OMNISCIA

May 20, 2024

SMART CONTRACT AUDIT REPORT

Euler Finance
Vault Kit

omniscia.io

info@omniscia.io

Online report: euler-finance-vault-kit

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

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

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

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


omniscia.io


info@omniscia.io

Vault Kit Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
a44880b0c0	April 9th 2024	896c6e7291
a44880b0c0	April 9th 2024	1ee9af79b1
fb2dd77a6f	April 24th 2024	67e65422eb
0f2192ac81	May 16th 2024	1498d443bd
0f2192ac81	May 20th 2024	8983dc326a

Audit Overview

We were tasked with performing an audit of the Euler Finance codebase and in particular their Euler Vault Kit that closely integrates with their Ethereum Vault Connector to facilitate the creation of a lending and borrowing market.

High-Level Description

The Euler Vault Kit represents a collection of modular contracts that are meant to be combined via a purpose-built proxy to create an **EIP-4626** vault with lending and/or borrowing capabilities that has a high level of customization, permitting most features of the vault to be enabled, disabled, and/or gatekept.

The EVK relies on the Ethereum Vault Connector (EVC) to operate properly, integrating the sub-account authorization system of the EVC as well as ensuring most mutating calls are forcibly routed via the EVC to permit the EVK to register itself, and potentially an account, for post-interaction health / safety checks.

Proxy Model

The EVK utilizes a purpose-built proxy relay system that will append three addresses at the end of each call that provide data the vault instance can utilize, permitting the same logic to be utilized for multiple deployments without mutating the vault's bytecode (i.e. `immutable` variables) or storage.

We identified certain flaws in the proxy model that arise from insecure `assembly` practices, however, they represent latent vulnerabilities that would manifest in future updates as they would result in corruption of the memory layout of the call which presently does not result in an actively exploitable flaw.

Custom Data Types

The EVK codebase employs multiple custom data types that wrap primitive number types to expose utility functions globally.

We believe that this approach is clear, concise, and would advise the Euler Finance team to closely monitor the Solidity compiler utilized for any potential compiler-related vulnerabilities that may arise from custom types, as they are a recently released feature.

External Integrations

The EVK heavily relies on the proper operation of the EVC to conduct its security checks, as in most instances these checks are deferred until the end of a batch's execution at the EVC level. In practical terms, the health checks of the protocol as well as of an account are performed **after all interactions that the user wishes to conduct with the vault have concluded**.

We validated that the integration is securely performed, and did not identify any improprieties in relation to the EVC calls performed.

The system natively supports an `IBalanceTracker` integration as well, recording balance changes to an external contract whenever they occur for a particular vault.

In relation to this integration, we identified a potential way a loan violator may force their liquidation to fail by propagating an error from the `IBalanceTracker` to the vault's context.

Finally, a hook system is implemented that effectively permits access control on certain vault operations to be guarded by an external contract with arbitrary logic. As no hook was in scope of the audit, we were not able to assess whether this particular integration has been securely performed.

Lending & Borrowing Market

The EVK at its base implements a system similar to the original Euler Finance vision; a debt (`DToken`) and collateral (`EToken`) token representation for a lending & borrowing market.

The mechanisms of the lending and borrowing market were evaluated, and two flaws were identified in relation to its operation and specifically in relation to its liquidation and interest accumulator modules.

We observed a discrepancy in the former whereby toxic liquidations are permitted to occur by offering a discount on the collateral of a lending position that would otherwise be considered as solvent.

In the latter, we noted a potential misbehaviour in how interest rate updates are handled. Specifically, the conditions under which interest rate updates are reflected are two:

We do not consider the second bullet mandatory, and in reality an interest rate update that could be reflected for active borrow positions will not be due to not being able to capture protocol fees which is unsound.

Inferred Security

A significant portion of the codebase will presently operate properly due to inferred security traits that may be enforced from upstream functions, defined as documentational constraints, or inherited via mathematical guarantees.

While the Euler Finance team has introduced multiple lines of in-line documentation to attempt to elaborate on certain seemingly insecure decisions made, we advise this effort to be continued as certain seemingly insecure statements are inadequately substantiated, such as the prohibition of self-transfers, or the discardment of the `RPow::rpow` result.

Overview Conclusion

We advise the Euler Finance 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 Euler Finance team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on, the PR it was included in, as well as an extensive document describing the Euler Finance team's assessment of each exhibit.

We evaluated all alleviations performed by Euler Finance and have identified that certain exhibits have been partially alleviated, or require a re-visit after further elaborations were introduced by our team. We advise the Euler Finance team to revisit the following exhibits: `BUL-01M`, `BUL-01C`, `BUS-01M`, `LNO-01M`

Additionally, we have introduced context to the following exhibits which may prompt the Euler Finance team to conduct further remediative actions: `BPY-02C`, `ESH-04C`, `MPD-01M`, `MPD-03M`, `MPD-01C`

All other exhibits have either been properly alleviated, been safely acknowledged, or been marked as nullified after clarifications were suppled to us by the Euler Finance team.

The Euler Finance team provided follow-up alleviations and responses that are reflected in the next post-audit conclusion chapter.

Post-Audit Conclusion (0f2192ac81)

The Euler Finance team revisited all aforementioned exhibits, either applying further alleviations or supplying us with clarifications as to why the exhibits should no longer be considered open.

With regard to exhibit `LNO-01M`, we deliberated the liquidation mechanism we proposed with the Euler Finance team and have concluded that the original approach of the codebase is considered the most optimal when taking partial liquidations into account.

We consider all outputs of the audit report properly consumed by the Euler Finance team and no further actions are expected in relation to the audit report.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	1	0	0
Informational	63	54	0	9
Minor	5	3	0	2
Medium	2	2	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **20 findings utilizing static analysis** tools as well as identified a total of **51 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Scope

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

Target

- Repository: **https://github.com/euler-xyz/euler-vault-kit**
- Commit: a44880b0c051bec87bd3bf35383666fd6e9053d2
- Language: Solidity
- Network: Ethereum
- Revisions: **a44880b0c0**, **fb2dd77a6f**, **0f2192ac81**

Contracts Assessed

File	Total Finding(s)
src/EVault/shared/types/Assets.sol (AST)	0
src/EVault/shared/types/AmountCap.sol (ACP)	0
src/EVault/shared/AssetTransfers.sol (ATS)	0
src/EVault/shared/Base.sol (BES)	2
src/EVault/modules/Borrowing.sol (BGN)	1
src/GenericFactory/BeaconProxy.sol (BPY)	5
src/EVault/shared/BorrowUtils.sol (BUS)	4
src/EVault/shared/BalanceUtils.sol (BUL)	3
src/EVault/modules/BalanceForwarder.sol (BFR)	2
src/EVault/shared/Cache.sol (CEH)	3

src/EVault/shared/Constants.sol (CST)	1
src/EVault/shared/types/ConfigAmount.sol (CAT)	1
src/EVault/shared/lib/ConversionHelpers.sol (CHS)	1
src/EVault/DToken.sol (DTN)	0
src/EVault/Dispatch.sol (DHC)	2
src/Synths/ESynth.sol (ESH)	4
src/EVault/EVault.sol (EVT)	0
src/EVault/shared/Errors.sol (ESR)	0
src/EVault/shared/Events.sol (EST)	0
src/EVault/shared/EVCClient.sol (EVC)	2
src/Synths/ERC20Collateral.sol (ERC)	0
src/Synths/EulerSavingsRate.sol (ESE)	1
src/EVault/shared/types/Flags.sol (FSG)	0
src/EVault/modules/Governance.sol (GEC)	6

src/GenericFactory/GenericFactory.sol (GFY)	7
src/Synths/IRMSynth.sol (IRM)	1
src/EVault/modules/Initialize.sol (IEZ)	1
src/InterestRateModels/IRMLinearKink.sol (IRL)	1
src/EVault/shared/types/LTVType.sol (LTV)	0
src/EVault/shared/LTVUtils.sol (LTU)	0
src/EVault/shared/types/LTVConfig.sol (LTC)	1
src/EVault/modules/Liquidation.sol (LNO)	3
src/EVault/shared/LiquidityUtils.sol (LUS)	0
src/GenericFactory/MetaProxyDeployer.sol (MPD)	4
src/EVault/shared/types/Owed.sol (ODE)	0
src/EVault/shared/lib/ProxyUtils.sol (PUS)	0
src/ProtocolConfig/ProtocolConfig.sol (PCG)	4
src/Synths/PegStabilityModule.sol (PSM)	5

src/EVault/shared/lib/RPow.sol (RPW)	0
src/EVault/shared/lib/RevertBytes.sol (RBS)	0
src/EVault/modules/RiskManager.sol (RMR)	1
src/EVault/shared/types/Shares.sol (SSE)	0
src/EVault/shared/Storage.sol (SEG)	2
src/EVault/shared/types/Snapshot.sol (STO)	1
src/EVault/shared/lib/SafeERC20Lib.sol (SER)	0
src/EVault/modules/Token.sol (TNE)	0
src/EVault/shared/types/Types.sol (TSE)	0
src/EVault/shared/types/UserStorage.sol (USE)	2
src/EVault/modules/Vault.sol (VTL)	0
src/EVault/shared/types/VaultCache.sol (VCE)	0
src/EVault/shared/types/VaultStorage.sol (VSE)	0

Compilation

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

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

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.23` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.0`).

We advise them to be locked to `0.8.23` (`=0.8.23`), the same version utilized for our static analysis as well as optimizational review of the codebase.

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

Static Analysis

The execution of our static analysis toolkit identified **93 potential issues** within the codebase of which **48 were ruled out to be false positives** or negligible findings.

The remaining **45 issues** were validated and grouped and formalized into the **20 exhibits** that follow:

ID	Severity	Addressed	Title
BES-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
BES-02S	Informational	Yes	Inexistent Visibility Specifiers
BPY-01S	Informational	Yes	Inexistent Visibility Specifiers
BGN-01S	Informational	Yes	Multiple Top-Level Declarations
CST-01S	Informational	Yes	Illegible Numeric Value Representation
DHC-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
EVC-01S	Informational	Yes	Inexistent Sanitization of Input Address
EVC-02S	Informational	Yes	Inexistent Visibility Specifier
ESE-01S	Informational	Yes	Inexistent Visibility Specifier
GFY-01S	Informational	Yes	Inexistent Sanitization of Input Address

GFY-02S	Informational	Yes	Inexistent Visibility Specifiers
GFY-03S	Informational	Acknowledged	Multiple Top-Level Declarations
GEC-01S	Informational	Yes	Inexistent Visibility Specifiers
IRM-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
IEZ-01S	Informational	Yes	Inexistent Visibility Specifiers
LNO-01S	Informational	Yes	Inexistent Visibility Specifier
PSM-01S	Informational	Yes	Illegible Numeric Value Representation
PSM-02S	Informational	Yes	Inexistent Sanitization of Input Addresses
PCG-01S	Informational	Yes	Inexistent Sanitization of Input Addresses
SEG-01S	Informational	Yes	Inexistent Visibility Specifiers

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Euler Finance's Vault Kit.

As the project at hand implements an EIP-4626 based borrowing and lending vault, intricate care was put into ensuring that the **flow of funds & assets within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed multiple medium-level vulnerabilities** within the system which could have had **moderate ramifications** to its overall operation; more details can be observed in the audit's summary.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a certain extent, however, we strongly recommend it to be expanded at certain complex points such as Euler Finance.

A total of **51 findings** were identified over the course of the manual review of which **19 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
BFR-01M	Unknown	Yes	Inexplicable Permittance of Re-Entrancies
BUL-01M	Medium	Nullified	Potentially Insecure Balance Tracker Invocation
BPY-01M	Informational	Acknowledged	Insecure Memory Array Expansion
BPY-02M	Informational	Yes	Unhandled Misbehaviour of Memory Load Operation
BUS-01M	Informational	Yes	Improper Event Emittance

CEH-01M	Minor	Yes	Inexistent Upward-Rounding Operation of Total Borrows
CEH-02M	Medium	Yes	Improper Interest Rate Update
CHS-01M	Minor	Acknowledged	Inefficient Imposition of Virtual Deposit
GFY-01M	Informational	Acknowledged	Potentially Dangerous Push Ownership Pattern
GFY-02M	Informational	Yes	Potentially Improper Exposure of Data
GEC-01M	Informational	Nullified	Potentially Weak Validation of Caps
GEC-02M	Minor	Nullified	Insecure Clearance of LTV
LNO-01M	Informational	Nullified	Improper Liquidation Incentive Calculations
MPD-01M	Informational	Nullified	Inexistent Allocation of Memory

MPD-02M	Informational	Yes	Inexistent Appendment of Metadata Length (EIP-3448 Discrepancy)
MPD-03M	Informational	Yes	Inexistent Handling of Creation Failure
PSM-01M	Informational	Yes	Inexistent Sanitization of Fees
PSM-02M	Minor	Acknowledged	Inexistent Reversion of Rounding Direction
RMR-01M	Minor	Nullified	Overprotective Security Mechanism

Code Style

During the manual portion of the audit, we identified **32 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
BFR-01C	Informational	Yes	Inefficient `mapping` Lookups
BUL-01C	Informational	Yes	Inefficient `mapping` Lookups
BUL-02C	Informational	Yes	Inexistent Propagation of Optimized Arithmetics
BPY-01C	Informational	Yes	Repetitive Value Literal
BPY-02C	Informational	Acknowledged	Unconditional Appendment of Metadata
BUS-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
BUS-02C	Informational	Yes	Inefficient Update of User's Debt
BUS-03C	Informational	Yes	Inefficient `mapping` Lookups
CEH-01C	Informational	Yes	Inconsistent Unwrapping Mechanism
CAT-01C	Informational	Yes	Repetitive Value Literal

