunciation of Ownership

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | CollSurplusPool.sol:L48 |

### Description:

The `CollSurplusPool::setAddresses` function will invoke the `OwnableUpgradeable::renounceOwnership` function which in turn will apply the `onlyOwner` modifier redundantly.

### Example:

contracts/CollSurplusPool.sol

SOL

```sol
function setAddresses(
    address _activePoolAddress,
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _vesselManagerOperationsAddress
) external override initializer {
    require(!isInitialized, "Already initialized");
    isInitialized = true;

    __Ownable_init();

    activePoolAddress = _activePoolAddress;
    borrowerOperationsAddress = _borrowerOperationsAddress;
    vesselManagerAddress = _vesselManagerAddress;
    vesselManagerOperationsAddress = _vesselManagerOperationsAddress;

    renounceOwnership();
}
```

### Recommendation:

We advise the `OwnableUpgradeable::_transferOwnership` function to be utilized directly, transferring ownership to the zero address.

### Alleviation:

Ownership of the contract is no longer renounced in the latest iteration of the codebase rendering this exhibit inapplicable.

## CSP-02C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | CollSurplusPool.sol:L70, L71, L78, L80, L84 |

### Description:

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

### Example:

contracts/CollSurplusPool.sol

SOL

```sol
63  function accountSurplus(
64      address _asset,
65      address _account,
66      uint256 _amount
67  ) external override {
68      _requireCallerIsVesselManager();
69
70      uint256 newAmount = userBalances[_account][_asset].add(_amount);
71      userBalances[_account][_asset] = newAmount;
72
73      emit CollBalanceUpdated(_account, newAmount);
74  }
```

### 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.

### Alleviation:

All inefficient `mapping` lookups have been significantly optimized per our recommendation, rendering this exhibit fully alleviated.

# CSP-03C: Inexplicable Ownable Pattern

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | CollSurplusPool.sol:L41, L48 |

## Description:

The `CollSurplusPool` inherits the `OwnableUpgradeable` implementation redundantly as it initializes it within the `CollSurplusPool::setAddresses` function and consequently renounces ownership in the same call.

## Example:

contracts/CollSurplusPool.sol

SOL

```sol
function setAddresses(
    address _activePoolAddress,
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _vesselManagerOperationsAddress
) external override initializer {
    require(!isInitialized, "Already initialized");
    isInitialized = true;

    __Ownable_init();

    activePoolAddress = _activePoolAddress;
    borrowerOperationsAddress = _borrowerOperationsAddress;
    vesselManagerAddress = _vesselManagerAddress;
    vesselManagerOperationsAddress = _vesselManagerOperationsAddress;

    renounceOwnership();
}
```

### Recommendation:

We advise it to be removed, inheriting the `Initializable` implementation of OpenZeppelin instead which is properly put in use within the contract.

### Alleviation:

While the renunciation has been removed, the `OwnableUpgradeable` contract is still inherited by the `CollSurplusPool`. To properly alleviate this exhibit, we advise the `OwnableUpgradeable` contract to be omitted from the `CollSurplusPool` entirely.

# CSP-04C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | CollSurplusPool.sol:L23, L37-L39 |

### Description:

The `CollSurplusPool` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the `CollSurplusPool::setAddresses` function. As such, the manual `isInitialized` flag is redundant.

### Example:

contracts/CollSurplusPool.sol

SOL

```sol
23  bool public isInitialized;
24
25  // deposited ether tracker
26  mapping(address => uint256) balances;
27  // Collateral surplus claimable by vessel owners
28  mapping(address => mapping(address => uint256)) internal userBalances;
29
30  // --- Contract setters ---
31
32  function setAddresses(
33      address _activePoolAddress,
34      address _borrowerOperationsAddress,
35      address _vesselManagerAddress,
36      address _vesselManagerOperationsAddress
37  ) external override initializer {
38      require(!isInitialized, "Already initialized");
39      isInitialized = true;
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# DebtToken Code Style Findings

## DTN-01C: Variable Mutability Specifier (Immutable)

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | DebtToken.sol:L22 |

### Description:

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

### Example:

contracts/DebtToken.sol

SOL

```sol
42  constructor(
43      address _vesselManagerAddress,
44      address _stabilityPoolAddress,
45      address _borrowerOperationsAddress,
46      address _timelockAddress
47  ) ERC20("GRAI", "GRAI") {
48      vesselManagerAddress = _vesselManagerAddress;
49      timelockAddress = _timelockAddress;
50      stabilityPool = IStabilityPool(_stabilityPoolAddress);
51      borrowerOperationsAddress = _borrowerOperationsAddress;
52  }
```

### Recommendation:

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

### Alleviation:

The `timelockAddress` has been set as `immutable`, greatly optimizing its read-access gas cost.

# DefaultPool Code Style Findings

## DPL-01C: Inefficient Renunciation of Ownership

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | DefaultPool.sol:L48 |

### Description:

The `DefaultPool::setAddresses` function will invoke the `OwnableUpgradeable::renounceOwnership` function which in turn will apply the `onlyOwner` modifier redundantly.

### Example:

contracts/DefaultPool.sol

SOL

```sol
function setAddresses(address _vesselManagerAddress, address _activePoolAddress)
    external
    initializer
{
    require(!isInitialized, "Already initialized");
    isInitialized = true;

    __Ownable_init();

    vesselManagerAddress = _vesselManagerAddress;
    activePoolAddress = _activePoolAddress;

    renounceOwnership();
}
```

### Recommendation:

We advise the `OwnableUpgradeable::_transferOwnership` function to be utilized directly, transferring ownership to the zero address.

### Alleviation:

Ownership of the contract is no longer renounced in the latest iteration of the codebase rendering this exhibit inapplicable.

# DPL-02C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | DefaultPool.sol:L73, L79, L88, L89, L97, L98 |

### Description:

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

### Example:

contracts/DefaultPool.sol

SOL

