

Kiln: Railnet Phase 1 Security Review

Auditors

Alireza Arjmand, Lead Security Researcher

Optimum, Lead Security Researcher

StErMi, Lead Security Researcher

Zigtur, Lead Security Researcher

Report prepared by: Lucas Goiriz

September 3, 2026

Contents

1 About Spearbit **3**

2 Introduction **3**

3 Risk classification **3**
3.1 Impact . . . 3
3.2 Likelihood . . . 3
3.3 Action required for severity levels . . . 3

4 Executive Summary **4**
4.1 Scope . . . 4
4.2 Trust assumptions . . . 7
4.2.1 Merkl Trust Assumptions . . . 7

5 Overview **8**

6 Findings **9**
6.1 Medium Risk . . . 9
6.1.1 `AaveV3Vehicle._maxDeposit` implementation logic contains multiple errors . . . 9
6.1.2 `CompoundV3Vehicle._maxRedeem` Can Misestimate Available Liquidity and Overstate Redeemable Shares . . . 9
6.2 Low Risk . . . 14
6.2.1 Share decimals and asset decimals are swapped in the fee manager calls . . . 14
6.2.2 Shares and assets downscaling always round down instead of using the `rounding` input parameter . . . 17
6.2.3 Scoped-role management functions should follow the same sanity rules defined in `AccessControlDefaultAdminRules` . . . 20
6.2.4 Euler EVault view functions integration safety relies on the governance . . . 21
6.2.5 `AaveV3Vehicle` prevents users from redeeming shares when the Aave reserve is frozen . . . 21
6.2.6 `BaseVehicle.ready` is not compliant with the `STEAM` standard . . . 22
6.2.7 `BaseVehicle._payout` should never be able to revert . . . 23
6.2.8 `FeeManager.onUpdate` should revert when `$.currentFeesConfigId == bytes32(0)` . . . 23
6.2.9 Aave offers multiple distribution reward systems which are currently not supported by the Aave V3 Vehicle . . . 24
6.2.10 The `AaveV3Merkl` must be fully refactored . . . 24
6.2.11 Incorrect Asset Accounting if Vehicle Is Morpho Market Fee Recipient . . . 26
6.2.12 Effects of rounding on both the vault's shares and underlying protocol shares . . . 26
6.3 Informational . . . 33
6.3.1 Use of deprecated contract comparison operators . . . 33
6.3.2 STEAM specifications precision about redeem optional output . . . 34
6.3.3 Scaling factor calculation in `_previewOnOperations` is prone to edge-cases . . . 34
6.3.4 Natspec issues . . . 35
6.3.5 [Research] AAVE Reward Distribution . . . 37
6.3.6 `Account.performCall` incorrectly reverts with `Unauthorized` error . . . 39
6.3.7 `ExternalRBAC` initializers are not disabled . . . 39
6.3.8 `isCategory` incorrectly reverts instead of returning false . . . 39
6.3.9 Dead code . . . 40
6.3.10 `setDeprecated` allows to ”undeprecate” . . . 41
6.3.11 Query ID is calculated at each round loop . . . 41
6.3.12 Requirements for `resume()` Implementation in Derived Vehicles . . . 42
6.3.13 Missing non-zero check for underlying protocol minted shares . . . 42
6.3.14 Missing `msg.sender` in `SpawnedExternalAccessControl` Event . . . 43
6.3.15 Missing `receive()` function prevents `Account` from accepting ETH . . . 43
6.3.16 Missing State Transition Validation in `_setState()` . . . 44

6.3.17 Functions renamings . . . 44
6.3.18 Resiliency Against Non-Standard ERC-4626 Vault Withdrawal Fees . . . 45
6.3.19 Repository-Wide Code Cleanup and Logic Consolidation . . . 45
6.3.20 Consider using OZ's `Ownable2Step` in the `Account` contract . . . 46
6.3.21 Consider reverting if the setter/updater function does not perform any state change . . . 46
6.3.22 Sanity Checks . . . 46
6.3.23 Bulk Informational Issues . . . 47
6.3.24 `forbiddenAddresses` configured by the factories could be seen as meaningless . . . 48
6.3.25 `spawn` function can be improved for some of the existing factories . . . 49
6.3.26 Consider updating the STEAM standard to require the Query to provide a non-empty `Asset[] output` array . . . 49
6.3.27 Use `forceApprove` instead of `safeIncreaseAllowance` in the `Factory` . . . 50
6.3.28 `Vehicle` functions `convertToAssets` and `convertToShares` should be better documented 51
6.3.29 Remove the `_recover` function's implementation from all those vehicle that cannot transition to the `RECOVERING` state . . . 51
6.3.30 Consider allowing integrator to fetch the fee detail snapshotted to the Query . . . 51
6.3.31 Consider avoiding taking payout fees when the query has been generated by the `FeeManager` 52
6.3.32 Some ”open” functions allow anyone to generate spammy events . . . 52
6.3.33 Vehicles do not follow a common style and best practice in their implementation . . . 53
6.3.34 Inefficient `O(n^2)` Duplicate Check in `_setRecipients` . . . 54
6.3.35 Redundant Admin Check in `grantRole` and `revokeRole` . . . 54
6.3.36 `CompoundV3Vehicle` and `MorphoBlueVehicle` Cannot Claim Incentive Rewards . . . 54
6.3.37 `CoreFactory` Mishandles ETH Forwarding and Refund Accounting Across CreateX and Clone Flows . . . 55
6.3.38 `Created` event emission does not respect the STEAM specification . . . 56
6.3.39 `STEAM` standard fee's lifecycle section is outdated . . . 56
6.3.40 Improve the `totalAssets()` definition in the `STEAM` standard . . . 57
6.3.41 Consider refactoring the `ModulesManager` module management/allowance logic . . . 57
6.3.42 Overriding `_shouldHandleFees` function in future implementations can have unexpected impacts . . . 59

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Kiln: Railnet Phase 1 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 24 days in total, Kiln engaged with Spearbit to review the railnet protocol. In this period of time a total of **56** issues were found.

Summary

Project Name	Kiln
Repository	railnet
Commit	9ca80f3b
Type of Project	DeFi, Yield Aggregator
Audit Timeline	Feb 23rd to Mar 19th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	12	10	2
Gas Optimizations	0	0	0
Informational	42	36	6
Total	**56**	**48**	**8**

4.1 Scope

The security review had the following components in scope for railnet on commit hash 9ca80f3b:

```
├── abstracts
│   ├── ExternalRBAC.sol
│   ├── ExternalRBACUpgradeable.sol
│   ├── FactoryBase.sol
│   └── Interceptor.sol
├── common
│   ├── Account.sol
│   ├── ExternalAccessControl.sol
│   ├── FreezablePausableBeacon.sol
│   └── interfaces
│       ├── IExternalAccessControl.sol
│       └── IFreezablePausableBeacon.sol
├── docs
│   └── STEAM.md
├── factories
│   ├── common
│   │   └── AccessControlFactory.sol
│   ├── CoreFactory.sol
```

```
│   ├── interfaces
│   │   └── ICoreFactory.sol
│   └── vehicles
│       ├── AaveV3VehicleFactory.sol
│       ├── CompoundV3VehicleFactory.sol
│       ├── ERC4626VehicleFactory.sol
│       ├── FeeManagerFactory.sol
│       ├── ModulesManagerFactory.sol
│       └── MorphoBlueVehicleFactory.sol
├── INDEX.md
├── libs
│   ├── AccessControl.sol
│   ├── Assembly.sol
│   ├── Asset.sol
│   ├── Check.sol
│   ├── Constant.sol
│   ├── Error.sol
│   ├── Factory.sol
│   ├── Keeper.sol
│   ├── Query.sol
│   ├── Rejection.sol
│   ├── Roles.sol
│   ├── Shares.sol
│   └── Vehicle.sol
├── steam
│   ├── IVehicle.sol
│   └── Query.sol
└── vehicles
    ├── aave_v3
    │   ├── AaveV3Vehicle.sol
    │   ├── abstracts
    │   │   ├── AaveV3VehicleErrors.sol
    │   │   └── AaveV3VehicleEvents.sol
    │   ├── facets
    │   │   └── AaveV3VehicleFacets.sol
    │   ├── interfaces
    │   │   ├── IPool.sol
    │   │   ├── IPoolAddressesProvider.sol
    │   │   ├── IPoolDataProvider.sol
    │   │   └── IScaledBalanceToken.sol
    │   ├── libs
    │   │   ├── AaveV3VehicleStore.sol
    │   │   └── AaveV3VehicleStructs.sol
    │   ├── modules
    │   │   └── AaveV3Merkl.mod.sol
    │   └── VEHICLE.md
    ├── base
    │   ├── abstracts
    │   │   ├── BaseVehicleAssetUtils.sol
    │   │   ├── BaseVehicleChecks.sol
    │   │   ├── BaseVehicleErrors.sol
    │   │   ├── BaseVehicleEvents.sol
    │   │   ├── BaseVehicleInitialization.sol
    │   │   ├── BaseVehicleQueryStateGetter.sol
    │   │   └── BaseVehicleQueryStateSetter.sol
    │   ├── BaseVehicle.sol
    │   ├── FeeManager.sol
    │   ├── interfaces
```

```
│   │   ├── facets
│   │   │   ├── ICreateFacet.sol
│   │   │   ├── IInitializeFacet.sol
│   │   │   ├── IRecoverFacet.sol
│   │   │   ├── IResumeFacet.sol
│   │   │   └── IUnlockFacet.sol
│   │   ├── IBaseVehicle.sol
│   │   ├── IDistributionCreator.sol
│   │   ├── IFeeManager.sol
│   │   ├── IModule.sol
│   │   ├── IModulesManager.sol
│   │   └── IModuleTarget.sol
│   ├── libs
│   │   ├── BaseVehicleConstants.sol
│   │   └── BaseVehicleStore.sol
│   ├── ModulesManager.sol
│   └── SingleAssetBaseVehicle.sol
├── compound_v3
│   ├── abstracts
│   │   ├── CompoundV3VehicleErrors.sol
│   │   └── CompoundV3VehicleEvents.sol
│   ├── CompoundV3Vehicle.sol
│   ├── facets
│   │   └── CompoundV3VehicleFacets.sol
│   ├── interfaces
│   │   └── IComet.sol
│   ├── libs
│   │   ├── CompoundV3VehicleStore.sol
│   │   └── CompoundV3VehicleStructs.sol
│   └── VEHICLE.md
├── erc4626
│   ├── abstracts
│   │   ├── ERC4626VehicleErrors.sol
│   │   └── ERC4626VehicleEvents.sol
│   ├── ERC4626Vehicle.sol
│   ├── facets
│   │   └── ERC4626VehicleFacets.sol
│   ├── libs
│   │   ├── ERC4626VehicleStore.sol
│   │   └── ERC4626VehicleStructs.sol
│   └── VEHICLE.md
└── morpho_blue
    ├── abstracts
    │   ├── MorphoBlueVehicleErrors.sol
    │   └── MorphoBlueVehicleEvents.sol
    ├── facets
    │   └── MorphoBlueVehicleFacets.sol
    ├── libs
    │   ├── MorphoBlueVehicleStore.sol
    │   └── MorphoBlueVehicleStructs.sol
    ├── MorphoBlueVehicle.sol
    └── VEHICLE.md
```

4.2 Trust assumptions

4.2.1 Merkl Trust Assumptions

Trust assumptions for usage of Merkl via the reward distribution system are listed below, it is recommended to keep these in mind when utilizing the protocol:

- Merkl will assign the same campaign ID to campaigns that differ only in total amount while keeping all other parameters identical, this will result in the revert of the second transaction that lands.
- Both the contract and the caller are expected to be properly whitelisted by Merkl before interacting with the system.
- Merkl enforces a minimum threshold on the total campaign amount to avoid economically insignificant campaigns.
- Merkl will enforce a minimum threshold on the distributed amount to prevent negligible distributions.
- Users themselves and whitelisted operators approved by Merkl are be able to claim rewards for of users.

5 Overview

Railnet is Kiln's vault platform built around **STEAM**, an in-house standard for vault operations that supports both synchronous and asynchronous flows. Every operation is a *query* — a struct describing the owner, receiver, input/output assets, and mode (deposit or redeem) — that moves through a fixed state machine (`EMPTY → PROCESSING → PAUSED / UNLOCKING / RECOVERING / REJECTED → SETTLED`) via the `create` , `resume` , `unlock` , and `recover` entrypoints. Synchronous vehicles settle a query in two calls; asynchronous ones park it until the underlying protocol settles.

The core of the phase is the **BaseVehicle** framework: an upgradeable ERC-20 vault whose shares are priced against the assets a concrete vehicle holds in its yield source. It uses namespaced (ERC-7201) storage throughout, delegates lifecycle logic to per-vehicle *facet* contracts to stay under the bytecode limit, and validates queries through a rejection model in which invalid queries become recoverable rather than reverting. `SingleAssetBaseVehicle` specializes the framework for the common single-asset-in/shares-out shape. Two satellite contracts extend a vehicle's behavior: the `FeeManager` (management, performance, deposit, and redeem fees, with per-query fee-config snapshots) and the `ModulesManager` (timelocked registration of modules that vehicles execute via delegatecall).

Surrounding infrastructure includes a shared role-based access control system (`ExternalAccessControl` , supporting global, scoped, and public roles), beacon-proxy upgradeability with pause and permanent-freeze controls (`FreezablePausableBeacon`), and a factory stack: `CoreFactory` performs deterministic deployments via CreateX, and per-component factories spawn beacon, proxy, and an initial burned deposit that protects new vaults against share-inflation attacks.

Four synchronous yield-source vehicles are in scope, each following the same layout (main contract, facets, store library): **Aave v3** (supplies to the Aave pool), **Compound v3** (supplies to a Comet market, was fully removed in later commits), **ERC-4626** (deposits into an external vault), and **Morpho Blue** (supplies to a Morpho market). A set of small shared libraries (asset/query handling, share conversion, roles, errors, keeper events) completes the scope.

6 Findings

6.1 Medium Risk

6.1.1 `AaveV3Vehicle._maxDeposit` implementation logic contains multiple errors

Severity: Medium Risk.

Context: AaveV3Vehicle.sol#L153-L176.

Description: The current implementation of the `AaveV3Vehicle._maxDeposit` function contains multiple logic and interpretation errors that incorrectly estimate the max deposit value that the vehicle can supply into the underlying Aave V3 protocol.

`_accruedToTreasuryScaled` could be outdated: The logic is assuming that the `_accruedToTreasuryScaled` value returned by `poolDataProvider.getReserveData(...)` is always up to date. If `lastUpdateTimestamp` (always returned by the same function) is lower than `block.timestamp` it means that it's possible that **more shares** could be accounted to the Aave treasury (if there's interest to be accrued). If that's the case (which is high likely) it means that `_accruedToTreasuryScaled` is **outdated** and so it could end up returning an **overestimated** `maxDeposit` value that could make the deposit operation of the vehicle revert.

Wrong assumptions on the value types: `_accruedToTreasuryScaled` and `_scaledTotalSupply` are values expressed in ”share” terms (in the Aave context, ”scaled” can be compared to ”shares”). Those values **must** be multiplied by the `_liquidityIndex` to bring them back to the ”non-scaled” value in which `_supplyCap` is expressed with.

Recommendation: Kiln should consider contacting the BGD team to properly integrate the Aave platform and extract the correct logic to determine the max available amount of supply that can be deposited into the Aave v3 Pool for the specific asset.

Kiln: Fixed in commit 53b6e6d.

Spearbit: Verified fixes.

6.1.2 `CompoundV3Vehicle._maxRedeem` Can Misestimate Available Liquidity and Overstate Redeemable Shares

Severity: Medium Risk.

Context: CompoundV3Vehicle.sol#L218.

Description: `CompoundV3Vehicle._maxRedeem` derives redeemable liquidity from:

```
$.compoundV3Market.totalSupply() - $.compoundV3Market.totalBorrow()
```

and then converts that amount into shares. This assumption is not always valid in Compound V3 and can cause the vehicle to overestimate how much can actually be redeemed.

Two cases make this especially problematic:

- **Borrows can exceed supply net of reserves:** In Compound V3, borrows can effectively eat into reserves, so `totalBorrow()` is not guaranteed to stay below `totalSupply()`. As a result, `totalSupply() - totalBorrow()` can underflow or otherwise stop being a reliable proxy for withdrawable base liquidity.

- **Absorbs can inflate `totalSupply()` without fresh incoming base assets:** During `absorb`, Compound updates accounting by increasing total supply and decreasing total borrow as debt is socialized onto reserves, but this does not mean new base tokens were actually transferred into the protocol. If reserves were previously withdrawn, an absorb can make `totalSupply() - totalBorrow()` look healthier than the protocol's real withdrawable base balance, causing `_maxRedeem` to overestimate redeemable shares.

As a result, `maxRedeem` may report more redeemable shares than can actually be withdrawn from Compound V3, leading to incorrect vehicle limits and failed redemption attempts.

