Hyperdrive February 2024 Security Review

Auditors

Christoph Michel, Lead Security Researcher

Saw-Mon and Natalie, Lead Security Researcher

M4rio.eth, Security Researcher

Deivitto, Junior Security Researcher

Report prepared by: Lucas Goiriz

March 11, 2024

Contents

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 hyperdrive 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 15 days in total, the DELV team engaged with Spearbit to review the hyperdrive protocol. In this period of time a total of **72** issues were found.

Summary

Project Name	Hyperdrive
Repository	hyperdrive
Commit	d363f4...f273ea
Type of Project	DeFi, Yield
Audit Timeline	Jan 29 to Feb 16 2024
Two week fix period	Feb 16 - Feb 29 2024

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	7	6	1
Low Risk	19	12	7
Gas Optimizations	8	8	0
Informational	36	21	15
Total	**72**	**49**	**23**

5 Findings

5.1 High Risk

5.1.1 Withdrawal shares of `msg.sender` are burnt incorrectly in `_removeLiquidity` flow

Severity: High Risk

Context: HyperdriveLP.sol#L279-L296, HyperdriveLP.sol#L397-L402

Description: When `_removeLiquidity` is called, the `_lpShares` of `AssetId._WITHDRAWAL_SHARE_ASSET_ID` is minted for `_options.destination`.

```
_mint(
     AssetId._WITHDRAWAL_SHARE_ASSET_ID,
     _options.destination, // <---
     _lpShares
);
```

But after distributing the excess idle in `_redeemWithdrawalSharesInternal` we burn the clamped amount of withdrawal shares for the `msg.sender` and not the `_options.destination`:

```
_burn(
    AssetId._WITHDRAWAL_SHARE_ASSET_ID,
    msg.sender, // <---
    withdrawalSharesRedeemed
);
```

Note that burning for the `msg.sender` in `_redeemWithdrawalShares` makes sense.

Recommendation: Make sure in the flow `_removeLiquidity ->_redeemWithdrawalSharesInternal` we would burn the withdrawal shares for `_options.destination` instead of `msg.sender`.

DELV: Fixed in PR 763.

Spearbit: Verified.

5.1.2 Closing netted, non-matured shorts can lead to insolvency

Severity: High Risk

Context: HyperdriveShort.sol#L190-L196

Description: Closing netted shorts can increase the long exposure in case the now non-netted longs are not also closed. This happens when calling `closeShort` before maturity. As the long exposure increases, a solvency check is required so that all outstanding longs can be fulfilled at maturity. This is not done in `closeShort` and the protocol can end up insolvent.

Recommendation: Consider checking if the protocol is solvent after updating the long exposure in `_closeShort` for non-matured shorts. (Closing shorts at maturity via `applyCheckpoint` should not require a solvency check as any long exposure created from closing netted shorts is offset by closing the same amount of netted matured longs).

Proof of concept:

```
function test_close_short_needs_to_check_solvency() external {
    uint256 apr = 0.1e18;
    uint256 ttm = uint256(0.5e18);
    vm.assume(apr >= 0.01e18 && apr <= 0.5e18);
    vm.assume(ttm <= ONE && ttm >= 0);

    // Deploy the pool and initialize the market
    {
        uint256 timeStretchApr = 0.02e18;
```

```
        deploy(alice, timeStretchApr, 0, 0, 0, 0);
    }
    uint256 contribution = 500e18;
    uint256 lpShares = initialize(alice, apr, contribution);
    contribution -= 2 * hyperdrive.getPoolConfig().minimumShareReserves;

    // open netted longs and shorts
    uint256 longBase = hyperdrive.calculateMaxLong();
    console2.log("longBase", hyperdrive.calculateMaxLong());
    (uint256 longMaturityTime, uint256 nettedBondAmount) = openLong(celine, longBase);
    console2.log("opened first long dy=", nettedBondAmount);
    (, uint256 baseDeposit) = openShort(celine, nettedBondAmount);
    console2.log("opened first short with same bond amount. deposit=", baseDeposit);
    assertEq(
        hyperdrive.balanceOf(AssetId.encodeAssetId(AssetId.AssetIdPrefix.Long, longMaturityTime),
↪   celine),
            hyperdrive.balanceOf(AssetId.encodeAssetId(AssetId.AssetIdPrefix.Short, longMaturityTime),
↪   celine),
        "!netted"
    );

    // open a long exposure
    longBase = hyperdrive.calculateMaxLong();
    (, uint256 netLongBonds) = openLong(celine, longBase);
    console2.log("opened second long dy=", netLongBonds);

    // time advances until we're at ttm
    uint256 timeAdvanced = POSITION_DURATION.mulDown(ONE - ttm);
    advanceTime(timeAdvanced, int256(0));

    // Celine closes her short.
    // this call makes it insolvent but passes. we gain a lot of long exposure by closing the netted
↪   short
    uint256 baseProceeds = closeShort(celine, longMaturityTime, nettedBondAmount);
    console2.log("closed shorts", baseProceeds);

    // Celine closes her longs at maturity
    console2.log("trying to close longs ...");
    timeAdvanced = POSITION_DURATION.mulDown(ttm);
    advanceTime(timeAdvanced, int256(0));
    hyperdrive.checkpoint(longMaturityTime);
    // closeLong(celine, longMaturityTime, nettedBondAmount + netLongBonds);
}
```

