Dahlia protocol

Security Review

Cantina Managed review by:

Saw-mon and Natalie, Lead Security Researcher

Kankodu, Security Researcher

Yorke Rhodes, Associate Security Researcher

January 7, 2025

Contents

1 Introduction 2
1.1 About Cantina 2
1.2 Disclaimer 2
1.3 Risk assessment 2
1.3.1 Severity Classification 2

2 Security Review Summary 3

3 Findings 4
3.1 High Risk 4
3.1.1 getLastMarketState does not update the market properly 4
3.1.2 Interest is not accrued before calling BorrowImpl.internalBorrow in Dahlia.supplyAndBorrow 5
3.1.3 WrappedVault share transfers results in stuck funds 5
3.1.4 Protocol and Reserve Fees cannot be withdrawn 6
3.2 Medium Risk 6
3.2.1 If a rewards schedule's start time is set as 0, reward interests do not get accumulated 6
3.2.2 The rewardsAdded calculation in extendRewardsInterval is incorrect 7
3.2.3 Inflation of zero totalBorrowShares to disable borrowing is possible 8
3.2.4 Inconsistency between precision factors / dimensional analysis 8
3.2.5 Unnecessary payable functions may lead to permanently locked funds 9
3.2.6 Vault tokens lose fungibility after a user claims interest 9
3.2.7 Protocol and Reserves shares calculation is incorrect 10
3.3 Low Risk 11
3.3.1 Rounding directions 11
3.3.2 Incorrect source of funds for supplyAndBorrow and repayAndWithdraw calls 11
3.3.3 Loss of precision in Variable IRM 12
3.3.4 WrappedVault doesn't follow ERC4626 standard correctly 12
3.4 Gas Optimization 13
3.4.1 Use calldata instead of memory for input location in updatePermissionWithSig 13
3.4.2 Pause and unpausing a market do extra unnecessary market.status checks 13
3.4.3 Remove unnecessary check when adding a new rewards token 13
3.5 Informational 14
3.5.1 Outdated gas snapshot, unused git submodules, unused remappings 14
3.5.2 Typos, Comments, Unused Code, 14
3.5.3 Events related findings 15
3.5.4 A deprecated market can be deprecated again 15
3.5.5 newRatePerSec and newFullUtilizationRate might need to be updated even if interestEarnedAssets is 0 16
3.5.6 market.updatedAt is only updated when interestEarnedAssets is non-zero in executeMarketAccrueInterest 16
3.5.7 Unreached code 16
3.5.8 Check for other VariableIrm.Config parameters missing in IrmFactory.createVariableIrm 17
3.5.9 When creating a Uniswap oracle make sure the token pairs are supported 17
3.5.10 Enforce minimum TWAP duration 17
3.5.11 Add proper check when setting reserve fees 17
3.5.12 Inconsistent condition checks when calling callbacks 18
3.5.13 Mixed use of returned parameters in WrappedVault.redeem 18
3.5.14 Allowed IRM cannot be disallowed 18
3.5.15 ChainlinkOracleWithMaxDelayBase price precision analysis 18

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
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.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Dahlia is a permissionless, modular lending protocol that emphasizes advanced risk control and liquidity aggregation, built atop the Royco Protocol. Designed to optimize capital efficiency and expand access to liquidity across a wider array of assets, Dahlia offers a flexible framework for market participants to manage risk while facilitating seamless borrowing and lending.

From Nov 19th to Dec 1st the Cantina team conducted a review of dahlia-protocol on commit hash 8d83ed30. The team identified a total of **33** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 4
- Medium Risk: 7
- Low Risk: 4
- Gas Optimizations: 3
- Informational: 15

3 Findings

3.1 High Risk

3.1.1 getLastMarketState does not update the market properly

Severity: High Risk

Context: InterestImpl.sol#L80-L101

Description: getLastMarketState is used in:

- previewLendRateAfterDeposit
- getMarket
- getMaxBorrowableAmount
- getPositionLTV
- getPositionInterest

In all these endpoints except previewLendRateAfterDeposit the input parameter lendAssets supplied is 0. For those cases the calculations are almost correct, but since previewLendRateAfterDeposit provides a potentially non-zero value for lendAssets the calculation is not performed correctly.

previewLendRateAfterDeposit is supposed to:

1. First simulate the effect of accruing interest then...
2. Simulate the effect of calling lend on the specified market.

But in the current implementation these orders are not followed but instead accruing interest is performed after lending and also the lendAssets is not added to the market.totalLendAssets. This will make previewLendRateAfterDeposit to return an incorrect rate which will effect the rate returned by WrappedVault.previewRateAfterDeposit:

```
if (reward == address(DEPOSIT_ASSET)) {
   uint256 dahliaRate = dahlia.previewLendRateAfterDeposit(marketId, assets);
   rewardsRate += dahliaRate;
}
```

and thus affect the following inequality checked in VaultMarketHub.allocateOffer:

```
if (offer.incentivesRatesRequested[i] >
↪   WrappedVault(offer.targetVault).previewRateAfterDeposit(offer.incentivesRequested[i], fillAmount)) {
    revert OfferConditionsNotMet();
}
```

Recommendation: Apply the following changes:

```
diff --git a/src/core/impl/InterestImpl.sol b/src/core/impl/InterestImpl.sol
index f672414..2ef20b9 100644
--- a/src/core/impl/InterestImpl.sol
+++ b/src/core/impl/InterestImpl.sol
@@ -75,29 +75,40 @@ library InterestImpl {
         feeShares = (interestEarnedAssets * feeRate * totalLendShares) / (Constants.FEE_PRECISION *
         ↪   (totalLendAssets + interestEarnedAssets));
     }

-    /// @notice Gets the expected market balances after interest accrual.
+    /// @notice Gets the expected market balances after interest accrual and lending.
     /// @return Updated market balances
     function getLastMarketState(IDahlia.Market memory market, uint256 lendAssets) internal view returns
     ↪   (IDahlia.Market memory) {
         uint256 totalBorrowAssets = market.totalBorrowAssets;
         uint256 deltaTime = block.timestamp - market.updatedAt;
-        if ((deltaTime != 0 || lendAssets != 0) && totalBorrowAssets != 0 && address(market.irm) !=
↪   address(0)) {
-            uint256 totalLendAssets = market.totalLendAssets + lendAssets;
-            uint256 totalLendShares = market.totalLendShares;
-            uint256 fullUtilizationRate = market.fullUtilizationRate;
-            uint256 reserveFeeRate = market.reserveFeeRate;
-            uint256 protocolFeeRate = market.protocolFeeRate;
-            (uint256 interestEarnedAssets, uint256 newRatePerSec, uint256 newFullUtilizationRate) =
-                IIrm(market.irm).calculateInterest(deltaTime, totalLendAssets, totalBorrowAssets,
↪   fullUtilizationRate);

-            uint256 protocolFeeShares = calcFeeSharesFromInterest(totalLendAssets, totalLendShares,
↪   interestEarnedAssets, protocolFeeRate);
-            uint256 reserveFeeShares = calcFeeSharesFromInterest(totalLendAssets, totalLendShares,
↪   interestEarnedAssets, reserveFeeRate);
+        // 1. accrue market interest
+        if (deltaTime != 0 && totalBorrowAssets != 0 && address(market.irm) != address(0)) {
+            uint256 totalLendAssets = market.totalLendAssets;
+
+            (uint256 interestEarnedAssets, uint256 newRatePerSec, uint256 newFullUtilizationRate) =
+                IIrm(market.irm).calculateInterest(deltaTime, totalLendAssets, totalBorrowAssets,
↪   market.fullUtilizationRate);

-            market.totalLendShares = totalLendShares + protocolFeeShares + reserveFeeShares;
             market.fullUtilizationRate = uint64(newFullUtilizationRate);
             market.ratePerSec = uint64(newRatePerSec);
-            market.totalBorrowAssets += interestEarnedAssets;
-            market.totalLendAssets += interestEarnedAssets;
+
+            if (interestEarnedAssets > 0) {
+                uint256 totalLendShares = market.totalLendShares;
+                uint256 protocolFeeShares = calcFeeSharesFromInterest(totalLendAssets, totalLendShares,
↪   interestEarnedAssets, market.protocolFeeRate);
+                uint256 reserveFeeShares = calcFeeSharesFromInterest(totalLendAssets, totalLendShares,
↪   interestEarnedAssets, market.reserveFeeRate);
+
+                market.totalLendShares = totalLendShares + protocolFeeShares + reserveFeeShares;
+
+                market.totalBorrowAssets += interestEarnedAssets;
+                market.totalLendAssets += interestEarnedAssets;
+            }
         }
+
+        if (lendAssets > 0) {
+            market.totalLendAssets += lendAssets;
+            market.totalLendShares += lendAssets.toSharesDown(market.totalLendAssets,  market.totalLendShares);
+        }
+
+        market.updatedAt = uint48(block.timestamp);
         return market;
     }
 }
```

The new implementation would look like this:

```
function getLastMarketState(IDahlia.Market memory market, uint256 lendAssets) internal view returns
↪   (IDahlia.Market memory) {
    uint256 totalBorrowAssets = market.totalBorrowAssets;
    uint256 deltaTime = block.timestamp - market.updatedAt;

    // 1. accrue market interest
    if (deltaTime != 0 && totalBorrowAssets != 0 && address(market.irm) != address(0)) {
        uint256 totalLendAssets = market.totalLendAssets;

        (uint256 interestEarnedAssets, uint256 newRatePerSec, uint256 newFullUtilizationRate) =
            IIrm(market.irm).calculateInterest(deltaTime, totalLendAssets, totalBorrowAssets,
            ↪   market.fullUtilizationRate);

        market.fullUtilizationRate = uint64(newFullUtilizationRate);
        market.ratePerSec = uint64(newRatePerSec);

        if (interestEarnedAssets > 0) {
            uint256 totalLendShares = market.totalLendShares;
            uint256 protocolFeeShares = calcFeeSharesFromInterest(totalLendAssets, totalLendShares,
            ↪   interestEarnedAssets, market.protocolFeeRate);
            uint256 reserveFeeShares = calcFeeSharesFromInterest(totalLendAssets, totalLendShares,
            ↪   interestEarnedAssets, market.reserveFeeRate);

            market.totalLendShares = totalLendShares + protocolFeeShares + reserveFeeShares;

            market.totalBorrowAssets += interestEarnedAssets;
            market.totalLendAssets += interestEarnedAssets;
        }
    }

    if (lendAssets > 0) {
        market.totalLendAssets += lendAssets;
        market.totalLendShares += lendAssets.toSharesDown(market.totalLendAssets,  market.totalLendShares);
    }

    market.updatedAt = uint48(block.timestamp);
    return market;
}
```

Notes:

1. `market.updatedAt` is updated at the end unconditionally to make calling this function multiple times cheaper, since the interest accrual would need to only happen once.
2. `market.fullUtilizationRate` and `market.ratePerSec` are updated irregardless of whether `interestEarnedAssets` is non-zero or not.
3. This new and old implementation have the assumption that the interest accrual only need to happen when `totalBorrowAssets` is non-zero although there could be some custom `IIrm` implementations that would return values for even if `totalBorrowAssets` is `0`.
4. The above implementation is not the most gas-efficient one. Further optimisations can be applied.