Proof of Concept: The proof of concept can be seen below:

```
// SPDX-License-Identifier: BUSL-1.1
// SPDX-FileCopyrightText: 2025 Kiln <contact@kiln.fi>
//
//                     d8b  888                     888
//                     Y8P  888                     888
//                          888                     888
//    888d888  8888b.   888  888  88888b.    .d88b.   888888
//    888P"       "88b  888  888  888 "88b  d8P  Y8b  888
//    888     .d888888  888  888  888  888  88888888  888
//    888     888  888  888  888  888  888  Y8b.      Y88b.
//    888     "Y888888  888  888  888  888   "Y8888    "Y888
//
pragma solidity >=0.8.33;

import {Test, console2} from "forge-std/Test.sol";

import {ICreateX} from "@createx/ICreateX.sol";
import {IERC20} from "@openzeppelin-contracts/token/ERC20/IERC20.sol";

import {Asset, EstimationType, Query, Mode, State} from "src/steam/Query.sol";

import {ExternalAccessControl, IExternalAccessControl} from
↪  "src/common/ExternalAccessControl.sol";
import {FreezablePausableBeacon} from "src/common/FreezablePausableBeacon.sol";
import {CoreFactory} from "src/factories/CoreFactory.sol";
import {CompoundV3VehicleFactory} from "src/factories/vehicles/CompoundV3VehicleFactory.sol";
import {FeeManagerFactory} from "src/factories/vehicles/FeeManagerFactory.sol";
import {Roles} from "src/libs/Roles.sol";
import {FeeManager} from "src/vehicles/base/FeeManager.sol";
import {IFeeManager} from "src/vehicles/base/interfaces/IFeeManager.sol";
import {ModulesManager} from "src/vehicles/base/ModulesManager.sol";
import {CompoundV3Vehicle} from "src/vehicles/compound_v3/CompoundV3Vehicle.sol";
import {IComet} from "src/vehicles/compound_v3/interfaces/IComet.sol";

contract CompoundV3VehicleMinimalForkTest is Test {
    address internal constant MAINNET_CREATEX = 0xba5Ed099633D3B313e4D5F7bdc1305d3c28ba5Ed;
    address internal constant COMET_USDC_V3 = 0xc3d688B66703497DAA19211EEdff47f25384cdc3;
    address internal constant USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;
    address internal constant WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
    uint256 internal constant BLOCK_BEFORE_24524672 = 24_524_671;
    address internal constant ABSORB_ABSORBER = 0x6905Cca5716Cc4FAe5692647F3b09A54F2754a87;
    address internal constant ABSORBED_ACCOUNT = 0x26922eFd8491b61Ea21F101D075D4abd1d8a953e;
    uint256 internal constant BOB_WETH_COLLATERAL = 200_000 ether;

    uint256 internal constant INITIAL_DEPOSIT = 100e6;
    uint256 internal constant ALICE_DEPOSIT = 1_000e6;
    uint256 internal constant BOB_DEPOSIT = 1e6;
    uint256 internal constant DUST_DEPOSIT = 1;

    address internal ADMIN;
```

```
address internal DEPLOYER;
address internal ALICE;
address internal BOB;
address internal TREASURY;

CompoundV3Vehicle internal vehicle;
CoreFactory internal coreFactory;
ExternalAccessControl internal accessControl;
CompoundV3VehicleFactory internal compoundV3Factory;

function setUp() external {
    vm.createSelectFork(vm.envString("MAINNET_RPC"));

    ADMIN = makeAddr("Admin");
    DEPLOYER = makeAddr("Deployer");
    ALICE = makeAddr("Alice");
    BOB = makeAddr("Bob");
    TREASURY = makeAddr("Treasury");

    _deployVehicleOnCurrentFork();
}

function _deployVehicleOnCurrentFork() internal {
    coreFactory = new CoreFactory(ICreateX(MAINNET_CREATEX));

    IExternalAccessControl.RoleAttribution[] memory initialRoles = new
    ↪  IExternalAccessControl.RoleAttribution[](1);
    initialRoles[0] = IExternalAccessControl.RoleAttribution({role: Roles.FACTORY_SPAWN,
    ↪  account: DEPLOYER});
    accessControl = new ExternalAccessControl(1 days, ADMIN, initialRoles);

    CompoundV3Vehicle implementation = new CompoundV3Vehicle();
    FreezablePausableBeacon beacon = new FreezablePausableBeacon(address(implementation),
    ↪  accessControl);

    compoundV3Factory = new CompoundV3VehicleFactory(coreFactory, accessControl, beacon,
    ↪  0);

    deal(USDC, DEPLOYER, 10_000e6);
    vm.startPrank(DEPLOYER);
    IERC20(USDC).approve(address(compoundV3Factory), type(uint256).max);

    vehicle = compoundV3Factory.spawn(
        CompoundV3VehicleFactory.SpawnParams({
            compoundV3Market: COMET_USDC_V3,
            accessControl: ExternalAccessControl(address(0)),
            feeManager: FeeManager(address(0)),
            modulesManager: ModulesManager(address(0)),
            querySalt: keccak256("bootstrap-query"),
            deploymentSalt: keccak256("bootstrap-deployment"),
            initialDepositSize: INITIAL_DEPOSIT,
            initialExpectedSupply: 1
        })
    );
    vm.stopPrank();

    assertEq(vehicle.asset(), USDC);
    assertEq(IComet(COMET_USDC_V3).baseToken(), USDC);
    assertTrue(vehicle.isEnabled());
}

function test_absorb_can_make_max_redeem_optimistic1() external {
```

```
        // "Start of block 24524672" corresponds to state at the end of block 24524671.
        vm.createSelectFork(vm.envString("MAINNET_RPC"), BLOCK_BEFORE_24524672);
        _deployVehicleOnCurrentFork();

        deal(USDC, ALICE, ALICE_DEPOSIT);

        uint256 aliceShares = _depositAndUnlock(vehicle, ALICE, ALICE_DEPOSIT,
        ↪  keccak256("alice-absorb-deposit"));
        assertGt(aliceShares, 0);

        uint256 bobBorrowed = _borrowAvailableLiquidityAsBob();
        assertGt(bobBorrowed, 0);

        uint256 cometUsdcCashBeforeAbsorb = IERC20(USDC).balanceOf(COMET_USDC_V3);
        uint256 reportedLiquidityBeforeAbsorb = _reportedCometAvailableLiquidity();

        address[] memory absorbAccounts = new address[](1);
        absorbAccounts[0] = ABSORBED_ACCOUNT;

        bytes memory cometTotalSupply;
        bytes memory cometTotalBorrow;

        (, cometTotalSupply) = COMET_USDC_V3.call(abi.encodeWithSignature("totalSupply()"));
        (, cometTotalBorrow) = COMET_USDC_V3.call(abi.encodeWithSignature("totalBorrow()"));
        console2.log("cometTotalSupply ", uint256(bytes32(cometTotalSupply)));
        console2.log("cometTotalBorrow ", uint256(bytes32(cometTotalBorrow)));

        // MethodID 0xc3cecfd2: absorb(address,address[])
        // (bool absorbSuccess,) =
        //     COMET_USDC_V3.call(abi.encodeWithSelector(bytes4(0xc3cecfd2), ABSORB_ABSORBER,
        ↪  absorbAccounts));
        // assertTrue(absorbSuccess);

        // (, cometTotalSupply) = COMET_USDC_V3.call(abi.encodeWithSignature("totalSupply()"));
        // (, cometTotalBorrow) = COMET_USDC_V3.call(abi.encodeWithSignature("totalBorrow()"));
        // console2.log("cometTotalSupply ", uint256(bytes32(cometTotalSupply)));
        // console2.log("cometTotalBorrow ", uint256(bytes32(cometTotalBorrow)));

        // Asset[] memory maxRedeemAssets = vehicle.maxRedeem(ALICE); --> fails

        Query memory redeemQuery = Query({
            owner: ALICE,
            receiver: ALICE,
            input: _singleAsset(address(vehicle), 1),
            output: new Asset[](0),
            mode: Mode.REDEEM,
            salt: keccak256("alice-max-redeem-after-absorb"),
            data: ""
        });

        vm.startPrank(ALICE);
        IERC20(address(vehicle)).approve(address(vehicle), 1);
        vm.expectRevert();
        vehicle.create(redeemQuery); // FAIL: panic: arithmetic underflow or overflow (0x11)
        //   cometTotalSupply  381832487932364
        //   cometTotalBorrow  381832487932375
        vm.stopPrank();
    }

    function _borrowAvailableLiquidityAsBob() internal returns (uint256 borrowedAssets) {
        deal(WETH, BOB, BOB_WETH_COLLATERAL);
```

```
        vm.startPrank(BOB);
        IERC20(WETH).approve(COMET_USDC_V3, BOB_WETH_COLLATERAL);
        IComet(COMET_USDC_V3).supply(WETH, BOB_WETH_COLLATERAL);
        vm.stopPrank();

        uint256 reportedLiquidity = _reportedCometAvailableLiquidity();
        assertGt(reportedLiquidity, 1);

        // Target near-full utilization. If account constraints prevent full drain, find Bob's
        ↪  max.
        uint256 borrowTarget = reportedLiquidity + 10; // ---> Here we want to exceed the
        ↪  supply rate
        borrowedAssets = _findMaxBorrowableAsBob(borrowTarget);
        assertGt(borrowedAssets, 0);

        bytes memory cometTotalSupply;
        bytes memory cometTotalBorrow;

        (, cometTotalSupply) = COMET_USDC_V3.call(abi.encodeWithSignature("totalSupply()"));
        (, cometTotalBorrow) = COMET_USDC_V3.call(abi.encodeWithSignature("totalBorrow()"));
        console2.log("cometTotalSupply ", uint256(bytes32(cometTotalSupply)));
        console2.log("cometTotalBorrow ", uint256(bytes32(cometTotalBorrow)));

        vm.prank(BOB);
        IComet(COMET_USDC_V3).withdraw(USDC, borrowedAssets);

        (, cometTotalSupply) = COMET_USDC_V3.call(abi.encodeWithSignature("totalSupply()"));
        (, cometTotalBorrow) = COMET_USDC_V3.call(abi.encodeWithSignature("totalBorrow()"));
        console2.log("cometTotalSupply ", uint256(bytes32(cometTotalSupply)));
        console2.log("cometTotalBorrow ", uint256(bytes32(cometTotalBorrow)));
    }

    function _findMaxBorrowableAsBob(uint256 upperBound) internal returns (uint256 maxBorrow) {
        uint256 low = 0;
        uint256 high = upperBound;

        while (low < high) {
            uint256 mid = low + (high - low + 1) / 2;
            uint256 snap = vm.snapshotState();
            vm.prank(BOB);
            (bool success,) =
            ↪  COMET_USDC_V3.call(abi.encodeWithSelector(IComet.withdraw.selector, USDC,
            ↪  mid));
            vm.revertToState(snap);

            if (success) {
                low = mid;
            } else {
                high = mid - 1;
            }
        }

        return low;
    }

    function _reportedCometAvailableLiquidity() internal view returns (uint256) {
        uint256 totalSupply_ = IComet(COMET_USDC_V3).totalSupply();
        uint256 totalBorrow_ = IComet(COMET_USDC_V3).totalBorrow();
        return totalBorrow_ >= totalSupply_ ? 0 : totalSupply_ - totalBorrow_;
    }
```

```
    function _singleAsset(address asset, uint256 value) internal pure returns (Asset[] memory
    ↪  assets) {
        assets = new Asset[](1);
        assets[0] = Asset({asset: asset, value: value});
    }

    function _depositAndUnlock(CompoundV3Vehicle target, address owner, uint256 assets,
    ↪  bytes32 salt)
        internal
        returns (uint256 sharesReceived)
    {
        Asset[] memory input = _singleAsset(USDC, assets);
        Query memory query = Query({
            owner: owner,
            receiver: owner,
            input: input,
            output: target.estimate(input, Mode.DEPOSIT, EstimationType.OUTPUT),
            mode: Mode.DEPOSIT,
            salt: salt,
            data: ""
        });

        vm.startPrank(owner);
        IERC20(USDC).approve(address(target), assets);
        State createState = target.create(query);
        assertEq(uint256(createState), uint256(State.UNLOCKING));
        (State unlockState, Asset[] memory unlockedAssets) = target.unlock(query);
        vm.stopPrank();

        assertEq(uint256(unlockState), uint256(State.SETTLED));
        assertEq(unlockedAssets.length, 1);
        assertEq(unlockedAssets[0].asset, address(target));
        sharesReceived = unlockedAssets[0].value;
    }
}
```

Recommendation: Do not use `totalSupply() - totalBorrow()` as a proxy for immediately withdrawable base liquidity.

Comet doesn't cap the amount that can be withdrawn to any address, effectively capping it to the minimum of the user's balance or the protocol's balance.

Kiln: Fixed by PR 375.

Spearbit: Verified fix, the `compoundV3Vehicle` now uses the contract's remaining base asset balance as reference for maximum redeemable amount. Note: the CompoundV3Vehicle has since been fully removed from the codebase in later commits.

6.2 Low Risk

6.2.1 Share decimals and asset decimals are swapped in the fee manager calls

Severity: Low Risk.

Context: BaseVehicle.sol#L452-L453.

Description: `BaseVehicle` calls both `onOperations()` and `previewOnOperations()` as part of the contract.

`FeeManager.onOperations()` is declared as the following.

```
function onOperations(
    uint256 currentTotalSupply,
    uint256 currentTotalAssets,
    uint8 assetDecimals,
    uint8 sharesDecimals
) external returns (uint256 feeSharesToMint) {
```

However, `BaseVehicle` calls both `onOperations()` and `previewOnOperations()` with `sharesDecimals` and `assetDecimals` in the wrong order.

```
function _handleFeesBeforeOperation(uint256 totalSupply_, uint256 totalAssets_, Id qid)
↪  internal returns (uint256) {
    // ...

        uint256 _feeSharesToMint =
            _feeManager.onOperations(totalSupply_, totalAssets_, decimals(), _assetDecimals());

    // ...
}
```

Proof of Concept: The following test can be imported in `test/vehicles/base/FeeManager.t.sol`.

```
// ============================================================================
    // Decimals Confusion Tests
    // ============================================================================

    /// @notice Demonstrates that swapping assetDecimals and sharesDecimals in
    ↪  previewOnOperations
    ///         produces different fee share amounts, exposing the parameter ordering confusion
    ///         between onOperations (sharesDecimals, assetDecimals) and previewOnOperations
    ↪  (assetDecimals, sharesDecimals).
    function testPreviewOnOperationsDecimalsConfusion() public {
        // Setup: 20% performance fee
        IFeeManager.Fees memory _fees = IFeeManager.Fees({
            performanceFeeBps: 2000,
            managementFeeBps: 200, // 2% annually
            depositFeeBps: 0,
            redeemFeeBps: 0
        });

        IFeeManager.Fees memory _maxFees = IFeeManager.Fees({
            performanceFeeBps: 5000, managementFeeBps: 1000, depositFeeBps: 1000,
            ↪  redeemFeeBps: 1000
        });

        IFeeManager.FeeRecipient[] memory _recipients = _createDefaultRecipients();
        $feeManager = _deployFeeManager(_fees, _maxFees, _recipients);

        uint256 _initialAssets = 10e6; // 6-decimal asset (e.g. USDC)
        uint256 _currentAssets = 11e6; // 100e6 profit
        uint256 _totalSupply = 0; // 18-decimal shares

        // Set initial cache
        vm.prank($mockVehicle);
        $feeManager.onUpdate(_initialAssets);

        // Fast forward 1 year (so management fee also kicks in)
        vm.warp(block.timestamp + 365 days);
```

```
        // Call 1: Correct ordering - assetDecimals=6, sharesDecimals=18
        vm.prank($mockVehicle);
        uint256 _feeSharesCorrect = $feeManager.previewOnOperations(_totalSupply,
        ↪  _currentAssets, 6, 18);

        // Call 2: Swapped ordering - assetDecimals=18, sharesDecimals=6 (the confusion)
        vm.prank($mockVehicle);
        uint256 _feeSharesSwapped = $feeManager.previewOnOperations(_totalSupply,
        ↪  _currentAssets, 18, 6);

        emit log_named_uint("Fee shares (assetDecimals=6, sharesDecimals=18)",
        ↪  _feeSharesCorrect);
        emit log_named_uint("Fee shares (assetDecimals=18, sharesDecimals=6)",
        ↪  _feeSharesSwapped);
        emit log_named_uint("Difference", _feeSharesCorrect > _feeSharesSwapped ?
        ↪  _feeSharesCorrect - _feeSharesSwapped : _feeSharesSwapped - _feeSharesCorrect);

        // These should differ if the decimals actually matter for the conversion
        // This demonstrates the impact of the confusion
        if (_feeSharesCorrect != _feeSharesSwapped) {
            emit log("DECIMALS CONFUSION: different results when swapping assetDecimals and
            ↪  sharesDecimals");
        } else {
            emit log("No difference detected (same decimals or scaling cancels out)");
        }
    }
```

Recommendation: The `FeeManager.onOperations` declaration must be fixed such that the ordering is correct.

The following patch fixes the **external functions declarations**.

Please note that internal functions are still using the order pattern that led to the issue (i.e. `shareSupply, assetSupply, assetDecimals, shareDecimals`). This is error-prone.

```
  diff --git a/src/vehicles/base/FeeManager.sol b/src/vehicles/base/FeeManager.sol
  index 3042b703..86420ead 100644
-  -- a/src/vehicles/base/FeeManager.sol
+  ++ b/src/vehicles/base/FeeManager.sol
  @@ -273,8 +273,8 @@ contract FeeManager is IFeeManager, Interceptor,
  ↪  ReentrancyGuardUpgradeable, Ext
       function onOperations(
           uint256 currentTotalSupply,
           uint256 currentTotalAssets,
-          uint8 assetDecimals,
-          uint8 sharesDecimals
+          uint8 sharesDecimals,
+          uint8 assetDecimals
       ) external returns (uint256 feeSharesToMint) {
           (uint256 _managementFeeSharesToMint, uint256 _performanceFeeSharesToMint) =
               _previewOnOperations(currentTotalSupply, currentTotalAssets, assetDecimals,
               ↪  sharesDecimals);
  @@ -302,8 +302,8 @@ contract FeeManager is IFeeManager, Interceptor,
  ↪  ReentrancyGuardUpgradeable, Ext
       function previewOnOperations(
           uint256 currentTotalSupply,
           uint256 currentTotalAssets,
-          uint8 assetDecimals,
-          uint8 sharesDecimals
+          uint8 sharesDecimals,
+          uint8 assetDecimals
```

```
    ) external view returns (uint256 feeSharesToMint) {
        (uint256 _managementFeeSharesToMint, uint256 _performanceFeeSharesToMint) =
            _previewOnOperations(currentTotalSupply, currentTotalAssets, assetDecimals,
            ↪  sharesDecimals);
```

Kiln: Fixed in commit 35678cd and commit 644c809.

Spearbit: Fixed. The ”on operations” functions are now the correct ordering for input parameters.