DHC-01C	Informational	Yes	Redundant Copy of Call Data
ESH-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
ESH-02C	Informational	Yes	Inefficient Loop Limit Evaluation
ESH-03C	Informational	Yes	Inefficient Structure Mutability Specifiers
ESH-04C	Informational	Nullified	Loop Iterator Optimization
GFY-01C	Informational	Acknowledged	Generic Typographic Mistake
GFY-02C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
GEC-01C	Informational	Yes	Inefficient Negation of Conditional
GEC-02C	Informational	Yes	Potentially Misleading Variable Naming
GEC-03C	Informational	Yes	Redundant Parenthesis Statement
IRL-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
LTC-01C	Informational	Yes	Inefficient Conditional Exclusivity
LNO-01C	Informational	Acknowledged	Repetitive Value Literal
MPD-01C	Informational	Nullified	Inefficient Memory Assignment

PSM-01C	Informational	Acknowledged	Inefficient Re-Calculation of Fee Percentages
PCG-01C	Informational	Acknowledged	Optimization of Structure Assignments
PCG-02C	Informational	Yes	Repetitive Value Literal
PCG-03C	Informational	Yes	Variable Grouping
STO-01C	Informational	Yes	Potentially Inefficient Storage Structure
SEG-01C	Informational	Nullified	Potential Combination of Data Points
USE-01C	Informational	Yes	Inconsistent Bit Clearance Style
USE-02C	Informational	Yes	Repetitive Value Literal

Base Static Analysis Findings

BES-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	Base.sol:L29-L33

Description:

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

Impact:

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

Example:

src/EVault/shared/Base.sol

SOL

```sol
29  constructor(Integrations memory integrations) EVCClient(integrations.evc) {
30      protocolConfig = IProtocolConfig(integrations.protocolConfig);
31      balanceTracker = IBalanceTracker(integrations.balanceTracker);
32      permit2 = integrations.permit2;
33  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The code was updated to utilize a newly introduced `AddressUtils` contract that will verify the address contains code instead, acting as a superset of the non-zero check and providing better sanitization of the referenced arguments.

To note, sanitization was solely introduced for the `integrations.protocolConfig` argument as the other contracts were deemed to be acceptable as `0` (i.e. unset).

BES-02S: Inexistent Visibility Specifiers

Type	Severity	Location
Code Style	Informational	Base.sol:L18, L19, L20

Description:

The linked variables have no visibility specifier explicitly set.

Example:

src/EVault/shared/Base.sol

SOL

```
18  IProtocolConfig immutable protocolConfig;
```

Recommendation:

We advise them 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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `internal` visibility specifier has been introduced to all referenced variables, preventing potential compilation discrepancies and addressing this exhibit.

BeaconProxy Static Analysis Findings

BPY-01S: Inexistent Visibility Specifiers

Type	Severity	Location
Code Style	Informational	BeaconProxy.sol:L7, L9, L11, L13-L18

Description:

The linked variables have no visibility specifier explicitly set.

Example:

src/GenericFactory/BeaconProxy.sol

SOL

```sol
7   bytes32 constant BEACON_SLOT =
0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50;
```

Recommendation:

We advise them 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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `internal` visibility specifier has been introduced to all referenced variables, preventing potential compilation discrepancies and addressing this exhibit.

Borrowing Static Analysis Findings

BGN-01S: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	Borrowing.sol:L16, L23

Description:

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

Example:

src/EVault/modules/Borrowing.sol

SOL

```sol
/// @notice Definition of callback method that flashLoan will invoke on your contract
interface IFlashLoan {
    function onFlashLoan(bytes memory data) external;
}

/// @title BorrowingModule
/// @author Euler Labs (https://www.eulerlabs.com/)
/// @notice An EVault module handling borrowing and repaying of vault assets
abstract contract BorrowingModule is IBorrowing, Base, AssetTransfers, BalanceUtils, LiquidityUtils {
```

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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `IFlashLoan` interface has been relocated to its dedicated `IFlashLoan.sol` file and is correspondingly imported, addressing this exhibit.

Constants Static Analysis Findings

CST-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	Constants.sol:L17

Description:

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

Example:

src/EVault/shared/Constants.sol

SOL

```
17  uint256 constant MAX_ALLOWED_INTEREST_RATE = 291867236321699131285;
```

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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and opted to retain the current representation in place as the literal is not meant to be human-readable but rather machine-generated per the comment that already accompanied it.

As such, we consider this exhibit addressed in the sense that we are aligned with the Euler Finance team's assessment of the literal's canonical representation.

Dispatch Static Analysis Findings

DHC-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	Dispatch.sol:L52-L61

Description:

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

Impact:

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

Example:

src/EVault/Dispatch.sol

SOL

```sol
52  constructor(Integrations memory integrations, DeployedModules memory modules)
Base(integrations) {
53      MODULE_INITIALIZE = modules.initialize;
54      MODULE_TOKEN = modules.token;
55      MODULE_VAULT = modules.vault;
56      MODULE_BORROWING = modules.borrowing;
57      MODULE_LIQUIDATION = modules.liquidation;
58      MODULE_RISKMANAGER = modules.riskManager;
59      MODULE_BALANCE_FORWARDER = modules.balanceForwarder;
60      MODULE_GOVERNANCE = modules.governance;
61  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The code was updated to utilize a newly introduced `AddressUtils` contract that will verify the address contains code instead, acting as a superset of the non-zero check and providing better sanitization of the referenced arguments.

EVCClient Static Analysis Findings

EVC-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	EVCClient.sol:L28-L30

Description:

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

Impact:

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

Example:

src/EVault/shared/EVCClient.sol

SOL

```sol
28  constructor(address _evc) {
29      evc = IEVC(_evc);
30  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The code was updated to utilize a newly introduced `AddressUtils` contract that will verify the address contains code instead, acting as a superset of the non-zero check and providing better sanitization of the referenced argument.

EVC-02S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	EVCClient.sol:L18

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/EVault/shared/EVCClient.sol

SOL

```
18  IEVC immutable evc;
```

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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

EulerSavingsRate Static Analysis Findings

ESE-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	EulerSavingsRate.sol:L29

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/Synths/EulerSavingsRate.sol

SOL

```
29  uint256 totalAssetsDeposited;
```

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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

GenericFactory Static Analysis Findings

GFY-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	GenericFactory.sol:L65-L73

Description:

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

Impact:

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

Example:

src/GenericFactory/GenericFactory.sol

SOL

```sol
65  constructor(address admin) {
66      emit Genesis();
67
68      reentrancyLock = REENTRANCYLOCK__UNLOCKED;
69
70      upgradeAdmin = admin;
71
72      emit SetUpgradeAdmin(admin);
73  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

GFY-02S: Inexistent Visibility Specifiers

Type	Severity	Location
Code Style	Informational	GenericFactory.sol:L15, L16

Description:

The linked variables have no visibility specifier explicitly set.

Example:

src/GenericFactory/GenericFactory.sol

SOL

```sol
15  uint256 constant REENTRANCYLOCK__UNLOCKED = 1;
```

Recommendation:

We advise them 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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `internal` visibility specifier has been introduced to all referenced variables, preventing potential compilation discrepancies and addressing this exhibit.

GFY-03S: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	GenericFactory.sol:L8, L12

Description:

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

Example:

src/GenericFactory/GenericFactory.sol

SOL

```sol
8   interface IComponent {
9       function initialize(address creator) external;
10  }
11
12  contract GenericFactory is MetaProxyDeployer {
```

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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and opted to retain the current declaration structure as they do not consider the `IComponent` declaration to be reducing the overall file's legibility.

Governance Static Analysis Findings

GEC-01S: Inexistent Visibility Specifiers

Type	Severity	Location
Code Style	Informational	Governance.sol:L22, L23, L24

Description:

The linked variables have no visibility specifier explicitly set.

Example:

src/EVault/modules/Governance.sol

SOL

```sol
22  uint16 constant MAX_PROTOCOL_FEE_SHARE = 0.5e4;
```

Recommendation:

We advise them 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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `internal` visibility specifier has been introduced to all referenced variables, preventing potential compilation discrepancies and addressing this exhibit.

IRMSynth Static Analysis Findings

IRM-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	IRMSynth.sol:L28-L34

Description:

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

Impact:

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

Example:

src/Synths/IRMSynth.sol

SOL

```sol
28  constructor(address synth_, address referenceAsset_, address oracle_) {
29      synth = synth_;
30      referenceAsset = referenceAsset_;
31      oracle = IPriceOracle(oracle_);
32
33      irmStorage = IRMData({lastUpdated: uint40(block.timestamp), lastRate:
BASE_RATE});
34  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

Initialize Static Analysis Findings

IEZ-01S: Inexistent Visibility Specifiers

Type	Severity	Location
Code Style	Informational	Initialize.sol:L21, L22

Description:

The linked variables have no visibility specifier explicitly set.

Example:

src/EVault/modules/Initialize.sol

SOL

```sol
21  uint256 constant INITIAL_INTEREST_ACCUMULATOR = 1e27; // 1 ray
```

Recommendation:

We advise them 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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `internal` visibility specifier has been introduced to all referenced variables, preventing potential compilation discrepancies and addressing this exhibit.

Liquidation Static Analysis Findings

LNO-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	Liquidation.sol:L19

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/EVault/modules/Liquidation.sol

SOL

```sol
19  uint256 constant MAXIMUM_LIQUIDATION_DISCOUNT = 0.2 * 1e18;
```

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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

PegStabilityModule Static Analysis Findings

PSM-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	PegStabilityModule.sol:L13

Description:

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

Example:

src/Synths/PegStabilityModule.sol

```sol
13  uint256 public constant BPS_SCALE = 10000;
```

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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced value literal has been updated in its representation to `100_00` in accordance with the recommendation's underscore style, addressing this exhibit.

PSM-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	PegStabilityModule.sol:L21-L28

Description:

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

Impact:

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

Example:

src/Synths/PegStabilityModule.sol

SOL

```sol
21  constructor(address _evc, address _synth, address _underlying, uint256
toUnderlyingFeeBPS, uint256 toSynthFeeBPS)
22      EVCUtil(IEVC(_evc))
23  {
24      synth = ESynth(_synth);
25      underlying = IERC20(_underlying);
26      TO_UNDERLYING_FEE = toUnderlyingFeeBPS;
27      TO_SYNTH_FEE = toSynthFeeBPS;
28  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

ProtocolConfig Static Analysis Findings

PCG-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	ProtocolConfig.sol:L55-L62

Description:

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

Impact:

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

Example:

src/ProtocolConfig/ProtocolConfig.sol

SOL

```sol
55  constructor(address admin_, address feeReceiver_) {
56      admin = admin_;
57      feeReceiver = feeReceiver_;
58
59      minInterestFee = 0.1e4;
60      maxInterestFee = 1e4;
61      protocolFeeShare = 0.1e4;
62  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

Storage Static Analysis Findings

SEG-01S: Inexistent Visibility Specifiers

Type	Severity	Location
Code Style	Informational	Storage.sol:L12, L16, L19

Description:

The linked variables have no visibility specifier explicitly set.

Example:

src/EVault/shared/Storage.sol

```sol
12  bool initialized;
```

Recommendation:

We advise them 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 (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `internal` visibility specifier has been introduced to all referenced variables, preventing potential compilation discrepancies and addressing this exhibit.

BalanceForwarder Manual Review Findings

BFR-01M: Inexplicable Permittance of Re-Entrancies

Type	Severity	Location
Standard Conformity	Unknown	BalanceForwarder.sol:L23, L36

Description:

The `BalanceForwarderModule::enableBalanceForwarder` and `BalanceForwarderModule::disableBalanceForwarder` functions do not impose re-entrancy protections, permitting them to be invoked in between vault operations.

Impact:

The actual severity of this exhibit is directly correlated with the integrated `balanceTracker` implementation and thus cannot be quantified.

Example:

src/EVault/modules/BalanceForwarder.sol

SOL

```sol
22  /// @inheritdoc IBalanceForwarder
23  function enableBalanceForwarder() public virtual reentrantOK {
24      if (address(balanceTracker) == address(0)) revert
E_BalanceForwarderUnsupported();
25
26      address account = EVCAuthenticate();
27      bool wasBalanceForwarderEnabled =
vaultStorage.users[account].isBalanceForwarderEnabled();
28
29      vaultStorage.users[account].setBalanceForwarder(true);
30      balanceTracker.balanceTrackerHook(account,
vaultStorage.users[account].getBalance().toUint(), false);
31
```

Example (Cont.):

```sol
        if (!wasBalanceForwarderEnabled) emit BalanceForwarderStatus(account, true);
    }