Dahlia: We agree with the findings but we propose a better solution in PR 15.

Additionally, during the review of the finding "`market.updatedAt` is only updated when `interestEarnedAssets` is non-zero in `executeMarketAccrueInterest`" we wrote a test to verify `updatedAt` logic and we had to add the same logic to skip modification of any fields if no interest accrued in `getLastMarketState`, see commit e10e43f6.

3.1.2 Interest is not accrued before calling `BorrowImpl.internalBorrow` in `Dahlia.supplyAndBorrow`

Severity: High Risk

Context: Dahlia.sol#L252

Description: Interest is not accrued before calling `BorrowImpl.internalBorrow` in `Dahlia.supplyAndBorrow`. Accuring interest before calling `BorrowImpl.internalBorrow` is important as otherwise:

1. `borrowedShares` would be calculated with stale data

2. The following inequality (BorrowImpl.sol#L81-L85) would also use stale data from before accruing interest:

   ```
   // Check if user has enough collateral
   (uint256 borrowedAssets, uint256 maxBorrowAssets) = MarketMath.calcMaxBorrowAssets(market,
   ↪  ownerPosition, collateralPrice);
   if (borrowedAssets > maxBorrowAssets) {
       revert Errors.InsufficientCollateral(borrowedAssets, maxBorrowAssets);
   }
   ```

   `borrowedAssets` would be calculated with stale data.

Recommendation: Make sure to accrue interest before calling `BorrowImpl.internalBorrow`:

```
_accrueMarketInterest(positions, market);
(borrowedAssets, borrowedShares) = BorrowImpl.internalBorrow(market, ownerPosition, borrowAssets, 0, owner,
↪  receiver, 0);
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.1.3 `WrappedVault` share transfers results in stuck funds

Severity: High Risk

Context: WrappedVault.sol#L455, WrappedVault.sol#L462

Description: Only the `wrappedVault` is allowed to `lend` and `withdraw` assets to/from the Dahlia lending contract. Users can interact with `wrappedVault` through its `deposit` and `withdraw` functions.

When a user deposits assets into `wrappedVault`, they are minted shares of `wrappedVault` as expected. Internally, `wrappedVault` calls `dahlia.lend` and assigns the Dahlia shares to the receiver (the depositor) instead of keeping them for itself. This ensures that individual accounting for `claimInterest` works correctly for each depositor.

However, an issue arises when a user transfers their `wrappedVault` shares to another address:

- The sender's `wrappedVault` balance is reduced, and the recipient's balance increases. The corresponding Dahlia shares, however, remain assigned to the original sender and do not update to reflect the transfer.

This leads to a critical issue:

- The recipient cannot withdraw funds because they lack sufficient Dahlia shares.
- The original sender cannot withdraw funds because their `wrappedVault` balance is insufficient, despite still holding the Dahlia shares.

Proof of Concept: Add below test in `test/core/integration/WrappedVault.t.sol`:

```
function testTransferDoesNotWork() public {
    address fromAddress = REGULAR_USER;
    address toAddress = REFERRAL_USER;

    uint256 depositAmount = 1 ether;
    MockERC20(address(token)).mint(fromAddress, depositAmount);

    vm.startPrank(fromAddress);
    token.approve(address(testIncentivizedVault), depositAmount);
    uint256 shares = testIncentivizedVault.deposit(depositAmount, fromAddress);

    testIncentivizedVault.transfer(toAddress, shares);
    vm.stopPrank();

    // neither the fromAddress, nor the toAddress can get funds back

    //this fails because in dahlia accounting toAddress doesn't not have the shares
    vm.startPrank(toAddress);
    vm.expectRevert();
    testIncentivizedVault.withdraw(depositAmount, toAddress, toAddress);
    vm.stopPrank();

    //this fails because wrapperVault balance has been transferred to toAddress
    vm.startPrank(fromAddress);
    vm.expectRevert();
    testIncentivizedVault.withdraw(depositAmount, fromAddress, fromAddress);
    vm.stopPrank();
}
```

Recommendation: Update the transfer logic to ensure proper transfer of Dahlia shares when wrappedVault shares are transferred.

Dahlia:: Fixed in commit dcc59d44.

Cantina Managed:

- Missing `Transfer` Events

  A token contract that creates new tokens **SHOULD** trigger a `Transfer` event with the `_from` address set to `0x0` when tokens are created.

  1. When `WrappedVault` shares are minted in the `WrappedVault._deposit` function, no `Transfer` event is emitted with `from = address(0)` and `to = receiver`. Consider adding this for full ERC20 compliance.
  2. Additionally, the same `Transfer` event should be emitted when the `protocolFeeRecipient` and `reserveFeeRecipient` have their shares minted.
  3. An equivalent `Transfer` event is also required when a `withdraw` occurs, with the `to` address set to `0x0`.
  4. This should also apply when a user claims interest. If tokens are being "burned", it must be reflected via a `Transfer` event.

Dahlia: Addressed `Transfer` event in PR 25.

3.1.4 Protocol and Reserve Fees cannot be withdrawn

Severity: High Risk

Context: InterestImpl.sol#L51

Description: Shares equivalent to the protocol and reserve fees are assigned to the correct recipient. However, they cannot withdraw because only the `wrappedVault` is allowed to call the `Dahlia.withdraw` function. The fee recipients do not have any vault shares minted, so they cannot withdraw through vault either.

Proof of Concept: Update the `test/core/integration/AccrueInterestIntegration.t.sol:AccrueInterestIntegrati` `int_accrueInterest_withFees` to have below additional checks.

Here's the code with the `diff` formatting preserved and structured for better readability:

```
function test_int_accrueInterest_withFees(
    TestTypes.MarketPosition memory pos,
    uint256 blocks,
    uint32 fee
) public {
    vm.pauseGasMetering();
    pos = vm.generatePositionInLtvRange(pos, TestConstants.MIN_TEST_LLTV, $.marketConfig.lltv);
    vm.dahliaSubmitPosition(pos, $.carol, $.alice, $);

    uint32 protocolFee = uint32(bound(uint256(fee), BoundUtils.toPercent(2), BoundUtils.toPercent(5)));
    uint32 reserveFee = uint32(bound(uint256(fee), BoundUtils.toPercent(1), BoundUtils.toPercent(2)));

    vm.startPrank($.owner);
    if (protocolFee != $.dahlia.getMarket($.marketId).protocolFeeRate) {
        $.dahlia.setProtocolFeeRate($.marketId, protocolFee);
    }
    if (reserveFee != $.dahlia.getMarket($.marketId).reserveFeeRate) {
        $.dahlia.setReserveFeeRate($.marketId, reserveFee);
    }
    vm.stopPrank();

    blocks = vm.boundBlocks(blocks);

    IDahlia.Market memory state = $.dahlia.getMarket($.marketId);
    uint256 totalBorrowBeforeAccrued = state.totalBorrowAssets;
    uint256 totalLendBeforeAccrued = state.totalLendAssets;
    uint256 totalLendSharesBeforeAccrued = state.totalLendShares;

    uint256 deltaTime = blocks * TestConstants.BLOCK_TIME;
    (uint256 interestEarnedAssets, uint256 newRatePerSec,) =
        $.marketConfig.irm.calculateInterest(
            deltaTime,
            state.totalLendAssets,
            state.totalBorrowAssets,
            state.fullUtilizationRate
        );

    uint256 protocolFeeShares = InterestImpl.calcFeeSharesFromInterest(
        state.totalLendAssets,
        state.totalLendShares,
        interestEarnedAssets,
        protocolFee
    );
    uint256 reserveFeeShares = InterestImpl.calcFeeSharesFromInterest(
        state.totalLendAssets,
        state.totalLendShares,
        interestEarnedAssets,
      reserveFee
    );

    vm.forward(blocks);
    if (interestEarnedAssets > 0) {
        vm.expectEmit(true, true, true, true, address($.dahlia));
        emit IDahlia.DahliaAccrueInterest(
            $.marketId,
            newRatePerSec,
            interestEarnedAssets,
            protocolFeeShares,
            reserveFeeShares
        );
    }

    $.dahlia.accrueMarketInterest($.marketId);

    IDahlia.Market memory stateAfter = $.dahlia.getMarket($.marketId);
    assertEq(stateAfter.totalLendAssets, totalLendBeforeAccrued + interestEarnedAssets, "total supply");
    assertEq(stateAfter.totalBorrowAssets, totalBorrowBeforeAccrued + interestEarnedAssets, "total borrow");
    assertEq(
        stateAfter.totalLendShares,
        totalLendSharesBeforeAccrued + protocolFeeShares + reserveFeeShares,
        "total lend shares"
    );

    IDahlia.UserPosition memory userPos1 = $.dahlia.getPosition($.marketId,
    ↪  ctx.wallets("PROTOCOL_FEE_RECIPIENT"));
```

```
      IDahlia.UserPosition memory userPos = $.dahlia.getPosition($.marketId,
      ↪  ctx.wallets("RESERVE_FEE_RECIPIENT"));
      assertEq(userPos1.lendShares, protocolFeeShares, "protocolFeeRecipient's lend shares");
      assertEq(userPos.lendShares, reserveFeeShares, "reserveFeeRecipient's lend shares");

      if (interestEarnedAssets > 0) {
          assertEq(stateAfter.updatedAt, block.timestamp, "last update");
      }