6.2.2 Shares and assets downscaling always round down instead of using the `rounding` input parameter

Severity: Low Risk.

Context: Shares.sol#L131-L178.

Description: The `convertToAssets` and `convertToShares` functions in the `Shares.sol` file implement an upscaling for calculations and then a downscaling of the result. These functions provide a `rounding` input parameter to select which rounding direction to use: `Floor` or `Ceil`.

However when `rounding = Ceil`, only the `Math.mulDiv` is rounding up and not the `downscaleShares/downscaleAssets` operation. This leads to rounding down when it should round up.

In the current state of the codebase, the issue occurs only on estimation calculations.

Proof of Concept: Import the following as `test/vehicles/erc4626/SharesCeilRoundingBug.t.sol` and then execute the following commands.

```
forge test --match-contract "ERC4626VehicleCeilRoundingBugTest" --mt
↪  test_vehicle_estimateDepositInput_CeilRoundingBug -vv
```

```
// SPDX-License-Identifier: BUSL-1.1
pragma solidity >=0.8.33;

import {Test} from "@forge-std/Test.sol";
import {Math} from "@openzeppelin-contracts/utils/math/Math.sol";
import {SharesLib} from "src/libs/Shares.sol";
import {Query, Route, Mode, Asset, State, EstimationType} from "src/steam/Query.sol";
import {ERC4626VehicleFactory} from "src/factories/vehicles/ERC4626VehicleFactory.sol";
import {ERC4626Mock} from "test/vehicles/mocks/ERC4626Mock.sol";
import {ERC20Mock} from "test/vehicles/mocks/ERC20Mock.sol";
import {STEAMSingleAssetTester} from "test/test_utils/STEAM.SingleAsset.t.sol";
import {Tester, TesterCommon, TesterTypes} from "test/test_utils/steam/utils/Tester.sol";
import {ExternalAccessControl} from "src/common/ExternalAccessControl.sol";
import {FeeManager} from "src/vehicles/base/FeeManager.sol";
import {ModulesManager} from "src/vehicles/base/ModulesManager.sol";
import {BaseVehicle} from "src/vehicles/base/BaseVehicle.sol";
import {IERC20Metadata} from
↪  "@openzeppelin-contracts/token/ERC20/extensions/IERC20Metadata.sol";

/// @title PoC: SharesLib.convertToAssets Ceil rounding defeated by downscale truncation
/// @notice When sharesDecimals(18) > assetDecimals(6), the floor division in
///         downscaleAssets() negates the Ceil rounding from Math.mulDiv().
///         This violates the invariant: convertToAssets(..., Ceil) >= convertToAssets(...,
↪  Floor)
///         for the same inputs when the mathematical result is non-integer.
///
///         Affected production paths (all with USDC/USDT/WBTC underlying):
///         - SingleAssetBaseVehicle._convertToAssets (estimate DEPOSIT+INPUT)
```

```
///         - MultiVehicle._convertToAssets (estimate DEPOSIT+INPUT)
///         - Conduit._convertConduitSharesToVehicleShares (estimate INPUT)



// ============================================================================
// PART 1: ERC4626 Vehicle end-to-end test
// ============================================================================

contract ERC4626VehicleCeilRoundingBugTest is STEAMSingleAssetTester {
    using Tester for Query;
    using TesterCommon for TesterTypes.AssertBalance[];
    using TesterCommon for TesterTypes.AssertAmount[];

    ERC4626Mock internal $underlyingAsset;

    function setUp() public virtual {
        // Deploy USDC-like token (6 decimals) and ERC4626 vault
        $asset = new ERC20Mock("USDC", "USDC", 6);
        $underlyingAsset = new ERC4626Mock(address($asset), "Vault", "vUSDC", 12);

        // Deploy vehicle through factory (initial deposit = 1 USDC = 1e6)
        ($vehicle,) = _deployERC4626Vehicle(
            _deployCoreFactory(),
            ERC4626VehicleFactory.SpawnParams({
                vault: address($underlyingAsset),
                accessControl: ExternalAccessControl(address(0)),
                feeManager: FeeManager(address(0)),
                modulesManager: ModulesManager(address(0)),
                querySalt: keccak256("INITIAL_DEPOSIT_QUERY"),
                deploymentSalt: keccak256("DEPLOYMENT_SALT"),
                initialDepositSize: 1e6, // 1 USDC
                initialExpectedSupply: 1e18
            })
        );

        vm.label(address($asset), "USDC");
        vm.label(address($underlyingAsset), "vUSDC-vault");
        vm.label(address($vehicle), "vehicle");
    }

    /// @notice End-to-end PoC through the vehicle's estimate() function.
    ///
    ///   Setup:
    ///     1. After setUp: totalSupply=1e18, totalAssets=1e6 (1:1 rate)
    ///     2. Alice deposits 2 USDC → totalSupply=3e18, totalAssets=3e6
    ///     3. Simulate loss: set vault USDC to 2e6 → totalSupply=3e18, totalAssets=2e6
    ///
    ///   Bug demonstration:
    ///     estimate([{vehicle, 1e18}], DEPOSIT, INPUT) asks:
    ///       "How many USDC do I need to deposit to get 1e18 shares?"
    ///     Uses Ceil rounding (favor protocol: charge user more).
    ///     math: 1e18 * 2e6 / 3e18 = 666_666.67 → Ceil = 666_667 USDC-wei
    ///     But returns: 666_666 (same as Floor, user is undercharged by 1 USDC-wei)
    function test_vehicle_estimateDepositInput_CeilRoundingBug() public {
        address alice = makeAddr("alice");

        // Step 1: Verify initial state
        assertEq($vehicle.totalSupply(), 1e18, "Initial totalSupply");
        assertEq($vehicle.totalAssets(), 1e6, "Initial totalAssets");
        assertEq($vehicle.decimals(), 18, "Vehicle shares = 18 decimals");
        assertEq($asset.decimals(), 6, "USDC = 6 decimals");
```

```
// Step 2: Alice deposits 2 USDC to get totalSupply=3e18
uint256 depositAmount = 2e6;
Query memory _depositQuery = _get_and_prepare_query(alice, alice, Mode.DEPOSIT,
↪  depositAmount);
vm.prank(alice);
$vehicle.create(_depositQuery);
vm.prank(alice);
$vehicle.unlock(_depositQuery);

assertEq($vehicle.totalSupply(), 3e18, "totalSupply after deposit");
assertEq($vehicle.totalAssets(), 3e6, "totalAssets after deposit");

// Step 3: Simulate a loss — reduce vault's USDC balance to 2e6
// This creates totalSupply=3e18, totalAssets=2e6 (non-trivial exchange rate)
deal(address($asset), address($underlyingAsset), 2e6);
assertEq($vehicle.totalAssets(), 2e6, "totalAssets after simulated loss");
assertEq($vehicle.totalSupply(), 3e18, "totalSupply unchanged");

// Step 4: Estimate DEPOSIT+INPUT — "How many USDC for 1e18 shares?"
// This path uses Ceil rounding (SingleAssetBaseVehicle._convertToAssets with
↪  roundUp=true)
Asset[] memory depositSharesOutput = new Asset[](1);
depositSharesOutput[0] = Asset({asset: address($vehicle), value: 1e18});

Asset[] memory assetsNeeded = $vehicle.estimate(depositSharesOutput, Mode.DEPOSIT,
↪  EstimationType.INPUT);

// Mathematical result: 1e18 * 2e6 / 3e18 = 666_666.666... USDC-wei
// With Ceil: should be 666_667 (round up to charge user more, favor protocol)
// BUG: returns 666_666 (same as Floor, protocol is shortchanged)
assertEq(assetsNeeded[0].asset, address($asset), "Output asset is USDC");
assertEq(assetsNeeded[0].value, 666_666, "BUG: estimate returns 666_666 (should be
↪  666_667)");

// Verify Floor also returns 666_666 using SharesLib directly
uint256 floorResult = SharesLib.convertToAssets(
    1e18, 18, 6, $vehicle.totalSupply(), $vehicle.totalAssets(), Math.Rounding.Floor
);
assertEq(floorResult, 666_666, "Floor = 666_666");

// Ceil == Floor, violating the rounding invariant
assertEq(assetsNeeded[0].value, floorResult, "BUG: Ceil estimate == Floor");

// Step 5: Actually deposit the estimated amount and verify shares received
// The estimate said 666_666 USDC buys 1e18 shares — let's test that claim.
address bob = makeAddr("bob");
uint256 estimatedCost = assetsNeeded[0].value; // 666_666

uint256 bobSharesBefore = $vehicle.balanceOf(bob);
assertEq(bobSharesBefore, 0, "Bob starts with 0 shares");

Query memory _bobDeposit = _get_and_prepare_query(bob, bob, Mode.DEPOSIT,
↪  estimatedCost);
vm.prank(bob);
$vehicle.create(_bobDeposit);
vm.prank(bob);
$vehicle.unlock(_bobDeposit);

uint256 bobSharesAfter = $vehicle.balanceOf(bob);
uint256 actualSharesReceived = bobSharesAfter - bobSharesBefore;
```

```
        // Bob deposited exactly what the estimate said was needed for 1e18 shares,
        // but received fewer: 999_999_000_000_000_000 instead of 1_000_000_000_000_000_000
        // The shortfall is 1_000_000_000_000 (1e12) shares.
        assertEq(actualSharesReceived, 999_999_000_000_000_000, "Actual shares received");
        assertTrue(actualSharesReceived < 1e18, "BUG: Got fewer shares than the estimate
        ↪  promised");
        assertEq(1e18 - actualSharesReceived, 1e12, "Shortfall = 1e12 shares");
    }
}
```

Recommendation: The downscaling should round up when `Ceil` is used.

Note: double rounding up should be avoided.

Kiln: Fixed in commit 10bbcb2.

Spearbit: Fixed. The `rounding` parameter is now used.

However, there is now a double ceil rounding in the function. In the current state of the codebase, it does not have any impact. It should be documented as it could become and issue in the future.

6.2.3 Scoped-role management functions should follow the same sanity rules defined in `AccessControlDefaultAdminRules`

Severity: Low Risk.

Context: ExternalAccessControl.sol#L128, ExternalAccessControl.sol#L141, ExternalAccessControl.sol#L153.

Description: The `grantScopedRole`, `revokeScopedRole` and `renounceScopedRole` are not following the same sanity check rules adopted by the `AccessControlDefaultAdminRules` contract which `ExternalAccessControl` is inheriting from.

The ”grant” operation, even for ”scoped roles”, should **revert** if the `role` is the `DEFAULT_ADMIN_ROLE`.

Unlike `ExternalAccessControl.grantRole`, which invokes `AccessControlDefaultAdminRules.grantRole`, the custom one ”reimplement” the logic.

Note that it will **also** emit the event `RoleGranted`.

You should do the following changes to align the behavior to the one expected by `AccessControlDefaultAdminRules`: `grantScopedRole` should revert with the error `AccessControlEnforcedDefaultAdminRules` when `role == DEFAULT_ADMIN_ROLE`.

```
function grantScopedRole(bytes32 role, address scope, address account)
    public
    virtual
    onlyRole(getRoleAdmin(role))
    notPublicScopedRole(role, scope)
{
    if (role == DEFAULT_ADMIN_ROLE) {
        revert AccessControlEnforcedDefaultAdminRules();
    }
    _grantScopedRole(role, scope, account);
}

function _grantScopedRole(bytes32 role, address scope, address account) internal virtual
↪  returns (bool) {
    bytes32 _encodedRole = _encodeScopedRole(role, scope);
    bool granted = _grantRole(_encodedRole, account);
```

```
    if( granted ) {
        emit ScopedRoleGranted(role, scope, account, _msgSender());
    }
    return granted;
}
```

The same sanity check should be implemented in the `revokeScopedRole` and `renounceScopedRole` functions.

Recommendation: Kiln should revert the `grantScopedRole`, `revokeScopedRole` and `renounceScopedRole` when the `role` is `DEFAULT_ADMIN_ROLE` to follow the same logic and behavior adopted by the `AccessControlDefaultAdminRules` from which it is inheriting from.

Kiln: Acknowledged. Multiple scoped DEFAULT_ADMIN_ROLE holders per scope is intentional behavior. Only the global DEFAULT_ADMIN_ROLE can grant scoped admin roles, scoped admins cannot sub-delegate. This diverges from AccessControlDefaultAdminRules' single-admin constraint by design, as we see valid operational scenarios for multiple scoped admins existing in parallel. Enforcing uniqueness would require a significant refactor to disambiguate global vs encoded scoped roles which we decided against given the trust model and the added complexity it would require. NatSpec has been updated to clearly disclose this behavior.

Spearbit: Acknowledged.

6.2.4 Euler EVault view functions integration safety relies on the governance

Severity: Low Risk.

Context: ERC4626Vehicle.sol#L225.

Description: `_maxRedeem` and `_maxDeposit` integrates with Euler EVault through `_underlyingVault.maxWithdraw(...)` and `_underlyingVault.maxDeposit(...)`.

In Euler docs about hooks, there is the specific mention that:

> ERC-4626 Compliance.
> - Some hook configurations may cause the vault to not be fully ERC-4626 compliant.
> - The max* functions may be inaccurate with hooks installed.

The `maxDeposit` and `maxRedeem` functions may not be accurate. However, the vehicle relies on them.

This ”non-compliance”, combined with the fact that hooks can be modified on EVaults that have a governance can lead to breaking the ERC4626 vehicle (denial of service).

Recommendation: The EVault governance actions should be monitored for vehicles integrating with an EVault.

Kiln: Acknowledged.

Spearbit: Acknowledged.

6.2.5 `AaveV3Vehicle` prevents users from redeeming shares when the Aave reserve is frozen

Severity: Low Risk.

Context: AaveV3Vehicle.sol#L197-L213, BaseVehicle.sol#L205-L214, BaseVehicle.sol#L299-L302.

Description: When the Aave v3 reserve (for the specific underlying) is frozen, only the Supply and Borrow operations are blocked. The user can always repay or withdraw from that reserve.

The current implementation of `AaveV3Vehicle._maxRedeem` is instead setting `assets[0].value = 0` when the reserve is frozen, not allowing the user to redeem the vehicle's shares when the `if (!AssetLib.assetsMatching(query.input, _max)) {` check is executed.

```
{
    Asset[] memory _max = query.mode == Mode.DEPOSIT
        ? _maxDeposit(msg.sender, __totalSupply, __totalAssets)
        : _maxRedeem(msg.sender, __totalSupply, __totalAssets);

    if (!AssetLib.assetsMatching(query.input, _max)) {
        if (query.mode == Mode.DEPOSIT) {
            revert ErrorLib.MaxDepositTooLow(query.input, _max);
        } else {
            revert ErrorLib.MaxRedeemTooLow(query.input, _max);
        }
    }
}
```

This wrong assumption will also influence the value returned by the `BaseVehicle.maxRedeem` when the reserve is frozen and all the integrators and contract that will base their logic on such value.

Recommendation: Kiln should remove the `!_isFrozen` check from the `if (_isActive && !_isPaused && !_isFrozen) {` condition in the `AaveV3Vehicle._maxRedeem` logic.

The `bool _isFrozen` flag can be fully ignored when fetched from the `_poolDataProvider.getReserveConfigurationData(address($.asset));` response.

Kiln: Fixed by commit `cc73b4c40b3cef483ed0554b128ecf45031a23bc`.

Spearbit: Fixed.

6.2.6 `BaseVehicle.ready` is not compliant with the `STEAM` standard

Severity: Low Risk.

Context: BaseVehicle.sol#L380-L383.

Description: The STEAM standard documentation relative to the `ready()` function states.

> Indicates whether the Vehicle is operational and ready to accept queries.
> **MUST** return `true` if the Vehicle is operational and can accept `create()` calls.

The current implementation of the function is **always** returning `true` even if the vehicle has not been **enabled** yet. When the vehicle is not enabled, only the `$.deployer` address will be able to execute the `create` function and continue the query's lifecycle.

Recommendation: Kiln should update the `ready` function to return `true` only when the vehicle has been fully initialized and enabled and everyone can execute the `create` function.

Kiln should also consider to "merge" the `ready()` and `isEnabled()` functions if they express the same meaning.

Kiln: Fixed by commit f3ae2878587c18b4a81ea386c2fa9bc1d6042b4e.

Spearbit: Fixed.

6.2.7 `BaseVehicle._payout` should never be able to revert

Severity: Low Risk.

Context: BaseVehicle.sol#L504-L506, BaseVehicle.sol#L520-L522, BaseVehicle.sol#L524-L526.

Description: The current implementation of the `_payout` flow in the `BaseVehicle` is executed when the Query is in the `RECOVERING` or `UNLOCKING` state.

For the sake of the review we will only describe the `UNLOCKING` scenario given that no vehicle, for this specific review, can enter the `RECOVERING` state.

To be in the `UNLOCKING` scenario, the user must have executed the `create` flow and so deposited funds (underlying or vehicle's share) into the vehicle.

After executing `unlock` the user would receive back the vehicle's shares (if the query has been created with `mode=DEPOSIT`) or underlying (if the query has been created with `mode=REDEEM`).

If the `_payout` function reverts, the user's funds will be stuck inside the vehicle without a way to recover them.

Right now the `_payout` function can revert in three cases.

1) `paidAssets.length != assets.length || (_hasFees && _feeAssets.length != assets.length` → `revert InvalidPayoutArrayLength`.
2) `_payoutAmount + _feeAmount != assets[_idx].value` → `revert BaseVehicleErrors.InvalidPayout`.
3) `_payoutAsset != IERC20(assets[_idx].asset) || _feeAsset != _payoutAsset` → `revert InvalidPayoutAsset`.

In all of these three cases the only possible reason to revert is that the `FeeManager` is not working as expected or is broken.

It's fair to assume that the `FeeManager` is a fully trusted, vetted and always working core component of the Railnet protocol. Given such assumption all of these three revert checks should be removed.

Recommendation: Given the assumption made on the `FeeManager`, Kiln should remove the three mentioned `revert` cases in the `BaseVehicle._payout`.