```sol
73  assetsBalances[_asset] = assetsBalances[_asset].sub(_amount);
74
75      IERC20Upgradeable(_asset).safeTransfer(activePool, safetyTransferAmount);
76      IDeposit(activePool).receivedERC20(_asset, _amount);
77
78
79  emit DefaultPoolAssetBalanceUpdated(_asset, assetsBalances[_asset]);
```

### 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.

### Alleviation:

All inefficient `mapping` lookups have been significantly optimized per our recommendation, rendering this exhibit fully alleviated.

# DPL-03C: Inexplicable Ownable Pattern

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | DefaultPool.sol:L43, L48 |

### Description:

The `DefaultPool` inherits the `OwnableUpgradeable` implementation redundantly as it initializes it within the **`DefaultPool::setAddresses`** function and consequently renounces ownership in the same call.

### Example:

contracts/DefaultPool.sol

SOL

```sol
36  function setAddresses(address _vesselManagerAddress, address _activePoolAddress)
37      external
38      initializer
39  {
40      require(!isInitialized, "Already initialized");
41      isInitialized = true;
42
43      __Ownable_init();
44
45      vesselManagerAddress = _vesselManagerAddress;
46      activePoolAddress = _activePoolAddress;
47
48      renounceOwnership();
49  }
```

### Recommendation:

We advise it to be removed, inheriting the `Initializable` implementation of OpenZeppelin instead which is properly put in use within the contract.

### Alleviation:

While the renunciation has been removed, the `OwnableUpgradeable` contract is still inherited by the `DefaultPool`. To properly alleviate this exhibit, we advise the `OwnableUpgradeable` contract to be omitted from the `DefaultPool` entirely.

# DPL-04C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | DefaultPool.sol:L29, L38, L40-L41 |

### Description:

The `DefaultPool` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the **`DefaultPool::setAddresses`** function. As such, the manual `isInitialized` flag is redundant.

### Example:

contracts/DefaultPool.sol

SOL

```sol
29  bool public isInitialized;
30
31  mapping(address => uint256) internal assetsBalances;
32  mapping(address => uint256) internal debtTokenBalances;
33
34  // --- Dependency setters ---
35
36  function setAddresses(address _vesselManagerAddress, address _activePoolAddress)
37      external
38      initializer
39  {
40      require(!isInitialized, "Already initialized");
41      isInitialized = true;
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# ERC20Decimals Code Style Findings

## ERD-01C: Non-Standard Interface Name

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | ERC20Decimals.sol:L5 |

### Description:

The referenced `interface` does not conform to the `IXXX` naming convention.

### Example:

contracts/Dependencies/ERC20Decimals.sol

SOL

```
5   interface ERC20Decimals {
```

### Recommendation:

We advise the `ERC20Decimals` interface and file to be aptly renamed to `IERC20Decimals`, properly illustrating its purpose.

### Alleviation:

The interface and file have both been aptly renamed with an `I` prefixed, signalling that they are meant to represent an `interface` rather than a `contract` implementation.

# ERC20Permit Code Style Findings

## ERC-01C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | ERC20Permit.sol:L96, L104 |

### Description:

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

### Example:

contracts/Dependencies/ERC20Permit.sol

SOL

```
95  bytes32 hashStruct = keccak256(
96      abi.encode(PERMIT_TYPEHASH, owner, spender, amount, _nonces[owner].current(), dead
97  );
98
99  bytes32 _hash = keccak256(abi.encodePacked(uint16(0x1901), DOMAIN_SEPARATOR, hashStruc
100
101 address signer = ecrecover(_hash, v, r, s);
102 require(signer != address(0) && signer == owner, "ERC20Permit: Invalid signature");
103
104 _nonces[owner].increment();
```

### 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.

### Alleviation:

All inefficient `mapping` lookups have been significantly optimized per our recommendation, rendering this exhibit fully alleviated.

# ERC-02C: Multiple Top-Level Declarations

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | ERC20Permit.sol:L7, L50 |

### Description:

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

### Example:

contracts/Dependencies/ERC20Permit.sol

SOL

```sol
interface IERC2612Permit {
    /**
     * @dev Sets `amount` as the allowance of `spender` over `owner`'s tokens,
     * given `owner`'s signed approval.
     *
     * IMPORTANT: The same issues {IERC20-approve} has related to transaction
     * ordering also apply here.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     * - `deadline` must be a timestamp in the future.
     * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
     * over the EIP712-formatted function arguments.
     * - the signature must use ``owner``'s current nonce (see {nonces}).
     *
     * For more information on the signature format, see the
     * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
     * section].
     */
    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    /**
     * @dev Returns the current ERC2612 nonce for `owner`. This value must be
     * included whenever a signature is generated for {permit}.
     *
     * Every successful call to {permit} increases ``owner``'s nonce by one. This
     * prevents a signature from being used multiple times.
     */
    function nonces(address owner) external view returns (uint256);
}

abstract contract ERC20Permit is ERC20, IERC2612Permit {
```

### 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:

The `IERC2612Permit` interface declaration has been relocated to its dedicated file and is now imported by the codebase, optimizing the project's structure.

# ERC-03C: Redundant Low-Level Assembly Blocks

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | ERC20Permit.sol:L62-L65, L115-L119 |

### Description:

The referenced `assembly` block within the contract's `constructor` yields the `chainid` of the execution context, however, the same value can be extracted without an `assembly` block by accessing `block.chainid`. Additionally, the **`ERC20Permit::chainId`** function is redundant as the value can be acquired via the same syntax in other contexts.

### Example:

contracts/Dependencies/ERC20Permit.sol

SOL

```sol
62  uint256 chainID;
63  assembly {
64      chainID := chainid()
65  }
```

### Recommendation:

We advise the `block.chainid` syntax to be utilized, standardizing the codebase's style and rendering the **`ERC20Permit::chainId`** function redundant.

### Alleviation:

The `block.chainid` variable is now utilized in the **`ERC20Permit::constructor`** as advised.

# ERC-04C: Variable Mutability Specifier (Immutable)

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | ERC20Permit.sol:L59, L67 |

### Description:

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

### Example:

contracts/Dependencies/ERC20Permit.sol

SOL

```sol
59  bytes32 public DOMAIN_SEPARATOR;
60
61  constructor() {
62      uint256 chainID;
63      assembly {
64          chainID := chainid()
65      }
66
67      DOMAIN_SEPARATOR = keccak256(
68          abi.encode(
69              keccak256(
70                  "EIP712Domain(string name,string version,uint256 chainId,address verif
71              ),
72              keccak256(bytes(name())),
73              keccak256(bytes("1")), // Version
74              chainID,
75              address(this)
76          )
77      );
78  }
```

### Recommendation:

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

### Alleviation:

The `DOMAIN_SEPARATOR` variable has been set as `immutable`, greatly optimizing its read-access gas cost.