+     IDahlia.Market memory m = $.dahlia.getMarket($.marketId);
+     WrappedVault vault = WrappedVault(address(m.vault));

+     // Ensure both fee recipients have zero WrappedVault balances
+     assertEq(
+         vault.balanceOf(address(ctx.wallets("RESERVE_FEE_RECIPIENT"))),
+         0,
+         "reserveFeeRecipient's wrappedVault balance"
+     );
+     assertEq(
+         vault.balanceOf(address(ctx.wallets("PROTOCOL_FEE_RECIPIENT"))),
+         0,
+         "protocolFeeRecipient's wrappedVault balance"
+     );

+     // Verify that withdrawing directly from Dahlia fails for both fee recipients
+     address reserveFeeRecipient = ctx.wallets("RESERVE_FEE_RECIPIENT");
+     vm.startPrank(reserveFeeRecipient);
+     vm.expectRevert();
+     $.dahlia.withdraw($.marketId, userPos.lendShares, reserveFeeRecipient, reserveFeeRecipient);
+     vm.stopPrank();

+     address protocolFeeRecipient = ctx.wallets("PROTOCOL_FEE_RECIPIENT");
+     vm.startPrank(protocolFeeRecipient);
+     vm.expectRevert();
+     $.dahlia.withdraw($.marketId, userPos1.lendShares, protocolFeeRecipient, protocolFeeRecipient);
+     vm.stopPrank();
  }
```

Recommendation: Add an additional function that allows protocol fee recipient and reserve fee recipient to withdraw.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.2 Medium Risk

3.2.1 If a rewards schedule's start time is set as `0`, reward interests do not get accumulated

Severity: Medium Risk

Context: WrappedVault.sol#L361-L362

Description: When one creates a reward schedule it is allowed to set the `start` to `0`. But in `_calculateRewardsPerToken` the interest accumulation process is skipped which causes the user rewards to not be updated:

```
if (rewardsInterval_.start == 0) return rewardsPerTokenOut;
```

Perhaps this was due to the assumption that if the start time is `0` then there couldn't be any active reward schedules. Which this is not true. Here is a simple PoC which can be added to `test/core/integration/WrappedVaultTakeRewards.t.sol`:

```
function testTakeRewardsWithStartIntervalZero() public {
    // !!!!!! change this params for checking rewards
    uint256 rewardAmount = 100_000 * 10 ** TestLib.rewardERC20decimals; // 1000 USDC rewards
    uint256 depositAmount = 500 * 10 ** TestLib.vaultERC20decimals; // 500 ETH

    uint32 start = 0; // <--- note the `0` start time
    uint32 duration = 30 days + uint32(block.timestamp);
    console.log("duration (seconds):", duration);

    testIncentivizedVault.addRewardsToken(address(rewardToken1));
    rewardToken1.mint(address(this), rewardAmount);
    rewardToken1.approve(address(testIncentivizedVault), rewardAmount);
    testIncentivizedVault.setRewardsInterval(address(rewardToken1), start, start + duration, rewardAmount,
    ↪  DEFAULT_FEE_RECIPIENT);


    RewardMockERC20(address(token)).mint($.alice, depositAmount);

    vm.startPrank($.alice);
    token.approve(address(testIncentivizedVault), depositAmount);
    uint256 d1 = testIncentivizedVault.deposit(depositAmount, $.alice);
    vm.stopPrank();

    console.log("user1 deposit:        ", d1);
    console.log("undistributed rewards:", rewardToken1.balanceOf(address(testIncentivizedVault)));
    console.log("user1 rewards:        ", rewardToken1.balanceOf($.alice));

    vm.warp(start + duration + 1);
    vm.startPrank($.alice);
    testIncentivizedVault.claim($.alice);
    vm.stopPrank();

    console.log("\n #### End of rewards period. Expecting DEFAULT_FRONTEND_FEE and DEFAULT_PROTOCOL_FEE stay");
    console.log("undistributed rewards:", rewardToken1.balanceOf(address(testIncentivizedVault)));
    console.log("user1 rewards:        ", rewardToken1.balanceOf($.alice));
}
```

Recommendation: To check whether is an active reward schedule or not it is best to check non-zeroness of the `rate` instead. As when one is creating a reward schedule non-zeroness of the final `rate` gets checked.

1. In `setRewardsInterval` (WrappedVault.sol#L318), `rate` is checked to be non-zero.
2. In `extendRewardsInterval` (WrappedVault.sol#L272), `rate` has been check to make sure it does not decrease (if non-zero stays non-zero).

Apply the following changes:

```
diff --git a/src/royco/contracts/WrappedVault.sol b/src/royco/contracts/WrappedVault.sol
index 0578e04..c684483 100644
--- a/src/royco/contracts/WrappedVault.sol
+++ b/src/royco/contracts/WrappedVault.sol
@@ -358,8 +358,8 @@ contract WrappedVault is Owned, ERC20, IWrappedVault {
         // No changes if the program hasn't started
         if (block.timestamp < rewardsInterval_.start) return rewardsPerTokenOut;

-        // No changes if the start value is zero
-        if (rewardsInterval_.start == 0) return rewardsPerTokenOut;
+        // No changes if the rate is zero, ie there is no active reward scheudle for the reward token in
↪   context
+        if (rewardsInterval_.rate == 0) return rewardsPerTokenOut;

         // Stop accumulating at the end of the rewards interval
         uint256 updateTime = block.timestamp < rewardsInterval_.end ? block.timestamp : rewardsInterval_.end;
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed by disallowing the start time to be `0`.

3.2.2 The `rewardsAdded` calculation in `extendRewardsInterval` is incorrect

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The `rewardsAdded` calculation in `extendRewardsInterval` is incorrect. The `rewardsAdded` and the new `rate` are calculated as below in `extendRewardsInterval`:

```
uint256 remainingRewards = rewardsInterval.rate * (rewardsInterval.end - newStart);
uint256 rate = (rewardsAdded - frontendFeeTaken - protocolFeeTaken + remainingRewards) / (newEnd - newStart);
rewardsAdded = (rate - rewardsInterval.rate) * (newEnd - newStart) + frontendFeeTaken + protocolFeeTaken;
```

we have:

$$a_r = r_{old}(t_{e,0} - t_{s,1})$$

$$r_{new} = \left\lfloor \frac{a_0 + a_r - f}{t_{e,1} - t_{s,1}} \right\rfloor$$

and so a_1 should have been calcualted as:

$$a_0 \geq a_1 = r_{new}(t_{e,1} - t_{s,1}) - a_r + f$$

or

$$a_1 = r_{new}(t_{e,1} - t_{s,1}) - r_{old}(t_{e,0} - t_{s,1}) + f$$

but the current implementation incorectly calcualtes a_1 as:

$$a_{1,incorrect} = (r_{new} - r_{old})(t_{e,1} - t_{s,1}) + f$$

which assumes that $t_{e,1}$ and $t_{e,0}$ are equal. The effect would be that the vault pulls less funds from the owner of the vault which would not match the newly set `rate`. The discrepncy is the following:

$$a_1 - a_{1,incorrect} = r_{old}(t_{e,1} - t_{e,0}) = r_{old}\Delta t_e$$

parameter	description
r_{old}	`rewardsInterval.rate`
r_{new}	`rate`
a_r	`remainingRewards`
a_0	`rewardsAdded` before update
a_1	`rewardsAdded` after update
f	`frontendFeeTaken + protocolFeeTaken`
$t_{s,0}$	`rewardsInterval.start` before update
$t_{s,1}$	`newStart`
$t_{e,0}$	`rewardsInterval.end`
$t_{e,1}$	`newEnd`

Recommendation: Use the following calcualtion instead:

```
rewardsAdded = rate * (newEnd - newStart) - remainingRewards + frontendFeeTaken + protocolFeeTaken;
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.2.3 Inflation of zero `totalBorrowShares` to disable borrowing is possible

Severity: Medium Risk

Context: SharesMathLib.sol#L20-L22

Description:

- When `totalBorrowShares` equals zero, an attacker can exploit the rounding by borrowing `SHARES_OFFSET - 1` shares.
  - Initially, `totalBorrowShares = 0` and `totalBorrowAssets = 0`.
  - The borrow amount (`borrowAssets`) is calculated as:

```
    borrowAssets = (borrowShares * totalBorrowAssets) / (totalBorrowShares + SHARES_OFFSET)
                 = (SHARES_OFFSET - 1) * 1 / SHARES_OFFSET = 0
```

  - The attacker receives `SHARES_OFFSET` borrowShares and 0 borrowAssets, causing `totalBorrowShares` to increase by `SHARES_OFFSET - 1`.
  - Repeating this process in a loop, the attacker exponentially inflates `totalBorrowShares` by borrowing `totalBorrowShares + SHARES_OFFSET -1` of shares each time, while keeping `totalBorrowAssets` at 0.
- The attacker continues until `totalBorrowShares` reaches `type(uint128).max`. No collateral is required since the attacker hasn't borrowed any assets.
- At this point, genuine users are unable to borrow even a minimal amount because borrowing 1 wei results in an overflow: `borrowShares minted > type(uint).max`.
- Additionally, since repaying debts on behalf of others is allowed, the attacker can reset `totalBorrowShares` to zero when the market is in use to execute the attack.

Proof of Concept: Add below test in `test/core/integration/BorrowIntegration.t.sol`:

```
function test_int_borrow_disable_borrow_attack() public {
    uint256 SHARES_OFFSET = 1e6;

    // Ensure that the current borrow shares are zero
    assertEq($.dahlia.getMarket($.marketId).totalBorrowShares, 0);

    // Users have lend some tokens
    vm.dahliaLendBy($.carol, 1 ether, $);

    // The attacker supplies enough collateral to borrow 1 wei of assets
    vm.dahliaSupplyCollateralBy($.alice, 1, $);


    uint256 i;
    while ($.dahlia.getMarket($.marketId).totalBorrowShares < type(uint128).max / 2 ) {
        uint256 totalBorrowShares = $.dahlia.getMarket($.marketId).totalBorrowShares;

        // The attacker borrows (totalBorrowShares + SHARES_OFFSET - 1) shares
        vm.prank($.alice);
        $.dahlia.borrow($.marketId, 0, totalBorrowShares + SHARES_OFFSET - 1, $.alice, $.alice);