    /// @inheritdoc IBalanceForwarder
    function disableBalanceForwarder() public virtual reentrantOK {
        if (address(balanceTracker) == address(0)) revert E_BalanceForwarderUnsupported();

        address account = EVCAuthenticate();
        bool wasBalanceForwarderEnabled = vaultStorage.users[account].isBalanceForwarderEnabled();

        vaultStorage.users[account].setBalanceForwarder(false);
        balanceTracker.balanceTrackerHook(account, 0, false);

        if (wasBalanceForwarderEnabled) emit BalanceForwarderStatus(account, false);
    }
```

Recommendation:

We advise re-entrancy guards to be set in place as any re-entrancy could affect the `balanceTracker` implementation and compromise the security guarantees that the `balanceTracker` expects in relation to `IBalanceTracker::balanceTrackerHook` invocations.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced functions have been marked as non-reentrant, preventing their invocation during potential re-entrancy-based attack vectors.

BalanceUtils Manual Review Findings

BUL-01M: Potentially Insecure Balance Tracker Invocation

Type	Severity	Location
Language Specific	Medium	BalanceUtils.sol:L122-L129

Description:

The `BalanceUtils::tryBalanceTrackerHook` function is meant to interact with the `balanceTracker` in a non-prohibitive manner, ensuring that whatever failure may be encountered in the `balanceTracker` cannot affect the sensitive operations of the `BalanceUtils` contract.

We do not consider the present safety precautions as adequate due to the following two possibilities:

Despite what the syntax of the `BalanceUtils::tryBalanceTrackerHook` function infers, any returned payload by the `IBalanceTracker` will be decoded and in case it is maliciously large will result in an out-of-gas error regardless of the gas stipend supplied to the call.

On the other hand, any unbounded gas operation in the `IBalanceTracker` (i.e. due to invalid EVM operations, an unbounded loop, propagation of the aforementioned error via the `balanceTracker`, etc.) will result in the `BalanceUtils` context and to not be able to complete even if the `63/64` rule is observed.

Depending on the trust level of the configured `balanceTracker`, the severity of this vulnerability ranges from high-to-moderate as it can either be maliciously imposed or may result inadvertently.

Impact:

The present `BalanceUtils::tryBalanceTrackerHook` integration with the `IBalanceTracker` may result in an out-of-gas error, preventing sensitive balance adjustments from being executable.

Example:

src/EVault/shared/BalanceUtils.sol

```sol
function tryBalanceTrackerHook(address account, uint256 newAccountBalance, bool forfeitRecentReward)
    private
    returns (bool success)
{
    (success,) = address(balanceTracker).call(
        abi.encodeCall(IBalanceTracker.balanceTrackerHook, (account, newAccountBalance, forfeitRecentReward))
    );
}
```

Recommendation:

We advise the `BalanceUtils::tryBalanceTrackerHook` function to instead utilize a low-level `assembly` block to avoid decoding the returned payload unconditionally.

As an added security measure, the code may impose a fixed gas stipend that any `IBalanceTracker` compliant contract will need to abide by if it wishes to be integrated properly by the Euler Finance system.

Alleviation (fb2dd77a6f):

The Euler Finance team has clarified that the Balance Tracker is a trusted contract and as such is expected to behave properly. We consider this statement to contradict the code's present implementation which will perform a low-level call opportunistically instead of ensuring that the call succeeds under all circumstances.

The code should either adopt a complete-trust model whereby the integration is performed directly without a low-level call and ignorance of the `success` flag, or a zero-trust model whereby the integration is performed with maximum security sanitizations enabled.

We originally assumed that a no-trust model is meant to be adhered to, and in light of the new information supplied by the Euler Finance team advise that the integration is performed directly by wrapping the `balanceTracker` in the `IBalanceTracker` interface and invoking the relevant function.

Alleviation (0f2192ac81):

The Euler Finance team evaluated our follow-up recommendation and proceeded with removing the `BalanceUtils::tryBalanceTrackerHook` function from the contract, instead performing the `IBalanceTracker` integration directly as advised.

As such, we consider this exhibit properly nullified as the originally described vulnerability was incorrect based on the complete-trust model that the code presently adheres to.

BeaconProxy Manual Review Findings

BPY-01M: Insecure Memory Array Expansion

Type	Severity	Location
Language Specific	Informational	BeaconProxy.sol:L38-L41

Description:

The construction mechanism of a contract with `immutable` variables entails retaining the `immutable` variable values in memory until the `BeaconProxy::constructor` concludes.

The present code will pad the length of the `trailingData` input argument to ensure the ABI decode operation that ensues succeeds, however, there is no guarantee that the trailing bytes will be zero or in other words that the `MLOAD` operations that occur under-the-hood of the `abi.decode` operation will read zero-value bytes.

Impact:

In its present state, the referenced statement will decode the `trailingData` payload and then assign to the respective `immutable` slots. As a result, the read operations from the slots beyond the original length of the `trailingData` are somewhat safe but invalidate memory layout assumptions of the Solidity language.

To ensure the code behaves as expected across compilation versions and to prevent compilation bugs from manifesting, we advise the code to adopt the slot-by-slot approach we advised.

Example:

src/GenericFactory/BeaconProxy.sol

SOL

```sol
22  constructor(bytes memory trailingData) {
23      emit Genesis();
24
25      require(trailingData.length <= MAX_TRAILING_DATA_LENGTH, "trailing data too long");
26
27      // Beacon is always the proxy creator; store it in immutable
28      beacon = msg.sender;
29
30      // Store the beacon address in ERC-1967 slot for compatibility with block explorers
31      assembly {
```

Example (Cont.):

```sol
            sstore(BEACON_SLOT, caller())
        }

        // Record length as immutable
        metadataLength = trailingData.length;

        // Pad length with uninitialized memory so the decode will succeed
        assembly {
            mstore(trailingData, 128)
        }
        (metadata0, metadata1, metadata2, metadata3) = abi.decode(trailingData, (bytes32, bytes32, bytes32, bytes32));
    }
```

Recommendation:

We advise consecutive `if` statements to be introduced, reading 32-byte chunks from the `trailingData` and storing them to their respective `metadata` prefixed slot in sequence.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and clarified that they acknowledge the behaviour outlined as whatever is written to the `metadataX` data points beyond the `metadataLength` is ignored by the appendment performed in consequent calls.

As such, we consider this exhibit safely acknowledged **provided that the compiler version utilized for the project does not prevent uninitialized memory reads from occurring**.

BPY-02M: Unhandled Misbehaviour of Memory Load Operation

Type	Severity	Location
Logical Fault	Informational	BeaconProxy.sol:L56, L61

Description:

The `BeaconProxy::fallback` function assumes that the `IBeacon::implementation()` function will always yield a 32-byte payload, however, that may not be the case.

Impact:

A contradiction exists in the codebase presently whereby a failure of the `GenericFactory::implementation` function is explicitly handled even though it should never occur, while a failure in how it behaves remains unhandled.

In detail, the code will presently load from the `0` memory pointer meaning that the `IMPLEMENTATION_SELECTOR` will be yielded as the "implementation" incorrectly.

If the code is meant to be compatible with other implementations than the `GenericFactory`, the `returndatasize` should be explicitly handled.

If the code is expected to be solely compatible with the `GenericFactory`, the code should not evaluate whether the `staticcall` failed as a `staticcall` to a compiler-generated getter function cannot practically fail unless insufficient gas was supplied in which case the remaining code will fail out-of-gas regardless.

Example:

src/GenericFactory/BeaconProxy.sol

SOL

```
56  let result := staticcall(gas(), beacon_, 0, 4, 0, 32)
57  if iszero(result) {
58      returndatacopy(0, 0, returndatasize())
59      revert(0, returndatasize())
60  }
61  let implementation := mload(0)
```

Recommendation:

As a solution, the code should ensure that the `returndatasize()` of the `staticcall` is equal to `32` so as to ensure that the memory pointer at `0` has been properly overwritten.

Alternatively, the code should omit handling of the `GenericFactory::implementation` function call's failure as the code will succeed in all circumstances.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The code was updated with the latter of our two recommendations as it expects the call fetching the implementation to always succeed, addressing this exhibit.

BorrowUtils Manual Review Findings

BUS-01M: Improper Event Emittance

Type	Severity	Location
Standard Conformity	Informational	BorrowUtils.sol:L93, L94

Description:

The `BorrowUtils::transferBorrow` function will emit two events when performing a transfer of debt from one account to another, both of which will either originate or end at the zero-address indicating mint or burn operations respectively.

These data points will be obfuscated off-chain as the borrow change of both the `from` and `to` account will encompass any uncaptured interest, rendering off-chain evaluation of the actual amount transferred computationally expensive and reliant on historical data.

Impact:

As the impact would be purely observable off-chain and for potential integrators, we consider this exhibit to be informational in nature.

Example:

src/EVault/shared/BorrowUtils.sol

SOL

```sol
70  function transferBorrow(VaultCache memory vaultCache, address from, address to,
Assets assets) internal {
71      Owed amount = assets.toOwed();
72
73      (Owed fromOwed, Owed fromOwedPrev) = updateUserBorrow(vaultCache, from);
74      (Owed toOwed, Owed toOwedPrev) = updateUserBorrow(vaultCache, to);
75
76      // If amount was rounded up, or dust is left over, transfer exact amount owed
77      if ((amount > fromOwed && (amount - fromOwed).isDust()) || (amount < fromOwed
&& (fromOwed - amount).isDust()))
78      {
79          amount = fromOwed;
```

Example (Cont.):

```sol
    }

    if (amount > fromOwed) revert E_InsufficientBalance();

    unchecked {
        fromOwed = fromOwed - amount;
    }

    toOwed = toOwed + amount;

    vaultStorage.users[from].setOwed(fromOwed);
    vaultStorage.users[to].setOwed(toOwed);

    logBorrowChange(from, fromOwedPrev, fromOwed);
    logBorrowChange(to, toOwedPrev, toOwed);
}
```

Recommendation:

We advise the code to emit events that synchronize the latest borrow balance, and then emit an event actually signifying the transfer of balance thus greatly aiding off-chain services in parsing the Euler Finance debt-related events.

Alleviation (fb2dd77a6f):

The Euler Finance team specified that they alleviated this exhibit in their response document, however, it appears to remain open in the final commit hash and may have been lost in between commits.

We attempted to identify a fix for this finding within the commit history of the PR supplied to us, however, we did not find any commit hash's name to signify a fix for this exhibit.

As the Euler Finance team intended to alleviate it, we advise them to revisit this exhibit as it remains open.

Alleviation (0f2192ac81):

The Euler Finance team clarified that the exhibit had been properly addressed in the original iteration of the final report albeit via changes at a different contract.

The `Borrowing::transferBorrow` function is invoked in two scenarios; a liquidation (**`Liquidation::liquidate`**) and a debt pull (**`Borrowing::pullDebt`**). The Euler Finance team identified that the liquidation execution path already emitted the debt transferred, and thus proceeded to introduce a `PullDebt` event to the **`Borrowing::pullDebt`** function as a gas optimal alleviation to this exhibit.

In light of this clarification, we consider this exhibit originally alleviated in the previous commit and thus addressed in the latest one as well.

Cache Manual Review Findings

CEH-01M: Inexistent Upward-Rounding Operation of Total Borrows

Type	Severity	Location
Mathematical Operations	Minor	Cache.sol:L99

Description:

The share-based fee captured as part of a borrow interest update will calculate the shares to mint based on a `newTotalAssets` evaluation **that does not round the borrows upwards**, resulting in an under-evaluation of the `newTotalAssets` and thus an over-evaluation of the fees captured.

Impact:

Although the fees captured will be over-evaluated, the error by which this overvaluation will occur is small rendering this exhibit to be of minor severity.

Example:

src/EVault/shared/Cache.sol

SOL

```
98  if (feeAssets != 0) {
99              uint256 newTotalAssets = vaultCache.cash.toUint() + (newTotalBorrows >>
INTERNAL_DEBT_PRECISION);
100             newTotalShares = newTotalAssets * newTotalShares / (newTotalAssets -
feeAssets);
101             newAccumulatedFees += newTotalShares -
vaultCache.totalShares.toUint();
102         }
103
104         // Store new values in vaultCache, only if no overflows will occur. Fees
are not larger than total shares, since they are included in them.
105
106         if (newTotalShares <= MAX_SANE_AMOUNT && newTotalBorrows <=
MAX_SANE_DEBT_AMOUNT) {
107             vaultCache.totalBorrows = newTotalBorrows.toOwed();
```

Example (Cont.):

```sol
            vaultCache.interestAccumulator = newInterestAccumulator;
            vaultCache.lastInterestAccumulatorUpdate = uint48(block.timestamp);

            if (newTotalShares != Shares.unwrap(vaultCache.totalShares)) {
                vaultCache.accumulatedFees = newAccumulatedFees.toShares();
                vaultCache.totalShares = newTotalShares.toShares();
            }
        }
    }
}

function totalAssetsInternal(VaultCache memory vaultCache) internal pure returns (uint256) {
    // total assets can exceed Assets max amount (MAX_SANE_AMOUNT)
    return vaultCache.cash.toUint() + vaultCache.totalBorrows.toAssetsUp().toUint();
}
```

Recommendation:

We advise the `newTotalAssets` calculation to properly use the `Cache::totalAssetsInternal` function, ensuring that the total assets calculated properly represent the expected total assets after the interest rate update is reflected in the contract's storage.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The calculation will properly round upwards by using the newly introduced `OwedLib::toAssetsUpUint256` function, alleviating this exhibit and ensuring consistency in the total asset evaluation across the system.

CEH-02M: Improper Interest Rate Update

Type	Severity	Location
Logical Fault	Medium	Cache.sol:L106-L115

Description:

In the current system, whether the interest rate update succeeds in the `Cache::initVaultCache` function relies on whether the `newTotalShares` and `newTotalBorrows` variables can be represented in their respective types.

Impact:

In the low-likelihood scenario whereby the `vaultCache.totalShares` value is very close to the `MAX_SANE_AMOUNT` but the `vaultCache.totalBorrows` value is relatively low, an interest rate update **will be forfeited incorrectly**. As the impact is high, we consider a medium-severity assessment as appropriate for this finding.

Example:

src/EVault/shared/Cache.sol

SOL