Kiln: Fixed by commit `d23ce2b1ad361377d9a7daaa4cc591e5dda909e6`.

Spearbit: Fixed.

6.2.8 `FeeManager.onUpdate` should revert when `$.currentFeesConfigId == bytes32(0)`

Severity: Low Risk.

Context: BaseVehicleInitialization.sol#L118, FeeManager.sol#L262-L270.

Description: When `FeeManager.onUpdate` is executed and `$.currentFeesConfigId` is equal to `bytes32(0)` it means that the `__FeeManager_init` has not been executed yet.

When a Vehicle is deployed and initialized, the `onUpdate` function is **always** executed. The Vehicle, should not be able to initialize itself if the `FeeManager` has not been also properly initialized.

Recommendation: Kiln should revert in the `onUpdate` flow if the `FeeManager` has not been properly initialized yet.

Kiln: Fixed by commit `e1d58cee3f90fb81f4535e16b2649679d1de3e6b`.

Spearbit: Fixed. Kiln has decided to avoid reverting directly in the `FeeManager.onUpdate` as suggested and has opted to revert during `BaseVehicleInitialization.__BaseVehicle_init` when the Fee Config ID returned by the `optionalFeeManager_` is equal to `bytes32(0)`.

6.2.9 Aave offers multiple distribution reward systems which are currently not supported by the Aave V3 Vehicle

Severity: Low Risk.

Description: The current Aave V3 vehicle implementation does only support the "Aave Reward Controller" reward system which seems to be going to be deprecated in favor of other systems.

In the specific, by looking at least at their "Core Instance" markets for the Mainnet chain we see that rewards are distributed via:

- Custom reward programs offered "directly" by the underlying protocol.
  - `Wrapped eETH` has a "custom" rewards program that is distributed **directly** by Ether.fi (Loyalty Points). See loyalty-points.
  - `USDe` has a "custom" rewards program (+ Merkl distributor) that is distributed **directly** by Ethena (Referral). See join.
  - `USDe` has a "custom" rewards program that is distributed **directly** by Kernel DAO (Kernel Points). See kernel-points-guide.
- ACI Merit Program: This is a program initiated and implemented by the Aave Chan Initiative (ACI).
- Merkl: This is a program initiated by the Aave DAO and implemented by Merkl. An integration guide can be seen here: earning-with-merkl.

Recommendation: Kiln should contact the Aave DAO to.

- Confirm that the old "Aave Reward Controller" system has been deprecated. If this is confirmed the existing `AaveV3Merkl` module can be removed from the codebase.
- List and integrate all the other reward distribution systems that are used for the existing market.

Kiln: Fixed by commit `65dcdd174c20eea6fafb5c4e91cc9dca1b2f5c0e`.

We decided to remove the Aave V3 module and focus on Merkl rewards only at launch via the Interceptor pattern.

Spearbit: Fixed. Note: Aave V3 rewards are not distributed only via Merkl but also via other custom distribution channels.

6.2.10 The `AaveV3Merkl` must be fully refactored

Severity: Low Risk.

Description: The current implementation of the `AaveV3Merkl` contains multiple critical issues and wrong assumptions that need to be fixed.

1) The `_rewardAsset` passed to `rewardsController.claimAllRewards` is **NOT** the underlying token (like `USDC`) but the `AToken` or `VToken` (in our case it will always be an `AToken` given that the protocol never borrows and only supplies). This wrong assumption does not only influence the interaction with the reward controller but the whole logic that needs a full refactor of the module itself.
2) For a single asset (`AToken`) you can receive **multiple** rewards, not a single one.
3) For a single asset (`AToken`) you could receive as a reward (part of the multiple rewards accrued) the asset itself (an `AToken`).

Recommendation: ⚠ NOTE: it's high likely that the "Aave Reward Controller" reward distribution system has been already discontinued and won't be used anymore by Aave. If that's the case, the whole module can be just removed from the codebase without applying any fixes.

Kiln **must** refactor the whole `AaveV3Merkl` to properly integrate with the Aave Reward system. Below are some suggestions that can be used to kickstart the refactoring.

The `AaveV3MerklModule.exec` function is executed in this way: someone calls `ModulesManager.exec(aaveVehicle, marklModuleId, merklRewardDistributionCampaignData)`. This will execute `aaveVehicle.exec(marklModuleId, merklRewardDistributionCampaignData)` which will execute in `DELEGATECALL` mode `AaveV3MerklModule.exec(marklModuleId, merklRewardDistributionCampaignData)`. This means that the `AaveV3MerklModule.exec` logic is executed with the `AaveV3Vehicle` context.

1) Get the vehicle's underlying asset by executing `address baseVaultAsset = _asset()`.
2) Get the corresponding `aToken` by calling `(address _aToken,,) = _poolDataProvider.getReserveTokensAddresses(baseVaultAsset)`.
3) Execute `(address[] memory rewardsList, uint256[] memory claimedAmounts) = rewardsController.claimAllRewardsToSelf([baseVaultAsset])`.
4) Iterate over the returned list to create, for each `rewardsList[i]` a Merkl Distribution if `claimedAmounts[i] > 0`.

Relative to the Merkl distribution:

There are some important things to note and be aware of when `distributionCreator.createCampaign` is executed (or the bulk version of the function called `createCampaigns`).

1) The `AaveV3Vehicle` must have SIGNED the `distributionCreator` agreement before calling `distributionCreator.createCampaign`. `AaveV3Vehicle` must execute (one time) `distributionCreator.acceptConditions()` otherwise `distributionCreator.createCampaign` will revert.
2) Check if `rewardsList[i]` has been whitelisted `distributionCreator.rewardTokenMinAmounts(rewardsList[i]) > 0`. What should the Module do if it has not been whitelisted? Avoid calling the `rewardsController.claimAllRewardsToSelf` for that reward?
3) Check if the reward amount is enough to satisfy the requirement `claimedAmounts[i] * HOUR >= distributionCreator.rewardTokenMinAmounts(rewardsList[i]) * newCampaign.duration`. What should the Module do if `claimedAmounts[i]` is not enough? Avoid calling the `rewardsController.claimAllRewardsToSelf` for that reward?

⚠ NOTE 2: Merkl could take a fee on the distributed rewards. The final amount of rewards received by the users could be less than `claimedAmounts[i]`.

⚠ NOTE 1: it's possible to claim, for an `asset` just a **specific** reward. The logic can be changed, it would be more complex but it's not so problematic.

Kiln: Fixed by commit `65dcdd174c20eea6fafb5c4e91cc9dca1b2f5c0e`.

We decided to remove the Aave V3 module and focus on Merkl rewards only at launch via the Interceptor pattern.

Spearbit: Fixed.

6.2.11 Incorrect Asset Accounting if Vehicle Is Morpho Market Fee Recipient

Severity: Low Risk.

Context: MorphoBlueVehicle.sol#L129-L137.

Description: The `MorphoBlueVehicle` calculates its total assets using Morpho's helper and relies on `MorphoBalancesLib.expectedSupplyAssets`. However, Morpho explicitly documents that this function is **incorrect when the queried address is the market `feeRecipient`**:

```
/// @dev Warning: Wrong for `feeRecipient` because their supply shares increase is not taken
↪  into account.
```

In Morpho Blue, protocol fees are realized by minting additional supply shares to the `feeRecipient` during interest accrual. This occurs in the core Morpho contract when fees are applied to the market's supply shares, sending them to the `feeRecipient` address.

Because `expectedSupplyAssets` does not correctly account for these fee shares, querying it for the `feeRecipient` will produce an incorrect asset value.

Recommendation: Ensure that the vehicle address can never be configured as the Morpho market `feeRecipient`.

Also consider:

- Document this as a hard integration invariant.
- Enforce it during initialization by verifying that the market's `feeRecipient` is not the vehicle address. It must be noted, that even if this check passes, owner can later change the `feeRecipient`.

Kiln: This is unlikely to happen and as the `feeRecipient` can change, we will document it and not enforce it at initialisation. Fixed by PR 375.

Spearbit: Verified fix. The documentation now explicitly mentions this behaviour.

6.2.12 Effects of rounding on both the vault's shares and underlying protocol shares

Severity: Low Risk.

Description: This is research I've done to explore the effects that the rounding directions can have on both the user's vault's share and the underlying protocol's share. In this specific case I have used Aave V3 as the underlying protocol (and vehicle) given that it's the one I'm more familiar with.

In a vehicle the user can perform two operations: deposit assets and redeem the vault's share (at least for the Single Asset type of Vehicles). The underlying protocol will (mostly) always defend the protocol's itself by rounding "against" the user (and in favor of the protocol):

- During a `DEPOSIT` operation (supply on Aave) Aave V3 will **round DOWN** the amount of shares minted to the Vehicle.
- During a `REDEEM` operation (withdraw on Aave, Aave V3 does not have a `redeem` function) Aave V3 will **round UP** the amount of shares to be burned.
- The `balanceOf` operation will **round DOWN** the balance of the user (the vehicle in our case).

On the Vehicle's side, the Vehicle will follow these rounding rules:

- During a `DEPOSIT` operation the Vehicle will **round DOWN** the amount of shares minted to the user: `SharesLib.convertToShares(..., Math.Rounding.Floor)`.
- During a `REDEEM` operation the Vehicle will **round DOWN** the amount of assets to be withdrawn by redeeming the specified shares. `SharesLib.convertToAssets(..., Math.Rounding.Ceil)`.

⚠ Note: the amount of shares minted / burned by the protocol to/from the user during a `DEPOSIT` or `REDEEM` operation does not "match" the one minted/burned by the underlying protocol's as a result of the operation. The Vehicle use it's own "logic" implemented in the `Shares` contract:

```
function convertToAssets(
    uint256 shares,
    uint8 sharesDecimals,
    uint8 assetDecimals,
    uint256 totalSupply,
    uint256 totalAssets,
    Math.Rounding rounding
) internal pure returns (uint256 assets) {

    if (totalSupply == 0) {
        return scale(shares, sharesDecimals, assetDecimals);
    }
    return downscaleAssets(
        Math.mulDiv(
            upscaleShares(shares, sharesDecimals, assetDecimals),
            upscaleAssets(totalAssets, sharesDecimals, assetDecimals),
            upscaleShares(totalSupply, sharesDecimals, assetDecimals),
            rounding
        ),
        sharesDecimals,
        assetDecimals
    );
}

function convertToShares(
    uint256 assets,
    uint8 sharesDecimals,
    uint8 assetDecimals,
    uint256 totalSupply,
    uint256 totalAssets,
    Math.Rounding rounding
) internal pure returns (uint256 shares) {
    if (totalSupply == 0) {
        return scale(assets, assetDecimals, sharesDecimals);
    }
    if (totalAssets == 0) {
        revert ZeroTotalAssets();
    }
    return downscaleShares(
        Math.mulDiv(
            upscaleAssets(assets, sharesDecimals, assetDecimals),
            upscaleShares(totalSupply, sharesDecimals, assetDecimals),
```

```
            upscaleAssets(totalAssets, sharesDecimals, assetDecimals),
            rounding
        ),
        sharesDecimals,
        assetDecimals
    );
}
```

Because there's not a 1:1 match between the Vehicle's shares minted/burned to/from the user and the underlying protocol's shares minted/burned to/from the Vehicle (on behalf of the user) the reasoning around the rounding becomes more complex.

The result of each operation and the rounding effect depends on the current exchange rate (on the vehicle) and the exchange rate on the underlying protocol (in the case of Aave V3, we use an **index**, not a ”real” exchange rate).

By Fork testing and fuzzing we can see that there are **both** cases where the user operation **could erode (LOSS)** the Vehicle's original underlying balance in the protocol or it could even make the Vehicle **EARN** because the user is leaving money on the plate.

The Fork test I have created performs **1 single deposit** and **multiple chunked redeem**. What I can assume given how the Vehicle's share logic works and how Aave's share logic works, is that.

1) When the Vehicle's **LOSS** is mainly because Aave is rounding against the Vehicle and the Vehicle is **NOT** rounding against the user.
2) When the Vehicle's **EARN** is because Aave is **NOT** rounding against the Vehicle and the Vehicle is rounding against the user.

It's important to note that **every** loss or earn consequences that influence the Vehicle are **also** influencing the existing suppliers' balances. When the Vehicle ”earn” the existing suppliers' share value increases, when the Vehicle ”lose” the existing suppliers' share value decreases.

The test: There are two fuzz tests implemented that internally apply the same logic but they try to prove two different things:

- `testStudyEarn` try to prove that the Vehicle can **EARN** more than `threshold` wei at the end of the test.
- `testStudyLoss` try to prove that the Vehicle can **LOSE** more than `threshold` wei at the end of the test.

Both the tests will always perform the following actions:

1) Deploy the Vehicle with an initial deposit of 1 unit of asset (in this case 1 USDC).
2) `u1` deposit the fuzzed `initialDeposit` amount of USDC.
3) `u1` will redeem in loop `split` times an amount of shares equal to `$vehicle.balanceOf(u1) / split`. The last iteration will redeem the whole remaining balance.
4) Depending on the `proofType` it will try to assert that the Vehicle is indeed earning/losing `threshold` of wei.

```
// SPDX-License-Identifier: BUSL-1.1
pragma solidity >=0.8.33;

import {
    STEAMSingleAssetTester,
    IERC20Metadata,
    AaveV3VehicleFactory,
    Asset,
    Query,
    State,
    Mode,
    EstimationType,
    ErrorLib,
```

```
    IERC20,
    Tester,
    Route
} from "test/vehicles/aave_v3/AaveV3Vehicle.t.sol";
import {IPoolAddressesProvider} from
↪  "src/vehicles/aave_v3/interfaces/IPoolAddressesProvider.sol";
import {IPoolDataProvider} from "src/vehicles/aave_v3/interfaces/IPoolDataProvider.sol";
import {ForkUtils} from "test/test_utils/ForkUtils.sol";
import {IExternalAccessControl, ExternalAccessControl} from
↪  "src/common/ExternalAccessControl.sol";
import {FeeManager} from "src/vehicles/base/FeeManager.sol";
import {ModulesManager} from "src/vehicles/base/ModulesManager.sol";
import {Vm} from "@forge-std/Test.sol";
import {CoreFactory, ICreateX} from "src/factories/CoreFactory.sol";
import {Roles} from "src/libs/Roles.sol";
import {AccessControlFactory} from "src/factories/common/AccessControlFactory.sol";
import {ModulesManagerFactory} from "src/factories/vehicles/ModulesManagerFactory.sol";
import {AaveV3MerklModule} from "src/vehicles/aave_v3/modules/AaveV3Merkl.mod.sol";
import {SafeERC20} from "@openzeppelin-contracts/token/ERC20/utils/SafeERC20.sol";
import {IDistributionCreator} from "src/vehicles/base/interfaces/IDistributionCreator.sol";
import {BaseVehicleErrors} from "src/vehicles/base/abstracts/BaseVehicleErrors.sol";
import 'forge-std/console.sol';

interface IAToken {
    function balanceOf(address account) external view returns (uint256);
    function scaledBalanceOf(address user) external view returns (uint256);
}

interface IPool {
    function deposit(address asset, uint256 amount, address onBehalfOf, uint16 referralCode)
    ↪  external;
}

contract _SErosionStudySingleTest is STEAMSingleAssetTester, ForkUtils {
    using Tester for Query;
    using SafeERC20 for IERC20;

    address internal constant BURN_ADDRESS =
    ↪  address(0x000000000000000000000000000000000000dEaD);
    address internal constant $USDC = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;
    address internal constant $DAI = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
    address internal constant $poolAddressesProvider =
    ↪  0x2f39d218133AFaB8F2B819B1066c7E434Ad94E9e;
    address internal constant $distributionCreator =
    ↪  0x8BB4C975Ff3c250e0ceEA271728547f3802B36Fd;
    address internal $pool;
    address internal $poolDataProvider;
    ExternalAccessControl internal $accessControl;
    ModulesManager internal $modulesManager;
    Vm.Wallet private $admin;
    CoreFactory _coreFactory;
    uint256 salt;

    function setUp() external {
        _startFork("MAINNET");
        $asset = IERC20Metadata($USDC);
        $admin = vm.createWallet("admin");

        _coreFactory =
        ↪  _deployCoreFactory(ICreateX(0xba5Ed099633D3B313e4D5F7bdc1305d3c28ba5Ed));
```

```
        ExternalAccessControl.RoleAttribution[] memory _roles = new
        ↪  IExternalAccessControl.RoleAttribution[](5);
        _roles[0] = IExternalAccessControl.RoleAttribution({role: Roles.EXEC, account:
        ↪  $admin.addr});
        _roles[1] = IExternalAccessControl.RoleAttribution({role: Roles.MODULE_MANAGER,
        ↪  account: $admin.addr});
        _roles[2] = IExternalAccessControl.RoleAttribution({role: Roles.CANCEL_MODULE,
        ↪  account: $admin.addr});
        _roles[3] = IExternalAccessControl.RoleAttribution({role: Roles.UPDATE_TIMELOCK,
        ↪  account: $admin.addr});
        _roles[4] = IExternalAccessControl.RoleAttribution({role: Roles.VEHICLE_ALLOW,
        ↪  account: $admin.addr});

        ($accessControl,) = _deployAccessControl(
            _coreFactory,
            AccessControlFactory.SpawnParams({
                initialDelay: 0,
                initialDefaultAdmin: makeAddr("access_control_admin"),
                initialRoles: _roles,
                deploymentSalt: keccak256("vehicle_access_control")
            })
        );

        ($modulesManager,) = _deployModulesManager(
            _coreFactory,
            ModulesManagerFactory.SpawnParams({
                initialTimelock: 0, accessControl: $accessControl, deploymentSalt:
                ↪  "MODULES_MANAGER"
            })
        );

        ($vehicle,) = _deployAaveV3Vehicle(
            _coreFactory,
            AaveV3VehicleFactory.SpawnParams({
                asset: address($asset),
                poolAddressesProvider: $poolAddressesProvider,
                accessControl: $accessControl,
                feeManager: FeeManager(address(0)),
                modulesManager: $modulesManager,
                querySalt: keccak256("INITIAL_DEPOSIT_QUERY"),
                deploymentSalt: keccak256("DEPLOYMENT_SALT"),
                initialDepositSize: _unit(1),
                initialExpectedSupply: 1e18
            })
        );

        vm.startPrank(makeAddr("access_control_admin"));
        $accessControl.setRolePublic(Roles.VEHICLE_STEAM, true);
        vm.stopPrank();

        $pool = IPoolAddressesProvider($poolAddressesProvider).getPool();
        $poolDataProvider =
        ↪  IPoolAddressesProvider($poolAddressesProvider).getPoolDataProvider();
        $multiDepositRangeMin = 4;
        $multiDepositRangeMax = 16;


        vm.startPrank($admin.addr);
        IDistributionCreator($distributionCreator).acceptConditions();
        vm.stopPrank();
    }
```

```
enum ProofType {
    LOSS,
    EARN
}

function testStudyEarn(uint256 initialDeposit, uint256 split) public {
    initialDeposit = bound(initialDeposit, _unit(1), _unit(1_000_000));
    split = bound(split, 1, 1000);
    execTest(ProofType.EARN, 10, initialDeposit, split);
}

function testStudyLoss(uint256 initialDeposit, uint256 split) public {
    initialDeposit = bound(initialDeposit, _unit(1), _unit(1_000_000));
    split = bound(split, 1, 1000);
    execTest(ProofType.LOSS, 10, initialDeposit, split);
}

function execTest(ProofType proofType, uint256 threshold, uint256 initialDeposit, uint256
↪   split) public {
    address es = makeAddr("existingSupplier");
    _deposit(es, _unit(1));

    address u1 = makeAddr("u1");

    (uint256 aTokenScaledBalanceStart, uint256 aTokenBalanceStart) =
    ↪   fetchVehicleBalances(address($asset));
    uint256 vShareDeposit = _deposit(u1, initialDeposit);
    uint256 u1Shares = $vehicle.balanceOf(u1);
    uint256 redeemAmount = u1Shares / split;

    for( uint256 i = 0; i < split; i++ ) {
        (uint256 aTokenScaledBalanceBeforeRedeem, uint256 aTokenBalanceBeforeRedeem) =
        ↪   fetchVehicleBalances(address($asset));
        if( i == split - 1) redeemAmount = $vehicle.balanceOf(u1);
        uint256 tokenWithdrawn = _redeem(u1, redeemAmount);
        (uint256 aTokenScaledBalanceAfterRedeem, uint256 aTokenBalanceAfterRedeem) =
        ↪   fetchVehicleBalances(address($asset));
        console.log('v token burned.  ',
        ↪   aTokenScaledBalanceBeforeRedeem-aTokenScaledBalanceAfterRedeem);
        console.log('u token burned.  ', redeemAmount);
        console.log('v token withdrawn',
        ↪   aTokenBalanceBeforeRedeem-aTokenBalanceAfterRedeem);
        console.log('u token withdrawn', tokenWithdrawn);

    }
    (uint256 aTokenScaledBalanceEnd, uint256 aTokenBalanceEnd) =
    ↪   fetchVehicleBalances(address($asset));

    console.log('vb ini', aTokenBalanceStart);
    console.log('vb end', aTokenBalanceEnd);
    console.log('ub ini', initialDeposit);
    console.log('ub end', IERC20(address($asset)).balanceOf(u1));

    // how much has changed the existing suppliers balance in value?
    console.log('es value', estimateUserBalance(es));

    if( aTokenBalanceEnd > aTokenBalanceStart ) {
        // Vehicle is EARNING
        console.log('v EARN', aTokenBalanceEnd-aTokenBalanceStart);
        console.log('u LOSS', initialDeposit-IERC20(address($asset)).balanceOf(u1));
        if( proofType == ProofType.EARN ) assertLt(aTokenBalanceEnd-aTokenBalanceStart,
        ↪   threshold);
```

```
    } else if ( aTokenBalanceEnd < aTokenBalanceStart ) {
        // Vehicle is LOSING
        console.log('v LOSS', aTokenBalanceStart-aTokenBalanceEnd);
        console.log('u LOSS', initialDeposit-IERC20(address($asset)).balanceOf(u1));
        if( proofType == ProofType.LOSS ) assertLt(aTokenBalanceStart-aTokenBalanceEnd,
        ↪  threshold);
    } else {
        // no-loss-earn scenario
    }
}

function estimateUserBalance(address user) public returns (uint256) {
    uint256 userVehicleBalance = $vehicle.balanceOf(user);
    if(userVehicleBalance == 0) return 0;
    Asset[] memory assets = new Asset[](1);
    assets[0] = Asset({asset: address($vehicle), value: userVehicleBalance});
    Asset[] memory output = $vehicle.estimate(assets, Mode.REDEEM, EstimationType.OUTPUT);
    return output[0].value;
}

function _deposit(address user, uint256 amount) public returns (uint256 output) {
    // uint256 _depositAmount = _unit(amount);
    uint256 _depositAmount = amount;
    deal(address($asset), user, _depositAmount);
    vm.startPrank(user);
    IERC20(address($asset)).approve(address($vehicle), _depositAmount);
    Asset[] memory _depositAssets = new Asset[](1);
    _depositAssets[0] = Asset({asset: address($asset), value: _depositAmount});
    Asset[] memory _depositOutput = new Asset[](1);
    _depositOutput[0] = Asset({asset: address($vehicle), value: 0});
    Query memory _query = Query({
        owner: user,
        receiver: user,
        input: _depositAssets,
        mode: Mode.DEPOSIT,
        salt: bytes32(salt++),
        output: _depositOutput,
        data: ""
    });
    $vehicle.create(_query);
    (State newState, Asset[] memory assets) = $vehicle.unlock(_query);
    vm.stopPrank();

    return assets[0].value;
}

function _redeem(address user, uint256 amount) public returns (uint256 output) {
    Asset[] memory _assets = new Asset[](1);
    _assets[0] = Asset({asset: address($vehicle), value: amount});
    Asset[] memory _output = new Asset[](1);
    _output[0] = Asset({asset: address($asset), value: 0});
    Query memory _query = Query({
        owner: user,
        receiver: user,
        input: _assets,
        mode: Mode.REDEEM,
        salt: bytes32(salt++),
        output: _output,
        data: ""
    });

    vm.startPrank(user);
```

```
        $vehicle.create(_query);
        (State newState, Asset[] memory assets) = $vehicle.unlock(_query);
        vm.stopPrank();

        return assets[0].value;
    }

    function fetchVehicleBalances(address underlying) public returns (uint256, uint256) {
        (address _aToken,,) =
        ↪  IPoolDataProvider($poolDataProvider).getReserveTokensAddresses(underlying);
        uint256 aTokenScaledBalance = IAToken(_aToken).scaledBalanceOf(address($vehicle));
        uint256 aTokenBalance = IAToken(_aToken).balanceOf(address($vehicle));
        return (aTokenScaledBalance, aTokenBalance);
    }

}
```

Recommendations: Kiln should:

1) Deploy enough initial assets into the Vehicle (when the vehicle is initially deployed) to avoid any possible "erosion" attack vector to be performed by cycling the redeem operation that could reduce the Vehicle's initial total assets in the underlying protocol.
2) Monitor the Vehicle's "erosion" issue during the Vehicle's lifecycle and have pre-defined plan to eventually donate on behalf of the vehicle to reimburse the loss in share price that the existing suppliers could incur.
3) Document and disclose the "erosion" behavior: the user's actions **could** erode the vehicle's total assets in the underlying protocol, reducing, as a consequence, the existing suppliers share's value.

Kiln: Fixed by commit `152376e7b86c94009f3c2817ee9b067a03bb5b85`.

Spearbit: Kiln has introduced the `AssetRegistry` contract that will.

- Manage a whitelist of assets that the vehicles can pick from when deployed.
- Manage, for each asset, the minimum asset's amount to be deployed and deposited on the vehicle's deployment.

Kiln has also documented and acknowledged the "erosion" issue described by the Finding.

6.3 Informational

6.3.1 Use of deprecated contract comparison operators

Severity: Informational.

Context: AccessControl.sol#L71.

Description: The solidity 0.8.31 version release deprecates the comparison of contract type variables.

> Comparison of variables of contract type is deprecated and scheduled for removal. Use an explicit cast to address type and compare the addresses instead.

Reference:

- solidity-0.8.31-release-announcement.

Recommendation: Cast the contract type variables to address before comparison.

Kiln: Fixed in commit cdfb38f.

Spearbit: Fixed. The contract type comparisons have been replaced by address comparisons.

6.3.2 STEAM specifications precision about redeem optional output

Severity: Informational.

Description: The STEAM specifications state the following about the output asset of a query:

> For `DEPOSIT` operations, this optionally specifies the output asset the user expects to receive as shares or other tokens resulting from the deposit. For `REDEEM` operations, this specifies the desired output asset the user wishes to receive from the Vehicle upon redemption.

The implementation of `SingleAssetBaseVehicle` shows that both `DEPOSIT` and `REDEEM` operations use optional outputs.

Recommendation: Add details in the specifications that the asset output is also optional for `REDEEM` operations.

> For `DEPOSIT` operations, this optionally specifies the output asset the user expects to receive as shares or other tokens resulting from the deposit. For `REDEEM` operations, this **optionally** specifies the desired output asset the user wishes to receive from the Vehicle upon redemption.

Kiln: Fixed in commit 555323a.

Spearbit: Fixed. The specifications section has been refactored and now correctly mentions that both deposit and redeem operations use the `output` array as an optional field.

6.3.3 Scaling factor calculation in `_previewOnOperations` is prone to edge-cases

Severity: Informational.

Context: FeeManager.sol#L496.

Description: The `FeeManager._previewOnOperations` function calculates and uses a scaling factor in case `_totalFeeAssets > _maxFees`.

In such calculation, if `totalFeeAssets` is greater than `10 ** assetDecimals * _maxFees` then `_scalingFactor = 1`. After this, `_managementFeeAssets = _managementFeeAssets * 1 / 10 ** assetDecimals`. In extreme edge-cases, this could still result in `_totalFeeAssets > currentTotalAssets` which would later revert.

```
function _previewOnOperations(
    // ...

    uint256 _totalFeeAssets = _performanceFeeAssets + _managementFeeAssets;
    // Cap fee assets to prevent underflow when cache is uninitialized (lastTimestamp == 0)
    uint256 _maxFees = Math.mulDiv(currentTotalAssets, ConstantLib.BPS_MAX - 1,
    ↪  ConstantLib.BPS_MAX);
    if (_totalFeeAssets > _maxFees) {
        // _performanceFeeAssets and _managementFeeAssets are capped proportionally
        uint256 _scalingFactor = Math.mulDiv(10 ** assetDecimals, _maxFees, _totalFeeAssets,
        ↪  Math.Rounding.Ceil); // @audit
        _performanceFeeAssets =
```

```
            Math.mulDiv(_performanceFeeAssets, _scalingFactor, 10 ** assetDecimals,
             ↪  Math.Rounding.Floor);
        _managementFeeAssets =
            Math.mulDiv(_managementFeeAssets, _scalingFactor, 10 ** assetDecimals,
             ↪  Math.Rounding.Floor);
        _totalFeeAssets = _performanceFeeAssets + _managementFeeAssets;
    }
    if (_totalFeeAssets != 0) {
        uint256 _totalAssetsWithoutFees = currentTotalAssets - _totalFeeAssets; // @audit could
         ↪  revert
```

Note: This issue seems unlikely to be exploitable. However, the recommended fix erases the edge-case while providing more readability.

Recommendation: Consider removing the scaling factor calculation and simplifying `_performanceFeeAssets` and `_managementFeeAssets`.

The following patch ensures that `_totalFeeAssets` can not be greater than `_maxFees` while removing the scaling factor.

```
  diff --git a/src/vehicles/base/FeeManager.sol b/src/vehicles/base/FeeManager.sol
  index 3042b703..0ef8eb65 100644
-  -- a/src/vehicles/base/FeeManager.sol
+  ++ b/src/vehicles/base/FeeManager.sol
  @@ -493,11 +493,9 @@ contract FeeManager is IFeeManager, Interceptor,
   ↪  ReentrancyGuardUpgradeable, Ext
            uint256 _maxFees = Math.mulDiv(currentTotalAssets, ConstantLib.BPS_MAX - 1,
             ↪  ConstantLib.BPS_MAX);
            if (_totalFeeAssets > _maxFees) {
                // _performanceFeeAssets and _managementFeeAssets are capped proportionally
-               uint256 _scalingFactor = Math.mulDiv(10 ** assetDecimals, _maxFees,
↪  _totalFeeAssets, Math.Rounding.Ceil);
-               _performanceFeeAssets =
-                   Math.mulDiv(_performanceFeeAssets, _scalingFactor, 10 ** assetDecimals,
↪  Math.Rounding.Floor);
-               _managementFeeAssets =
-                   Math.mulDiv(_managementFeeAssets, _scalingFactor, 10 ** assetDecimals,
↪  Math.Rounding.Floor);
+               _performanceFeeAssets = Math.mulDiv(_performanceFeeAssets, _maxFees,
↪  _totalFeeAssets, Math.Rounding.Floor);
+               _managementFeeAssets = _maxFees - _performanceFeeAssets;
+               // @audit or use: _managementFeeAssets = Math.mulDiv(_managementFeeAssets,
↪  _maxFees, _totalFeeAssets, Math.Rounding.Floor);
                _totalFeeAssets = _performanceFeeAssets + _managementFeeAssets;
            }
            if (_totalFeeAssets != 0) {
```

Kiln: Fixed in commit a75c49c.

Spearbit: Fixed. Recommended patch has been applied.

6.3.4 Natspec issues

Severity: Informational.

Context: ExternalAccessControl.sol#L158, Factory.sol#L47-L58, Vehicle.sol#L74-L78, Vehicle.sol#L92-L96, Vehicle.sol#L111-L117, Vehicle.sol#L135-L141, BaseVehicleChecks.sol#L34-L36, BaseVehicle.sol#L670, BaseVehicle.sol#L699, FeeManager.sol#L492.

Description: Several natspec comments and documentation strings are inaccurate, incomplete, or misleading:

- ☒ FreezablePausableBeacon.sol#L153: The `getStatus()` note states "implementationAddress may be inaccessible if the beacon is paused", but the paused state has no impact on retrieving the `$implementation` storage variable. Only the `implementation()` view function is affected by the pause.
- ☒ IFreezablePausableBeacon.sol#L48: The `BeaconFrozen()` error comment states it is "thrown when attempting to upgrade a frozen beacon". In practice, the error is also thrown on `pause()` and `pauseFor()` calls, and when attempting to freeze an already frozen beacon.
- ☒ IFreezablePausableBeacon.sol#L58: The `ExpectedPause()` error is declared but never used.
- ☒ IFreezablePausableBeacon.sol#L65: The `upgradeTo` natspec states "Can only be called when not frozen and not paused", but the implementation only enforces `whenNotFrozen`. The "and not paused" mention is incorrect and should be removed.
- ☒ IExternalAccessControl.sol#L80-L94: The natspec for `grantScopedRole()` and `revokeScopedRole()` states "Can only be called by accounts with DEFAULT_ADMIN_ROLE". The actual modifier is `onlyRole(getRoleAdmin(role))`, which is not necessarily `DEFAULT_ADMIN_ROLE`.
- ☒ IVehicle.sol#L92: The `error()` function natspec states it "Gets the error encountered by a query that led to RECOVERING or REJECTED state", but in practice `BaseVehicle.error` only returns an error when the state is `REJECTED`, not `RECOVERING`.
- ☒ Asset.sol#L65: The comment for output matching describes the opposite of the actual logic.
- ☒ Error.sol#L39: The `ZeroLength()` error comment is a copy-paste of the `ZeroValue()` comment. It should read "The error thrown when the length is 0.".
- ☒ Error.sol#L68: Typo: "Raise" should be "Raised".
- ☒ Factory.sol#L146: The return parameter of `deployContract` is named `vehicle` but should be `contract`.
- ☒ Factory.sol#L248: The `generateVehicleSymbol` natspec does not document the `count` parameter.
- ☒ Rejection.sol#L66: The natspec for `rejectWithInvalidInput` and `rejectWithInvalidOutput` states that current inputs/outputs are returned for comparison, but only the query inputs/outputs are encoded.
- ☒ Roles.sol#L103: Typo: "preform" should be "perform".
- ☒ Shares.sol#L23: The `ZeroTotalAssets()` error comment states it is thrown "during conversion", but it is only thrown during `convertToShares`.
- ☒ Vehicle.sol#L163: The `@return` comment ends with "false" and should read "false otherwise".
- ☒ BaseVehicleChecks.sol#L49: Comment says "Prevents calls when the vehicle is not in the expected state" but it should say "when the query is not in the expected state".
- ☒ BaseVehicleErrors.sol#L35: The errors `InvalidReceiver()`, `InvalidOwner()`, and `InvalidMode(Mode expected, Mode actual)` are declared but never used.
- ☒ BaseVehicleInitialization.sol#L71: The `forbiddenAddresses` param comment says "forbidden to create queries" but should say "forbidden to create or receive queries".
- ☒ BaseVehicleQueryStateSetter.sol#L89: Comment references `create` as a function that can transition to `REJECTED`, but `create()` no longer transitions to `REJECTED` per the current specs.
- ☒ FeeManager.sol#L50: `FeeManager` mentions `redeemVehicleShares` but it can't be found in the contract.
- ☒ FeeManager.sol#L78: Comment says "ModulesManager", but it is set on the "FeeManager".