        // The attacker borrowed 0 assets, but totalBorrowShares increased by (totalBorrowShares +
        ↪  SHARES_OFFSET - 1)
        assertEq(
            $.dahlia.getMarket($.marketId).totalBorrowShares,
            totalBorrowShares + (totalBorrowShares + SHARES_OFFSET - 1)
        );
        assertEq($.dahlia.getMarket($.marketId).totalBorrowAssets, 0);

        assertEq($.dahlia.getPosition($.marketId, $.alice).borrowShares,
        ↪  $.dahlia.getMarket($.marketId).totalBorrowShares);
        //according to the protocol, even with a lot borrowed shares, the attacker has only borrowed 1 wei of
        ↪  assets
        (uint256 attackerBorrowedAssets,,) = $.dahlia.getMaxBorrowableAmount($.marketId, $.alice);
        assertEq(attackerBorrowedAssets, 1);

        i++;
    }
    // very few iterations means gas spent is low. This is an extreme example where attacker disables borrowing
    ↪  more than 1 wei,
    // for disabling borrowing more than 1 ether for example, it takes even less loops
    assertEq(i,108);



    assertGt($.dahlia.getMarket($.marketId).totalBorrowShares, type(uint128).max / 2);
    // Now that totalBorrowShares exceed type(uint128).max / 2, borrowing even 2 wei of assets by a genuine
    // user will revert because borrowShares that will be minted will be greater than type(uint128).max and
    ↪  will overflow

    vm.dahliaSupplyCollateralBy($.bob, 2, $);

    // Expect a revert with `panic: arithmetic underflow or overflow (0x11)`
    // because totalBorrowShares will overflow
    vm.expectRevert();
    vm.dahliaBorrowBy($.bob, 2, $);
}
```

Recommendation: Do not use virtual shares calculation for borrow shares. There is no need for it if rounding directions are correct. Or make sure there is a lower bound check for `assets` if only the `shares` are provided to the `borrow` endpoint:

```
// Calculate assets or shares
if (assets > 0) {
    shares = assets.toSharesUp(market.totalBorrowAssets, market.totalBorrowShares);
} else {
    assets = shares.toAssetsDown(market.totalBorrowAssets, market.totalBorrowShares);
    require(assets > minimumBorrow(market.loanToken), Error());
}
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed by removing the option to supply `shares` to the `borrow` endpoint.

3.2.4 Inconsistency between precision factors / dimensional analysis

Severity: Medium Risk

Context: WrappedVault.sol#L511-L516

Description: The precision/dimensional analysis for `rewardsRate` and `dahliaRate` are matching:

$$\frac{10^{18}[I]}{[T][L]}$$

and in this case where $[\mathrm{I}] = [\mathrm{L}]$ simplifies to:

$$\frac{10^{18}}{[T]}$$

The discprencey arises when one looks at the code in `VaultMarketHub.allocateOffer(...)` (VaultMarketHub.sol#L232-L234):

```
offer.incentivesRatesRequested[i] >
↪  WrappedVault(offer.targetVault).previewRateAfterDeposit(offer.incentivesRequested[i], fillAmount)
```

In this comparison both sides should have the same precision/dimension. According to the NatSpec for `incentivesRatesRequested` (VaultMarketHub.sol#L23-L24) we have:

```
/// @custom:field incentivesRatesRequested The desired incentives per input token per second to fill the offer,
↪  measured in
/// wei of incentives per wei of deposited assets per second, scaled up by 1e18 to avoid precision loss
```

or in other words:

$$[\texttt{incentivesRatesRequested}] = 10^{18} \cdot \frac{(10^{18}/10^{d_I})[I]}{[T](10^{18}/10^{d_L})[L]} = \frac{10^{18+d_L-d_I}[I]}{[T][L]}$$

In the finding the notation $[\cdot]$ represents the precision ($\text{prec} = 10^x$) or dimension of a quantity.

parameter	description
I	reward
L	DEPOSIT_ASSET or loan token of the corresponding Dahlia market.
T	timestamp in seconds
d_X	X.decimals()

And so as soon as the reward and deposit token do not have the same decimals the precision of `WrappedVault(offer.targetVault).previewRateAfterDeposit(...)` and `offer.incentivesRatesRequested[I]` would not match.

Recommendation: Make sure that `rewardsRate` (WrappedVault.sol#L517) returned by `previewRateAfterDeposit` is scaled (up/or down) accordingly with the factor of $10^{d_L - d_I}$.

3.2.5 Unnecessary `payable` functions may lead to permanently locked funds

Severity: Medium Risk

Context: Dahlia.sol#L165

Description: The `payable` modifier on `withdraw` and `claimInterest` enables callers to pass nonzero `msg.value` which is not used by the function implementation. There is no functionality anywhere to withdraw these funds so they will be stuck permanently.

Recommendation: Remove the `payable` modifiers from these functions.

Dahlia: Acknowledged. We'll keep some of these `payable` modifiers to ensure gas efficiency and consistency with Royco's implementation.

Cantina Managed: Acknowledged.

3.2.6 Vault tokens lose fungibility after a user claims interest

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: As explained in the finding "`WrappedVault` share transfers results in stuck funds":

> Only the `wrappedVault` is allowed to `lend` and `withdraw` assets to/from the Dahlia lending contract. Users can interact with `wrappedVault` through its `deposit` and `withdraw` functions.
>
> When a user deposits assets into `wrappedVault`, they are minted shares of `wrappedVault` as expected. Internally, `wrappedVault` calls `dahlia.lend` and assigns the Dahlia shares to the receiver (the depositor) instead of keeping them for itself. This ensures that individual accounting for `claimInterest` works correctly for each depositor.

All the interest accrued is claimed for the depositor by the vault whenever the user calls the `vault.claim` function. This, in turn, calls `dahlia.claimInterest` for the depositor. The `dahlia.claimInterest` function decreases the lendShares equivalent to the interest accrued and sends it to the depositor. However, the problem is that the `wrappedVault` balance equivalent to the claimed interest is never burned.

This means that `wrappedVault.balanceOf(depositor)` is no longer the same as `wrappedVault.balanceOfDahlia(depositor)`. The last `wrappedVault` balance equivalent to the claimed interest can never be withdrawn or burned.

As a result, each `wrappedVault` share balance for a user is worth a different amount, and `wrappedVault` shares are no longer fungible.

Proof of Concept: Add the test below in `test/core/integration/WrappedVaultIntegration.t.sol`:

```
function test_wrappedVault_erc20_not_fungible() public {
    vm.startPrank(WrappedVault(address(marketProxy)).owner());
    WrappedVault(address(marketProxy)).addRewardsToken(address($.loanToken));
    vm.stopPrank();
    // Alice deposits some tokens
    uint256 assets = 1 ether;
    $.loanToken.setBalance($.alice, assets);

    vm.startPrank($.alice);
    $.loanToken.approve(address(marketProxy), assets);
    vm.resumeGasMetering();
    marketProxy.deposit(assets, $.alice);
    vm.stopPrank();

    // Bob deposits the same amount of tokens
    $.loanToken.setBalance($.bob, assets);

    vm.startPrank($.bob);
    $.loanToken.approve(address(marketProxy), assets);
    vm.resumeGasMetering();
    marketProxy.deposit(assets, $.bob);
    vm.stopPrank();

    // Carol supplies some collateral and borrows tokens
    vm.dahliaSupplyCollateralBy($.carol, 10 ether, $);

    // Carol borrows 1 ether.
    vm.prank($.carol);
    $.dahlia.borrow($.marketId, 1 ether, 0, $.carol, $.carol);

    // some time passes
    vm.warp(block.timestamp + 10 days);

    // Alice claims her interest
    vm.prank($.alice);
    WrappedVault(address(marketProxy)).claim($.alice);

    uint256 aliceVaultBalance = marketProxy.balanceOf($.alice);
    uint256 bobVaultBalance = marketProxy.balanceOf($.bob);

    // Assert that Alice's vault balance is still the same as Bob's
    // even though Bob has not claimed any interest yet
    assertEq(aliceVaultBalance, bobVaultBalance);

    // In reality, Alice's balance is worth less than Bob's balance
    // This demonstrates that the vault tokens are not fungible

    assertLt(marketProxy.maxWithdraw($.alice), marketProxy.maxWithdraw($.bob));

    // To resolve this issue, shares should be burned as interest is claimed.
}
```

Recommendation: `wrappedVault` tokens equivalent to the lendShares withdrawn to pay for the interest should be burned.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed by querying the lend shares from `Dahlia` and using it as wrapped vault shares.

3.2.7 Protocol and Reserves shares calculation is incorrect

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The protocol and reserves are being underpaid by a small amount each time. In the equation, `totalLendAssets` is increased by the full `interestEarnedAssets`, which is incorrect. This results in a slightly lower rate than what the protocol and reserves deserve, as the `protocolFee` and `reserveFee` are included in the `interestEarnedAssets`.

It's analogous to adding a depositor's assets to `totalLendAssets` before calculating shares using `totalLendShares` and `totalLendAssets` ratio.

Additionally, since virtual shares are not being accounted for, Dahlia underpays protocols and reserves by a multiple of 10ˆ6.