# FeeCollector Code Style Findings

## FCR-01C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | FeeCollector.sol:L182, L198, L207-L208 |

### Description:

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

### Example:

contracts/FeeCollector.sol

SOL

```sol
function _decreaseDebt(
    address _borrower,
    address _asset,
    uint256 _paybackFraction
) internal {
    uint256 NOW = block.timestamp;
    require(_paybackFraction <= 1 ether, "Payback fraction cannot be higher than 1 (@
    require(_paybackFraction > 0, "Payback fraction cannot be zero");
    FeeRecord memory mRecord = feeRecords[_borrower][_asset];
    if (mRecord.amount == 0) {
        // console.log("      decreaseDebt() :: no records found");
        return;
    }
    if (mRecord.to < NOW) {
        // console.log("      decreaseDebt() :: record is expired");
        _closeExpiredOrLiquidatedFeeRecord(_borrower, _asset, mRecord.amount);
    } else {
        // collect expired refund
        uint256 expiredAmount = _calcExpiredAmount(mRecord.from, mRecord.to, mRecord.a
        _collectFee(_borrower, _asset, expiredAmount);
        if (_paybackFraction == 1e18) {
            // full payback
            uint256 refundAmount = mRecord.amount - expiredAmount;
            _refundFee(_borrower, _asset, refundAmount);
            delete feeRecords[_borrower][_asset];
            emit FeeRecordUpdated(_borrower, _asset, NOW, 0, 0);
            // console.log("^^^ EVENT FeeRecordUpdated(%s, 0, 0)", NOW);
        } else {
            // refund amount proportional to the payment
            uint256 refundAmount = ((mRecord.amount - expiredAmount) * _paybackFractio
            // console.log("      decreaseDebt() :: %s = refund", f(refundAmount));
            _refundFee(_borrower, _asset, refundAmount);
            uint256 updatedAmount = mRecord.amount - expiredAmount - refundAmount;
            feeRecords[_borrower][_asset].amount = updatedAmount;
            feeRecords[_borrower][_asset].from = NOW;
            // console.log("      decreaseDebt() :: %s left", f(updatedAmount));
            emit FeeRecordUpdated(_borrower, _asset, NOW, mRecord.to, updatedAmount);
            // console.log("^^^ EVENT FeeRecordUpdated(%s, %s, %s)", NOW, mRecord.to,
        }
    }
}
```

### 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.

### Alleviation:

All inefficient `mapping` lookups have been significantly optimized per our recommendation, rendering this exhibit fully alleviated.

# FCR-02C: Inexistent Error Messages

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | FeeCollector.sol:L50, L51 |

### Description:

The linked `require` checks have no error messages explicitly defined.

### Example:

contracts/FeeCollector.sol

SOL

```sol
50  require(!isInitialized);
```

### Recommendation:

We advise each to be set so to increase the legibility of the codebase and aid in validating the `require` checks' conditions.

### Alleviation:

While the first `require` check is no longer present in the codebase, the second `require` check has not had an explicit error message introduced thereby rendering this exhibit unaddressed. Given that it pertains a style-related exhibit, we will consider this exhibit acknowledged.

# FCR-03C: Leftover Test Code

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | FeeCollector.sol:L364-L388 |

### Description:

The `FeeCollector::f` function is meant to be removed from the codebase as per its `TODO` comment.

### Example:

contracts/FeeCollector.sol

SOL

```sol
/**
 * TEMPORARY formatting method to help with debugging
 * TODO remove for production deployment
 */
function f(uint256 value) internal pure returns (string memory) {
    string memory sInput = Strings.toString(value);
    bytes memory bInput = bytes(sInput);
    uint256 len = bInput.length > 18 ? bInput.length + 1 : 20;
    string memory sResult = new string(len);
    bytes memory bResult = bytes(sResult);
    if (bInput.length <= 18) {
        bResult[0] = "0";
        bResult[1] = ".";
        for (uint256 i = 1; i <= 18 - bInput.length; i++) bResult[i + 1] = "0";
        for (uint256 i = bInput.length; i > 0; i--) bResult[--len] = bInput[i - 1];
    } else {
        uint256 c = 0;
        uint256 i = bInput.length;
        while (i > 0) {
            bResult[--len] = bInput[--i];
            if (++c == 18) bResult[--len] = ".";
        }
    }
    return string(bResult);
}
```

### Recommendation:

We advise this to be done so, bringing the code closer to a production deployment.

### Alleviation:

The leftover test code has been safely removed from the codebase as advised.

# FCR-04C: Loop Iterator Optimization

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | FeeCollector.sol:L142 |

### Description:

The linked `for` loop increments / decrements the iterator "safely" due to Solidity's built-in safe arithmetics (post-`0.8.X`).

### Example:

contracts/FeeCollector.sol

SOL

```sol
142 for (uint256 i = 0; i < borrowersLength; ++i) {
```

### Recommendation:

We advise the increment / decrement operation to be performed in an `unchecked` code block as the last statement within the `for` loop to optimize its execution cost.

### Alleviation:

The loop iterator increment has been optimized as advised, however, its `i++` counterpart is utilized instead of `++i`. We advise the latter to be set in use as it is more optimal than the present code.

# FCR-05C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | FeeCollector.sol:L49, L50, L62 |

### Description:

The `FeeCollector` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the **`FeeCollector::setAddresses`** function. As such, the manual `isInitialized` flag is redundant.