- ☒ FeeManager.sol#L449: Comment indicates "Performance fee is calculated on remaining profit (after management fee deduction)". However this is not accurate anymore. Performance fees are taken on top of management fees.
- ☐ FeeManager.sol#L492: remove the mentioned comment because it is referencing a scenario that does not exist anymore.
- ☒ IBaseVehicle.sol#L34-L36: Comment uses the outdated STEAM states "Empty → Pending → Accepted → Processing → Unlocking → Settled".
- ☒ IFeeManager.sol#L57: Comment mentions "Sets the maximum allowed fees" but it actually sets the fees and not the maximum.
- ☒ IFeeManager.sol#L81: The comment mentions that the state is being modified, but there is no state being updated and only an event is being emitted.
- ☒ ExternalAccessControl.sol#L158: The `ExternalAccessControl.setRolePublic` function references the wrong function. The scoped `DEFAULT_ADMIN_ROLE` role can be made public by using the `setRolePublic` and not the `setScopedRolePublic`. Rewrite the natspec comment correctly. Example: "However, scoped `DEFAULT_ADMIN_ROLE` can be made public through `setRolePublic()`. If passed an encoded role directly the code has no way to differentiate and prevent a scoped `DEFAULT_ADMIN_ROLE` being made public.".
- ☒ Factory.sol#L48-L49: The `Factory.InitialDepositParams` natspec is outdated. The `asset` and `amount` parameters have been replaced by `inputAsset` and `outputAsset`. Update it accordingly. Consider also to rename `amount` to `inputAmount` given that it will **exclusively** be used for the `input` parameter of the `Query`.
- ☒ Vehicle.sol#L74-L157: Consider improving the natspec of the `convertToAssets`, `convertToShares`, `estimateShares` and `estimateAssets` functions in the `Vehicle` to explicitly disclose which fees are applied in the underlying `vehicle.estimate` call.
- ☒ BaseVehicleChecks.sol#L34-L36: The `BaseVehicleChecks._onlyRoleWhenEnabled` function's natspec should explicitly disclose with a `@dev` comment that the function won't perform any role check if the Access Control has not been configured. Consider also renaming the function to be explicit in this behavior.
- ☒ BaseVehicle.sol#L670 + BaseVehicle.sol#L699: Both the `_maxDepositWithFees` and `_maxRedeemWithFees` functions natspec in the `BaseVehicle` contract wrongly use the term "minting". These are `view` function that don't mint any fee shares but "account" them into the calculation. Replace the "minting" term with the "accounting" one.

Recommendation: Review and correct the cited natspec comments and documentation strings to accurately reflect the implemented behavior. Remove unused error declarations.

Kiln: Fixed by commit `d1392bdd71ddffe4d612379368c44aa9e7a39df8`.

Spearbit: Fixed. Additional fixes provided in the commit commit `8fa112d7c9fe8fdf55529a562feb49287a8c3e33`.

6.3.5 [Research] AAVE Reward Distribution

Severity: Informational.

"Custom" reward programs:

Example 1: `Wrapped eETH`: `Wrapped eETH` has a "custom" rewards program that is distributed **directly** by Ether.fi (Loyalty Points). See loyalty-points.

Example 2: `USDe` : `USDe` has a ”custom” rewards program (+ Merkl distributor) that is distributed **directly** by Ethena (Referral). See join.

Example 3: `rsETH` : `rsETH` has a ”custom” rewards program that is distributed **directly** by Kernel DAO (Kernel Points). See kernel-points-guide.

ACI Merit Program: This is a program initiated and implemented by the Aave Chan Initiative (ACI). Aave Labs does not guarantee the program and accepts no liability. Learn more merit.

Merkl: This is a program initiated by the Aave DAO and implemented by Merkl. Aave Labs does not guarantee the program and accepts no liability. An integration guide can be seen here: earning-with-merkl.

⚠ **IMPORTANT NOTE:** it seems that each Merkl distribution campaign has its own ”requirements” which could be more than just ”supply the liquidity and hold it”.

AAVE Rewards Controller (Mainnet study): Mainnet AAVEV3 Default Incentives Controller: 0x8164Cc65827dcFe994AB23944CBC90e0aa80bFcb.

Asset List: These are the assets for which the user will get some rewards when they hold them in their balance. ⚠ **IMPORTANT NOTE:** These assets could be both `AToken` or `VToken` .

- Aave Ethereum ETHx (aEthETHx).
- Aave Ethereum Lido WETH (aEthLidoWETH).
- Aave Ethereum Lido wstETH (aEthLidowstETH).
- Aave Ethereum USDS (aEthUSDS).
- Aave Ethereum Lido USDC (aEthLidoUSDC).

Reward List: ⚠ **IMPORTANT NOTE:** the reward of an asset could be the asset's itself (usually an `AToken`).

- Aave Ethereum ETHx (aEthETHx).
  - (ENDED) Stader (SD).
- Aave Ethereum Lido WETH (aEthLidoWETH).
  - (ENDED) Aave Ethereum Lido WETH (aEthLidoWETH).
- Aave Ethereum Lido wstETH (aEthLidowstETH).
  - (ENDED) Wrapped liquid staked Ether 2.0 (wstETH).
  - (ENDED) Aave Ethereum Lido wstETH (aEthLidowstETH).
- Aave Ethereum USDS (aEthUSDS).
  - (ACTIVE **UNTIL** 04 March 2026) Aave Ethereum USDS (aEthUSDS).
- Aave Ethereum Lido USDC (aEthLidoUSDC).
  - (ENDED) Aave Ethereum Lido wstETH (aEthLidowstETH).

Kiln: Fixed by commit `65dcdd174c20eea6fafb5c4e91cc9dca1b2f5c0e` .

We decided to remove the Aave V3 module and focus on Merkl rewards only at launch via the Interceptor pattern.

Spearbit: Fixed. Note: Aave V3 rewards are not distributed only via Merkl but also via other custom distribution channels.

6.3.6 `Account.performCall` incorrectly reverts with `Unauthorized` error

Severity: Informational.

Context: Account.sol#L69-L71.

Description: `Account.performCall` reverts with the `Unauthorized` error through the following code.

```
if (msg.sender != $owner) {
    revert ErrorLib.Unauthorized(msg.sender, $owner);
}
```

The variable ordering is incorrect as `Unauthorized` error definition is:

```
error Unauthorized(address expectedCaller, address caller);
```

Recommendation: Swap the two variables.

```
  if (msg.sender != $owner) {
-     revert ErrorLib.Unauthorized(msg.sender, $owner);
+     revert ErrorLib.Unauthorized($owner, msg.sender);
  }
```

Kiln: Fixed in commit bc078b8.

Spearbit: Fixed. The variables have been swapped in the `Unauthorized` error.

6.3.7 `ExternalRBAC` initializers are not disabled

Severity: Informational.

Context: ExternalRBAC.sol#L29-L31.

Description: The `ExternalRBAC` contract is inherited by non-upgradeable contracts. However, it inherits `ExternalRBACUpgradeable` which inherits `Initializable`.

The initializers are not disabled while the contract is not upgradeable.

Recommendation: Add a call to `_disableInitializers()` in `ExternalRBAC.constructor`.

Kiln: Fixed in commit 9f9e388.

Spearbit: Fixed. The `_disableInitializers` function is now called in constructor.

6.3.8 `isCategory` incorrectly reverts instead of returning false

Severity: Informational.

Context: Vehicle.sol#L53-L55.

Description: The `isCategory` function in the `Vehicle` library reverts when there is no deposit route, no redeem route or when the number of deposit routes does not match the number of redeem routes.

Considering that the STEAM standard does not enforce such conditions and that future STEAM implementations could use this library, the `isCategory` function should not revert on these conditions.

```
/// @dev Checks if a vehicle belongs to a specific category based on its routes.
///      For SingleAsset category, validates that vehicle has routes for single asset deposit
↪  and single asset redeem.
/// @param vehicle The vehicle contract to check.
/// @param testedCategory The category to test against.
/// @return True if the vehicle matches the category, false otherwise.
function isCategory(IVehicle vehicle, VehicleCategory testedCategory) internal view returns
↪  (bool) {
    (Route[] memory _depositRoutes, Route[] memory _redeemRoutes) = vehicle.routes();

    if (_depositRoutes.length == 0 || _redeemRoutes.length == 0 || _depositRoutes.length !=
    ↪  _redeemRoutes.length) {
        revert InvalidRouteLengths(_depositRoutes, _redeemRoutes);
    }
```

Recommendation: Consider returning `false` instead of reverting in `isCategory`.

Kiln: Fixed in commit 84daa3f.

Spearbit: Fixed. The `false` boolean is now returned. The revert behavior should now be handled by the caller.

6.3.9 Dead code

Severity: Informational.

Context: Asset.sol#L58-L62.

Description: The codebase contains several instances of unused code like custom errors, library functions, and role constants that are defined but never referenced anywhere in the project.

- ☒ `src/common/interfaces/IFreezablePausableBeacon.sol:59`: The `ExpectedPause` error is defined in the `IFreezablePausableBeacon` interface but is never used anywhere in the codebase. The companion error `EnforcedPause` (line 56) is used, but `ExpectedPause` has no corresponding usage.
- ☒ `src/libs/Asset.sol:58-62`: The `Asset.getSingleAssetValueFromMemory` function is not used and can be removed.
- ☒ `src/libs/Error.sol:78,83,86,90,130`: The following errors defined in `ErrorLib` are never used anywhere in the codebase: `Rejected`, `Settled`, `OnlyDelegateCall`, `DuplicatedAsset`, and `InvalidInitialDepositAmount`.
- ☒ `src/libs/Factory.sol:122`: `FactoryLib` defines its own `FailedVehicleCreation(address factory, Query query)` error. This error is never used — the `FactoryLib.executeDeposit` function reverts with `FailedVehicleUnlock`, not `FailedVehicleCreation`. A different `FailedVehicleCreation(address factory)` error is defined in `ICoreFactory.sol` and used in `CoreFactory.sol`; the `FactoryLib` variant is a dead, conflicting duplicate.
- ☒ `src/libs/Rejection.sol:45,58`: `RejectionLib` provides two helper functions (`rejectWithMaxDepositTooLow` and `rejectWithMaxRedeemTooLow`) that are never called. The actual capacity-exceeded rejection logic in `BaseVehicle.sol` (lines 209–213) reverts directly with `ErrorLib.MaxDepositTooLow` and `ErrorLib.MaxRedeemTooLow` rather than using these `RejectionLib` encoders.

- ☒ `src/libs/Roles.sol:76` : The `FEE_MANAGER_REDEEM_VEHICLE_SHARES` role constant is defined in `Roles` but is not referenced anywhere in the codebase for access control checks or role assignments.
- ☒ `src/vehicles/base/abstracts/BaseVehicleErrors.sol:36,39,43` : The errors `InvalidReceiver`, `InvalidOwner`, and `InvalidMode(Mode expected, Mode actual)` are declared in `BaseVehicleErrors` but never used. The other errors defined in the same contract (`InvalidCaller`, `InvalidState`, `InvalidRecipient`) are actively used.

Recommendation: Dead code should be removed from the codebase where possible. Removing unused declarations reduces code size, lowers the risk of confusion for developers and auditors, and eliminates potential inconsistencies.

Kiln: Fixed in commit 0a93627 and commit 864d1c1.

Spearbit: Fixed. Dead code occurrences have been fixed by being deleted or being used.

6.3.10 `setDeprecated` allows to ”undeprecate”

Severity: Informational.

Context: FactoryBase.sol#L62-L66.

Description: The `FactoryBase.setDeprecated` function allows the authorized party to deprecate a factory. But this function also allows to cancel deprecation of a factory.

A deprecation should not be supposed to be cancellable.

Moreover, no event is emitted during the execution of `setDeprecated`.

Recommendation: Either change the ”deprecation” naming, or make the deprecation non-cancellable.

Kiln: Fixed in commit 2894381.

Spearbit: Fixed. `setDeprecated` has been renamed to `deprecate` and does not allow to cancel a deprecation. Moreover, the `Deprecated()` event is now emitted.

6.3.11 Query ID is calculated at each round loop

Severity: Informational.

Context: BaseVehicleAssetUtils.sol#L53.

Description: In multiple locations, `query.toId(address(this))` is executed in multiple loop rounds but this value will not change.

The locations are:

- `BaseVehicleAssetUtils:53`.
- `BaseVehicle.sol:546`.

Recommendation: Compute `query.toId(address(this))` outside of loops if possible.

Kiln: Fixed in commit bde6415.

Spearbit: Fixed. The query ID is calculated once and cached to be reused later.

6.3.12 Requirements for `resume()` Implementation in Derived Vehicles

Severity: Informational.

Description: The `BaseVehicle` contract defines the `resume()` function as a core part of the STEAM query lifecycle, delegating the specific execution logic to the internal `_resume()`. To ensure the economic integrity and share pricing of the vehicle, any derived contract or facet implementing the resumption logic must adhere to the following requirements:

- `resume()` must be the function that interacts with the underlying protocol and therefore drives the change in `totalAssets()` (either increase via deposit or decrease via redeem/withdraw).
- `resume()` must be the function that mints vehicle shares; otherwise, the vehicle share price will be calculated incorrectly.
- Minting vehicle shares means `resume()` should also include management and performance fees while calculating the shares to be minted.
- `resume()` should estimate the output for the user and validate the slippage. As part of this process, it may override the value of `feesConfigId` that was previously stored during the call to `create()`.

Recommendation: Consider implementing realistic vehicle contracts that fully implement functions such as `resume()` and `recover()`. This would make it easier for both the development team and security researchers to reason about the correctness of the STEAM state machine—particularly the logic in `BaseVehicle.create()`, which may be affected by the eventual implementation details of `resume()` and `recover()`.

Kiln: Fixed in 6038c67.

Spearbit: Fixed by implementing the reviewer's recommendation.

6.3.13 Missing non-zero check for underlying protocol minted shares

Severity: Informational.

Description: Across multiple vehicle implementations, the internal logic for handling deposits lacks a validation check to ensure that the number of underlying protocol shares to be minted is greater than zero, As we can see in the sample implementation of :

```
function _create_deposit(Query calldata query, Id qid, uint256 totalSupply_, uint256
↪  totalAssets_)
    internal
    virtual
    returns (State, State[] memory, Asset[] memory)
{
    ERC4626VehicleStore.Storage storage $ = ERC4626VehicleStore.getStorage();
    uint256 _assetsToDeposit = query.getSingleInputValue();
    IERC20 _inputAsset = IERC20(query.getSingleInputAddress());
    // [1] Calculate vehicle shares to mint using current exchange rate
    uint256 _sharesToMint = _preview_deposit(_assetsToDeposit, totalSupply_, totalAssets_,
    ↪  Math.Rounding.Floor);
    // [2] Approve ERC4626 vault to spend the deposited assets
    IERC4626 _underlyingVault = $.underlyingVault;
    SafeERC20.forceApprove(_inputAsset, address(_underlyingVault), _assetsToDeposit);
    // [3] Deposit assets into ERC4626 vault
```

```
    _underlyingVault.deposit(_assetsToDeposit, address(this));
    // [4] Mint vehicle shares to the vehicle itself (not yet to user)
    _mint(address(this), _sharesToMint);
    // [5] Store the vehicle share amount for the unlock phase
    $.values[qid].amountToUnlock = _sharesToMint;
    // [6] Transition to UNLOCKING state
    return _transition(qid, State.UNLOCKING, query.input);
}
```

The underlying protocol shares being received from the call to `_underlyingVault.deposit()` are never checked to be non-zero. If a user provides an amount of assets that, after accounting for fees or rounding, results in zero shares (due to rounding errors or any potential share price manipulation in the underlying protocol itself), the vehicle may still proceed with pulling assets from the user. This leads to a state where the user loses capital without receiving any representation of ownership in return.

Recommendation: `_create_deposit()` should be changed to support to calculate the protocol shares received from the underlying protocol (`_underlyingVault` in our code snippet example) and revert in case this value is 0.

Kiln: We acknowledge this issue.

Spearbit: Acknowledged.

6.3.14 Missing `msg.sender` in `SpawnedExternalAccessControl` Event

Severity: Informational.

Description: The spawn function in `AccessControlFactory` utilizes a guarded salt to prevent front-running, but the `SpawnedExternalAccessControl` event does not record the msg.sender. This deviates from the design of `CoreFactory`, where the `VehicleCreated` and `ContractCreated` events explicitly include the `msg.sender` for indexing and tracking.

Recommendation: Update the `SpawnedExternalAccessControl` event signature to include an indexed deployer address and pass `msg.sender` during emission.

Kiln: Fixed in 84a13a42.

Spearbit: Fixed by implementing the reviewer's recommendation.

6.3.15 Missing `receive()` function prevents `Account` from accepting ETH

Severity: Informational.

Description: The `Account` contract is designed to hold assets and interact with protocols like Ethena. While `performCall` is `payable`, the contract lacks a `receive()` or `fallback()` function. Any direct ETH transfer (e.g., from a vault, a rewards distributor, or the `owner`) will revert.

Impact: The contract cannot function as a long-term treasury or recipient for native ETH rewards. Any integration attempting to ”push” ETH to this address will fail.

Recommendation: Add a `receive()` function to allow the contract to accept native currency.

```
/// @dev Enable the contract to receive ETH.
receive() external payable {}
```

Kiln: Fixed in 197626f.

Spearbit: Fixed by implementing the reviewer's recommendation.

6.3.16 Missing State Transition Validation in `_setState()`

Severity: Informational.

Context: BaseVehicleQueryStateSetter.sol#L28.

Description: The `_setState()` function currently updates the query state in storage without verifying if the transition from the current state to `newState` is legally defined. Since this utility is the central entry point for all state changes, it lacks a critical post-condition check to ensure the state machine's integrity.

Impact: A buggy facet or subclass could accidentally trigger an illegal transition (e.g., moving a query from a terminal `REJECTED` state back to `ACTIVE`). Without a centralized check, these logic errors can propagate silently, leading to inconsistent contract states that are difficult to debug or recover.

Recommendation: Implement the existing transition table to encode valid `(from -> to)` pairs directly within `_setState()`. Reverting with a custom `InvalidTransition(State from, State to)` error ensures that all state changes are validated regardless of which internal utility or facet initiates them.

Kiln: Fixed in 5ad50ea8.

Spearbit: Fixed by implementing the reviewer's recommendation.

6.3.17 Functions renamings

Severity: Informational.

Description: Rename the following functions to better reflect their actual behavior, specifically regarding fee accounting and toggle logic.