Proof of Concept: Here's a test showing that the updated calculation is much closer to the correct value than the current buggy implementation, which under-calculates shares.

```
// SPDX-License-Identifier: BUSL-1.1
pragma solidity ^0.8.27;

import { Test } from "forge-std/Test.sol";
import { Constants } from "src/core/helpers/Constants.sol";
import { SharesMathLib } from "src/core/helpers/SharesMathLib.sol";
import { InterestImpl } from "src/core/impl/InterestImpl.sol";

contract FeeShareCalculationBugDemo is Test {
    function test_feeShares_bug_demo() public pure {
        uint256 totalLendAssets = 200 ether;
        uint256 totalLendShares = 100 ether;

        uint256 interestEarnedAssets = 10 ether;
        uint256 feeRate = 0.1e5; //10%

        uint256 feeSharesUsingBuggy = InterestImpl.calcFeeSharesFromInterest(totalLendAssets, totalLendShares,
        ↪  interestEarnedAssets, feeRate);
        uint256 feeSharesUsingFixed = fixedCalcFeeSharesFromInterest(totalLendAssets, totalLendShares,
        ↪  interestEarnedAssets, feeRate);

        uint256 feeAssetUsingBuggy =
            SharesMathLib.toAssetsDown(feeSharesUsingBuggy, totalLendAssets + interestEarnedAssets,
            ↪  totalLendShares + feeSharesUsingBuggy);
        uint256 feeAssetUsingFixed =
            SharesMathLib.toAssetsDown(feeSharesUsingFixed, totalLendAssets + interestEarnedAssets,
            ↪  totalLendShares + feeSharesUsingFixed);

        // We know from the above details that at a 10% fee rate, whatever feeShares we get, we should be able
        ↪  to convert it back to 1 ether of assets.
        // As shown below, the fixed calculation is much closer to 1 ether, which we know is the correct
        ↪  answer. It's not exactly 1 ether because of precision loss, and virtual shares are
        // not being accounted for.

        assertApproxEqAbs(feeAssetUsingBuggy, 1 ether, 1e16); //decrease the precision further and it fails
        assertApproxEqAbs(feeAssetUsingFixed, 1 ether, 1e4); //much closer to 1 ether
    }

    function fixedCalcFeeSharesFromInterest(uint256 totalLendAssets, uint256 totalLendShares, uint256
    ↪  interestEarnedAssets, uint256 feeRate)
        internal
        pure
        returns (uint256 feeShares)
    {
        feeShares = (interestEarnedAssets * feeRate * totalLendShares)
            / (Constants.FEE_PRECISION * (totalLendAssets + interestEarnedAssets - (interestEarnedAssets *
            ↪  feeRate / Constants.FEE_PRECISION)));
    }
}
```

Recommendation:

- The calcualtion provided by `fixedCalcFeeSharesFromInterest` is more accurate given:
  - It does not considering virtual shares/assets.
  - When one is dealing with only updating the shares of one position.
- Since here we are updating the shares of two different positions the more accurate formula (as suggested by @Saw-mon-and-Natalie) would be:

$$i \in \{0, 1\}$$

$$s_i = r_i \left(\frac{a(A_L + a)}{A_L^2 + (1 - r_0 + 1 - r_1)aA_L + (1 - r_0 - r_1)a^2} \right) S_L$$

Parameters:

Parameter	Description
r_0	protocolFeeRate / FEE_PRECISION

Parameter	Description
r_1	reserveFeeRate / FEE_PRECISION
S_L	totalLendShares
s_0	protocolFeeShares
s_1	reserveFeeShares
A_L	totalLendAssets
a	interestEarnedAssets

If one of the

$$r_i$$

values is 0, the above formula reduces to the one used in `fixedCalcFeeSharesFromInterest`.

- Update the `executeMarketAccrueInterest` as shown below should fix the problem without introducing too much complexity. `Dahlia` ensures that `totalLendAssets` are decreased by `protocolFeeAssets + reserveFeeAssets`, allows the protocol to "deposit" `protocolFeeAssets`, increases `totalLendAssets` by `protocolFeeAssets` and `totalLendShares` by `protocolFeeShares`, and then allows the reserve to "deposit".

```
function executeMarketAccrueInterest(
    IDahlia.Market storage market,
    IDahlia.UserPosition storage protocolFeeRecipientPosition,
    IDahlia.UserPosition storage reserveFeeRecipientPosition
) internal {
    if (address(market.irm) == address(0)) {
        return;
    }

    uint256 deltaTime = block.timestamp - market.updatedAt;
    if (deltaTime == 0) {
        return;
    }
    uint256 totalLendAssets = market.totalLendAssets;
    uint256 totalBorrowAssets = market.totalBorrowAssets;
    (uint256 interestEarnedAssets, uint256 newRatePerSec, uint256 newFullUtilizationRate) =
        IIrm(market.irm).calculateInterest(deltaTime, totalLendAssets, totalBorrowAssets,
        ↪  market.fullUtilizationRate);

    if (interestEarnedAssets > 0) {
        market.fullUtilizationRate = uint64(newFullUtilizationRate);
        market.ratePerSec = uint64(newRatePerSec);
        market.totalBorrowAssets += interestEarnedAssets;
        market.totalLendAssets += interestEarnedAssets;

        uint256 protocolFeeAssets = interestEarnedAssets * market.protocolFeeRate /
        ↪  Constants.FEE_PRECISION;
        uint256 reserveFeeAssets = interestEarnedAssets * market.reserveFeeRate /
        ↪  Constants.FEE_PRECISION;

        market.totalLendAssets -= protocolFeeAssets + reserveFeeAssets;

        // Calculate protocol fee
        uint256 protocolFeeShares = 0;
        uint256 protocolFeeRate = market.protocolFeeRate;

        if (protocolFeeRate > 0) {
            protocolFeeShares = protocolFeeAssets.toSharesDown(market.totalLendAssets,
            ↪  market.totalLendShares);
            protocolFeeRecipientPosition.lendShares += protocolFeeShares.toUint128();
            market.totalLendShares += protocolFeeShares;
            market.totalLendAssets += protocolFeeAssets;
        }

        // Calculate reserve fee
        uint256 reserveFeeShares = 0;
        uint256 reserveFeeRate = market.reserveFeeRate;
        if (reserveFeeRate > 0) {
            reserveFeeShares = reserveFeeAssets.toSharesDown(market.totalLendAssets,
            ↪  market.totalLendShares);
            reserveFeeRecipientPosition.lendShares += reserveFeeShares.toUint128();
            market.totalLendShares += reserveFeeShares;
```

```
              market.totalLendAssets += reserveFeeAssets;
          }

          emit IDahlia.DahliaAccrueInterest(market.id, newRatePerSec, interestEarnedAssets,
          ↪   protocolFeeShares, reserveFeeShares);
          market.updatedAt = uint48(block.timestamp);
      }
  }
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: The fix provided follows the following suggestion regarding the main `if` block:

```
if (interestEarnedAssets > 0) {
    totalLendAssets += interestEarnedAssets;

    uint256 protocolFeeAssets = interestEarnedAssets * market.protocolFeeRate / Constants.FEE_PRECISION;
    uint256 reserveFeeAssets  = interestEarnedAssets * market.reserveFeeRate  / Constants.FEE_PRECISION;
    uint256 totalLendShares   = market.totalLendShares;
    uint256 sumOfFeeAssets    = protocolFeeAssets + reserveFeeAssets;
    uint256 sumOfFeeShares    = sumOfFeeAssets.toSharesDown(totalLendAssets - sumOfFeeAssets, totalLendShares);

    totalLendShares += sumOfFeeShares;

    uint256 protocolFeeShares = protocolFeeAssets.toSharesDown(totalLendAssets, totalLendShares);
    uint256 reserveFeeShares  = sumOfFeeShares - protocolFeeShares;

    // update storage
    // the positions can be conditioned to be updated based on the non-zeroness of the fee shares
    protocolFeeRecipientPosition.lendShares += protocolFeeShares.toUint128();
        reserveFeeRecipientPosition.lendShares +=  reserveFeeShares.toUint128();

    market.totalLendShares     = totalLendShares;
    market.totalLendAssets     = totalLendAssets;
    market.totalBorrowAssets   = totalBorrowAssets + interestEarnedAssets;
    market.fullUtilizationRate = uint64(newFullUtilizationRate);
    market.ratePerSec          = uint64(newRatePerSec);
    market.updatedAt           = uint48(block.timestamp);

    emit IDahlia.DahliaAccrueInterest(market.id, newRatePerSec, interestEarnedAssets, protocolFeeShares,
    ↪   reserveFeeShares);
}
```

Plus two other modifications:

1. Updating `market.fullUtilizationRate` and `market.ratePerSec` before the `if` block.
2. Minting shares for the `protocolFeeRecipient` and `reserveFeeRecipient` in the corresponding market's wrapped vault:

```
    if (protocolFeeShares > 0) {
        positions[protocolFeeRecipient].lendShares += protocolFeeShares.toUint128();
        market.vault.mintFees(protocolFeeShares, protocolFeeRecipient);
    }
    if (reserveFeeShares > 0) {
        positions[reserveFeeRecipient].lendShares += reserveFeeShares.toUint128();
        market.vault.mintFees(reserveFeeShares, reserveFeeRecipient);
    }
```

3.3 Low Risk

3.3.1 Rounding directions

Severity: Low Risk

Context: MarketMath.sol#L78, MarketMath.sol#L122-L123, BorrowImpl.sol#L83

Description/Recommendation:

- BorrowImpl.sol#L83: `borrowedAssets` is rounded up, it would be best to underestimate this value when used to check the borrower does not overborrow past the allowed total borrow capacity.

- MarketMath.sol#L122-L123: `totalCollateralCapacity` should be underestimates to make sure `LTV` gets overestimated.
- MarketMath.sol#L78-L80: `reserveAssets` is overestimated even though it's converted from shares to assets. That would mean the lend shares for `reservePosition` are favoured more compared to regular user positions.

Dahlia: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Incorrect source of funds for `supplyAndBorrow` and `repayAndWithdraw` calls

Severity: Low Risk

Context: Dahlia.sol#L250, Dahlia.sol#L274

Description: The `Dahlia.supplyAndBorrow` function is protected by `isSenderPermitted(owner)` to ensure that only the owner or an address permitted by the owner can call it to supply collateral and borrow assets in a single function call. However, when pulling the supplied collateral, the protocol does it from `msg.sender` instead of the `owner`. This undermines the purpose of the function, as it forces the owner to transfer funds to the permitted address before calling this function.

A similar issue exists in the `Dahlia.repayAndWithdraw` function, where the assets required for repayment are pulled from `msg.sender` instead of the `owner`.

Recommendation: Modify the functions to pull the funds from the `owner` rather than `msg.sender`.

Dahlia: Added in commit df70af0d.

Cantina Managed: Fixed.

3.3.3 Loss of precision in Variable `IRM`

Severity: Low Risk

Context: VariableIrm.sol#L106-L108

Description: `newFullUtilizationRate` uses "division before multiplication" leading to excess loss of precision. `decayGrowth` is divided by `leftUtilization` before being multiplied with `fullUtilizationRate`.

See SolidityScan's blog entry for why this is important.

Recommendation: Always perform integer multiplication before division where possible.

3.3.4 `WrappedVault` doesn't follow ERC4626 standard correctly

Severity: Low Risk

Context: WrappedVault.sol#L557-L570, WrappedVault.sol#L613-L614, WrappedVault.sol#L635

Description:

- First instance:

```
function maxMint(address) external pure returns (uint256 maxShares) {
    maxShares = type(uint128).max;
}
```

```
function maxDeposit(address) external pure returns (uint256 maxAssets) {
    maxAssets = type(uint128).max;
}
```

  Given that `maxMint` is `type(uint128).max`, it doesn't make sense for `maxDeposit` to not also be `type(uint128).max`. Both amounts need to be convertible to each other.

- Second instance:

```
function maxWithdraw(address addr) external view returns (uint256 maxAssets) {
    maxAssets = convertToAssets(balanceOfDahlia(addr));
}
```

The maxWithdraw function doesn't account for the available funds in the market. If utilization is very high, a user might not be able to withdraw the full amount. According to the ERC-4626 standard, the function should underestimate the maximum amount if necessary:

> MUST return the maximum amount of assets that could be transferred from the owner through withdraw and not cause a revert, which MUST NOT be higher than the actual maximum that would be accepted (it should underestimate if necessary).

For example:

- Total assets**: 12,000
- Borrowed: 11,000
- User deposit: 2,000

If the user calls maxWithdraw(), it should return 1,000 instead of 2,000. The same issue applies to maxRedeem.

- Third instance: WrappedVault.sol#L557-L570, from ERC4626:

> Mints exactly shares Vault shares to receiver by depositing assets of underlying tokens.

Note that in the case of WrappedVault this might not be always true. Since shares is converted to assets and then fed into dahlia.lend(..., assets, ...) which returns the final shares' to be used/minted. So the initial shares input and the final minted shares might not be equal.

For the withdraw endpoint one has the following check (WrappedVault.sol#L577) that makes sure there is no discrepancy between the assets provided and actualAssets:

```
if (assets != actualAssets) revert InvalidWithdrawal();
```

Recommendation:

1. maxDeposit should return SharesMathLib.toAssetsUp(type(uint128).max, market.totalLendAssets, market.totalLendShares);.
2. Update the maxWithdraw and maxRedeem functions as suggested.
3. Make sure exactly the provided shares are minted for the user. This would mean one might need to potentially introduce a new endpoint on Dahlia side where instead of the assets as an input parameter, it would accept shares.

Dahlia: Fixed in commit ea7bb1ec.

Cantina Managed: Fixed.

3.4 Gas Optimization

3.4.1 Use calldata instead of memory for input location in updatePermissionWithSig

Severity: Gas Optimization

Context: Permitted.sol#L42, Permitted.sol#L45, IPermitted.sol#L31

Description: updatePermissionWithSig assigns memory as the location of signature which consumes more gas.

Recommendation: It would be best to use calldata here and also utilise ECDSA.recoverCalldata:

```
diff --git a/src/core/abstracts/Permitted.sol b/src/core/abstracts/Permitted.sol
index b674fa6..156fd15 100644
--- a/src/core/abstracts/Permitted.sol
+++ b/src/core/abstracts/Permitted.sol
@@ -39,10 +39,10 @@ abstract contract Permitted is IPermitted, EIP712, Nonces {
     }

     /// @inheritdoc IPermitted
-    function updatePermissionWithSig(Data memory data, bytes memory signature) external {
+    function updatePermissionWithSig(Data memory data, bytes calldata signature) external {
         require(block.timestamp <= data.deadline, Errors.SignatureExpired());
         bytes32 digest = hashTypedData(data);
-        address recoveredSigner = ECDSA.recover(digest, signature);
+        address recoveredSigner = ECDSA.recoverCalldata(digest, signature);
         require(data.signer == recoveredSigner, Errors.InvalidSignature());
         _useCheckedNonce(recoveredSigner, data.nonce);

diff --git a/src/core/interfaces/IPermitted.sol b/src/core/interfaces/IPermitted.sol
index b99cedf..dae15db 100644
--- a/src/core/interfaces/IPermitted.sol
+++ b/src/core/interfaces/IPermitted.sol
@@ -28,5 +28,5 @@ interface IPermitted {
     /// @dev Fails if the signature is reused or invalid
     /// @param data The permission details
     /// @param signature The EIP-712 signature
-    function updatePermissionWithSig(Data calldata data, bytes memory signature) external;
+    function updatePermissionWithSig(Data calldata data, bytes calldata signature) external;
 }
```

`pnpm run diff`:

```
test_int_proxy_withdrawByShares(uint256) (gas: -1 (-0.001%))
test_int_proxy_withdrawByAssets(uint256) (gas: -1 (-0.001%))
test_int_proxy_depositByShares(uint256) (gas: -1 (-0.001%))
test_int_proxy_depositByAssets(uint256) (gas: -1 (-0.001%))
test_int_lend_byAssets(uint256) (gas: -1 (-0.001%))
test_int_supplyAndBorrow_byAssets((uint256,uint256,uint256,uint256,uint24)) (gas: 1 (0.001%))
test_int_royco_deployWithOwner(address) (gas: -41 (-0.001%))
test_int_royco_deployWithNoOwner() (gas: -41 (-0.001%))
test_int_supplyCollateral_tokenIsIncorrect(uint256,address) (gas: -41 (-0.001%))
test_int_supplyAndBorrow_insufficientLiquidity((uint256,uint256,uint256,uint256,uint24)) (gas: 1 (0.001%))
test_int_flashLoan_success(uint256) (gas: 1 (0.002%))
test_int_proxy_withdrawWithPermit(uint256) (gas: -1 (-0.002%))
test_int_proxy_withdrawWithApprove(uint256) (gas: -1 (-0.002%))
test_int_supplyAndBorrow_unhealthyPosition((uint256,uint256,uint256,uint256,uint24)) (gas: 2 (0.003%))
test_int_repayAndWithdraw_byShares((uint256,uint256,uint256,uint256,uint24),uint256) (gas: 1 (0.003%))
test_int_repayAndWithdraw_byAssets((uint256,uint256,uint256,uint256,uint24),uint256) (gas: 1 (0.003%))
test_int_callback_supplyCollateral(uint256) (gas: -3 (-0.003%))
test_int_liquidate_noReserveShares((uint256,uint256,uint256,uint256,uint24)) (gas: -1 (-0.003%))
test_int_callback_liquidate((uint256,uint256,uint256,uint256,uint24)) (gas: -3 (-0.004%))
test_int_flashLoan_shouldRevertIfNotReimbursed(uint256) (gas: 1 (0.004%))
test_int_repay_maxOnBehalf(uint256) (gas: 1 (0.004%))
testRefundInterval(uint32,uint32,uint256) (gas: -13 (-0.005%))
test_int_callback_repay((uint256,uint256,uint256,uint256,uint24)) (gas: -2 (-0.006%))
test_int_liquidate_withReserveShares((uint256,uint256,uint256,uint256,uint24)) (gas: -2 (-0.006%))
test_int_proxy_withdrawWithTimelapByAssets((uint256,uint256,uint256,uint256,uint24)) (gas: 3 (0.008%))
test_int_setup_createDahlia_revert() (gas: -15 (-0.012%))
testRewardsAccrualWithMultipleUsers(uint256[],uint32) (gas: -150 (-0.013%))
test_int_withdraw_insufficientLiquidity((uint256,uint256,uint256,uint256,uint24)) (gas: -2 (-0.014%))
test_int_setup_createDahlia_withSalt() (gas: -4053 (-0.105%))
test_int_manage_deployMarketWhenLltvNotAllowed((address,address,address,address,uint256,uint256,string,address)) (gas: -41 (-0.121%))
test_int_manage_deployMarketWhenIrmNotAllowed((address,address,address,address,uint256,uint256,string,address)) (gas: -41 (-0.151%))
test_Permitted_withSigSuccess((address,address,bool,uint256,uint256),uint256) (gas: -130 (-0.185%))
test_Permitted_withSigWrongNonce((address,address,bool,uint256,uint256),uint256) (gas: -130 (-0.292%))
test_Permitted_withReusedSig((address,address,bool,uint256,uint256),uint256) (gas: -340 (-0.458%))
test_Permitted_withSigWrongPK((address,address,bool,uint256,uint256),uint256) (gas: -130 (-0.622%))
test_Permitted_withSignatureDeadlineOutdated((address,address,bool,uint256,uint256),uint256,uint256) (gas: -198 (-0.885%))
Overall gas change: -5372 (-0.009%)
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.4.2 Pause and unpausing a market do extra unnecessary `market.status` checks

Severity: Gas Optimization

Context: Dahlia.sol#L448, Dahlia.sol#L458

Description: In both `pauseMarket` and `unpauseMarket` the following check has been performed:

```
_validateMarketDeployed(market.status); // require(status != MarketStatus.None, ...);
```

Although the above check is guaranteed by the next lines:

```
require(market.status == MarketStatus.Active, Errors.CannotChangeMarketStatus());
```

or:

```
require(market.status == MarketStatus.Paused, Errors.CannotChangeMarketStatus());
```

Recommendation: `_validateMarketDeployed(market.status)` can be removed unless in the future an extra logic is considered to be added to this function:

```
diff --git a/src/core/contracts/Dahlia.sol b/src/core/contracts/Dahlia.sol
index 3dc8699..4db9d26 100644
--- a/src/core/contracts/Dahlia.sol
+++ b/src/core/contracts/Dahlia.sol
@@ -445,7 +445,6 @@ contract Dahlia is Permitted, Ownable2Step, IDahlia, ReentrancyGuard {
     function pauseMarket(MarketId id) external {
         Market storage market = markets[id].market;
         _checkDahliaOwnerOrVaultOwner(market.vault);
-        _validateMarketDeployed(market.status);
         require(market.status == MarketStatus.Active, Errors.CannotChangeMarketStatus());
         emit MarketStatusChanged(market.status, MarketStatus.Paused);
         market.status = MarketStatus.Paused;
@@ -455,7 +454,6 @@ contract Dahlia is Permitted, Ownable2Step, IDahlia, ReentrancyGuard {
     function unpauseMarket(MarketId id) external {
         Market storage market = markets[id].market;
         _checkDahliaOwnerOrVaultOwner(market.vault);
-        _validateMarketDeployed(market.status);
         require(market.status == MarketStatus.Paused, Errors.CannotChangeMarketStatus());
         emit MarketStatusChanged(market.status, MarketStatus.Active);
         market.status = MarketStatus.Active;
```