### Example:

contracts/FeeCollector.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _grvtStakingAddress,
    address _debtTokenAddress,
    address _treasuryAddress,
    bool _routeToGRVTStaking
) external initializer {
    require(!isInitialized);
    require(_treasuryAddress != address(0));
    borrowerOperationsAddress = _borrowerOperationsAddress;
    vesselManagerAddress = _vesselManagerAddress;
    grvtStaking = IGRVTStaking(_grvtStakingAddress);
    debtTokenAddress = _debtTokenAddress;
    treasuryAddress = _treasuryAddress;
    routeToGRVTStaking = _routeToGRVTStaking;
    if (_routeToGRVTStaking && address(grvtStaking) == address(0)) {
        revert FeeCollector__InvalidGRVTStakingAddress();
    }
    __Ownable_init();
    isInitialized = true;
}
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# GravitaBase Code Style Findings

## GBE-01C: Unused Function Declaration

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | GravitaBase.sol:L89-L91 |

### Description:

The `GravitaBase::_revertWrongFuncCaller` function remains unutilized in the codebase.

### Example:

contracts/Dependencies/GravitaBase.sol

SOL

```sol
89  function _revertWrongFuncCaller() internal pure {
90      revert("WFC");
91  }
```

### Recommendation:

We advise it to be safely omitted, reducing the bytecode size of the contract.

### Alleviation:

The unutilized `GravitaBase::_revertWrongFuncCaller` function has been safely removed from the codebase as advised.

# GravitaMath Code Style Findings

## GMH-01C: Illegible Representation of Value Literal

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | GravitaMath.sol:L101 |

### Description:

The `GravitaMath::_computeNominalCR` function will yield a value of `2 ** 256 - 1`, representing the maximum value of a `uint256` variable, when an "infinite" collateral ratio is meant to be yielded by it.

### Example:

contracts/Dependencies/GravitaMath.sol

SOL

```sol
function _computeNominalCR(uint256 _coll, uint256 _debt) internal pure returns (uint25
    if (_debt > 0) {
        return _coll.mul(NICR_PRECISION).div(_debt);
    }
    // Return the maximal value for uint256 if the Vessel has a debt of 0. Represents
    else {
        // if (_debt == 0)
        return 2 ** 256 - 1;
    }
}

function _computeCR(
    uint256 _coll,
    uint256 _debt,
    uint256 _price
) internal pure returns (uint256) {
    if (_debt > 0) {
        uint256 newCollRatio = _coll.mul(_price).div(_debt);

        return newCollRatio;
    }
    // Return the maximal value for uint256 if the Vessel has a debt of 0. Represents
    else {
        // if (_debt == 0)
        return type(uint256).max;
    }
}
```

### Recommendation:

We advise the same syntax as `GravitaMath::_computeCR` to be used, yielding `type(uint256).max` and increasing the legibility of the codebase.

### Alleviation:

The representation of the value literal has been standardized in the code utilizing `type(uint256).max` as advised.

# GMH-02C: Repetitive Value Literal

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | GravitaMath.sol:L62, L63 |

### Description:

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

### Example:

contracts/Dependencies/GravitaMath.sol

SOL

```sol
62  if (_minutes > 525600000) {
```

### Recommendation:

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

### Alleviation:

The referenced repetitive value literal has been relocated to a `constant` variable declaration labelled `EXPONENT_CAP`, optimizing the legibility of the codebase.

# PoolBase Code Style Findings

## PBE-01C: Significantly Inefficient Merging of Pending Gains / Distributed Funds

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | PoolBase.sol:L43-L68 |

### Description:

The `PoolBase::_leftSumColls` function is meant to merge whatever pending gains are denoted in `_tokens` and `_amounts` to the `_coll1` data entry, however, it does so significantly inefficiently. The same inefficiency is observed in the `PoolBase::_leftSubColls` function.

### Example:

contracts/Dependencies/PoolBase.sol

SOL

```sol
function _leftSumColls(
    Colls memory _coll1,
    address[] memory _tokens,
    uint256[] memory _amounts
) internal pure returns (uint256[] memory) {
    // If nothing on the right side then return the original.
    if (_amounts.length == 0) {
        return _coll1.amounts;
    }

    uint256 coll1Len = _coll1.amounts.length;
    uint256 tokensLen = _tokens.length;
    // Result will always be coll1 len size.
    uint256[] memory sumAmounts = new uint256[](coll1Len);

    uint256 i = 0;
    uint256 j = 0;

    // Sum through all tokens until either left or right side reaches end.
    while (i < tokensLen && j < coll1Len) {
        // If tokens match up then sum them together.
        if (_tokens[i] == _coll1.tokens[j]) {
            sumAmounts[j] = _coll1.amounts[j].add(_amounts[i]);
            ++i;
        }
        // Otherwise just take the left side.
        else {
            sumAmounts[j] = _coll1.amounts[j];
        }
        ++j;
    }
    // If right side ran out add the remaining amounts in the left side.
    while (j < coll1Len) {
        sumAmounts[j] = _coll1.amounts[j];
        ++j;
    }

    return sumAmounts;
}
```

### Recommendation:

We advise the code to instead sum / subtract the values in the `_coll1.amounts` data entry **directly**, rendering the new `sumAmounts` / `diffAmounts` array redundant. Additionally, this will significantly optimize the code as only the `_tokens` array would need to be iterated as the `_coll1.amounts` data entry will be "pre-filled" with the desirable amounts.

### Alleviation:

The code, now located within `StabilityPool`, has been refactored per our recommendation albeit in a different approach that is still relatively inefficient. We advise the `_tokens` array to be iterated rather than the `_coll1` array, iterating the `_coll1` array **inside the `_tokens` loop** and issuing a `break` statement when the correct `_coll1` entry has been found to further optimize the code.

# PBE-02C: Unused Error Declaration

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | PoolBase.sol:L22 |

### Description:

The `PoolBase__AdminOnly` error remains unused in the codebase.

### Example:

contracts/Dependencies/PoolBase.sol

SOL

```sol
22  error PoolBase__AdminOnly();
```

### Recommendation:

We advise it to be safely omitted from it.

### Alleviation:

The `PoolBase` contract is no longer present in the codebase rendering this exhibit no longer applicable.

# PriceFeed Code Style Findings

## PFD-01C: Inexistent Error Message

| Type | Severity | Location |
| --- | --- | --- |
| **Code Style** | Informational | PriceFeed.sol:L60 |

### Description:

The linked `require` check has no error message explicitly defined.

### Example:

contracts/PriceFeed.sol

SOL

```sol
60  require(!isInitialized);
```

### Recommendation:

We advise one to be set so to increase the legibility of the codebase and aid in validating the `require` check's condition.