DELV: Fixed in PR 761.

Spearbit: Verified.

5.2 Medium Risk

5.2.1 Invariants are not checked after calling `calculateDistributeExcessIdleShareProceedsNetLongEdgeCaseSafe`

Severity: Medium Risk

Context: LPMath.sol#L653-L666

Description: `calculateDistributeExcessIdleShareProceedsNetLongEdgeCaseSafe` returns:

$$z_{proceed} = z_{og} - \frac{z_{og}}{\zeta_{og}}\left(\frac{s_{LP}}{s_{LP} + s_W}\right)PV_0 - z_{flat}$$

or in other words:

$$z_{og} \longrightarrow \frac{1}{\zeta_{og}}\left(\frac{s_{LP}}{s_{LP} + s_W}\right)PV_0 - z_{flat}$$

$\cdot z_{og}$

1. This scaling is independent of the number of iterations performed in the Newton approximation loop. This is not really an issue just an observation that one the calculation ones picks this specific value only depending on the original point.
2. After one finds z we still need to check that `uint256(_params.netCurveTrade) <= maxBondAmount` is `false` on the updated curve $C(z, \zeta, y)$, otherwise the assumption where $z + z_{curve} - z_{min} = \left(\frac{\zeta_{og}}{z_{og}}\right)z$ would be wrong which would break:

$$\frac{PV_0}{s_{LP} + s_w} = \frac{PV_1}{s_{LP}}$$

3. $z_{proceed}$ is not compared to the idle share reserves to make sure it cannot be greater than that value, we need to check:

$$z_{proceed} \leq I$$

Recommendation: 2. After updating the point to the new point (z, ζ, y) by the scaling factor:

$$\lambda = \frac{1}{\zeta_{og}}\left(\frac{s_{LP}}{s_{LP} + s_W}\right)PV_0 - z_{flat}$$

Make sure that `uint256(_params.netCurveTrade) <= maxBondAmount` is `false`.

3. Check that $z_{proceed} \leq I$.

DELV:

2. has been applied in PR 800, which had an error that was fixed in PR 827.
3. The above PRs along with PR 771 below will guarantee 3.

Spearbit: Verified.

5.2.2 In some cases `maxShareReservesDelta` can be greater than the idle share reserves in `calculateDistributeExcessIdle`

Severity: Medium Risk

Context: LPMath.sol#L436, LPMath.sol#L964-L968, LPMath.sol#L1145

Description: In `calculateDistributeExcessIdle`, if the pool is net short and if we end up calculating `calculateMaxShareReservesDeltaInitialGuess`:

$$\Delta z_{guess} = \frac{z_{og}}{y_{og}} \max(0, y_{og} - (\Delta y_{net} + y_{optimal}))$$

This initial guess value for the start of the newton method might be bigger than the idle share reserves:

$$I < \Delta z_{guess}$$

and if in `calculateMaxShareReservesDelta` we only go through the main `loop` once and break or return early:

```
// Proceed with Newton's method for a few iterations.
for (uint256 i = 0; i < MAX_SHARE_RESERVES_DELTA_MAX_ITERATIONS; ) {
    //... // break or return

    if (maybeMaxShareReservesDelta > _params.idle) {
        maybeMaxShareReservesDelta = _params.idle;
    }
}
```

It is not guaranteed that the final returned value is claimed by idle share reserves

Recommendation: Make sure the initial guess for the `maybeMaxShareReservesDelta` is also clamped by the idle share reserves.