`pnpm run diff`:

```
test_int_proxy_withdrawByShares(uint256) (gas: -1 (-0.001%))
test_int_proxy_withdrawByAssets(uint256) (gas: -1 (-0.001%))
test_int_proxy_depositByShares(uint256) (gas: -1 (-0.001%))
test_int_proxy_depositByAssets(uint256) (gas: -1 (-0.001%))
test_int_lend_byAssets(uint256) (gas: -1 (-0.001%))
test_int_flashActions((uint256,uint256,uint256,uint256,uint24)) (gas: -4 (-0.002%))
test_int_proxy_withdrawWithPermit(uint256) (gas: -1 (-0.002%))
test_int_proxy_withdrawWithApprove(uint256) (gas: -1 (-0.002%))
test_int_supplyAndBorrow_unhealthyPosition((uint256,uint256,uint256,uint256,uint24)) (gas: 2 (0.003%))
test_int_proxy_withdrawWithTimelapByAssets((uint256,uint256,uint256,uint256,uint24)) (gas: 1 (0.003%))
test_int_liquidate_noReserveShares((uint256,uint256,uint256,uint256,uint24)) (gas: -1 (-0.003%))
test_int_flashLoan_success(uint256) (gas: 2 (0.003%))
test_int_supplyCollateral_success(uint256) (gas: 2 (0.004%))
test_int_supplyAndBorrow_byAssets((uint256,uint256,uint256,uint256,uint24)) (gas: 5 (0.005%))
test_int_supplyAndBorrow_insufficientLiquidity((uint256,uint256,uint256,uint256,uint24)) (gas: 4 (0.005%))
test_int_liquidate_withReserveShares((uint256,uint256,uint256,uint256,uint24)) (gas: -2 (-0.006%))
testRewardsAccrualWithMultipleUsers(uint256[],uint32) (gas: -89 (-0.008%))
test_int_flashLoan_shouldRevertIfNotReimbursed(uint256) (gas: 2 (0.008%))
test_int_repayAndWithdraw_byShares((uint256,uint256,uint256,uint256,uint24),uint256) (gas: 3 (0.008%))
test_int_repayAndWithdraw_byAssets((uint256,uint256,uint256,uint256,uint24),uint256) (gas: 3 (0.008%))
test_int_setup_createDahlia_revert() (gas: -15 (-0.012%))
testExtendRewardsInterval(uint256,uint256,uint256,uint256,uint256) (gas: 43 (0.012%))
test_int_repay_maxOnBehalf(uint256) (gas: 3 (0.012%))
test_int_withdraw_insufficientLiquidity((uint256,uint256,uint256,uint256,uint24)) (gas: -2 (-0.014%))
test_int_setup_createDahlia_withSalt() (gas: -3653 (-0.094%))
test_int_marketStatus_deprecate() (gas: -68 (-0.123%))
test_int_marketStatus_unpause() (gas: -408 (-0.292%))
test_int_marketStatus_pause() (gas: -408 (-0.292%))
Overall gas change: -4587 (-0.008%)
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.4.3 Remove unnecessary check when adding a new rewards token

Severity: Gas Optimization

Context: WrappedVault.sol#L174

Description: In Royco, the check was as follows:

```
if (rewardsToken == address(VAULT)) revert InvalidReward();
```

This ensures that the `rewardsToken` is not the vault token. This check makes sense because the Vault itself is an ERC20 token. However, in this case, we know that `dahlia` is not an ERC20 token. Therefore, removing the check makes more sense than replacing `VAULT` with `dahlia`.

Recommendation: Remove the check as suggested.

Dahlia: Fixed in commit 0a87386a.

Cantina Managed: Fixed.

3.5 Informational

3.5.1 Outdated gas snapshot, unused git submodules, unused remappings

Severity: Informational

Context: *(No context files were provided by the reviewer)*

1. `.gas-snapshot` is out dated run `pnpm run snapshot` to update.
2. The following git submodules are not used:

```
lib/v3-core
```

3. The `remappings.txt` contains unused aliases:

```
@uniswap/v3-core/=lib/v3-core/
@prb/math/=lib/prb-math/
```

Dahlia:

1. `.gas-snapshot` designed for manual run of `pnpm run snapshot`, make a change and run `pnpm run diff` to understand the gas difference. We initially had this part of pre-commit, but for some reason, it's not always providing the same `.gas-snapshot` on different platforms.
2. used as a transitive dependency, you can try to remove it to see the error.
3. `@prb/math` agreed not needed, it has been removed in commit bb8ce8fa.

Cantina Managed: `@prb/math` has been removed in commit bb8ce8fa.

3.5.2 Typos, Comments, Unused Code,

Severity: Informational

Context: Dahlia.sol#L489-L495, IDahlia.sol#L39, IDahlia.sol#L189, VariableIrm.sol#L84, WrappedVault.sol#L68-L69, WrappedVault.sol#L147, WrappedVault.sol#L162-L164, WrappedVault.sol#L414

Description/Recommendation:

- IDahlia.sol#L39: `fullUtilizationRate` has the type `uint64` which occupies `8` bytes although the comment mentions `3` bytes. Note that this also changes have the separation into storage slots are commented in this `struct` using `// --- ....`
- IDahlia.sol#L189: This parameter is called `prevBorrowRate` although in the implementation of `executeMarketAccrueInterest` the `newRatePerSec` is supplied.
- ManageMarketImpl.sol#L15: `FixedPointMathLib` is not used for `uint256` in `ManageMarketImpl`.
- WrappedVault.sol#L68: `intervaled` → `interval`.
- WrappedVault.sol#L69: `THe` → `The`.
- WrappedVault.sol#L162-L164: `minDuration` → `minEnd`.
- WrappedVault.sol#L147:

    // Burn 10,000 wei to stop 'first share' front running attacks on depositors

    - This comment is not correct. `Dahlia.lend` doesn't care about the `totalSupply` of `WrappedVault`. It keeps track of `totalLendShares` and `totalLendAssets` in its own accounting and converts shares into assets based off of those.
    - But the code itself is essential, since it would guarantee that `totalSupply` would never be 0 which is important when one does:

    ```
    rewardsPerTokenOut.accumulated = (rewardsPerTokenIn.accumulated +
    ↪  (elapsedScaled.mulDivDown(rewardsInterval_.rate, totalSupply)));
    ```

- VariableIrm.sol#L84, Timelock.sol#L117,WrappedVault.sol#L219, WrappedVault.sol#L230, WrappedVaultFactory.sol#L124: Prefix internal functions with an underscore, for example `getFullUtilizationInterest` → `_getFullUtilizationInterest`.
- WrappedVault.sol#L414: Unused return value.
- Dahlia.sol#L115: Unnecessary cast from `uint24` to itself.

Dahlia: Fixed in commit b4f01b7a.

Cantina Managed: Fixed.

3.5.3 Events related findings

Severity: Informational

Context: DahliaRegistry.sol#L17, DahliaRegistry.sol#L33, DahliaRegistry.sol#L56, Dahlia.sol#L49-L52, InterestImpl.sol#L64, ManageMarketImpl.sol#L4, ManageMarketImpl.sol#L15, IDahliaRegistry.sol#L12, IDahliaRegistry.sol#L18, IDahlia.sol#L189, ChainlinkOracleWithMaxDelayBase.sol#L40-L41, UniswapOracleV3SingleTwapBase.sol#L43-L49, DahliaOracleFactory.sol#L21-L24

Description/Recommendation:

- DahliaRegistry.sol#L33: The event `SetValue` is emitted first then the storage is updated. This doesn't follow the pattern used in other places like in `setAddress`.
- DahliaRegistry.sol#L56:
  1. Unlike the other events `AllowIrm` does not expose the `setter`.
  2. `DahliaRegistry` inherits from `IDahliaRegistry` and so here one could have just called `emit AllowIrm(irm)` like the other event emissions.
- DahliaRegistry.sol#L17: In the `DahliaRegistry`'s constructor, the value for `VALUE_ID_ROYCO_WRAPPED_VAULT_MIN_INITIAL_FRONTEND_FEE` has been set but the event `SetValue` is not emitted.
- Dahlia.sol#L49-L52: Event emission is missing for: `dahliaRegistry`, `protocolFeeRecipient`, `lltvRange`, `liquidationBonusRateRange`.
- InterestImpl.sol#L64: Storage update after emitting `DahliaAccrueInterest`.
- ChainlinkOracleWithMaxDelayBase.sol#L36: Events missing when `params` and `_maxDelays` has been set. For `_maxDelays` there is already a custom event that can be used `SetMaximumOracleDelay`.
- UniswapOracleV3SingleTwapBase.sol#L43: Events missing.
- DahliaOracleFactory.sol#L21: Events missing.
- IDahliaRegistry.sol#L18, IDahliaRegistry.sol#L24: In `SetAddress` and `SetValue` setter is always the current owner. There is already an event when ownership transfer happens and that can be used to know the current owner. There is no need to emit the setter in these events.
- WrappedVault.sol#L35, WrappedVault.sol#L213: `FeesClaimed` event does not emit `owed`.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.4 A deprecated market can be deprecated again

Severity: Informational

Context: Dahlia.sol#L465-L470

Description: In `deprecateMarket` we have:

```
function deprecateMarket(MarketId id) external onlyOwner {
    Market storage market = markets[id].market;
    _validateMarketDeployed(market.status);
    emit MarketStatusChanged(market.status, MarketStatus.Deprecated);
    market.status = MarketStatus.Deprecated;
}
```

Note that as long as the `market.status` is not `MarketStatus.None` the `owner` can call this function again and again and thus enforces multiple `MarketStatusChanged(market.status, MarketStatus.Deprecated)` events to be emitted for the same market (might be confusing for off-chain tooling).

Recommendation: Although not necessary, one can impose a stricter check to make sure this function can only be called when the market is either `Active` or `Paused`.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.5 newRatePerSec and newFullUtilizationRate might need to be updated even if interestEarnedAssets is 0

Severity: Informational

Context: InterestImpl.sol#L36-L41

Description: A custom IIrm implementation might return values for newRatePerSec and newFullUtilizationRate which might need to be updated even if interestEarnedAssets returned is 0.

Recommendation: It might be best to take updating those values out of the if block:

```
(uint256 interestEarnedAssets, uint256 newRatePerSec, uint256 newFullUtilizationRate) =
    IIrm(market.irm).calculateInterest(deltaTime, totalLendAssets, totalBorrowAssets,
    ↪  market.fullUtilizationRate);

market.fullUtilizationRate = uint64(newFullUtilizationRate);
market.ratePerSec = uint64(newRatePerSec);

if (interestEarnedAssets > 0) {
    // ...
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.6 market.updatedAt is only updated when interestEarnedAssets is non-zero in executeMarketAccrueInterest

Severity: Informational

Context: InterestImpl.sol#L65

Description: market.updatedAt is only updated when interestEarnedAssets is non-zero in executeMarketAccrueInterest

Cantina Managed: I think the current implementation might make more sense if market.updatedAt is updated even if interestEarnedAssets is 0, a malicious user might call accrueMarketInterest within small interval and thus force deltaTime to be very small which might make interestEarnedAssets == 0 and thus avoid letting interest accrue.