### Alleviation:

The `require` check is no longer present in the codebase rendering this exhibit no longer applicable.

# PFD-02C: Redundant External Self-Calls

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | PriceFeed.sol:L149, L171 |

### Description:

The referenced statements perform external calls to self via the `this.fetchPrice` syntax redundantly.

### Example:

contracts/PriceFeed.sol

SOL

```sol
148 function _calcEthPrice(uint256 ethAmount) internal returns (uint256) {
149     uint256 ethPrice = this.fetchPrice(address(0));
150     return ethPrice.mul(ethAmount).div(1 ether);
151 }
```

### Recommendation:

We advise the `PriceFeed::fetchPrice` function to be set as `public` and the calls to be made "internally" by removing the `this` call prefix.

### Alleviation:

The second referenced instance is no longer present in the codebase whereas the first instance has been properly corrected to perform an "internal" call rather than an "external" self-call.

# PFD-03C: Redundant Function Implementation

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | PriceFeed.sol:L183-L185 |

### Description:

The referenced function yields a contract-level `constant` variable.

### Example:

contracts/PriceFeed.sol

SOL

```sol
183 function _getOracleUpdateTimelock() internal view virtual returns (uint256) {
184     return ORACLE_UPDATE_TIMELOCK;
185 }
```

### Recommendation:

We advise it to be omitted and invocations of it to be replaced by the `constant` itself.

### Alleviation:

The redundant function has been safely removed from the codebase as advised.

# PFD-04C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | PriceFeed.sol:L33, L59-L61 |

### Description:

The `PriceFeed` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the **`PriceFeed::setAddresses`** function. As such, the manual `isInitialized` flag is redundant.

### Example:

contracts/PriceFeed.sol

SOL

```sol
function setAddresses(
    address _adminContract,
    address _rethToken,
    address _stethToken,
    address _wstethToken
) external initializer {
    require(!isInitialized);
    isInitialized = true;
    __Ownable_init();
    adminContract = _adminContract;
    rethToken = _rethToken;
    stethToken = _stethToken;
    wstethToken = _wstethToken;
}
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# PFD-05C: Suboptimal Struct Declaration Styles

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | PriceFeed.sol:L80, L82 |

### Description:

The linked declaration styles of the referenced structs are using index-based argument initialization.

### Example:

contracts/PriceFeed.sol

SOL

```sol
80  queuedOracles[_token] = OracleRecord(newOracle, timelockRelease, true, true, _isEthInd
```

### Recommendation:

We advise the key-value declaration format to be utilized instead in each instance, greatly increasing the legibility of the codebase.

### Alleviation:

The key-value declaration style is now in use in the code that both instances have been merged to, alleviating this exhibit.

# ReentrancyGuardUpgradeable Code Style Findings

## RGU-01C: Inefficient Reentrancy Guard Implementation

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | ReentrancyGuardUpgradeable.sol:L40-L44, L48-L50 |

### Description:

The `ReentrancyGuardUpgradeable` implementation present in the Gravita Protocol codebase represents an outdated OpenZeppelin version modified to not use the `Initializable` dependency, however, it is outdated and thus inefficient.

### Example:

contracts/Dependencies/ReentrancyGuardUpgradeable.sol

SOL

```sol
/**
 * @dev Prevents a contract from calling itself, directly or indirectly.
 * Calling a `nonReentrant` function from another `nonReentrant`
 * function is not supported. It is possible to prevent this from happening
 * by making the `nonReentrant` function external, and making it call a
 * `private` function that does the actual work.
 */
modifier nonReentrant() {
    // On the first call to nonReentrant, _notEntered will be true
    require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

    // Any calls to nonReentrant after this point will fail
    _status = _ENTERED;

    _;

    // By storing the original value once again, a refund is triggered (see
    // https://eips.ethereum.org/EIPS/eip-2200)
    _status = _NOT_ENTERED;
}
```

### Recommendation:

We advise the internal function paradigm that **the latest version of** `ReentrancyGuardUpgradeable` applies in OpenZeppelin to be replicated here, significantly optimizing the gas cost of the `ReentrancyGuardUpgradeable::nonReentrant` modifier.

### Alleviation:

The `ReentrancyGuardUpgradeable` contract has been removed from the codebase in favour of using the actual `ReentrancyGuardUpgradeable` dependency of OpenZeppelin as a result of this exhibit. As such, we consider this exhibit addressed.

# SortedVessels Code Style Findings

## SVS-01C: Inefficient Renunciation of Ownership

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | SortedVessels.sol:L48 |

### Description:

The `SortedVessels::setParams` function will invoke the `OwnableUpgradeable::renounceOwnership` function which in turn will apply the `onlyOwner` modifier redundantly.

### Example:

contracts/SortedVessels.sol

SOL

```sol
77  function setParams(address _vesselManagerAddress, address _borrowerOperationsAddress)
78      external
79      override
80      initializer
81  {
82      require(!isInitialized, "Already initialized");
83      isInitialized = true;
84
85      __Ownable_init();
86
87      vesselManager = IVesselManager(_vesselManagerAddress);
88      borrowerOperationsAddress = _borrowerOperationsAddress;
89
90      renounceOwnership();
91  }
```

### Recommendation:

We advise the `OwnableUpgradeable::_transferOwnership` function to be utilized directly, transferring ownership to the zero address.

### Alleviation:

Ownership of the contract is no longer renounced in the latest iteration of the codebase rendering this exhibit inapplicable.

# SVS-02C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | SortedVessels.sol:L143, L147-L148, L151-L153, L156-L158, L161-L164, L167, L184, L186, L189, L191, L195, L197, L201-L203, L205-L207, L212-L213, L216-L217, L370, L376, L383-L384, L404, L410, L417-L418 |

### Description:

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

### Example:

contracts/SortedVessels.sol

SOL