DELV: Fixed in PR 771.

Spearbit: Verified.

5.2.3 After trading on the curve in `closeShort` and applying the fees, spot price is not check to make sure it is still below `1`

Severity: Medium Risk

Context: HyperdriveShort.sol#L519-L538, HyperdriveShort.sol#L570

Description: In `_calculateCloseShort`, one checks the following inequality to make sure the non-matured portion of Δy which is $t_r \Delta y$ when given back to the pool does not bear negative interest:

$$\frac{\mu(z - \zeta + \Delta z_{curve})}{y - t_r \Delta y}$$

$^{t_s} \leq 1 - \phi_c(1 - p_{spot}) \leq 1$

The far right inequality should be right since we need to have checked that before trading on the curve we were in the region with $p_{spot} \leq 1$. Thus the negative interest inequality would guarantee that the point $(z - \zeta + \Delta z, y - t_r \Delta y)$ would also have a spot price of less than 1.

But after trading on the curve we add the LP/curve fees minus the governance fee to the share so:

$$(z - \zeta, y) \to (z - \zeta + \Delta z, y - t_r \Delta y) \to (z - \zeta + \Delta z + (1 - \phi_g) f_c, y - t_r \Delta y)$$

since the share reserves have increased when adding the pool fees, one does not necessarily know that the spot price at this point is still less than 1:

$$\frac{\mu(z - \zeta + \Delta z_{curve} + (1 - \phi_g)f_c)}{y - t_r \Delta y}$$

t_s

Recommendation: In `_calculateCloseShort` when adding the curve fees we still need to make sure the point we end up with has a spot price of less than 1:

$$\frac{\mu(z - \zeta + \Delta z_{curve} + (1 - \phi_g)f_c)}{y - t_r \Delta y}$$

$t_s \leq 1$

This should be equivalent to if not considering the error introduced in `pow` to the following:

$$\frac{\mu(z - \zeta + \Delta z_{curve} + (1 - \phi_g)f_c)}{y - t_r \Delta y} \leq 1$$

or even better to avoid division errors and costs:

$$\mu(z - \zeta + \Delta z_{curve} + (1 - \phi_g)f_c) \leq y - t_r \Delta y$$

DELV: Fixed in PR 770.

Spearbit: Verified.

5.2.4 Large vault share price updates can be captured by open/close short

Severity: Medium Risk

Context: HyperdriveShort.sol#L597, HyperdriveShort.sol#L447-L459

Description: When opening a short, the user deposits `c / c0 * dy - dz * c`, when closing in the same transaction, they receive base proceeds are also `c' / c0 * dy - dz * c'`, where `dz * c'  dz * c` as this represents the bond sell and buyback part of the short. This can lead to a profit when a vault share price transaction increasing `c` is sandwiched. The trader makes a profit and the loss must therefore come from the LPs in equal terms.

Recommendation: As fees and gas fees eat into the profits, this only becomes a problem for vaults that don't linearly increase the vault share price, but instead rely on fewer, larger share price update events. There's currently no simple mitigation as Hyperdrive's design always directly uses the underlying vault price instead of smoothing it out.

Proof of concept:

```
function test_sandwich_vault_share_price_update() external {
    uint256 apr = 0.1e18;

    // Deploy the pool and initialize the market
    {
        uint256 timeStretchApr = 0.02e18;
        deploy(alice, timeStretchApr, 0, 0, 0, 0);
    }
    uint256 contribution = 500e18;
    uint256 lpShares = initialize(alice, apr, contribution);
    contribution -= 2 * hyperdrive.getPoolConfig().minimumShareReserves;

    // sandwich vault price update
    // 1. open short
    uint256 shorts = 10e18;
    (uint256 maturityTime, uint256 baseDeposit) = openShort(celine, shorts);
    console2.log("opened shorts, paid ", baseDeposit);

    // 2. simulate vault share price increase
    int256 c1c0Growth = 0.01e18;
    MockHyperdrive(address(hyperdrive)).accrue(365 days, c1c0Growth);

    // 3. close short
    uint256 baseProceeds = closeShort(celine, maturityTime, shorts);
    console2.log("closed shorts, got ", baseProceeds);

    // 4. profit
    int256 profit = int256(baseProceeds) - int256(baseDeposit);
    console2.log("profit: %se16", profit / 1e16);
    assertGe(profit, 0);
}
```

DELV: Acknowledged. This is reasonable, and we were aware of this problem. Our mitigation for this will be to clearly document this in the factory documentation and to avoid using yield sources that would be problematic.