```sol
106 if (newTotalShares <= MAX_SANE_AMOUNT && newTotalBorrows <= MAX_SANE_DEBT_AMOUNT) {
107     vaultCache.totalBorrows = newTotalBorrows.toOwed();
108     vaultCache.interestAccumulator = newInterestAccumulator;
109     vaultCache.lastInterestAccumulatorUpdate = uint48(block.timestamp);
110
111     if (newTotalShares != Shares.unwrap(vaultCache.totalShares)) {
112         vaultCache.accumulatedFees = newAccumulatedFees.toShares();
113         vaultCache.totalShares = newTotalShares.toShares();
114     }
115 }
```

Recommendation:

We consider the current approach incorrect, as the total borrows and interest rate of the system should be updated **distinctly from the fees** permitting the system to accumulate interest on underlying borrows even if fees cannot be captured to increase the total share supply.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The code was updated to evaluate the borrow interest and accumulated fees updates distinctly, validating that the borrow interest rate can be applied at first and consequently evaluating whether fees can be safely accumulated.

As such, we consider this exhibit fully alleviated as a total borrow update taking precedence over a fee update is what we consider best practice.

ConversionHelpers Manual Review Findings

CHS-01M: Inefficient Imposition of Virtual Deposit

Type	Severity	Location
Standard Conformity	Minor	ConversionHelpers.sol:L20-L22

Description:

The `ConversionHelpers::conversionTotals` function will impose a virtual deposit system which enforces a `1:1` ratio between the assets and underlying shares, being equivalent in security to the "basic" level described by OpenZeppelin's **EIP-4626** research.

Impact:

While the **EIP-4626** first-deposit inflation attack is no longer profitable due to the `ConversionHelpers::conversionTotals` function's virtual offsets, it does not result in a net loss for a would-be attacker and thus could potentially be carried out as a griefing attack.

Example:

src/EVault/shared/lib/ConversionHelpers.sol

SOL

```sol
function conversionTotals(VaultCache memory vaultCache)
    internal
    pure
    returns (uint256 totalAssets, uint256 totalShares)
{
    unchecked {
        totalAssets =
            vaultCache.cash.toUint() + vaultCache.totalBorrows.toAssetsUp().toUint()
+ VIRTUAL_DEPOSIT_AMOUNT;
        totalShares = vaultCache.totalShares.toUint() + VIRTUAL_DEPOSIT_AMOUNT;
    }
```

Example (Cont.):

```sol
}
```

Recommendation:

We advise the system to offset the total shares by a single factor higher than the total assets, increasing the security level of the virtual deposit system in relation to first-deposit inflation attacks.

Specifically, a `1:1` ratio will cause the attack **to not be profitable but rather result in a net-zero benefit** which can still be capitalized as a griefing attack. A `1:10` ratio will cause the attack to actively hurt the would-be attacker, thereby acting as a much stronger deterrent against these types of attacks.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and assessed the ramifications of a virtual decimal offset to create downstream complications that would not be trivially resolved.

As such, they consider the basic version (`1:1`) of the **EIP-4626** security mechanism to be adequate for their intents and purposes as clarified in their thorough analysis blog post **located here**.

GenericFactory Manual Review Findings

GFY-01M: Potentially Dangerous Push Ownership Pattern

Type	Severity	Location
Input Sanitization	Informational	GenericFactory.sol:L108-L112

Description:

The referenced code will permit an overwrite of the `upgradeAdmin` in the `GenericFactory`, the only member with administrative privileges, in a push pattern.

Impact:

A mistake during administrator adjustments would result in permanent forfeiture of the `upgradeAdmin` role which we consider an ill-advised trait.

Example:

src/GenericFactory/GenericFactory.sol

SOL

```sol
108 function setUpgradeAdmin(address newUpgradeAdmin) external nonReentrant adminOnly
{
109     if (newUpgradeAdmin == address(0)) revert E_BadAddress();
110     upgradeAdmin = newUpgradeAdmin;
111     emit SetUpgradeAdmin(newUpgradeAdmin);
112 }
```

Recommendation:

We advise a pull pattern to be adopted instead, proposing a new administrator which would then have to explicitly accept administrator-ship via a separate dedicated function.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and clarified that they utilized a push-based ownership pattern in other instances of the codebase in addition to this one, and that they are comfortable with the push-based mechanism.

GFY-02M: Potentially Improper Exposure of Data

Type	Severity	Location
Standard Conformity	Informational	GenericFactory.sol:L22, L30, L118

Description:

The `proxyLookup` mapping is exposed publicly yet will perform mutations when queried via its dedicated getter function due to a potentially outdated data point.

Impact:

Direct invocations of the `GenericFactory::proxyLookup` compiler-generated getter function may yield outdated data points that callers may not be necessarily aware of.

Example:

src/GenericFactory/GenericFactory.sol

SOL

```sol
116 function getProxyConfig(address proxy) external view returns (ProxyConfig memory
config) {
117     config = proxyLookup[proxy];
118     if (config.upgradeable) config.implementation = implementation;
119 }
```

Recommendation:

We advise the `mapping` to not be accessible directly, as it should always be queried via the `GenericFactory::getProxyConfig` function.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `proxyLookup` member has been set as `internal` per our recommendation, ensuring that the data point is correctly fetched via the `GenericFactory::getProxyConfig` function at all times.

Governance Manual Review Findings

GEC-01M: Potentially Weak Validation of Caps

Type	Severity	Location
Input Sanitization	Informational	Governance.sol:L263, L266

Description:

The `Governance::setCaps` function will apply sanitization on the input `supplyCap` and `borrowCap` values, however, it will fail to ensure that non-zero caps result in non-zero representations in the `AmountCap` mantissa system.

Impact:

As the `Governance::setCaps` function is a governor-controlled function, we consider calls reviewed and thus do not anticipate this misconfiguration to manifest in production.

Example:

src/EVault/modules/Governance.sol

SOL

```sol
259 /// @inheritdoc IGovernance
260 function setCaps(uint16 supplyCap, uint16 borrowCap) public virtual nonReentrant governorOnly {
261     AmountCap _supplyCap = AmountCap.wrap(supplyCap);
262     // Max total assets is a sum of max pool size and max total debt, both Assets type
263     if (supplyCap > 0 && _supplyCap.toUint() > 2 * MAX_SANE_AMOUNT) revert E_BadSupplyCap();
264
265     AmountCap _borrowCap = AmountCap.wrap(borrowCap);
266     if (borrowCap > 0 && _borrowCap.toUint() > MAX_SANE_AMOUNT) revert E_BadBorrowCap();
267
268     vaultStorage.supplyCap = _supplyCap;
```

Example (Cont.):

```sol
    vaultStorage.borrowCap = _borrowCap;

    emit GovSetCaps(supplyCap, borrowCap);
}
```

Recommendation:

We advise the code to ensure that the `_supplyCap.toUint()` and `_borrowCap.toUint()` calculations result in a non-zero value when the respective input arguments are non-zero, preventing non-zero exponents with a zero value to be specified in the function.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and clarified that a zero-value mantissa with a non-zero exponent is considered a valid configuration of the cap as `0`, whereas a zero mantissa and zero exponent is considered a valid configuration of the cap as *unlimited*.

Based on this fact and the additional documentation introduced around this mechanism, we consider the original exhibit inapplicable as it describes desirable behaviour.

GEC-02M: Insecure Clearance of LTV

Type	Severity	Location
Logical Fault	Minor	Governance.sol:L225-L230

Description:

The `Governance::clearLTV` function will permit an LTV to be cleared via the `LTVConfigLib::clear` function which will set the `targetTimestamp` to `0`. In turn, this will cause the collateral to no longer be recognized permitting debt socialization as well as immediate liquidations on positions that relied on the cleared collateral.

Impact:

The present `Governance::clearLTV` function cannot be executed without affecting users negatively, and requires a redesign to function as expected.

Example:

src/EVault/modules/Governance.sol

SOL

```sol
224 /// @inheritdoc IGovernance
225 function clearLTV(address collateral) public virtual nonReentrant governorOnly {
226     uint16 originalLTV = getLTV(collateral, LTVType.LIQUIDATION).toUint16();
227     vaultStorage.ltvLookup[collateral].clear();
228
229     emit GovSetLTV(collateral, 0, 0, 0, originalLTV);
230 }
```

Recommendation:

We advise the `Governance::clearLTV` function to be revised, as the governance-based time delay may be insufficient in recovering positions back to a healthy state until the proposal is executed.

As potential remediations, we advise the function to be omitted entirely and the `Governance::setLTV` function to be utilized with an `ltv` of `0`, permitting LTV to gradually lower to `0`. Alternatively, we advise user positions that have a collateral which had its LTV cleared recently to be "immune" to liquidations for a brief window to permit those users to react to the event and re-stabilize their positions.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and clarified that the `GovernanceModule::clearLTV` function is an emergency mechanism that behaves precisely as described deliberately so as to protect against critical security threats that malicious collateral may pose within the EVK.

Documentation was introduced that further clarifies the purpose of the `Governance::clearLTV` function, and as such we consider this exhibit to be inapplicable given that it describes business-requirement aligned code functionality.

Liquidation Manual Review Findings

LNO-01M: Improper Liquidation Incentive Calculations

Type	Severity	Location
Logical Fault	Informational	Liquidation.sol:L158-L161

Description:

The `Liquidation::calculateMaxLiquidation` mechanism will **artificially create bad debt** due to the `discountFactor` for low LTV configurations. Specifically, a liquidate-able position may be able to recover fully with the collateral available yet can be liquidated to force the discounted collateral-attached debt to be socialized.

As an example configuration using simplistic terms:

In the above example, the `discountFactor` will be maximized (i.e. `80%`) and thus the `maxYieldValue` will exceed the available balance of the asset (i.e. `100 USD / 80% = 125 USD`).

As a result, the ensuing `if` conditional will **lower the `maxRepayValue` to `80 USD` even though the position has sufficient collateral to fulfil its liability obligations**.

Impact:

Under certain conditions, the liquidation incentives offered by the `Liquidation::calculateMaxLiquidation` function can be capitalized by liquidators to create bad debt by forcing a position into insolvency even though value-wise the position would be solvent.

Additionally, the present system effectively incentivizes users to liquidate their own collateral via a secondary account thus repaying their position at a discount.

Example:

src/EVault/modules/Liquidation.sol

```sol
function calculateMaxLiquidation(LiquidationCache memory liqCache, VaultCache memory vaultCache)
    private
    view
    returns (LiquidationCache memory)
{
    // Check account health

    (uint256 liquidityCollateralValue, uint256 liquidityLiabilityValue) =
        calculateLiquidity(vaultCache, liqCache.violator, liqCache.collaterals, LTVType.LIQUIDATION);

```

Example (Cont.):

```sol
    // no violation
    if (liquidityCollateralValue > liquidityLiabilityValue) return liqCache;

    // Compute discount

    uint256 discountFactor = liquidityCollateralValue * 1e18 / liquidityLiabilityValue; // health score = 1 - discount

    if (discountFactor < 1e18 - MAXIMUM_LIQUIDATION_DISCOUNT) {
        discountFactor = 1e18 - MAXIMUM_LIQUIDATION_DISCOUNT;
    }

    // Compute maximum yield

    uint256 collateralBalance = IERC20(liqCache.collateral).balanceOf(liqCache.violator);
    uint256 collateralValue =
        vaultCache.oracle.getQuote(collateralBalance, liqCache.collateral, vaultCache.unitOfAccount);

    if (collateralValue == 0) {
        // worthless collateral can be claimed with no repay
        liqCache.yieldBalance = collateralBalance;
        return liqCache;
    }

    uint256 liabilityValue = liqCache.owed.toUint();
    if (address(vaultCache.asset) != vaultCache.unitOfAccount) {
        // liquidation, in contrast to liquidity calculation, uses mid-point pricing instead of bid/ask
        liabilityValue =
            vaultCache.oracle.getQuote(liabilityValue, address(vaultCache.asset), vaultCache.unitOfAccount);
```

Example (Cont.):

```sol
    }

    uint256 maxRepayValue = liabilityValue;
    uint256 maxYieldValue = maxRepayValue * 1e18 / discountFactor;