```sol
data[_asset].nodes[_id].exists = true;

if (prevId == address(0) && nextId == address(0)) {
    // Insert as head and tail
    data[_asset].head = _id;
    data[_asset].tail = _id;
} else if (prevId == address(0)) {
    // Insert before `prevId` as the head
    data[_asset].nodes[_id].nextId = data[_asset].head;
    data[_asset].nodes[data[_asset].head].prevId = _id;
    data[_asset].head = _id;
} else if (nextId == address(0)) {
    // Insert after `nextId` as the tail
    data[_asset].nodes[_id].prevId = data[_asset].tail;
    data[_asset].nodes[data[_asset].tail].nextId = _id;
    data[_asset].tail = _id;
} else {
    // Insert at insert position between `prevId` and `nextId`
    data[_asset].nodes[_id].nextId = nextId;
    data[_asset].nodes[_id].prevId = prevId;
    data[_asset].nodes[prevId].nextId = _id;
    data[_asset].nodes[nextId].prevId = _id;
}

data[_asset].size = data[_asset].size.add(1);
```

### 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.

### Alleviation:

All inefficient `mapping` lookups have been significantly optimized per our recommendation, rendering this exhibit fully alleviated.

# SVS-03C: Inexplicable Ownable Pattern

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | SortedVessels.sol:L41, L48 |

## Description:

The `SortedVessels` inherits the `OwnableUpgradeable` implementation redundantly as it initializes it within the `SortedVessels::setParams` function and consequently renounces ownership in the same call.

## Example:

contracts/SortedVessels.sol

SOL

```sol
77  function setParams(address _vesselManagerAddress, address _borrowerOperationsAddress)
78      external
79      override
80      initializer
81  {
82      require(!isInitialized, "Already initialized");
83      isInitialized = true;
84
85      __Ownable_init();
86
87      vesselManager = IVesselManager(_vesselManagerAddress);
88      borrowerOperationsAddress = _borrowerOperationsAddress;
89
90      renounceOwnership();
91  }
```

### Recommendation:

We advise it to be removed, inheriting the `Initializable` implementation of OpenZeppelin instead which is properly put in use within the contract.

### Alleviation:

While the renunciation has been removed, the `OwnableUpgradeable` contract is still inherited by the `SortedVessels`. To properly alleviate this exhibit, we advise the `OwnableUpgradeable` contract to be omitted from the `SortedVessels` entirely.

## SVS-04C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | SortedVessels.sol:L80, L82-L83 |

### Description:

The `SortedVessels` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the **`SortedVessels::setParams`** function. As such, the manual `isInitialized` flag is redundant.

### Example:

contracts/SortedVessels.sol

SOL

```sol
77  function setParams(address _vesselManagerAddress, address _borrowerOperationsAddress)
78      external
79      override
80      initializer
81  {
82      require(!isInitialized, "Already initialized");
83      isInitialized = true;
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# StabilityPool Code Style Findings

## SPL-01C: Inefficient Renunciation of Ownership

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | StabilityPool.sol:L267 |

### Description:

The `StabilityPool::setAddresses` function will invoke the `OwnableUpgradeable::renounceOwnership` function which in turn will apply the `onlyOwner` modifier redundantly.

### Example:

contracts/StabilityPool.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _activePoolAddress,
    address _debtTokenAddress,
    address _sortedVesselsAddress,
    address _communityIssuanceAddress,
    address _adminContractAddress
) external initializer override {
    require(!isInitialized, "StabilityPool: Already initialized");

    isInitialized = true;
    __Ownable_init();
    __ReentrancyGuard_init();

    borrowerOperations = IBorrowerOperations(_borrowerOperationsAddress);
    vesselManager = IVesselManager(_vesselManagerAddress);
    activePool = IActivePool(_activePoolAddress);
    debtToken = IDebtToken(_debtTokenAddress);
    sortedVessels = ISortedVessels(_sortedVesselsAddress);
    communityIssuance = ICommunityIssuance(_communityIssuanceAddress);
    adminContract = IAdminContract(_adminContractAddress);

    P = DECIMAL_PRECISION;

    renounceOwnership();
}
```

### Recommendation:

We advise the `OwnableUpgradeable::_transferOwnership` function to be utilized directly, transferring ownership to the zero address.

### Alleviation:

Ownership of the contract is no longer renounced in the latest iteration of the codebase rendering this exhibit inapplicable.

# SPL-02C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | StabilityPool.sol:L404, L405, L661, L662, L836, L838-L841, L852, L856-L859 |

### Description:

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

### Example:

contracts/StabilityPool.sol

SOL

```sol
392 function _updateG(uint256 _GRVTIssuance) internal {
393     uint256 cachedTotalDebtTokenDeposits = totalDebtTokenDeposits; // cached to save a
394     /*
395      * When total deposits is 0, G is not updated. In this case, the GRVT issued can n
396      * depositors - it is missed out on, and remains in the balanceof the CommunityIss
397      *
398      */
399     if (cachedTotalDebtTokenDeposits == 0 || _GRVTIssuance == 0) {
400         return;
401     }
402     uint256 GRVTPerUnitStaked = _computeGRVTPerUnitStaked(_GRVTIssuance, cachedTotalDe
403     uint256 marginalGRVTGain = GRVTPerUnitStaked.mul(P);
404     epochToScaleToG[currentEpoch][currentScale] = epochToScaleToG[currentEpoch][curren
405     emit G_Updated(epochToScaleToG[currentEpoch][currentScale], currentEpoch, currentS
406 }
```

### 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.

### Alleviation:

All but the first instance pair have been optimized per our recommendation, rendering this exhibit partially alleviated.

# SPL-03C: Inexplicable Contract Member

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | StabilityPool.sol:L174, L615-L616, L806-L807 |

### Description:

The `pendingCollGains` member of the `StabilityPool` is utilized in multiple statements within the code, however, it results in a no-op as it remains filled with zero-values throughout its lifetime.

### Example:

contracts/StabilityPool.sol

SOL