Spearbit: Acknowledged.

5.2.5 Wrong zeta fee adjustments when closing longs

Severity: Medium Risk

Context: HyperdriveLong.sol#L578

Description: The `z` and `zeta` reserves are expected to update to `z'` and `zeta'` as follows (when closing longs):

- `z'` should reduce by the total trade but include LP fees (gov fees are not reinvested): `z' = z - shareProceeds + fees_total - govFees_total`.
- The effective reserves `z' - zeta'` should reduce by the curve trade `shareProceedsCurve` but reinvest the *curve* LP fees (`fee_c - govFee_c`): `z' - zeta' = (z - zeta) - shareProceedsCurve + fee_c - govFee_c`.
- To get there, the share adjustments need to be updated by the flat trade and the flat LP fees `zeta' = zeta - (shareProceeds - shareProceedsCurve) + (fee_f - govFee_f)`.

Therefore, `shareAdjustmentDelta = (shareProceeds - shareProceedsCurve) - (fee_f - govFee_f)` but in the code it is `shareAdjustmentDelta = (shareProceeds - shareProceedsCurve) - (fee_total - govFee_total) + fee_c`.

```
(dz_c, dy_c, shareProceeds) = HDM.calculateCloseLong(...)
dz_c' = dz_c - fee_c
shareProceeds' = shareProceeds - (fee_c + fee_f) = shareProceeds - fee_total
dz = shareProceeds' + govFee_total = shareProceeds - fee_total + govFee_total
    = shareProceeds - (fee_total - govFee_total)
dzeta = dz - dz_c' = dz - (dz_c - fee_c)
    = (shareProceeds - dz_c) - (fee_total - govFee_total) + fee_c
 it should be (shareProceeds - dz_c) - (fee_f - govFee_f)
    where (fee_f - govFee_f) can be obtained by (fee_total - govFee_total) - (fee_c - govFee_c)
```

Recommendation: Consider changing `shareCurveDelta -= curveFee;` to `shareCurveDelta -= (curveFee - governanceCurveFee);` instead:

```
  (
      curveFee, // shares
      flatFee, // shares
+     governanceCurveFee, // shares
      totalGovernanceFee // shares
  ) = _calculateFeesGivenBonds(
      _bondAmount,
      timeRemaining,
      spotPrice,
      _vaultSharePrice
  );

- shareCurveDelta -= curveFee;
+ shareCurveDelta -= (curveFee - governanceCurveFee);
```

DELV: Fixed in PR 767.

Spearbit: Verified.

5.2.6 LPs are paying twice for governance fees when opening longs

Severity: Medium Risk

Context: HyperdriveLong.sol#L435-L470,

Description: The `_calculateOpenLong` function first computes fees on the bond amount that is paid out to the trader. The governance fee measured in bonds is removed from the bond reserves, then converted to a vault shares amount, and again removed from the share reserves. The governance fees are essentially removed twice, once from the bond and once from the share reserves. Meaning, the LPs are paying twice for it.

Recommendation: Consider only removing the governance fees from the share reserves. We can imagine this as governance swapping the fees that were taken as bonds to shares. For this, the governance fees need to be *removed* from the `bondReservesDelta` variable (because `y - dy` will mean that it is *added back* to the bond reserves).

```
- bondReservesDelta = bondProceeds + governanceCurveFee;
+ bondReservesDelta = bondProceeds;
```

DELV: Fixed in PR 766.

Spearbit: Verified. The `_bondProceeds` are now the same as `_bondReservesDelta` which simplifies some of the code (but also makes reading the code a little confusing if you don't have that insight).

5.2.7 `checkpoint` can be re-entered

Severity: Medium Risk

Context: HyperdriveCheckpoint.sol#L32, HyperdriveShort.sol#L98

Description: Most public functions of `Hyperdrive` have reentrancy guards but the `checkpoint` function does not. It can change important contract state like reserves. This can interfere with other functions that don't expect state to change during a `_deposit` call.