    // Limit yield to borrower's available collateral, and reduce repay if necessary
    // This can happen when borrower has multiple collaterals and seizing all of this one won't bring the violator back to solvency

    if (collateralValue < maxYieldValue) {
        maxRepayValue = collateralValue * discountFactor / 1e18;
        maxYieldValue = collateralValue;
    }

    liqCache.repay = (maxRepayValue * liqCache.owed.toUint() / liabilityValue).toAssets();
    liqCache.yieldBalance = maxYieldValue * collateralBalance / collateralValue;

    return liqCache;
}
```

Recommendation:

We advise the `discountFactor` to solely be applied **if the `collateralValue` exceeds the `liabilityValue`** as otherwise toxic debt will be created at the expense of the system.

To note, the documentation line that precedes the referenced `if` conditional **should be corrected** as the conditional can trigger under the aforementioned conditions which do not fall under the "insolvency" category.

Alleviation (fb2dd77a6f):

The Euler Finance team evaluated this exhibit and has opted to acknowledge it as an inherent economical risk to lending protocols. Additionally, the Euler Finance team maintained that the proposed mitigation would cause toxic debt to be further exacerbated as no one would desire to liquidate it.

To clarify the exhibit's original intentions, the proposed mechanism is to offer a discount on excess `collateralValue` that exceeds the `liabilityValue` **solely when that case is applicable**. Specifically, we propose that a position that is able to be liquidated while recovering fully should offer a discount **on the extra collateral the user holds**.

Toxic liquidations (i.e. under-collateralized positions) should retain their present behaviour and impose a discount unconditionally, as at that point toxic debt is unavoidable. The exhibit's focus is toxic debt **that is avoidable without an expense to the protocol, simply by reducing the incentive of the liquidator rather than eliminating it**.

We invite the Euler Finance team to re-visit this exhibit, as well as adjust the documentation line outlined in the last sentence of the "Recommendation" chapter regardless of their action.

Alleviation (0f2192ac81):

We engaged in extensive discussions with the Euler Finance team debating the proposed liquidation incentive mechanism, the outputs of which have been reflected in the **Dutch Liquidation Analysis** authored by the Euler Finance team.

In summary, we consider the proposed mechanism sufficiently deliberated within the paper and we are aligned with all criticism levied; especially in relation to partial liquidations and their impact on the overall game-theory mechanics that surround liquidations.

As a result of these efforts, we consider this exhibit to be informational in nature and to be nullified as the original incentive model employed by the Euler Finance team does not lead to any active vulnerability and has been considered the optimal approach based on the business requirements of the Euler Finance

team.

MetaProxyDeployer Manual Review Findings

MPD-01M: Inexistent Allocation of Memory

Type	Severity	Location
Language Specific	Informational	MetaProxyDeployer.sol:L16

Description:

The `MetaProxyDeployer` contract will load the free memory pointer to start constructing its deployment payload, however, the code will never offset the free memory pointer to properly indicate the memory space has been allocated.

Impact:

The memory space utilized by the `MetaProxyDeployer` contract for the proxy's bytecode is never indicated as reserved in the free memory pointer, indicating non-standard usage of memory.

Example:

src/GenericFactory/MetaProxyDeployer.sol

SOL

```sol
8   /// @dev Creates a proxy for `targetContract` with metadata from `metadata`. Code modified from EIP-3448 reference implementation: https://eips.ethereum.org/EIPS/eip-3448
9   /// @return addr A non-zero address if successful.
10  function deployMetaProxy(address targetContract, bytes memory metadata) internal returns (address addr) {
11      // the following assembly code (init code + contract code) constructs a metaproxy.
12      assembly {
13          let offset := add(metadata, 32)
14          let length := mload(metadata)
15          // load free memory pointer as per solidity convention
16          let start := mload(64)
17          // keep a copy
```

Example (Cont.):

SOL

```
18          let ptr := start
19          // deploy code (11 bytes) + first part of the proxy (21 bytes)
20          mstore(ptr,
0x600b380380600b3d393df3363d3d373d3d3d3d60368038038091363936013d73)
21          ptr := add(ptr, 32)
22
23          // store the address of the contract to be called
24          mstore(ptr, shl(96, targetContract))
25          // 20 bytes
26          ptr := add(ptr, 20)
27
28          // the remaining proxy code...
29          mstore(ptr,
0x5af43d3d93803e603457fd5bf3000000000000000000000000000000000000000000)
30          // ...13 bytes
31          ptr := add(ptr, 13)
32
33          // copy the metadata
34          {
35              for { let i := 0 } lt(i, length) { i := add(i, 32) } {
mstore(add(ptr, i), mload(add(offset, i))) }
36          }
37          ptr := add(ptr, length)
38
39          // The size is deploy code + contract code + calldatasize - 4.
40          addr := create(0, start, sub(ptr, start))
41      }
42  }
```

Recommendation:

We advise the free memory pointer to be updated properly in adherence to Solidity conventions.

To note, a vulnerability does not manifest in the present code as the `ptr` variable that would be corrupted is atomically utilized within the `MetaProxyDeployer::deployMetaProxy`. Despite of this, we still advise the memory pointer to be updated accordingly to prevent any issues arising in future updates of the code.

Alleviation (fb2dd77a6f):

The Euler Finance team opted to acknowledge this exhibit based on the fact that they have copied their code from the reference code of the **EIP-3448** page.

While we understand the exhibit's acknowledgement, we would like to clarify that an EIP's example implementation is not meant to be secure but rather illustrate how the code should function.

Alleviation (0f2192ac81):

The Euler Finance team opted to create their own `MetaProxyDeployer` implementation based on the definition of the **EIP-3448** standard, significantly increasing the legibility of the implementation.

As the code no longer reserves memory from the free memory pointer, we consider this exhibit no longer applicable.

MPD-02M: Inexistent Appendment of Metadata Length (EIP-3448 Discrepancy)

Type	Severity	Location
Standard Conformity	Informational	MetaProxyDeployer.sol:L37

Description:

Per the **EIP-3448** standard definition as well as reference implementation, the trailing 32 bytes of the contract's bytecode MUST (RFC-2119) indicate the length of the metadata.

In the bytecode constructed by the `MetaProxyDeployer`, no size is appended at the end of the bytecode thereby generating code that is non-compliant with the **EIP-3448** standard.

Impact:

The `MetaProxyDeployer` contract generates bytecode that is not compliant with the **EIP-3448** standard.

As this discrepancy is acknowledged in the contract's documentation, we consider the exhibit to be informational. To note, we still advise the size of the data to be appended as the **EIP-3448** terminology in use is presently misleading given that the **EIP-3448** standard is closely correlated to the appendment of the metadata size per its MUST RFC-2119 terminology.

Example:

src/GenericFactory/MetaProxyDeployer.sol

SOL

```
33  // copy the metadata
34  {
35      for { let i := 0 } lt(i, length) { i := add(i, 32) } { mstore(add(ptr, i),
mload(add(offset, i))) }
36  }
37  ptr := add(ptr, length)
38
39  // The size is deploy code + contract code + calldatasize - 4.
40  addr := create(0, start, sub(ptr, start))
```

Recommendation:

We advise the `ptr` to be written to after being offset by the metadata length to store the `length` value itself, ensuring that the generated bytecode is compliant with the **EIP-3448** standard.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team clarified that the contract is meant to be inspired by the **EIP-3448** standard rather than adhering to it, and performed documentational adjustments to imply as such.

Based on these changes, we consider this exhibit to be properly addressed as the contract's purpose is no longer up to interpretation.

MPD-03M: Inexistent Handling of Creation Failure

Type	Severity	Location
Language Specific	Informational	MetaProxyDeployer.sol:L40

Description:

The `create` EVM operation code may yield the zero-address in case of failure, however, this is not explicitly handled by the `MetaProxyDeployer::deployMetaProxy` function nor the `GenericFactory::createProxy` function it is invoked in.

Impact:

Although unlikely, any failure of the `MetaProxyDeployer::deployMetaProxy` function will not be explicitly handled and will instead result in a failure implicitly in the `GenericFactory::createProxy` function and specifically the `IComponent::initialize` call it performs.

As the code is secure as-is, a severity of informational was deemed appropriate.

Example:

src/GenericFactory/MetaProxyDeployer.sol

SOL

```sol
8   /// @dev Creates a proxy for `targetContract` with metadata from `metadata`. Code modified from EIP-3448 reference implementation: https://eips.ethereum.org/EIPS/eip-3448
9   /// @return addr A non-zero address if successful.
10  function deployMetaProxy(address targetContract, bytes memory metadata) internal returns (address addr) {
11      // the following assembly code (init code + contract code) constructs a metaproxy.
12      assembly {
13          let offset := add(metadata, 32)
14          let length := mload(metadata)
15          // load free memory pointer as per solidity convention
16          let start := mload(64)
17          // keep a copy
```

Example (Cont.):

SOL

```
18          let ptr := start
19          // deploy code (11 bytes) + first part of the proxy (21 bytes)
20          mstore(ptr,
0x600b380380600b3d393df3363d3d373d3d3d3d60368038038091363936013d73)
21          ptr := add(ptr, 32)
22
23          // store the address of the contract to be called
24          mstore(ptr, shl(96, targetContract))
25          // 20 bytes
26          ptr := add(ptr, 20)
27
28          // the remaining proxy code...
29          mstore(ptr,
0x5af43d3d93803e603457fd5bf3000000000000000000000000000000000000000)
30          // ...13 bytes
31          ptr := add(ptr, 13)
32
33          // copy the metadata
34          {
35              for { let i := 0 } lt(i, length) { i := add(i, 32) } {
mstore(add(ptr, i), mload(add(offset, i))) }
36          }
37          ptr := add(ptr, length)
38
39          // The size is deploy code + contract code + calldatasize - 4.
40          addr := create(0, start, sub(ptr, start))
41      }
42  }
```

Recommendation:

Given that the `GenericFactory::createProxy` function expects the `BeaconProxy` failure to lead to a `revert`, we advise the `MetaProxyDeployer::deployMetaProxy` function to `revert` as well if the yielded address `iszero`.

Alleviation (fb2dd77a6f):

The Euler Finance team opted to acknowledge this exhibit based on the fact that they have copied their code from the reference code of the **EIP-3448** page.

While we understand the exhibit's acknowledgement, we would like to clarify that an EIP's example implementation is not meant to be secure but rather illustrate how the code should function.

Alleviation (0f2192ac81):

The Euler Finance team opted to create their own `MetaProxyDeployer` implementation based on the definition of the **EIP-3448** standard, significantly increasing the legibility of the implementation.

The latest `MetaProxyDeployer::deployMetaProxy` function implementation will properly validate that the contract was deployed by ensuring the `addr` is non-zero, alleviating this exhibit.

PegStabilityModule Manual Review Findings

PSM-01M: Inexistent Sanitization of Fees

Type	Severity	Location
Input Sanitization	Informational	PegStabilityModule.sol:L26, L27

Description:

The `TO_UNDERLYING_FEE` and `TO_SYNTH_FEE` fees must always be less than the `BPS_SCALE`, however, the **`PegStabilityModule::constructor`** does not presently sanitize them.

Impact:

The contract can always be redeployed if misconfigured, rendering this exhibit to be informational in nature.

Example:

src/Synths/PegStabilityModule.sol

SOL

```sol
21  constructor(address _evc, address _synth, address _underlying, uint256
toUnderlyingFeeBPS, uint256 toSynthFeeBPS)
22      EVCUtil(IEVC(_evc))
23  {
24      synth = ESynth(_synth);
25      underlying = IERC20(_underlying);
26      TO_UNDERLYING_FEE = toUnderlyingFeeBPS;
27      TO_SYNTH_FEE = toSynthFeeBPS;
28  }
```

Recommendation:

We advise them to be properly sanitized, ensuring that the calculations of the `PegStabilityModule::quoteTo`-prefixed functions will be properly performed.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced fees are adequately sanitized via `if-revert` pattern checks introduced to the `PegStabilityModule::constructor`, addressing this exhibit.

PSM-02M: Inexistent Reversion of Rounding Direction

Type	Severity	Location
Logical Fault	Minor	PegStabilityModule.sol:L70-L72, L78-L80

Description:

The `PegStabilityModule::quoteTo`-prefixed functions will always round towards `0` which is incorrect for the output-to-input conversion mechanisms which should instead round upwards.

Impact:

The accounting error that presently exists in the `PegStabilityModule` may result in discrepant output amounts for the same input amount depending on which conversion avenue was utilized and the amounts involved.

Example:

src/Synths/PegStabilityModule.sol

SOL

```sol
66  function quoteToUnderlyingGivenIn(uint256 amountIn) public view returns (uint256)
{
67      return amountIn * (BPS_SCALE - TO_UNDERLYING_FEE) / BPS_SCALE;
68  }
69
70  function quoteToUnderlyingGivenOut(uint256 amountOut) public view returns
(uint256) {
71      return amountOut * BPS_SCALE / (BPS_SCALE - TO_UNDERLYING_FEE);
72  }
73
74  function quoteToSynthGivenIn(uint256 amountIn) public view returns (uint256) {
75      return amountIn * (BPS_SCALE - TO_SYNTH_FEE) / BPS_SCALE;
```

Example (Cont.):

```sol
}

function quoteToSynthGivenOut(uint256 amountOut) public view returns (uint256) {
    return amountOut * BPS_SCALE / (BPS_SCALE - TO_SYNTH_FEE);
}
```

Recommendation:

We advise the output-to-input conversion mechanisms to round in the appropriate direction, ensuring that a user cannot supply less input than normally expected for a particular output.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and opted to acknowledge it as they believe that inverse rounding between the input and output conversion mechanisms would confuse integrators.

RiskManager Manual Review Findings

RMR-01M: Overprotective Security Mechanism

Type	Severity	Location
Logical Fault	Minor	RiskManager.sol:L105

Description:

The `RiskManager::checkVaultStatus` function which is invoked as part of the Ethereum Vault Connector's execution flow after all interactions with the vault have concluded will ensure that the `borrowCap` has not been breached.

A problem that arises from the current security mechanism is the fact that **the entire vault will be inoperable** if the total borrows have naturally exceeded the permitted `borrowCap`. For example, a user with collateral but no debt will not be able to withdraw their collateral which is a restriction that should not be imposed.

Impact:

In the extreme circumstance that the `borrowCap` has been reached due to natural interest accrual, the vault will be inoperable in functions that do not otherwise deal with borrow balances when it shouldn't be so.

Example:

src/EVault/modules/RiskManager.sol

SOL