```sol
805 // Reset pendingCollGains since those were all sent to the borrower
806 Colls memory tempPendingCollGains;
807 pendingCollGains[_to] = tempPendingCollGains;
```

### Recommendation:

We advise it to be re-evaluated and potentially omitted, significantly improving the gas costs of the functions it was utilized in.

### Alleviation:

The `pendingCollGains` contract member has been safely removed as advised.

# SPL-04C: Inexplicable Ownable Pattern

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | StabilityPool.sol:L254, L267 |

### Description:

The `StabilityPool` inherits the `OwnableUpgradeable` implementation redundantly as it initializes it within the `StabilityPool::setAddresses` function and consequently renounces ownership in the same call.

### Example:

contracts/StabilityPool.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _activePoolAddress,
    address _debtTokenAddress,
    address _sortedVesselsAddress,
    address _communityIssuanceAddress,
    address _adminContractAddress
) external initializer override {
    require(!isInitialized, "StabilityPool: Already initialized");

    isInitialized = true;
    __Ownable_init();
    __ReentrancyGuard_init();

    borrowerOperations = IBorrowerOperations(_borrowerOperationsAddress);
    vesselManager = IVesselManager(_vesselManagerAddress);
    activePool = IActivePool(_activePoolAddress);
    debtToken = IDebtToken(_debtTokenAddress);
    sortedVessels = ISortedVessels(_sortedVesselsAddress);
    communityIssuance = ICommunityIssuance(_communityIssuanceAddress);
    adminContract = IAdminContract(_adminContractAddress);

    P = DECIMAL_PRECISION;

    renounceOwnership();
}
```

### Recommendation:

We advise it to be removed, inheriting the `Initializable` implementation of OpenZeppelin instead which is properly put in use within the contract.

### Alleviation:

The contract no longer utilizes or inherits the `OwnableUpgradeable` implementation, addressing this exhibit in full.

# SPL-05C: Loop Iterator Optimizations

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | StabilityPool.sol:L635, L794, L835, L849, L896, L924, L934 |

### Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

### Example:

contracts/StabilityPool.sol

SOL

```sol
635 for (uint256 i = 0; i < assetsLen; ++i) {
```

### Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

### Alleviation:

The loop iterator increments have been optimized as advised where applicable, however, their `i++` counterpart is utilized instead of `++i`. We advise the latter to be set in use as it is more optimal than the present code.

# SPL-06C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | StabilityPool.sol:L250-L251, L253 |

### Description:

The `StabilityPool` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the **`StabilityPool::setAddresses`** function. As such, the manual `isInitialized` flag is redundant.

### Example:

contracts/StabilityPool.sol

SOL

```sol
function setAddresses(
    address _borrowerOperationsAddress,
    address _vesselManagerAddress,
    address _activePoolAddress,
    address _debtTokenAddress,
    address _sortedVesselsAddress,
    address _communityIssuanceAddress,
    address _adminContractAddress
) external initializer override {
    require(!isInitialized, "StabilityPool: Already initialized");

    isInitialized = true;
    __Ownable_init();
    __ReentrancyGuard_init();

    borrowerOperations = IBorrowerOperations(_borrowerOperationsAddress);
    vesselManager = IVesselManager(_vesselManagerAddress);
    activePool = IActivePool(_activePoolAddress);
    debtToken = IDebtToken(_debtTokenAddress);
    sortedVessels = ISortedVessels(_sortedVesselsAddress);
    communityIssuance = ICommunityIssuance(_communityIssuanceAddress);
    adminContract = IAdminContract(_adminContractAddress);

    P = DECIMAL_PRECISION;

    renounceOwnership();
}
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# SPL-07C: Suboptimal Struct Declaration Style

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | StabilityPool.sol:L614 |

### Description:

The linked declaration style of a struct is using index-based argument initialization.

### Example:

contracts/StabilityPool.sol

SOL

```sol
614 Colls(collateralsFromNewGains, amountsFromNewGains),
```

### Recommendation:

We advise the key-value declaration format to be utilized instead, greatly increasing the legibility of the codebase.

### Alleviation:

The referenced declaration of a `struct` is no longer present in the codebase, rendering this exhibit no longer applicable.

# Timelock Code Style Findings

## TKC-01C: Inefficient Application of Access Control

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | Timelock.sol:L113-L115 |

### Description:

The referenced statements replicate the behaviour of the `Timelock::adminOnly` modifier.

### Example:

contracts/Timelock.sol

SOL

```sol
106 function queueTransaction(
107     address target,
108     uint value,
109     string memory signature,
110     bytes memory data,
111     uint eta
112 ) public returns (bytes32) {
113     if (msg.sender != admin) {
114         revert Timelock__AdminOnly();
115     }
```

### Recommendation:

We advise the `modifier` to be utilized by the `Timelock::queueTransaction` function and the manual access control statements to be omitted.

### Alleviation:

The `Timelock::adminOnly` modifier is utilized in place of the manual check in the `Timelock::queueTransaction` as advised.

# TKC-02C: Redundant Function Implementation

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | Timelock.sol:L179-L181 |

### Description:

The `Timelock::getBlockTimestamp` function implementation is redundant as it yields a statement literal ( `block.timestamp`).

### Example:

contracts/Timelock.sol

SOL

```sol
179 function getBlockTimestamp() internal view returns (uint) {
180     return block.timestamp;
181 }
```

### Recommendation:

We advise all its invocations to be replaced by the `block.timestamp` statement directly, optimizing their gas cost.

### Alleviation:

The redundant **`Timelock::getBlockTimestamp`** function has been safely omitted from the codebase as advised.

# VesselManager Code Style Findings

## VMR-01C: Inefficient `mapping` Lookups

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | VesselManager.sol:L227-L228, L281-L283, L417-L418, L420, L422, L426-L427, L429, L504-L505, L522-L523, L533-L535, L541-L542, L547-L549, L554-L556, L558-L559, L594-L596, L598-L599, L610, L613, L619, L621-L622, L625, L692-L693, L701-L702, L711-L712, L721-L722, L731, L737 |

### Description:

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

### Example:

contracts/VesselManager.sol

SOL

```
500 function _getCurrentVesselAmounts(address _asset, address _borrower) internal view ret
501     uint256 pendingCollReward = getPendingAssetReward(_asset, _borrower);
502     uint256 pendingDebtReward = getPendingDebtTokenReward(_asset, _borrower);
503
504     uint256 currentAsset = Vessels[_borrower][_asset].coll.add(pendingCollReward);
505     uint256 currentDebt = Vessels[_borrower][_asset].debt.add(pendingDebtReward);
506
507     return (currentAsset, currentDebt);
508 }
```

### 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.

### Alleviation:

All inefficient `mapping` lookups have been significantly optimized per our recommendation, rendering this exhibit fully alleviated.

# VMR-02C: Redundant Data Point

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | VesselManager.sol:L692 |

### Description:

The `address` type `asset` data point present in each `Vessel` struct is redundant as the `Vessel` requires the `_asset` key to be accessed.

### Example:

contracts/VesselManager.sol

SOL