So maybe the current implementation is good. Overall it might make sense to create a test case to demonstrate the above attack type.

Dahlia: market.updatedAt should update only if there is any interest to avoid missing interest if we try to call every single block, for example, we have a tiny position in assets and should still accrue some interest in a big enough deltaTime.

I wrote a test to verify updatedAt logic and I had to add the same logic to skip modification of any fields if no interest accrued in getLastMarketState: commit e10e43f6.

I have used the PR 15 for fix of getLastMarketState does not update the market properly.

Cantina Managed: I think we come to an agreement that no fields should be updated if interestEarnedAssets is 0, thus closing this issue.

3.5.7 Unreached code

Severity: Informational

Context: WrappedVault.sol#L270-L272

Description:

- WrappedVault.sol#L270-L272: If rate < rewardsInterval.rate is true, then the following would already revert due to arithmetic underflow:

  ```
  rewardsAdded = (rate - rewardsInterval.rate) * (newEnd - newStart) + frontendFeeTaken +
  ↪  protocolFeeTaken;
  ```

  and thus revert RateCannotDecrease() cannot be reached.

Recommendation: Swap these lines:

```
uint256 rate = (rewardsAdded - frontendFeeTaken - protocolFeeTaken + remainingRewards) / (newEnd - newStart);
if (rate < rewardsInterval.rate) revert RateCannotDecrease();

rewardsAdded = (rate - rewardsInterval.rate) * (newEnd - newStart) + frontendFeeTaken + protocolFeeTaken;
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Change is not applied since the formula for `rewardsAdded` has been changed in commit bb8ce8fa.

3.5.8 Check for other `VariableIrm.Config` parameters missing in `IrmFactory.createVariableIrm`

Severity: Informational

Context: IrmFactory.sol#L20-L21

Description: Check for other `VariableIrm.Config` parameters missing in `IrmFactory.createVariableIrm` such as:

- `minFullUtilizationRate`.
- `maxFullUtilizationRate`.

Recommendation: Make sure `minFullUtilizationRate <= maxFullUtilizationRate`.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.9 When creating a Uniswap oracle make sure the token pairs are supported

Severity: Informational

Context: UniswapOracleV3SingleTwapBase.sol#L46-L48

Description: In `UniswapOracleV3SingleTwapBase.constructor` one does not check wether the provided tokens are supported in the static oracle.

Recommendation: Add the following check:

```
bool pairSupported = IStaticOracle(UNISWAP_STATIC_ORACLE_ADDRESS).isPairSupported(
   UNISWAP_V3_TWAP_BASE_TOKEN,
   UNISWAP_V3_TWAP_QUOTE_TOKEN
);

if (!pairSupported) {
 revert PairNotSupported(UNISWAP_V3_TWAP_BASE_TOKEN, UNISWAP_V3_TWAP_QUOTE_TOKEN);
}
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.10 Enforce minimum TWAP duration

Severity: Informational

Context: UniswapOracleV3SingleTwapBase.sol#L54

Description: When creating a Uniswap V3 TWAP, there is no enforcement of a minimum TWAP duration. Setting the TWAP duration too low undermines its purpose and can lead to price manipulation.

Recommendation: Consider implementing a reasonable minimum duration.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.11 Add proper check when setting reserve fees

Severity: Informational

Context: ManageMarketImpl.sol#L25-L31

Description: If newFee is being set to a non zero value, add a check to make sure that reserveFeeRecipient is non zero as well. Otherwise, reserve fees will be sent to the zero address (InterestImpl.sol#L60).

reserveFeeRecipient is not set in the constructor, which means there is a risk of an error if setReserveFeeRate is called before reserveFeeRecipient is assigned a non-zero value.

Recommendation: Add the check as suggested.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.12 Inconsistent condition checks when calling callbacks

Severity: Informational

Context: Dahlia.sol#L336

Description: In this context onDahliaSupplyCollateral is only called when callbackData.length is non-zero where as for the other callbacks the following check is performed:

```
if (callbackData.length > 0 && address(msg.sender).code.length > 0) { ... }
```

Recommendation: Make sure the conditions checked are consistent across the callback functionality or add an explanation was why the extra check address(msg.sender).code.length > 0 in this context is missing.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.13 Mixed use of returned parameters in WrappedVault.redeem

Severity: Informational

Context: WrappedVault.sol#L584-L587

Description: In this context, there is a mixed use of assets returned by previewRedeem and _assets returned by _withdraw.

```
function redeem(uint256 shares, address receiver, address owner) external returns (uint256 _assets) {
   uint256 assets = previewRedeem(shares);
   (_assets) = _withdraw(msg.sender, shares, receiver, owner);

   emit Withdraw(msg.sender, receiver, owner, assets, shares);
}
```

Recommendation: The line defining assets can be removed:

```
function redeem(uint256 shares, address receiver, address owner) external returns (uint256 assets) {
   (assets) = _withdraw(msg.sender, shares, receiver, owner);

   emit Withdraw(msg.sender, receiver, owner, assets, shares);
}
```

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.14 Allowed IRM cannot be disallowed

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: An IRM contract cannot be removed from the allowlist. Disallowing an allowed IRM ensures that future deployed markets cannot use that IRM anymore.

Recommendation: Add a method to disallow IRMs so that new markets cannot be created using those IRMs. Markets with IRMs that are no longer allowed due to a bug can be deprecated. If a market is not deprecated and the IRM is not allowed, users will understand that the IRM is discouraged but does not have a bug.

Dahlia: Fixed in commit bb8ce8fa.

Cantina Managed: Fixed.

3.5.15 `ChainlinkOracleWithMaxDelayBase` price precision analysis

Severity: Informational

Context: ChainlinkOracleWithMaxDelayBase.sol#L46-L51

Description: `ORACLE_PRECISION` constant is deifned as:

```
ORACLE_PRECISION = 10
    ** (
        36 + quoteTokenDecimals + params.quoteFeedPrimary.getDecimals() +
        ↪  params.quoteFeedSecondary.getDecimals() - baseTokenDecimals
            - params.baseFeedPrimary.getDecimals() - params.baseFeedSecondary.getDecimals()
    );
```

or in other words:

$$k = 10^{36+(d_q+d_{q,1}+d_{q,2})-(d_b+d_{b,1}+d_{b,2})}$$

and the final price is given by:

```
price = ORACLE_PRECISION.mulDiv(_basePrimaryPrice * _baseSecondaryPrice, _quotePrimaryPrice *
↪  _quoteSecondaryPrice);
```

$$p = 10^{36}\left[\frac{\frac{p_{b,1}}{10^{d_{b,1}}}\cdot\frac{p_{b,2}}{10^{d_{b,2}}}}{\frac{p_{q,1}}{10^{d_{q,1}}}\cdot\frac{p_{q,2}}{10^{d_{q,2}}}}\right]\cdot\frac{10^{d_q}}{10^{d_b}}$$

Note the $\frac{10^{d_q}}{10^{d_b}}$ factor in p which should be used for precision normalization.

parameter	description
p	price
k	ORACLE_PRECISION
d_q	quoteTokenDecimals
$d_{q,1}$	params.quoteFeedPrimary.getDecimals()
$d_{q,2}$	params.quoteFeedSecondary.getDecimals()
$p_{q,1}$	_quotePrimaryPrice
$p_{q,2}$	_quoteSecondaryPrice
d_b	baseTokenDecimals
$d_{b,1}$	params.baseFeedPrimary.getDecimals()
$d_{b,2}$	params.baseFeedSecondary.getDecimals()
$p_{b,1}$	_basePrimaryPrice
$p_{b,2}$	_baseSecondaryPrice
B	the base token unit/amount (non-normalized)
B_1	the base intermediary token unit/amount (non-normalized)
X	the common token unit/amount (non-normalized)
Q	the quote token unit/amount (non-normalized)
Q_1	the quote intermediary token unit/amount (non-normalized)
d_Z	decimals for token Z

we have:

$$p_{b,1} = \frac{B_1/10^{d_{B_1}}}{B/10^{d_b}} \cdot 10^{d_{b,1}}$$

$$p_{b,2} = \frac{X/10^{d_X}}{B_1/10^{d_{B_1}}} \cdot 10^{d_{b,2}}$$

$$p_{q,1} = \frac{Q_1/10^{d_{Q_1}}}{Q/10^{d_q}} \cdot 10^{d_{q,1}}$$

$$p_{q,2} = \frac{X/10^{d_X}}{Q_1/10^{d_{Q_1}}} \cdot 10^{d_{q,2}}$$

and so:

$$\frac{p_{b,1}}{10^{d_{b,1}}} \cdot \frac{p_{b,2}}{10^{d_{b,2}}} = \frac{X/10^{d_X}}{B/10^{d_b}}$$

$$\frac{p_{q,1}}{10^{d_{q,1}}} \cdot \frac{p_{q,2}}{10^{d_{q,2}}} = \frac{X/10^{d_X}}{Q/10^{d_q}}$$

and finally:

$$p = 10^{36} \cdot \frac{Q}{B}$$

which has the correct units when used in `collateralToLendUp` given B is the collateral/base token and Q is the loan/quote token.

Recommendation: No changes needed.