```sol
84  /// @inheritdoc IRiskManager
85  /// @dev See comment about re-entrancy for `checkAccountStatus`
86  function checkVaultStatus() public virtual reentrantOK onlyEVCChecks returns
(bytes4 magicValue) {
87      // Use the updating variant to make sure interest is accrued in storage
before the interest rate update
88      VaultCache memory vaultCache = updateVault();
89      uint256 newInterestRate = computeInterestRate(vaultCache);
90
91      logVaultStatus(vaultCache, newInterestRate);
92
93      // We use the snapshot to check if the borrows or supply grew, and if so then
we check the borrow and supply caps.
```

Example (Cont.):

```sol
        // If snapshot is initialized, then caps are configured.
        // If caps are set in the middle of a batch, then snapshots represent the state of the vault at that time.
        if (vaultCache.snapshotInitialized) {
            vaultStorage.snapshotInitialized = vaultCache.snapshotInitialized = false;

            Assets snapshotCash = snapshot.cash;
            Assets snapshotBorrows = snapshot.borrows;

            uint256 prevBorrows = snapshotBorrows.toUint();
            uint256 borrows = vaultCache.totalBorrows.toAssetsUp().toUint();

            if (borrows > vaultCache.borrowCap && borrows > prevBorrows) revert E_BorrowCapExceeded();

            uint256 prevSupply = snapshotCash.toUint() + prevBorrows;
            uint256 supply = totalAssetsInternal(vaultCache);

            if (supply > vaultCache.supplyCap && supply > prevSupply) revert E_SupplyCapExceeded();

            snapshot.reset();
        }

        magicValue = IEVCVault.checkVaultStatus.selector;
    }
```

Recommendation:

We advise the `borrowCap` limitation specifically to be applied solely for operations that **actively increase the cap** rather than for natural interest rate increases, ensuring that users can still interact with the vault albeit at a limited capacity when the `borrowCap` has been breached.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and concluded that it is incorrect as the snapshot being evaluated will have reflected any borrow interest accrual in the `prevBorrows` variable, effectively causing the `if` clause to **not trigger if the borrows were not increased as part of the overall user interaction**.

As such, a user will continue to be able to withdraw even when the borrow has exceeded the cap rendering this exhibit invalid.

BalanceForwarder Code Style Findings

BFR-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	BalanceForwarder.sol:L27, L29, L30, L40, L42

Description:

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

Example:

src/EVault/modules/BalanceForwarder.sol

SOL

```sol
22  /// @inheritdoc IBalanceForwarder
23  function enableBalanceForwarder() public virtual reentrantOK {
24      if (address(balanceTracker) == address(0)) revert
E_BalanceForwarderUnsupported();
25
26      address account = EVCAuthenticate();
27      bool wasBalanceForwarderEnabled =
vaultStorage.users[account].isBalanceForwarderEnabled();
28
29      vaultStorage.users[account].setBalanceForwarder(true);
30      balanceTracker.balanceTrackerHook(account,
vaultStorage.users[account].getBalance().toUint(), false);
31
```

Example (Cont.):

```sol
        if (!wasBalanceForwarderEnabled) emit BalanceForwarderStatus(account, true);
    }

    /// @inheritdoc IBalanceForwarder
    function disableBalanceForwarder() public virtual reentrantOK {
        if (address(balanceTracker) == address(0)) revert E_BalanceForwarderUnsupported();

        address account = EVCAuthenticate();
        bool wasBalanceForwarderEnabled = vaultStorage.users[account].isBalanceForwarderEnabled();

        vaultStorage.users[account].setBalanceForwarder(false);
        balanceTracker.balanceTrackerHook(account, 0, false);

        if (wasBalanceForwarderEnabled) emit BalanceForwarderStatus(account, false);
    }
```

Recommendation:

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

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

BalanceUtils Code Style Findings

BUL-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	BalanceUtils.sol:L27, L30, L49, L57, L71, L74, L84, L85, L111, L117

Description:

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

Example:

src/EVault/shared/BalanceUtils.sol

SOL

```sol
18  function increaseBalance(
19      VaultCache memory vaultCache,
20      address account,
21      address sender,
22      Shares amount,
23      Assets assets
24  ) internal {
25      if (account == address(0)) revert E_BadSharesReceiver();
26
27      (Shares origBalance, bool balanceForwarderEnabled) =
vaultStorage.users[account].getBalanceAndBalanceForwarder();
```

Example (Cont.):

```sol
        Shares newBalance = origBalance + amount;

        vaultStorage.users[account].setBalance(newBalance);
        vaultStorage.totalShares = vaultCache.totalShares = vaultCache.totalShares +
amount;

        if (balanceForwarderEnabled) {
            tryBalanceTrackerHook(account, newBalance.toUint(), false);
        }

        emit Transfer(address(0), account, amount.toUint());
        emit Deposit(sender, account, assets.toUint(), amount.toUint());
    }
```

Recommendation:

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

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

Alleviation (fb2dd77a6f):

The optimization has been partially applied as only the example referenced in the exhibit has been optimized. We advise all referenced instances by the exhibit (i.e. the `Location` lines) to be addressed, optimizing the codebase further.

Alleviation (0f2192ac81):

The optimization has been properly applied to all remaining instances rendering it addressed in full.

BUL-02C: Inexistent Propagation of Optimized Arithmetics

Type	Severity	Location
Gas Optimization	Informational	BalanceUtils.sol:L53-L55, L79-L81

Description:

The referenced `unchecked` code blocks will not properly propagate to the overridden `-` functions thereby causing the syntax to be ineffective.

Example:

src/EVault/shared/BalanceUtils.sol

SOL

```sol
52  Shares newBalance;
53  unchecked {
54      newBalance = origBalance - amount;
55  }
```

Recommendation:

We advise the syntax to be omitted from the `BalanceUtils` contract, and the `unchecked` code block to be introduced to the subtraction itself if needed.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced `unchecked` code blocks have been omitted, and the calculations are now performed utilizing the newly introduced **`SharesLib::subUnchecked`** function that will properly perform the calculations in an `unchecked` code block.

BeaconProxy Code Style Findings

BPY-01C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	BeaconProxy.sol:L11, L40

Description:

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

Example:

src/GenericFactory/BeaconProxy.sol

SOL

```sol
11  uint256 constant MAX_TRAILING_DATA_LENGTH = 128;
```

Recommendation:

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

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The statement within the **`BeaconProxy::constructor`** properly utilizes the `MAX_TRAILING_DATA_LENGTH` constant that already existed, optimizing the maintainability and legibility of the codebase.

BPY-02C: Unconditional Appendment of Metadata

Type	Severity	Location
Gas Optimization	Informational	BeaconProxy.sol:L65-L68

Description:

The `BeaconProxy::fallback` function will append the four metadata slots of the contract unconditionally to the delegated payload, instead transmitting a subset of it in the `delegatecall` instruction.

Example:

src/GenericFactory/BeaconProxy.sol

SOL

```sol
45  fallback() external payable {
46      address beacon_ = beacon;
47      uint256 metadataLength_ = metadataLength;
48      bytes32 metadata0_ = metadata0;
49      bytes32 metadata1_ = metadata1;
50      bytes32 metadata2_ = metadata2;
51      bytes32 metadata3_ = metadata3;
52
53      assembly {
54          // Fetch implementation address from the beacon
```

Example (Cont.):

SOL

```
        mstore(0, IMPLEMENTATION_SELECTOR)
        let result := staticcall(gas(), beacon_, 0, 4, 0, 32)
        if iszero(result) {
            returndatacopy(0, 0, returndatasize())
            revert(0, returndatasize())
        }
        let implementation := mload(0)

        // delegatecall to the implementation with trailing metadata
        calldatacopy(0, 0, calldatasize())
        mstore(calldatasize(), metadata0_)
        mstore(add(32, calldatasize()), metadata1_)
        mstore(add(64, calldatasize()), metadata2_)
        mstore(add(96, calldatasize()), metadata3_)
        result := delegatecall(gas(), implementation, 0, add(metadataLength_,
calldatasize()), 0, 0)
        returndatacopy(0, 0, returndatasize())

        switch result
        case 0 { revert(0, returndatasize()) }
        default { return(0, returndatasize()) }
    }
}
```

Recommendation:

As the variables involved are `immutable`, we advise a dedicated `BeaconProxy` implementation per slots required to be devised which would optimize the gas cost involved in relaying all payloads to the proxy.

Alleviation (fb2dd77a6f):

The following alleviation chapter is misconceived as the `immutable` aspect of the variables was ignored. Kindly consult the next alleviation chapter for an up-to-date evaluation of the optimization proposed.

The Euler Finance team evaluated this exhibit and specified that they do not envision the gas savings to justify the legibility decrease.

The gas savings acquired by the recommended optimization are significant, and will reduce the number of `SLOAD` operations by the number of unused metadata elements in addition to optimizing the function's overall memory usage.

The in-memory statically sized array recommendation advised as "potential" is inadequate, however, the `for` loop based `mstore` operations that are advised would correctly reflect the aforementioned reduction in gas.

As a final clarification and as the exhibit's original description details, the code appends the four metadata slots of the contract unconditionally to the delegated payload, instead transmitting a subset of it in the `delegatecall` instruction.

In other words, the memory location from `0` onwards will redundantly include the `metadataX_` items that are ultimately not transmitted by the `delegatecall` instruction.

We advise the exhibit's optimization to be revisited based on the elaborations we have supplied.

Alleviation (0f2192ac81):

Our original recommendation was improperly defined as it assumed that each variable was located in `storage` rather than being `immutable`.

All `immutable` variables are inaccessible in `assembly` blocks rendering an optimization under the same implementation impossible to achieve.

We proposed a dedicated `BeaconProxy` implementation to be introduced instead per number of `immutable` slots required which would permit the appendment of metadata to be optimally performed.

The Euler Finance team evaluated the gas savings that stem from such an approach, and did not consider them to outweigh the operational burden of maintaining multiple implementations and properly deploying each one.

As such, we consider this exhibit properly acknowledged.

BorrowUtils Code Style Findings

BUS-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	BorrowUtils.sol:L65, L77, L160, L164

Description:

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

Example:

src/EVault/shared/BorrowUtils.sol

SOL

```sol
159 if (owed > prevOwed) {
160     uint256 change = (owed.toAssetsUp() - prevOwed.toAssetsUp()).toUint();
161     emit Borrow(account, change);
162     DToken(dTokenAddress).emitTransfer(address(0), account, change);
163 } else if (prevOwed > owed) {
164     uint256 change = (prevOwed.toAssetsUp() - owed.toAssetsUp()).toUint();
165
166     emit Repay(account, change);
167     DToken(dTokenAddress).emitTransfer(account, address(0), change);
168 }
```

Recommendation:

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

Alleviation (fb2dd77a6f):

The optimization has been partially applied as only the example referenced in the exhibit has been optimized. We advise all referenced instances by the exhibit (i.e. the `Location` lines) to be addressed, optimizing the codebase further.

Alleviation (0f2192ac81):

The optimization proposed has been applied to the remaining instances, rendering this exhibit fully addressed.

BUS-02C: Inefficient Update of User's Debt

Type	Severity	Location
Gas Optimization	Informational	BorrowUtils.sol:L36, L46, L63

Description:

The `BorrowUtils::increaseBorrow` function will inefficiently adjust the `vaultStorage.users[account]` data location twice redundantly, once when synchronizing the user's borrow position and once when incrementing it.

Example:

src/EVault/shared/BorrowUtils.sol

SOL

```
29  function updateUserBorrow(VaultCache memory vaultCache, address account)
30      private
31      returns (Owed newOwed, Owed prevOwed)
32  {
33      prevOwed = vaultStorage.users[account].getOwed();
34      newOwed = getCurrentOwed(vaultCache, account, prevOwed);
35
36      vaultStorage.users[account].setOwed(newOwed);
37      vaultStorage.users[account].interestAccumulator =
vaultCache.interestAccumulator;
38  }
```

Example (Cont.):

```sol

function increaseBorrow(VaultCache memory vaultCache, address account, Assets assets) internal {
    (Owed owed, Owed prevOwed) = updateUserBorrow(vaultCache, account);

    Owed amount = assets.toOwed();
    owed = owed + amount;

    vaultStorage.users[account].setOwed(owed);
    vaultStorage.totalBorrows = vaultCache.totalBorrows = vaultCache.totalBorrows + amount;

    logBorrowChange(account, prevOwed, owed);
}
```

Recommendation:

We advise the `BorrowUtils::updateUserBorrow` function to accept an `Owed` / `uint256` argument that indicates whether the `newOwed` value should be incremented and by how much, optimizing the code's gas cost by one warm `SSTORE` operation.

A similar optimization can be applied for the `BorrowUtils::decreaseBorrow` function albeit with different operations based on an `Assets` instead of `Owed` argument.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The code has been optimized per our recommendation, splitting the `BorrowUtils::updateUserBorrow` function into the `BorrowUtils::loadUserBorrow` and `BorrowUtils::setUserBorrow` functions and thus permitting them to be invoked separately in an optimal way.

BUS-03C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	BorrowUtils.sol:L33, L36, L37

Description:

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

Example:

src/EVault/shared/BorrowUtils.sol

SOL