For example, the following call `StEthHyperdrive.openShort -> _openShort -> _deposit` performs a refund that happens in the middle of the `openShort` function. It then continues execution with the `_applyShort` function. One could re-enter `checkpoint` during `_deposit` to apply a checkpoint, changing the market state. This allows calculating a swap that would violate solvency requirements but can pass as solvency is only checked after `_deposit`. It is also possible that after the market state is updated in `_applyCheckpoint` there would be some excess idle to be distributed. If so the curve C is scaled down to λC. But when we call `_applyOpenShort` the inputs are from the deltas calculated when we traded on C and not λC which might push our point (z, ζ, y) further to the right (not past the solvency or minimum line but more than the amount it should).

Recommendation: All calls to `_calculateOpenShort` and `_applyOpenShort` should be atomic and the curve C should not change in between. Consider adding reentrancy guards to `checkpoint`.

DELV: Fixed in PR 765.

Spearbit: Verified.

5.3 Low Risk

5.3.1 The `_distributeExcessIdle` is not called when zombie interest is collected

Severity: Low Risk

Context: HyperdriveLong.sol#L199, HyperdriveShort.sol#L201

Description: When a position is closed by a checkpoint at maturity but the user does not close it, it can accumulate interest accordingly with the Zombie Interest algorithm.

The positive interest that is accumulated is added to the share reserves HyperdriveBase.sol#L401. This can create excess idle that should be distributed at the current share price.

Recommendation: Consider calling `_distributeExcessIdle` whenever a user closes a long or a short to distribute the excess idle at the current share price.

DELV: Fixed in PR 783.

Spearbit: Verified.

5.3.2 The derivative in `calculateMaxBuyBondsOutDerivativeSafe` is not calculated correctly

Severity: Low Risk

Context: LPMath.sol#L998-L1004, LPMath.sol#L1521

Description: The derivative in `calculateMaxBuyBondsOutDerivativeSafe` is not calculated correctly. Let:

$$F(z) = dy_{max}(z) = y - \frac{K}{\frac{c}{\mu} + 1}$$

$$\frac{1}{1 - t_s} - y^{net}_{curve}$$

where K is:

$$K = \frac{c}{\mu}(\mu(z-\zeta))^{1-t_s} + y^{1-t_s}$$

So we have:

$$\frac{dF}{dz} = \frac{dy}{dz} - \frac{1}{1-t_s}\,\frac{K}{\frac{c}{\mu}+1}$$

$$\frac{t_s}{1-t_s}\,\frac{\frac{dK}{dz}}{\frac{c}{\mu}+1}$$

$$\frac{dK}{dz} = (1-t_s)\cdot\, c(\mu(z-\zeta))^{-t_s}(1-\frac{d\zeta}{dz}) + y^{-t_s}\frac{dy}{dz}$$

and so we have:

$$\frac{dF}{dz} = \frac{dy}{dz} - \frac{K}{\frac{c}{\mu}+1}$$

$$\frac{t_s}{1-t_s}\,\frac{c(\mu(z-\zeta))^{-t_s}(1-\frac{d\zeta}{dz}) + y^{-t_s}\frac{dy}{dz}}{\frac{c}{\mu}+1}$$

or:

$$\frac{dF}{dz} = 1 - \frac{d\zeta}{dz}$$

$$\frac{dy}{dz-d\zeta} - \frac{K}{\frac{c}{\mu}+1}\,\frac{t_s}{1-t_s}\,\frac{c(\mu(z-\zeta))^{-t_s} + y^{-t_s}\frac{dy}{dz-d\zeta}}{\frac{c}{\mu}+1}$$

and since this derivative calculate using scaling of the variables:

$$(z,\zeta,y) \to \frac{z+dz}{z}(z,\zeta,y)$$

we would have:

$$\frac{dy}{dz} = \frac{y}{z}$$

$$\frac{d\zeta}{dz} = \frac{\zeta}{z}$$

and so

$$\frac{dF}{dz} = 1 - \frac{\zeta}{z}$$

$$\frac{y}{z-\zeta} - \frac{K}{\frac{c}{\mu}+1}\frac{t_s}{1-t_s}\frac{c(\mu(z-\zeta))^{-t_s} + y^{-t_s}\frac{y}{z-\zeta}}{\frac{c}{\mu}+1}$$

But the implemented/calculated derivative in the codebase is:

$$1 - \frac{\zeta_{og}}{z_{og}}$$

$$\frac{y_{og}}{z_{og}-\zeta_{og}} - \frac{K}{\frac{c}{\mu}+1}\frac{t_s}{1-t_s}\frac{c(\mu(z-\zeta))^{-t_s} + y^{-t_s}\frac{y_{og}}{z_{og}-\zeta_{og}}}{\frac{c}{\mu}+1}$$

which is a mix of formulas for derivative of F at point z and z_{og}.