- **Rename:** `previewApplyFees` → `previewApplyDepositRedeemFees`.
- **Rename:** `_previewApplyFees` → `_previewApplyDepositRedeemFees`.
- *Rationale:* Disambiguates between transaction fees and ongoing management fees.
- **Rename:** `_handleFeesAfterOperation` → `_updatePostFeeCheckpoints`.
- *Rationale:* The current name is misleading; the function captures state snapshots rather than executing fee logic.
- **Rename:** `convertToAssets()` → `previewAssetsWithFees()`.
- *Rationale:* Clarifies that the returned value is the amount remaining with fees.
- **Rename:** `BaseVehicle.allow()` → `BaseVehicle.setAccessStatus()`.
- *Rationale:* ”Allow” implies a one-way action; the new name reflects that it toggles both allow/disallow states.
- **Rename:** `BaseVehicle._handleFeesBeforeOperation()` → `handlePerformanceAndManagementFees()`.

- *Rationale:* Replaces a generic lifecycle name with the specific types of fees being processed.
- **Rename:** `Asset.assetsMatching(Asset[] memory queryOutput, Asset[] memory currentOutput)`: rename parameters as it is being used for input as well.
- **Rename:** In every Vehicle (and Facet) `_create_deposit` → `_createDeposit`, `_create_redeem` → `_createRedeem`. Follow the existing camel case style for the function's name.
- **Rename:** `SingleAssetBaseVehicle._preview_deposit` → `_previewDeposit`, `SingleAssetBaseVehicle._preview_redeem` → `_previewRedeem`. Follow the existing camel case style for the function's name.
- **Rename:** `BaseVehicleChecks._onlyState` → `_onlyQueryState`. The auth check is not about the contract's state but the query one.
- **Rename:** `BaseVehicleChecks._onlyOwner` → `_onlyQueryOwner`. The auth check is not about the contract's state but the query one.

Kiln: Fixed in cdeb706.

Spearbit: Fixed by implementing the reviewer's recommendation.

6.3.18 Resiliency Against Non-Standard ERC-4626 Vault Withdrawal Fees

Severity: Informational.

Context: ERC4626Vehicle.sol#L150.

Description: Some erc4626 vaults might deduct fees during the `erc4626Vault.withdraw()` call, returning fewer assets than requested, although the standard defines `withdraw` as the following: "Burns shares from owner and sends **exactly** assets of underlying tokens to receiver.".

The current logic assumes the amount received exactly equals `_assetsToWithdraw`. This discrepancy will cause `unlock()` to revert when it encounters a balance mismatch.

Recommendation: Compute `_assetsToWithdraw` based on the difference of the balances of the calling contract before and after the call to `withdraw` to ensure accurate accounting.

Kiln: We acknowledge this issue.

Spearbit: Acknowledged.

6.3.19 Repository-Wide Code Cleanup and Logic Consolidation

Severity: Informational.

Description:

1. BaseVehicle.sol: L426: `else if` should be used instead of `if`.
2. BaseVehicle.sol: L232: `_assets` variable is not used.
3. ERC4626Vehicle.sol: L84: The vehicle has facets but does not implement the create facet.
4. ERC4626VehicleFacets.sol: L70: `__BaseVehicle_init()` should be called in `initialize()` instead.
5. BaseVehicleInitialization.sol: L137: redundant initialization of `$enabled`.
6. ERC4626VehicleFactory.sol: L164: redundant check that already occurs in `_paramsChecks()`.

7. ERC4626VehicleFactory.sol: L168: redundant check that already occurs in `craftInitialDeposit()`.

Kiln: Fixed in a558b015.

Spearbit: Fixed by implementing the reviewer's recommendation.

6.3.20 Consider using OZ's `Ownable2Step` in the `Account` contract

Severity: Informational.

Description: The `Account` contract is implementing a basic ownership mechanism that does not offer all the utility (migration, auth checks, ...) and security features already implemented in the battletested `Ownable2Step` contract.

Recommendation: Kiln should consider removing the current `$owner` logic and replacing it by directly inheriting from the `Ownable2Step` contract.

Kiln: We ACK this issue. Using OZ.Ownable2Step introduced more code and required guardrails (overriding renounceOwnership to prevent soft locks for ex.) and since this contract will only be used by other contract they would not need nor be able to transfer ownership of the Account contract.

Spearbit: Acknowledged.

6.3.21 Consider reverting if the setter/updater function does not perform any state change

Severity: Informational.

Context: ExternalAccessControl.sol#L164, ExternalAccessControl.sol#L182.

Description: The current codebase is adopting the best practice to revert if a setter/updater function does not actually change the contract's state. This best practice is not always applied to all the functions.

Here are some examples:

- ☒ ExternalAccessControl.setRolePublic.
- ☒ ExternalAccessControl.setScopedRolePublic.

Recommendation: Kiln should consider to apply the above best practice across all the contracts to be coherent with the adopted strategy.

Kiln: Fixed by commit `d3cad61ed1dc62bf571379478b0ee366a4602d9f`. Additional fixes here commit `864d1c142f707ade12918de3b1e43439d90a4f2d`.

Spearbit: Fixed.

6.3.22 Sanity Checks

Severity: Informational.

Context: ExternalAccessControl.sol#L72, CoreFactory.sol#L46, AaveV3VehicleFacets.sol#L96-L97, BaseVehicleInitialization.sol#L118, CompoundV3VehicleFacets.sol#L62.

Description:

- ☒ ExternalAccessControl._grantRole: `_grantRole` allows granting the `role` role to `address(0)`. By implementing the sanity check at this point you can remove the same check from `AccessControlFactory._paramsChecks`.
- ☒ CoreFactory.sol#L46: Add the sanity check `CheckLib.checkContract(createX, ...);` for the `createX` input parameter in the `CoreFactory` constructor.
- ☒ AaveV3VehicleFacets.sol#L82: Consider reverting the Aave V3 vehicle initialization if the underlying Pool is not fully functional. Revert if `isActive == false || isPaused == true || isFrozen == true`.
- ☒ BaseVehicleInitialization.sol#L118: consider reverting the `BaseVehicleInitialization.__BaseVehicle_init` logic if the `FeeManager` has not been fully initializing. The function could fetch the `$.currentFeesConfigId` of the `FeeManager` by calling the `fee()` function and revert if `$.currentFeesConfigId == bytes32(0)`.
- ☒ CompoundV3VehicleFacets.sol#L62: Consider reverting the Compound V3 vehicle initialization if the underlying market is not fully functional. Revert if `isSupplyPaused == true || isWithdrawPaused == true`.

Recommendation: Kiln should consider implementing the above sanity checks suggested to improve the security and avoid misconfigurations.

Kiln: Fixed by commit `1be5700fe9d515723c7cbe9eb253ee5c83826afb`.

Spearbit: Fixed. The suggested revert during the execution of `BaseVehicleInitialization.__BaseVehicle_init` when the Fee Manager has not been initialized yet has been implemented in the commit `e1d58cee3f90fb81f4535e16b2649679d1de3e6b`.

6.3.23 Bulk Informational Issues

Severity: Informational.

Context: ExternalAccessControl.sol#L281, FreezablePausableBeacon.sol#L129-L131, AccessControlFactory.sol#L96-L98, CoreFactory.sol#L157, MorphoBlueVehicleFactory.sol#L129, Vehicle.sol#L89, Vehicle.sol#L108, BaseVehicleEvents.sol#L46-L60, BaseVehicleInitialization.sol#L131-L135, BaseVehicle.sol#L756-L757, FeeManager.sol#L265-L269.

Description:

- ☒ ExternalAccessControl.sol#L281: The `CannotModifyPublicRole` error in `ExternalAccessControl._notPublicScopedRole` name is incorrect. The error name and meaning are specific to the action where these modifiers/functions are called on (to notify a public role) but the function is more general as a guard that "denies" access to public roles to a flow. Consider one of these two options: Option 1) just return a `bool` (is public or not) and revert in the specific "root" function (where the meaning of the function is expressed). Option 2) use a more general-purpose error like `PublicRoleAuthDenied` or something similar.
- ☒ FreezablePausableBeacon.sol#L129-L131: the `paused()` sanity check in `FreezablePausableBeacon.implementation` implementation can be removed. The same check is already performed by the `whenNotPaused` modifier.
- ☒ AccessControlFactory.sol#L96-L98: In `AccessControlFactory.spawn` consider replacing the direct call to `CORE_FACTORY.deployContract` with `FactoryLib.deployContract` to be coherent with the rest of the best practice already adopted. Note: the same suggestion can be applied to the OOS code (in the Phase 0 context) in `MultiVehicleFactory._subSpawn`.

- ☒ MorphoBlueVehicleFactory.sol#L129: rename the input parameter `morphoImplementation` in the `MorphoBlueVehicleFactory` constructor to `morphoBlueVehicleImplementation` to avoid confusing it with the "real" Morpho Blue (market).
- ☒ CoreFactory.sol#L157: Rename the named return parameter `vehicle` for the `CoreFactory.cloneContract` function to `contract_`.
- ☐ Vehicle.sol#L89 + Vehicle.sol#L108: Consider refactoring the `convertToAssets` and `convertToShares` functions in `Vehicle` to return the whole `Asset` type instead of the raw value to offer a better DX.
- ☐ BaseVehicleEvents.sol#L46-L60: consider declaring the `address` type input of the `DeployerInitialized`, `FeeManagerInitialized`, `ModulesManagerInitialized` and `AccessControlInitialized` events as `indexed`.
- ☒ BaseVehicleInitialization.sol#L131-L135: consider emitting a specific event when the `$.forbiddenAddresses` are configured during the `BaseVehicleInitialization.__BaseVehicle_init` execution.
- ☒ BaseVehicleQueryStateSetter.sol#L97-L102: the transition to the `REJECTED` state should happen only via the execution of the `BaseVehicleQueryStateSetter._reject` function. Look in the whole codebase where `_transition(queryId, State.REJECTED, query.input);` is used and replace it with the `_reject` call.
- ☒ BaseVehicle.sol#L756-L757: consider swapping the order of the `ignorePayoutFees` and `ignoreOngoingFees` input parameters of the `BaseVehicle._estimate` function. The "ongoing fee" is applied **before** the "payout one".
- ☐ FeeManager.sol#L265-L270: consider emitting a specific event when `FeeManager.onUpdate` is executed and `$.cache[msg.sender].applicableConfigId != $.currentFeesConfigId` to signal that the new fee config has been applied to the calling vehicle.
- ☒ MorphoBlueVehicle.sol#L181-L192: `assets[0].asset = address(this);` is repeated in both else/if brackets and can be in the main body of the function.
- ☒ FeeManager.sol#L159: `NoRecipients` is not used.
- ☒ Consider refactoring every `Math.mulDiv` call to use **explicit** rounding directions. Some of the existing `Math.mulDiv` calls do not specify it.

Recommendation: Kiln should consider implementing the suggestions listed above.

Kiln: Fixed by commit `6a2265d0a75a081253f67e0beb1f68661b4b431b`.

Spearbit: Kiln has also provided the following additional commits: commit `b329803238bd0e77c5232f3eb24d06a0e156fe89`.

Fixed. The remaining points have been acknowledged by Kiln.

6.3.24 `forbiddenAddresses` configured by the factories could be seen as meaningless

Severity: Informational.

Context: AaveV3VehicleFactory.sol#L224.

Description: The `forbiddenAddresses` configured for a base vehicle is used to prevent the creation of a query that has the `owner` or the `receiver` (of the Query) part of these forbidden lists.

The Factories of these vehicles are pre-configuring the `forbiddenAddresses` with some specific addresses. Let's take a look at the `AaveV3VehicleFactory` to make an example:

```
address[] memory _forbiddenAddresses = new address[](1);
_forbiddenAddresses[0] = params.poolAddressesProvider;
```

While we can argue that indeed the Pool Address Provider address is an ”invalid” address to be configured as the query `owner` or `receiver`, we could also argue that also the underlying's `AToken`, `VToken`, `Pool` (and so on) are invalid `owner/receiver` of a Query. The list could go on and on.

The `forbiddenAddresses` should only be used to configure specific addresses that could break the Vehicle's logic or enable security exploits.

The current usage, like the one done by the `AaveV3VehicleFactory` (and all the other existing factories) could be seen as meaningless and limited given that the addresses to be configured with are infinite.

Recommendation: Kiln should, for each existing Vehicle Factories, look if there are specific addresses to be configured as ”forbidden”. Only addresses that could break the vehicle or create security exploits should be configured as forbidden.

Kiln: Fixed by commit `52aabac0518d9d9afd4a9c7621aaf02ac01e2f04`. Additional changes have been made in the commit `48b39a1c009007b6d1f04153c6589eeab08ebe0f`.

Spearbit: Fixed.

6.3.25 `spawn` function can be improved for some of the existing factories

Severity: Informational.

Context: AccessControlFactory.sol#L93-L94, AccountListFactory.sol#L128-L129, OwnerRegistryFactory.sol#L119-L120, FeeManagerFactory.sol#L125-L126, ModulesManagerFactory.sol#L121-L122.

Description: The `spawn` function of the `FeeManagerFactory`, `ModulesManagerFactory`, `AccessControlFactory`, `AccountListFactory` and `OwnerRegistryFactory` factories can be refactored and improved.

The below suggestion reference the `FeeManagerFactory` code but it can be applied to all the above listed factories:

1) Take `_feeManagerAddress` from the `FactoryLib.deployContract` and not from `this.getDeploymentAddress`.
2) Declare `getDeploymentAddress` as `external` and not `public` (it's not needed public anymore).
3) Move the `SpawnedFeeManager` event to the bottom of the function's logic (like the in the Vehicle factories).

Recommendation: Kiln should consider to implement the above suggestions.

Kiln: Fixed by commit `102df6af2ce5f5a09191e7100bba394e70dd2c1d`.

Spearbit: Fixed.

6.3.26 Consider updating the STEAM standard to require the Query to provide a non-empty `Asset[] output` array

Severity: Informational.

Context: STEAM.md#L255-L260, Asset.sol#L73-L75, SingleAssetBaseVehicle.sol#L83-L85.

Description: The current `STEAM` standard and the corresponding code implementation allow the query creator to specify an **empty** array for the `query.output` attribute.

This choice create inconsistencies and less strict code behaviors that should be avoided. For a standard it's always a good design choice to require being explicit and avoid any possible confusion or misconfiguration.

If the user or caller does not care about the query results and accepts any possible outcome from a `DEPOSIT` or `REDEEM` operation, it should explicitly express it as a valid, non-empty `query.output`.

Let's make a practical example: the user wants to deposit `USDC` in the `AaveV3Vehicle` and accept any possible outcome from such an operation.

Right now the user can choose to: specify an empty `query.output` or be explicit and set it to `[{asset: address(vehicle), value: 0}]`.

By being explicit we can ensure that the query has been correctly configured and the user has made no mistakes. This will become even more crucial when the protocol will manage more complex and exotic vehicles that could output multiple assets.

Recommendation: Kiln should consider changing the `STEAM` standard and consider an empty Query `output` as an invalid state.

After performing such a change in the standard, the changes should also be reflected in the code:

- The `Asset.assetsMatching` function should remove the `if (_queryOutputLength == 0) { return true; }` early return statement.
- The `SingleAssetBaseVehicle._validateOutputs` function should revert with `RejectionLib.rejectWithZeroInputValue()` when `outputs.length == 0`.

Kiln: Fixed by commit `55e570e6baa88d886d73e9ea984767fcf2c01884`.

Spearbit: Fixed.

6.3.27 Use `forceApprove` instead of `safeIncreaseAllowance` in the `Factory`

Severity: Informational.

Context: Factory.sol#L160.

Description: In `Factory.craftInitialDeposit` the `safeIncreaseAllowance` is currently used.

1) The Vehicle has just been deployed so we can assume that it won't have any existing allowance for `address(this)`.
2) We expect the Vehicle to pull the whole `params.initialDepositSize` when `BaseVehicle.create` is executed to manage the initial deposit query.

Recommendation: Kiln should replace `safeIncreaseAllowance` with `forceApprove`.

Kiln: Fixed by commit `557f5ea`.

Spearbit: Fixed.

6.3.28 `Vehicle` functions `convertToAssets` and `convertToShares` should be better documented

Severity: Informational.

Context: Vehicle.sol#L85-L87, Vehicle.sol#L104-L106.

Description: The `Vehicle` is a utility library that is conceived to support multi-asset vehicles but the `convertToAssets` and `convertToShares` functions will revert if the vehicle is **not** a single-asset vehicle.

Recommendation: Kiln should consider renaming and documenting those functions and be explicit on the usage.

Kiln: Fixed by commit `04b0b9d7f7dda617ebaf4ac9b7024e7962fac4ab`.

Spearbit: Fixed.

6.3.29 Remove the `_recover` function's implementation from all those vehicle that cannot transition to the `RECOVERING` state

Severity: Informational.

Context: AaveV3Vehicle.sol#L83-L91, CompoundV3Vehicle.sol#L155-L163, ERC4626Vehicle.sol#L157-L165, MorphoBlueVehicle.sol#L103-L111, WrapperVehicle.sol#L182-L190.

Description: The `AaveV3Vehicle`, `CompoundV3Vehicle`, `ERC4626Vehicle`, `MorphoBlueVehicle` and `WrapperVehicle` cannot logically transact a query to the `RECOVERING` phase. This means that everytime that someone calls the vehicle's `recover(Query calldata query)` function, the vehicle will revert with the `BaseVehicleErrors.InvalidState` result when the `_onlyState` function (triggered by the `steamOperation(query, State.RECOVERING)` modifier) is executed.

To avoid any confusion and possible future mistakes, all those vehicles should fully remove the `function _recover(Query calldata query, Id qid)` implementation from their contract.

As a general rule of thumb, if the state cannot be reached, the vehicle's should never implement the `internal` corresponding function that would manage the state.

Recommendation: Kiln should remove the `_recover` function's implementation from all those vehicles that cannot transition to the `RECOVERING` state for a query.