```sol
function updateUserBorrow(VaultCache memory vaultCache, address account)
    private
    returns (Owed newOwed, Owed prevOwed)
{
    prevOwed = vaultStorage.users[account].getOwed();
    newOwed = getCurrentOwed(vaultCache, account, prevOwed);

    vaultStorage.users[account].setOwed(newOwed);
    vaultStorage.users[account].interestAccumulator = vaultCache.interestAccumulator;
}
```

Recommendation:

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

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The exhibit's recommendation has been applied to the partial version of the original function under `BorrowUtils::setUserBorrow`, rendering the exhibit alleviated.

Cache Code Style Findings

CEH-01C: Inconsistent Unwrapping Mechanism

Type	Severity	Location
Code Style	Informational	Cache.sol:L94, L101, L111

Description:

The `Shares` data type is converted to its `uint256` representation either via the dedicated **`SharesLib::toUint`** function or via the native `Shares::unwrap` function.

Example:

src/EVault/shared/Cache.sol

SOL

```sol
94  uint256 newTotalShares = vaultCache.totalShares.toUint();
95  uint256 feeAssets =
96      interestFee.mulDiv(newTotalBorrows - vaultCache.totalBorrows.toUint(), 1 <<
INTERNAL_DEBT_PRECISION);
97
98  if (feeAssets != 0) {
99      uint256 newTotalAssets = vaultCache.cash.toUint() + (newTotalBorrows >>
INTERNAL_DEBT_PRECISION);
100     newTotalShares = newTotalAssets * newTotalShares / (newTotalAssets -
feeAssets);
101     newAccumulatedFees += newTotalShares - vaultCache.totalShares.toUint();
102 }
103
```

Example (Cont.):

```sol
// Store new values in vaultCache, only if no overflows will occur. Fees are not larger than total shares, since they are included in them.

if (newTotalShares <= MAX_SANE_AMOUNT && newTotalBorrows <= MAX_SANE_DEBT_AMOUNT) {
    vaultCache.totalBorrows = newTotalBorrows.toOwed();
    vaultCache.interestAccumulator = newInterestAccumulator;
    vaultCache.lastInterestAccumulatorUpdate = uint48(block.timestamp);

    if (newTotalShares != Shares.unwrap(vaultCache.totalShares)) {
```

Recommendation:

We advise a uniform approach to be utilized, ensuring consistency in the codebase as the result of those functions is identical.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `Shares::unwrap` type-based function has been replaced by the `SharesLib::toUint` function, ensuring consistency across the contract as advised.

ConfigAmount Code Style Findings

CAT-01C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	ConfigAmount.sol:L18, L25, L32, L45

Description:

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

Example:

src/EVault/shared/types/ConfigAmount.sol

```sol
18  return uint256(self.toUint16()) * multiplier / (1e4 * divisor);
```

Recommendation:

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

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced repetitive value literal `1e4` has been declared as a `constant` at the `Constants` contract under the label `CONFIG_SCALE`, optimizing the legibility of the codebase.

Dispatch Code Style Findings

DHC-01C: Redundant Copy of Call Data

Type	Severity	Location
Gas Optimization	Informational	Dispatch.sol:L119

Description:

The referenced assembly statement will redundantly copy `PROXY_METADATA_LENGTH` bytes from the `calldata` that will ultimately not be relayed by the `staticcall` operation.

Example:

src/EVault/Dispatch.sol

SOL

```
112 function delegateToModuleView(address module) private view {
113     assembly {
114         // Construct optimized custom call data for `this.viewDelegate()`
115         // [selector 4B][module address 32B][calldata with stripped proxy metadata]
116         // Proxy metadata will be appended back by the proxy on staticcall
117         mstore(0, 0x1fe8b953000000000000000000000000000000000000000000000000000000000000000000)
118         mstore(4, module)
119         calldatacopy(36, 0, calldatasize())
120         // insize: calldatasize + 36 (signature and address) - proxy metadata size
121         let result := staticcall(gas(), address(), 0, sub(add(calldatasize(), 36), PROXY_METADATA_LENGTH), 0, 0)
```

Example (Cont.):

```sol
        returndatacopy(0, 0, returndatasize())
        switch result
        case 0 { revert(0, returndatasize()) }
        default { return(0, returndatasize()) }
    }
}
```

Recommendation:

We advise the `calldatacopy` operation to subtract the `PROXY_METADATA_LENGTH` from the `calldatasize()` result, optimizing the code's gas cost.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The redundant `calldatacopy` operation has been optimized per our recommendation, reducing the code's gas cost and preventing potentially dirty bits from being read.

ESynth Code Style Findings

ESH-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	ESynth.sol:L73

Description:

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

Example:

src/Synths/ESynth.sol

SOL

```sol
73  minterCache.minted = minterCache.minted > amount ? minterCache.minted -
uint128(amount) : 0; // down-casting is safe because amount < minted <= max uint128
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

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

ESH-02C: Inefficient Loop Limit Evaluation

Type	Severity	Location
Gas Optimization	Informational	ESynth.sol:L138

Description:

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

Example:

src/Synths/ESynth.sol

SOL

```sol
138 for (uint256 i = 0; i < ignoredForTotalSupply.length(); i++) {
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced loop limit evaluation (`ignoredForTotalSupply.length()`) has been properly cached outside the `for` loop to a dedicated local variable `ignoredLength` that is consequently utilized, optimizing the loop's gas cost significantly.

ESH-03C: Inefficient Structure Mutability Specifiers

Type	Severity	Location
Gas Optimization	Informational	ESynth.sol:L45, L55, L65, L74

Description:

The `MinterData` structures that are loaded in the **`ESynth::mint`** and **`ESynth::burn`** functions are declared as `storage` yet they are re-written to their respective `mapping` slot after being mutated.

Example:

src/Synths/ESynth.sol

SOL

```sol
/// @notice Mints a certain amount of tokens to the account.
/// @param account The account to mint the tokens to.
/// @param amount The amount of tokens to mint.
function mint(address account, uint256 amount) external nonReentrant {
    address sender = _msgSender();
    MinterData storage minterCache = minters[sender];

    if (
        amount > type(uint128).max - minterCache.minted
            || minterCache.capacity < uint256(minterCache.minted) + amount
```

Example (Cont.):

SOL

```sol
        ) {
            revert E_CapacityReached();
        }

        minterCache.minted += uint128(amount); // safe to down-cast because amount <= capacity <= max uint128
        minters[sender] = minterCache;

        _mint(account, amount);
    }

    /// @notice Burns a certain amount of tokens from the accounts balance. Requires the account, except the owner to have an allowance for the sender.
    /// @param account The account to burn the tokens from.
    /// @param amount The amount of tokens to burn.
    function burn(address account, uint256 amount) external nonReentrant {
        address sender = _msgSender();
        MinterData storage minterCache = minters[sender];

        // The allowance check should be performed if the spender is not the account with the exception of the owner burning from this contract.
        if (account != sender && !(account == address(this) && sender == owner())) {
            _spendAllowance(account, sender, amount);
        }

        // If burning more than minted, reset minted to 0
        minterCache.minted = minterCache.minted > amount ? minterCache.minted - uint128(amount) : 0; // down-casting is safe because amount < minted <= max uint128
        minters[sender] = minterCache;

        _burn(account, amount);
    }
```

Recommendation:

We advise both variables to be set as `memory`, optimizing the gas cost of each function to one `SLOAD` and one `SSTORE` operation.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The data locations of the two referenced local `MinterData` declarations have been properly updated to `memory`, optimizing each function's gas cost.

ESH-04C: Loop Iterator Optimization

Type	Severity	Location
Gas Optimization	Informational	ESynth.sol:L138

Description:

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

Example:

src/Synths/ESynth.sol

SOL

```sol
138 for (uint256 i = 0; i < ignoredForTotalSupply.length(); 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 (fb2dd77a6f):

The recommendation has been indirectly addressed due to the alleviation of exhibit **`ESH-02C`** and thus the optimizer's application.

To note, the original loop iterator was not optimized as the `ignoredForTotalSupply.length()` statement was considered dynamic and thus the optimizer did not assume the increment would be securely performed.

Alleviation (0f2192ac81):

After further testing of the proposed optimization, we concluded that the built-in optimizations of the Solidity compiler the project uses (`0.8.23`) would have covered the iterator case described in the exhibit per the **official requirements of the optimization**.

As such, we consider the optimization nullified as it would not have resulted in a gas reduction in its original form.

GenericFactory Code Style Findings

GFY-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	GenericFactory.sol:L26

Description:

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

Example:

src/GenericFactory/GenericFactory.sol

SOL

```sol
26  uint256 private reentrancyLock;
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit, opting to accept it as valid and acknowledge it.

GFY-02C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	GenericFactory.sol:L133, L134, L135

Description:

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

Example:

src/GenericFactory/GenericFactory.sol

SOL

```sol
129 function getProxyListSlice(uint256 start, uint256 end) external view returns
(address[] memory list) {
130     if (end == type(uint256).max) end = proxyList.length;
131     if (end < start || end > proxyList.length) revert E_BadQuery();
132
133     list = new address[](end - start);
134     for (uint256 i; i < end - start; ++i) {
135         list[i] = proxyList[start + i];
136     }
137 }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit, opting to accept it as valid and acknowledge it.

Governance Code Style Findings

GEC-01C: Inefficient Negation of Conditional

Type	Severity	Location
Gas Optimization	Informational	Governance.sol:L207

Description:

The referenced `if` clause will evaluate a comparative conditional and negate it which is inefficient.

Example:

src/EVault/modules/Governance.sol

SOL

```
207 if (!(newLTVAmount < origLTV.getLTV(LTVType.LIQUIDATION)) && rampDuration > 0)
revert E_LTVRamp();
```

Recommendation:

We advise the negation to be reflected in the comparator itself, adjusting the `<` comparator to `>=` so as to optimize the statement by one negation.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The inefficient negation has been properly assimilated into the conditional itself, optimizing its cost.

GEC-02C: Potentially Misleading Variable Naming

Type	Severity	Location
Code Style	Informational	Governance.sol:L23, L24, L281

Description:

The `GUARANTEED_INTEREST_FEE`-prefixed minimum and maximum values, despite what their names imply, are not guaranteed and may be exceeded or subceeded if the newly configured interest fee is approved by the `IProtocolConfig` implementation.

Example:

src/EVault/modules/Governance.sol

SOL

```sol
275 function setInterestFee(uint16 newInterestFee) public virtual nonReentrant
governorOnly {
276     // Update vault to apply the current interest fee to the pending interest
277     VaultCache memory vaultCache = updateVault();
278     logVaultStatus(vaultCache, vaultStorage.interestRate);
279
280     // Interest fees in guaranteed range are always allowed, otherwise ask
protocolConfig
281     if (newInterestFee < GUARANTEED_INTEREST_FEE_MIN || newInterestFee >
GUARANTEED_INTEREST_FEE_MAX) {
282         if (!protocolConfig.isValidInterestFee(address(this), newInterestFee))
revert E_BadFee();
283     }
284
```

Example (Cont.):

```sol
    vaultStorage.interestFee = newInterestFee.toConfigAmount();

    emit GovSetInterestFee(newInterestFee);
}
```

Recommendation:

We advise the names to be adjusted, properly highlighting that they are the **governor limitations rather than the protocol limitations**.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team acknowledged that identifying a descriptive variable name for the referenced `constant` declarations is difficult, and opted to retain the existing naming convention while introducing annotation as to why they consider it valid.

We concur with the Euler Finance team's assessment, and consider the introduced comments to be sufficient in elaborating on each variable given that they are only internally exposed.

GEC-03C: Redundant Parenthesis Statement

Type	Severity	Location
Code Style	Informational	Governance.sol:L114

Description:

The referenced statement is redundantly wrapped in parenthesis (`()`).

Example:

src/EVault/modules/Governance.sol

SOL

```sol
114 return (vaultStorage.configFlags.toUint32());
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The redundant parenthesis statement has been safely omitted as advised, optimizing the code's legibility.

IRMLinearKink Code Style Findings

IRL-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	IRMLinearKink.sol:L53

Description:

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

Example:

src/InterestRateModels/IRMLinearKink.sol

SOL

```sol
49  if (utilisation <= kink) {
50      ir += utilisation * slope1;
51  } else {
52      ir += kink * slope1;
53      ir += slope2 * (utilisation - kink);
54  }
```

Recommendation:

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced subtraction was optimized securely, performing it within an `unchecked` code block without wrapping the multiplication that it is utilized in.

LTVConfig Code Style Findings

LTC-01C: Inefficient Conditional Exclusivity

Type	Severity	Location
Gas Optimization	Informational	LTVConfig.sol:L42

Description:

The first `if` clause within the **`LTVConfigLib::getLTV`** function is meant to yield the `targetLTV` immediately if it is greater than the `originalLTV`, however, the conditional is inefficient as the equality case is not covered.

Example:

src/EVault/shared/types/LTVConfig.sol

SOL

```sol
40  function getLTV(LTVConfig memory self, LTVType ltvType) internal view returns (ConfigAmount) {
41      if (
42          ltvType == LTVType.BORROWING || block.timestamp >= self.targetTimestamp || self.targetLTV > self.originalLTV
43      ) {
44          return self.targetLTV;
45      }
46
47      uint256 ltv = self.originalLTV.toUint16();
48
49      unchecked {
```

Example (Cont.):

```sol
            uint256 timeElapsed = self.rampDuration - (self.targetTimestamp - block.timestamp);

            // targetLTV < originalLTV and timeElapsed < rampDuration
            ltv = ltv - ((ltv - self.targetLTV.toUint16()) * timeElapsed / self.rampDuration);
        }
        // because ramping happens only when LTV decreases, it's safe to down-cast the new value
        return ConfigAmount.wrap(uint16(ltv));
    }
```

Recommendation:

We advise the code to yield the `targetLTV` if the `targetLTV` is greater-than-or-equal-to the `originalLTV`, optimizing the equality case's gas cost.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The conditional was updated to be inclusive in relation to its LTV comparison, optimizing the code's gas cost.

Liquidation Code Style Findings

LNO-01C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	Liquidation.sol:L129, L130

Description:

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

Example:

src/EVault/modules/Liquidation.sol

SOL