```sol
687 function setVesselStatus(
688     address _asset,
689     address _borrower,
690     uint256 _num
691 ) external override onlyBorrowerOperations {
692     Vessels[_borrower][_asset].asset = _asset;
693     Vessels[_borrower][_asset].status = Status(_num);
694 }
```

### Recommendation:

We advise the data point to be safely omitted as it is not utilized within the contract.

### Alleviation:

The `asset` data point has been safely removed from the data entry.

# VMR-03C: Redundant External Self-Call

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | VesselManager.sol:L238 |

### Description:

The referenced statement performs an external call to self via the `this.getVesselStatus` syntax redundantly.

### Example:

contracts/VesselManager.sol

SOL

```sol
237 function isVesselActive(address _asset, address _borrower) public view override return
238     return this.getVesselStatus(_asset, _borrower) == uint256(Status.active);
239 }
```

### Recommendation:

We advise the `VesselManager::getVesselStatus` function to be set as `public` and the call to be made "internally" by removing the `this` call prefix.

### Alleviation:

The redundant self-call has been replaced by an "internal" call of its `public` function as advised.

# VMR-04C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | VesselManager.sol:L134-L136 |

## Description:

The `VesselManager` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the **`VesselManager::setAddresses`** function. As such, the manual `isInitialized` flag is redundant.

## Example:

contracts/VesselManager.sol

SOL

```sol
124 function setAddresses(
125     address _borrowerOperationsAddress,
126     address _stabilityPoolAddress,
127     address _gasPoolAddress,
128     address _collSurplusPoolAddress,
129     address _debtTokenAddress,
130     address _feeCollectorAddress,
131     address _sortedVesselsAddress,
132     address _vesselManagerOperationsAddress,
133     address _adminContractAddress
134 ) external override initializer {
135     require(!isInitialized, "Already initialized");
136     isInitialized = true;
137     __Ownable_init();
138     borrowerOperations = _borrowerOperationsAddress;
139     stabilityPool = IStabilityPool(_stabilityPoolAddress);
140     gasPoolAddress = _gasPoolAddress;
141     collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
142     debtToken = IDebtToken(_debtTokenAddress);
143     feeCollector = IFeeCollector(_feeCollectorAddress);
144     sortedVessels = ISortedVessels(_sortedVesselsAddress);
145     vesselManagerOperations = IVesselManagerOperations(_vesselManagerOperationsAddress
146     adminContract = IAdminContract(_adminContractAddress);
147 }
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# VesselManagerOperations Code Style Findings

## VMO-01C: Loop Iterator Optimizations

| Type | Severity | Location |
| --- | --- | --- |
| **Gas Optimization** | Informational | VesselManagerOperations.sol:L485, L548, L594, L782 |

### Description:

The linked `for` loops increment / decrement their iterator "safely" due to Solidity's built - in safe arithmetics (post-`0.8.X`).

### Example:

contracts/VesselManagerOperations.sol

SOL

```sol
485 for (vars.i = 0; vars.i < _vesselArray.length; vars.i++) {
```

### Recommendation:

We advise the increment / decrement operations to be performed in an `unchecked` code block as the last statement within each `for` loop to optimize their execution cost.

### Alleviation:

The loop iterator increments have been optimized as advised where applicable, however, their `i++` counterpart is utilized instead of `++i`. We advise the latter to be set in use as it is more optimal than the present code.

# VMO-02C: Redundant Initialization Paradigm

| Type | Severity | Location |
| --- | --- | --- |
| Gas Optimization | Informational | VesselManagerOperations.sol:L66-L67, L75 |

## Description:

The `VesselManagerOperations` contract inherits the OpenZeppelin `OwnableUpgradeable` implementation which contains the `Initializable` implementation, put in use within the `VesselManagerOperations::setAddresses` function. As such, the manual `isInitialized` flag is redundant.

## Example:

contracts/VesselManagerOperations.sol

SOL

```sol
function setAddresses(
    address _vesselManagerAddress,
    address _sortedVesselsAddress,
    address _stabilityPoolAddress,
    address _collSurplusPoolAddress,
    address _debtTokenAddress,
    address _adminContractAddress
) external initializer {
    require(!isInitialized, "Already initialized");
    __Ownable_init();
    vesselManager = IVesselManager(_vesselManagerAddress);
    sortedVessels = ISortedVessels(_sortedVesselsAddress);
    stabilityPool = IStabilityPool(_stabilityPoolAddress);
    collSurplusPool = ICollSurplusPool(_collSurplusPoolAddress);
    debtToken = IDebtToken(_debtTokenAddress);
    adminContract = IAdminContract(_adminContractAddress);
    isInitialized = true;
}
```

### Recommendation:

We advise it and its validations to be omitted from the codebase as it is ineffectual and duplicates the purpose of the `Initializable::initializer` modifier.

### Alleviation:

The manual initialization methodology has been removed from the contract as advised.

# VMO-03C: Suboptimal Struct Declaration Styles

| Type | Severity | Location |
| --- | --- | --- |
| Code Style | Informational | VesselManagerOperations.sol:L97-L101, L347, L770 |

### Description:

The linked declaration styles of the referenced structs are using index-based argument initialization.

### Example:

contracts/VesselManagerOperations.sol

SOL

```sol
LiquidationContractsCache memory contractsCache = LiquidationContractsCache(
    adminContract.activePool(),
    adminContract.defaultPool(),
    sortedVessels
);
```

### Recommendation:

We advise the key-value declaration format to be utilized instead in each instance, greatly increasing the legibility of the codebase.

### Alleviation:

The key-value declaration style is now in use in all referenced instances of the exhibit, addressing it in full.

# 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.

## External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

## 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 Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

## Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. 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 contract-level variable that is present in the inheritance chain of the local execution level's context.

## Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

## Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

## Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

## 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.

## Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

## Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (`require` statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

# 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.