Kiln: Fixed by commit `4269470b84c8a95f7c4838eabdc3e550400360b7`.

We removed the overrides + we also overriden `resume()` and `recover()` to immediately revert and reduce bytecode size (no Facet handling).

Spearbit: Fixed.

6.3.30 Consider allowing integrator to fetch the fee detail snapshotted to the Query

Severity: Informational.

Description: When a Query has been created, the `BaseVehicle` snapshot a specific query configuration to be applied in the unlocking phase. At the moment there's not a way for the query's owner to retrieve that information properly.

Kiln should offer these two functions:

- In `BaseVehicle` a function that given a `Query q` it fetches the `feesConfigId` snapshotted to it and retrieves the `depositFeeBps` and `redeemFeeBps` associated to it in the `FeeManager`.
- In the `FeeManager` a function that given a `bytes32 configId` it returns the `depositFeeBps` and `redeemFeeBps`.

Recommendation: Kiln should consider improving the user's and integrator DX and allow external entities to fetch the fee configuration associated with a query.

Kiln: Fixed by commit `53abd3e97df3480356d1227dd46db558421220e7`.

Spearbit: Kiln has also provided the following additional commits: commit `b8b982fa266355b1494bd05779be377c784bad4f` and commit `3e680e46f128e2ff9359d99ddfa2af52c864bb7e`.

Now the config can be queried directly from the Vehicle itself and the FeeManager will revert if the specified `configId` does not exist.

6.3.31 Consider avoiding taking payout fees when the query has been generated by the `FeeManager`

Severity: Informational.

Description: When the user executes the `unlock` flow to receive the vehicle's shares (from a `DEPOSIT` operation), the `FeeManager` will take an ”operation fee” that will transfer part of the shares to the `FeeManager`.

At some point the entity with the role `FEE_MANAGER_DISPATCH_ERC20` will execute `FeeManager.dispatchERC20` and dispatch those shares to every `feeRecipients` that will need to execute a `REDEEM` operation on the same vehicle to receive the underlying asset. To the generated query, the `FeeManager` will also take a payout fee that will distribute the fees (as underlying assets) to the `FeeManager` again.

Recommendation: Kiln should consider to avoid taking the ”operation fee” on the redeem query generated to withdraw the fee distributed to the `feeRecipients`.

Kiln: Acknowledged. According to our discussion, we finally decided to revert the changes and to come back to the original behavior, see the revert commit 7ae0f3b7.

Spearbit: Kiln will acknowledge the Finding. A solution has been proposed but it was reverted since the complexity of the implementation and the possible side effects were not worth the effort and the degradation of overall security and gas cost.

6.3.32 Some ”open” functions allow anyone to generate spammy events

Severity: Informational.

Context: Keeper.sol#L64, Keeper.sol#L71, Keeper.sol#L79, FeeManager.sol#L282, FeeManager.sol#L296.

Description: In the codebase there are functions that allow anyone to execute them and generate ”spammy” events that need to be excluded by the Kiln monitoring and analytics system.

- `FeeManager.onOperations` → `OperationalFeesCollected` event.
- `FeeManager.applyFees` → `TransactionalFeeCollected` event.
- `KeeperLib.startJob` → `JobStarted` event.
- `KeeperLib.stopJob` → `JobDone` event.
- `KeeperLib.cancelJob` → `JobCancelled` event.

Recommendation: Kiln should be aware of all these ”open” functions and that the events generated by those functions **must** be properly filtered by their monitoring and analytics tools.

Kiln: Fixed by commit `d2881fec31784f6bb85923dc8852537e00b81b27`.

Spearbit: Fixed.

6.3.33 Vehicles do not follow a common style and best practice in their implementation

Severity: Informational.

Context: MorphoBlueVehicleFacets.sol#L109.

Description: A lot of refactor has been already done and we can see big improvements but there are still these incoherences that should be fixed and refactored in the context of how each Vehicle has been implemented.

We can see for example some differences comparing the Morpho Vehicle with the Aave V3 Vehicle:

1) In `AaveV3VehicleFacets` the approval is executed **on demand** when the `_create_deposit` is executed. In `MorphoBlueVehicleFacets` instead is performed during the initialization phase and for the infinite amount (`type(uint256).max`).
2) `AaveV3VehicleFacets` it's called `_inputAsset.forceApprove(...)` while in Morpho it is called `SafeERC20.forceApprove(IERC20($.asset), ...)`.
3) The code in `AaveV3VehicleFacets` is ”dense” (no new line between instructions), while `MorphoBlueVehicleFacets` is instead sparse with new lines between instructions.

Recommendation: Kiln should push to create a common and standard style and best practice guide that should be strictly applied to every vehicle developed.

In the specific context of the above example, we suggest Kiln the following reccomendations:

1) Avoid the ”infinite approval” and implement, like in the Aave V3 vehicle, on-demand ad-hoc approval.
2) Adopt the `SafeERC20` usage used in the Aave V3 vehicle.
3) Adopt (and apply) in the other vehicles the style used in the Morpho Vehicle.

Kiln: Fixed by commit `bc84f3292727262be48411ec69af4394b48dfefc`.

Spearbit: Fixed. Note: the suggestion to avoid the infinite approval in the MorphoBlueVehicle was intentionally not applied. Kiln kept the infinite approval because the underlying Morpho Blue protocol is immutable, which the reviewer agreed with.

6.3.34 Inefficient `O(n^2)` Duplicate Check in `_setRecipients`

Severity: Informational.

Context: FeeManager.sol#L429-L433.

Description: The `_setRecipients` function checks for duplicate recipient addresses using a nested loop, resulting in `O(n^2)` complexity. For each recipient, it compares the address with all subsequent recipients to detect duplicates. This approach is unnecessarily inefficient and does not scale well in case the count of addresses of the recipients grows large.

Recommendation: Require the `newRecipients` array to be sorted by address and replace the nested duplicate check with a single pass that verifies `previousAddress < currentAddress`. This reduces the validation complexity from `O(n^2)` to `O(n)` and improves gas efficiency.

Kiln: We ACK this one as this is not a common execution path (rarely called) so we prefer UX over efficiency here.

Spearbit: Acknowledged. The issue can be left unresolved as it is a gas optimization and does not actively threaten the protocol.

6.3.35 Redundant Admin Check in `grantRole` and `revokeRole`

Severity: Informational.

Context: ExternalAccessControl.sol#L100.

Description: The `grantRole` and `revokeRole` functions apply `onlyRole(getRoleAdmin(role))` even though the same admin check is already enforced in the inherited OpenZeppelin `grantRole` and `revokeRole` through the `super` call chain. This results in redundant authorization checks and unnecessary gas overhead. This applies only to `grantRole` and `revokeRole`, while the check must remain in `grantScopedRole` and `revokeScopedRole` since they implement custom scoped-role logic.

Recommendation: Remove the redundant `onlyRole(getRoleAdmin(role))` modifier from `grantRole` and `revokeRole`, since the inherited OpenZeppelin implementation already enforces the same admin check. Keep the modifier on `grantScopedRole` and `revokeScopedRole`.

Kiln: Fixed by PR 375.

Spearbit: Verified Fix, `grantRole` and `revokeRole` now check the `onlyRole(getRoleAdmin(role))` only through the `super` call chain.

6.3.36 `CompoundV3Vehicle` and `MorphoBlueVehicle` Cannot Claim Incentive Rewards

Severity: Informational.

Description: The `CompoundV3Vehicle` and `MorphoBlueVehicle` integrations currently do not provide a mechanism to claim incentive rewards generated by their positions.

Both Comet and Morpho Blue distribute rewards separately from the base lending yield, and these rewards must be explicitly claimed through their respective reward distribution contracts. Since the vehicles do not expose any functionality to trigger reward claims, incentives accrued by the vehicles remain unclaimed.

- Compound rewards documentation: https://docs.compound.finance/.
- Morpho rewards documentation, new rewards via Merkl: https://docs.morpho.org/learn/concepts/rewards/.

As a result, any rewards accumulated by these vehicles are not collected or redistributed within the system until a module is added.

Recommendation: Consider implementing a rewards claiming module for `CompoundV3Vehicle` and `MorphoBlueVehicle`.

Kiln: We ACK this issue.

- We do not plan yet on supporting CompoundV3Vehicle additional rewards via a module.
- The Interceptor pattern in place helps us support Merkl natively on all our contracts.

Spearbit: Acknowledged. Not fixing this issue will result in the Vehicle not being able to redeem compound rewards for the time-being and the protocol has accepted the risk. A module can be added in the future that enables receiving the rewards. Note: the CompoundV3Vehicle has since been fully removed from the codebase in later commits.

6.3.37 `CoreFactory` Mishandles ETH Forwarding and Refund Accounting Across CreateX and Clone Flows

Severity: Informational.

Context: CoreFactory.sol#L62-L123.

Description: `CoreFactory` does not consistently handle `msg.value` and ETH refunds across its deployment paths. This can lead to ETH being unintentionally trapped inside the factory and later used in ways that are not clearly tied to the caller's transaction.

Several situations illustrate this behavior:

- **ETH can be trapped when `data.length == 0`:** In `deployVehicle` and `deployContract`, the branch `if (data.length == 0)` calls `CREATEX.deployCreate2(salt, initCode)` without forwarding ETH. If `msg.value > 0`, the ETH remains in `CoreFactory` instead of being used for deployment or refunded to the caller.
- **Refunds from `CreateX` accumulate in the factory:** In initialization flows using `deployCreate2AndInit`, `CreateX` sends any excess ETH to the provided `refundAddress`, which in this integration resolves to `CoreFactory`. The factory accepts these refunds via `receive()` but does not forward them to the original caller, allowing ETH to accumulate in the contract.
- **Refunds may not correspond to the current deployment:** `CreateX` refunds its entire balance, which may include ETH previously forced into the contract (e.g., via `SELFDESTRUCT`). This means the refund amount is not necessarily attributable only to the current transaction.
- **Factory balance can later be used during clone initialization:** In `cloneVehicle` and `cloneContract`, initialization calls forward `values.initCallAmount` rather than bounding the ETH to `msg.value`. If the factory already holds ETH, callers may trigger initialization calls that spend funds previously accumulated in the factory.
- **Constructor value assumptions:** If `values.constructorAmount > 0`, the constructor of the provided `initCode` must be payable; otherwise deployment will revert.

Recommendation: Standardize ETH handling across all deployment paths:

- Revert when `data.length == 0 && msg.value > 0`, or ensure ETH is forwarded to the deployment call.
- Avoid retaining refunds from `CreateX` inside `CoreFactory`. Refund excess ETH directly to the caller or explicitly forward it after deployment.
- Ensure initialization calls cannot spend ETH already held by the factory. The ETH forwarded during initialization should be bounded by the current `msg.value`.
- Validate or clearly document that deployments using a nonzero `constructorAmount` require a payable constructor.

Kiln: Fixed in commit db83c21.

Spearbit: Fixes verified.

6.3.38 `Created` event emission does not respect the STEAM specification

Severity: Informational.

Context: STEAM.md#L294, BaseVehicle.sol#L238.

Description: The `STEAM` spec for the `Created` event states:

> MUST be emitted before any state transitions occur for the query.

The current `BaseVehicle.create` implementation does not follow the spec definition by emitting the event at the very end of the flow after that the internal state of the contract has been updated by the execution of `_create`.

Recommendation: Kiln should perform the following change to the code for the `BaseVehicle.create` function:

```
+  emit Created(_qid, query);
  (newState, _possibleNext, _assets) = _create(query, _qid, __totalSupply, __totalAssets);

  if (_handleFees) {
      _handleFeesAfterOperation();
  }

-  emit Created(_qid, query);
```

Kiln: Fixed by commit `3b3a631ac66439f9c1551b2fb01b643b9be269a1`.

Spearbit: Fixed.

6.3.39 `STEAM` standard fee's lifecycle section is outdated

Severity: Informational.

Context: STEAM.md#L516, BaseVehicle.sol#L833-L835.

Description: The ”Fee Application During Query Lifecycle” section of the `STEAM` standard states.

> - Fees are **only applied** when transitioning from `EMPTY` to `PROCESSING` or `UNLOCKING` states via `create()` .
> - Fees are **not applied** during recovery operations (`recover()` method), allowing users to reclaim assets without additional charges.

But vehicles do not strictly apply these rules. A vehicle implementation can **override** the `_shouldHandleFees` function and apply the fees not only when the query is in the `EMPTY` state but also when it's in the `UNLOCKING` or `RECOVERING` state.

```
function _shouldHandleFees(Mode, State queryState) internal pure virtual returns (bool) {
    return queryState == State.EMPTY;
}
```

Recommendation: Kiln should update the `STEAM` standard to correctly define all those cases when the ”Ongoing Fees” and ”Operational Fees” should be applied.

Kiln: Fixed by commit `8e26ac51bdee8dfd43d7c1a9204e474db09cb3c5`.

Spearbit: Fixed.

6.3.40 Improve the `totalAssets()` definition in the `STEAM` standard

Severity: Informational.

Context: STEAM.md#L580.

Description: The `STEAM` standard for the `totalAssets()` definition states:

> - **MUST** return a `uint256` representing the total amount of assets (both in-contract and invested).

The current implementation, at least for the Vehicles that have been reviewed in the current Phase, does **not** include the underlying `_asset()` balance that has been **donated directly** to the vehicle itself.

Recommendation: Kiln should update the `STEAM` standard definition for the `totalAssets()` to specify that underlying donations directly to the Vehicle won't be included in the value returned by the function itself.

Kiln: Fixed by commit `21b5a98b263ac45e830f101e65926b0a37250ec3`.

Spearbit: Fixed.

6.3.41 Consider refactoring the `ModulesManager` module management/allowance logic

Severity: Informational.

Description: The current implementation of the `ModulesManager` works like this:

A module (a ”feature”) is identified by a `bytes32 id` and an implementation contract (of such feature) `address module` The `MODULE_MANAGER` role can.

- Add a module via `addModule(bytes32 id, address module)`.
- Update a module via `updateModule(bytes32 id, address module)`.
- Remove a module via `removeModule(bytes32 id)`.
- Approve a ”pending module” via `approvePendingModule(bytes32 id)`.
- Cancel a ”pending module” via `cancelPendingModule(bytes32 id)`.

The `addModule` and `updateModule` do not instantly add/update the module to the new address but they will create a ”pending module request” that needs to pass a timelock period and then be finalized via the `approvePendingModule` action.

On the other side the Vehicle will allow/disallow the execution of a module via the `allowModule(bytes32 id)` and `disallowModule(bytes32 id)` functions.

The Vehicles are allowing/disallowing the ”feature” (the `bytes32 id`) and **not** the specific implementation of the feature which is instead identified by the `(bytes32 id, address module)` tuple.

Let's assume that ”Feature A” has been added by the Module Manager and has been finalized with the tuple `(featureA_id, featureA_address)` and let's assume the `Vehicle1` has approved it.

What should happen in these scenarios?

1) The feature is **removed** via `removeModule(featureA_id)`, is ”replaced” via `addModule(featureA_id, featureB_address)` and finalized via `approvePendingModule(featureA_id)`. The `Vehicle1` has **”blindly”** allowed the ”feature” which now has changed implementation.
2) The feature implementation is **updated** via `updateModule(featureA_id, featureB_address)` and finalized via `approvePendingModule(featureA_id)`. The `Vehicle1` has **”blindly”** allowed the ”feature” which now has changed implementation.
3) The feature implementation is migrated to a new implementation `featureA_new_address` but the update is **still pending approval**. Should the `Vehicle1` be able to still execute `featureB_address` implementation even if it is being replaced for some specific reason? Or if there's a ”pending” approval the execution should be blocked?

Recommendation: Assuming that the `ModulesManager` is not unique to each Vehicle, Kiln should consider refactoring and improving the `ModulesManager` to offer better management and security for the Vehicle that will execute in a **delegatecall** context those ”module implementations”.

The Vehicle should be able to approve the **final** implementation of a ”feature” that will be executed, which is identified by the `(moduleId, moduleImplementationAddess)`.

Kiln: Fixed by commit `278c93f7b9e7f1e91a70ab7e18e5bc30e68f17c1`.

Spearbit: Fixed. The `BaseVehicle.exec` function now tracks the execution of the external module via the `BaseVehicleEvents.ModuleExecuted` event, the additional suggestion has been implemented in the commit `7bff310be492403ad0dcbfdcf3bb446ba1a5ba29`.

6.3.42 Overriding `_shouldHandleFees` function in future implementations can have unexpected impacts

Severity: Informational.

Context: BaseVehicle.sol#L220.

Description: The `_shouldHandleFees` function is marked as virtual. In future implementations, it may get overridden.

However, such implementation will most probably have to review the `ignoreOngoingFees` parameter used during the call to `_estimate`.

In the current `create` function, the ongoing fees are ignored because `_handleFees` is true and fees were already minted in `_handleFeesBeforeOperation`. This is shown in the `_estimate` call, it ignores these fees.

```
function create(Query calldata query)
    // ...
{
    // ...

    bool _handleFees = _shouldHandleFees(query.mode, State.EMPTY);

    if (_handleFees) {
        __totalSupply = _handleFeesBeforeOperation(__totalSupply, __totalAssets, _qid);
    }

    // ...

    // Check if the required output matches the estimated output
    {
        (Asset[] memory _output, bytes32 _appliedFeesConfigId) =
            _estimate(query.input, query.mode, EstimationType.OUTPUT, false, true,
            ↪  __totalSupply, __totalAssets);
        BaseVehicleStore.getStorage().queries[_qid].feesConfigId = _appliedFeesConfigId;

        AssemblyLib.revertIfBytes(_validateConstraints(query, _output));
    }
```

Recommendation: Overriding `_shouldHandleFees` is likely to require overriding the BaseVehicle `create` function too. This should be documented somewhere.

Kiln: Fixed in commit 231c593 by using `_shouldHandleFees` value to ignore the ongoing fees in `create`.

Spearbit: Fixed. The fix bounds two behaviors together: ”fees are being taken” and ”fees are excluded from estimation”.

This fixes the issue. However, it is a potential edge-case for future vehicles.