```sol
129 if (discountFactor < 1e18 - MAXIMUM_LIQUIDATION_DISCOUNT) {
```

Recommendation:

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

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit but opted to acknowledge it explicitly as they consider the current representation sufficient.

MetaProxyDeployer Code Style Findings

MPD-01C: Inefficient Memory Assignment

Type	Severity	Location
Gas Optimization	Informational	MetaProxyDeployer.sol:L35

Description:

The referenced `for` loop will iterate from `0` in multiples of `32`, however, the first iteration will inefficiently perform two `add` operations that are unnecessary as the `i` iterator will be `0`.

Example:

src/GenericFactory/MetaProxyDeployer.sol

SOL

```
10  function deployMetaProxy(address targetContract, bytes memory metadata) internal returns (address addr) {
11      // the following assembly code (init code + contract code) constructs a metaproxy.
12      assembly {
13          let offset := add(metadata, 32)
14          let length := mload(metadata)
15          // load free memory pointer as per solidity convention
16          let start := mload(64)
17          // keep a copy
18          let ptr := start
19          // deploy code (11 bytes) + first part of the proxy (21 bytes)
```

Example (Cont.):

```sol
            mstore(ptr,
0x600b380380600b3d393df3363d3d373d3d3d3d60368038038091363936013d73)
            ptr := add(ptr, 32)

            // store the address of the contract to be called
            mstore(ptr, shl(96, targetContract))
            // 20 bytes
            ptr := add(ptr, 20)

            // the remaining proxy code...
            mstore(ptr,
0x5af43d3d93803e603457fd5bf3000000000000000000000000000000000000000000)
            // ...13 bytes
            ptr := add(ptr, 13)

            // copy the metadata
            {
                for { let i := 0 } lt(i, length) { i := add(i, 32) } { mstore(add(ptr,
i), mload(add(offset, i))) }
            }
            ptr := add(ptr, length)

            // The size is deploy code + contract code + calldatasize - 4.
            addr := create(0, start, sub(ptr, start))
        }
    }
```

Recommendation:

We advise the loop and ensuing `ptr` assignment to be wrapped in an `if lt(0, length)` construct, ensuring that the loop and `ptr` offset is executed solely when a non-zero metadata payload has been specified.

Afterwards, the first metadata memory store can occur before the `for` loop and the `for` loop's `i` iterator can begin at `32`, optimizing the code as advised whilst also optimizing the zero-metadata case of the `MetaProxyDeployer::deployMetaProxy` function.

Alleviation (fb2dd77a6f):

The Euler Finance team opted to acknowledge this exhibit based on the fact that they have copied their code from the reference code of the **EIP-3448** page.

While we understand the exhibit's acknowledgement, we would like to clarify that an EIP's example implementation is not meant to be secure but rather illustrate how the code should function.

Alleviation (0f2192ac81):

The Euler Finance team opted to create their own `MetaProxyDeployer` implementation based on the definition of the **EIP-3448** standard, significantly increasing the legibility of the implementation.

The latest version of the code does not perform a low-level `for` loop within an `assembly` block rendering the described optimization inapplicable.

PegStabilityModule Code Style Findings

PSM-01C: Inefficient Re-Calculation of Fee Percentages

Type	Severity	Location
Gas Optimization	Informational	PegStabilityModule.sol:L26, L27, L67, L71, L75, L79

Description:

The referenced statements will constantly perform a known subtraction due to mixing `constant` and `immutable` values.

Example:

src/Synths/PegStabilityModule.sol

SOL

```
66  function quoteToUnderlyingGivenIn(uint256 amountIn) public view returns (uint256)
{
67      return amountIn * (BPS_SCALE - TO_UNDERLYING_FEE) / BPS_SCALE;
68  }
69
70  function quoteToUnderlyingGivenOut(uint256 amountOut) public view returns
(uint256) {
71      return amountOut * BPS_SCALE / (BPS_SCALE - TO_UNDERLYING_FEE);
72  }
73
74  function quoteToSynthGivenIn(uint256 amountIn) public view returns (uint256) {
75      return amountIn * (BPS_SCALE - TO_SYNTH_FEE) / BPS_SCALE;
```

Example (Cont.):

```sol
}

function quoteToSynthGivenOut(uint256 amountOut) public view returns (uint256) {
    return amountOut * BPS_SCALE / (BPS_SCALE - TO_SYNTH_FEE);
}
```

Recommendation:

We advise the code to instead assign the result of the subtraction to the relevant `immutable` variables, optimizing each referenced statement by one subtraction operation.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and opted to acknowledge it as they do not believe the minimal gas savings to justify the introduction of extra code and an `immutable` variable.

ProtocolConfig Code Style Findings

PCG-01C: Optimization of Structure Assignments

Type	Severity	Location
Gas Optimization	Informational	ProtocolConfig.sol:L174-L175, L195-L196

Description:

A structure assignment that is performed entry-by-entry is more optimal than an overwrite, as the overwrite has implicit memory expansion costs due to instantiating a `struct` entry before assigning it to the respective data location.

Example:

src/ProtocolConfig/ProtocolConfig.sol

SOL

```
174 _interestFeeRanges[vault] =
175     InterestFeeRange({exists: exists_, minInterestFee: minInterestFee_,
maxInterestFee: maxInterestFee_});
```

Recommendation:

We advise the referenced assignments to be performed key-by-key, optimizing their overall gas cost.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this exhibit and opted to acknowledge it as the optimization would be observed in infrequently invoked functions and thus was assessed as inconsequential.

PCG-02C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	ProtocolConfig.sol:L137, L151, L172, L193

Description:

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

Example:

src/ProtocolConfig/ProtocolConfig.sol

SOL

```sol
137 if (newProtocolFeeShare > 1e4) revert E_InvalidConfigValue();
```

Recommendation:

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

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The referenced repetitive value literal `1e4` has been declared as a `constant` at the `Constants` contract under the label `CONFIG_SCALE`, optimizing the legibility of the codebase.

PCG-03C: Variable Grouping

Type	Severity	Location
Code Style	Informational	ProtocolConfig.sol:L28-L29, L35-L36

Description:

The referenced variables are meant to represent the default configuration of the `ProtocolFeeConfig`, however, their declarations are separated by multiple variables.

Example:

src/ProtocolConfig/ProtocolConfig.sol

SOL

```sol
28  /// @dev protocol fee receiver, unless a vault has it configured otherwise
29  address public feeReceiver;
30
31  /// @dev min interest fee, except for vaults configured otherwise
32  uint16 internal minInterestFee;
33  /// @dev max interest fee, except for vaults configured otherwise
34  uint16 internal maxInterestFee;
35  /// @dev protocol fee share, except for vaults configured otherwise
36  uint16 internal protocolFeeShare;
```

Recommendation:

We advise them to be located together, optimizing the legibility of the code.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `protocolFeeShare` variable was relocated right after the `feeReceiver`, optimizing the code's legibility.

Snapshot Code Style Findings

STO-01C: Potentially Inefficient Storage Structure

Type	Severity	Location
Code Style	Informational	Snapshot.sol:L16, L23, L28

Description:

The `SnapshotLib` implementation will ensure that at least one bit remains as `1` whenever a `Snapshot` entry is mutated as a form of gas optimization, however, the location of this bit may cause issues if the `Snapshot` structure is expanded.

Specifically, the value of `1 << 31` (i.e. `10000000000000000000000000000000` in binary) will follow any data point in the `Snapshot` structure that precedes the `_stamp` value. The issue that arises from this is that expansions of the `Snapshot` structure cannot be made as any data points introduced would have to accommodate for a potentially dirty most significant bit.

Example:

src/EVault/shared/types/Snapshot.sol

SOL

```sol
9   struct Snapshot {
10      // Packed slot: 14 + 14 + 4 = 32
11      // vault's cash holdings
12      Assets cash;
13      // vault's total borrows in assets, in regular precision
14      Assets borrows;
15      // stamp occupies the rest of the storage slot and makes sure the slot is
non-zero for gas savings
16      uint32 stamp;
17  }
18
```

Example (Cont.):

```sol
/// @title SnapshotLib
/// @author Euler Labs (https://www.eulerlabs.com/)
/// @notice Library for working with the `Snapshot` struct
library SnapshotLib {
    uint32 constant STAMP = 1 << 31; // non zero initial value of the snapshot slot to save gas on SSTORE
```

Recommendation:

We advise the `STAMP` to be set to `1`, ensuring the least significant bit of the overall `Snapshot` data point is configured and thus permitting the introduction of new variables after `borrows`.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The `STAMP` variable has been updated to represent `1`, ensuring that the `Snapshot` data structure can expand if needed and addressing this exhibit.

Storage Code Style Findings

SEG-01C: Potential Combination of Data Points

Type	Severity	Location
Gas Optimization	Informational	Storage.sol:L12, L16

Description:

The `Snapshot` structure has an empty variable meant to remain as "dirty" at all times to prevent the `Snapshot` storage slot from ever being zeroed out as part of a gas optimization. The `initialized` flag, on the other hand, is meant to remain `true` during the contract's normal operation.

Example:

src/EVault/shared/Storage.sol

SOL

```
7   /// @title Storage
8   /// @author Euler Labs (https://www.eulerlabs.com/)
9   /// @notice Contract that defines the EVault's data storage
10  abstract contract Storage {
11      /// @notice Flag indicating if the vault has been initialized
12      bool initialized;
13
14      /// @notice Snapshot of vault's cash and borrows created at the beginning of
an operation or a batch of operations
15      /// @dev The snapshot is separate from VaultStorage, because it could be
implemented as transient storage
16      Snapshot snapshot;
```

Example (Cont.):

```sol
17
18      /// @notice A singleton VaultStorage
19      VaultStorage vaultStorage;
20  }
```

Recommendation:

We advise the `initialized` and `snapshot` data points to be potentially combined, optimizing the `Storage` contract's storage layout.

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The Euler Finance team evaluated this recommendation and opted to retain the data structures separate as the current layout is better suited for their future plans, such as utilization of transient storage (i.e. **EIP-1153**).

As such, we consider this exhibit inapplicable in light of the Euler Finance team's future plans.

UserStorage Code Style Findings

USE-01C: Inconsistent Bit Clearance Style

Type	Severity	Location
Code Style	Informational	UserStorage.sol:L50, L56, L62

Description:

The `BALANCE_FORWARDER_MASK` configuration function will clear its bit by performing a bitwise AND (`&`) operation between the unwrapped data point and the negated value of the `BALANCE_FORWARDER_MASK`, whereas the two other functions for the `OWED_MASK` and `SHARES_MASK` respectively will use a bitwise AND (`&`) operation with a bitwise OR (`|`) combination of other masks.

Example:

src/EVault/shared/types/UserStorage.sol

SOL

```
47  function setBalanceForwarder(UserStorage storage self, bool newValue) internal {
48      self.data = newValue
49          ? PackedUserSlot.wrap(PackedUserSlot.unwrap(self.data) |
BALANCE_FORWARDER_MASK)
50          : PackedUserSlot.wrap(PackedUserSlot.unwrap(self.data) &
~BALANCE_FORWARDER_MASK);
51  }
52
53  function setOwed(UserStorage storage self, Owed owed) internal {
54      uint256 data = PackedUserSlot.unwrap(self.data);
55
56      self.data = PackedUserSlot.wrap((owed.toUint() << 112) | (data &
(BALANCE_FORWARDER_MASK | SHARES_MASK)));
```

Example (Cont.):

```sol
}

function setBalance(UserStorage storage self, Shares balance) internal {
    uint256 data = PackedUserSlot.unwrap(self.data);

    self.data = PackedUserSlot.wrap(balance.toUint() | (data &
(BALANCE_FORWARDER_MASK | OWED_MASK)));
}
```

Recommendation:

We advise a uniform bit erasure style to be adopted in the codebase, and we advise the former to be adopted as it results in less operations and thus higher legibility.

Specifically, we advise the negated form of the mask being written-to to be utilized in each referenced bitwise AND (`&`) clause (i.e. `data & ~OWED_MASK` instead of `data & (BALANCE_FORWARDER_MASK | SHARES_MASK)`)

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

A uniform bit clearance style has been adopted by the library, utilizing a bitwise `AND` (`&`) operation with the inverse (i.e. bitwise `NOT`, a.k.a. `~`) of the desired mask. As such, we consider this exhibit fully addressed.

USE-02C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	UserStorage.sol:L28, L56

Description:

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

Example:

src/EVault/shared/types/UserStorage.sol

SOL

```
28  uint256 constant OWED_OFFSET = 112;
```

Recommendation:

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

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

Alleviation (fb2dd77a6ff9b7f710edb48e7eb5437e0db4fc1a):

The statement within **`UserStorageLib::setOwed`** properly utilizes the `OWED_OFFSET` constant that already existed, optimizing the maintainability and legibility of the codebase.

Finding Types

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

Input Sanitization

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

Indeterminate Code

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

Language Specific

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

Curve Specific

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

Code Style

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

Mathematical Operations

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

Logical Fault

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

Privacy Concern

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

Proof Concern

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

Severity Definition

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

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

Severity Levels

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

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

Unknown Severity

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

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

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

Informational Severity

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

Minor Severity

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

Medium Severity

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

Major Severity

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

Likelihood & Impact Assessment

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

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

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

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.