Recommendation: Make sure $\frac{dF}{dz}$ is calculated as:

$$\frac{dF}{dz} = 1 - \frac{\zeta}{z}$$

$$\frac{y}{z-\zeta} - \frac{K}{\frac{c}{\mu}+1}\frac{t_s}{1-t_s}\frac{c(\mu(z-\zeta))^{-t_s} + y^{-t_s}\frac{y}{z-\zeta}}{\frac{c}{\mu}+1}$$

This correction would not be required if the scaling approach from the issue below is used since the Newton and derivatives would be removed, see the issue "Deriving `maxShareReservesDelta` in `calculateDistributeExcessIdle` can be simplified using a scaling trick".

DELV: This function has been removed in PR 771 as is no longer needed.

Spearbit: Verified.

5.3.3 Solvency is not checked during the excess idle distribution when the `netCurveTrade` is `0`

Severity: Low Risk

Context: LPMath.sol#L564-L571

Description: Solvency is not checked during the excess idle distribution when the `netCurveTrade` is `0`.

In `calculateDistributeExcessIdle` if `netCurveTrade` is `0` we would have:

$$\Delta z_{max} = I (\text{or} z_{idle})$$

also in this case we would have:

$$PV_0 = z - \frac{L_o - S_o}{c} - z_{min} = I + \frac{L_{exp} - (L_o - S_o)}{c} = I + \frac{\sum_{t_m \in B} \max(0, s_{S,t_m} - s_{L,t_m})}{c}$$

and after updating the liquidity by removing $\Delta z_{max} = I$ we would have:

$$PV_1 = \frac{\sum_{t_m \in B} \max(0, s_{S,t_m} - s_{L,t_m})}{c}$$

and the `withdrawalSharesRedeemed` is calculated as:

$$\Delta s_w = (s_{LP} + s_w)\frac{I}{PV_0}$$

Now if we end up in the `else` branch of:

```
if (withdrawalSharesRedeemed == 0) { return (0, 0); }
else if (withdrawalSharesRedeemed <= _params.withdrawalSharesTotalSupply) { return
↪  (withdrawalSharesRedeemed, maxShareReservesDelta); }
else { withdrawalSharesRedeemed = _params.withdrawalSharesTotalSupply; }
```

We know that:

$$\Delta s_w = (s_{LP} + s_w)\frac{I}{PV_0} > s_w$$

and so `calculateDistributeExcessIdleShareProceeds` would return the following as `shareProceeds` since `netCurveTrade` is `0`:

$$\Delta z = \frac{s_w}{s_{LP} + s_w}$$

PV_0

and we need to check:

$$\Delta z = \frac{s_w}{s_{LP} + s_w}$$

$PV_0 \leq I$

But the above inequality can be derived from the inequality we got from the `else` branch, unless due to some division errors the inequality's direction would flip.

Recommendation: It might be best to still check for the solvency when due in theory without division errors it should be guaranteed.

DELV: Fixed in PR 788. The fix also is depending on PR 771.

Combining these two PRs we would have post `calculateDistributeExcessIdleShareProceeds` calculation the following guarantee:

$$z_{proceeds} \leq \Delta z_{max} \leq I = z_{idle}$$

Spearbit: Verified.

5.3.4 `calculateUpdateLiquidity` does not check to make sure the effective share reserves `ze` is not below the minimum threshold

Severity: Low Risk

Context: LPMath.sol#L46, HyperdriveLP.sol#L472, HyperdriveMath.sol#L153

Description: In `calculateUpdateLiquidity` one is scaling the point (z, ζ, y) by the provided delta of share reserves:

$$(z, \zeta, y) \longrightarrow \frac{z + \Delta z}{z}(z, \zeta, y)$$

The following check is included in the implantation:

$$z + \Delta z \geq z_{min}$$

But the following check is missing:

$$\frac{z + \Delta z}{z}(z - \zeta) \geq z_{min}$$

Recommendation: Make sure the inequality below is checked in `calculateUpdateLiquidity`:

$$\frac{z + \Delta z}{z}(z - \zeta) \geq z_{min}$$

In general, one can have two utility functions:

```
_updateCurvePointMemory_(CurvePoint memory p, CurvePointDelta memory dp) internal view ...
_updateCurvePointStorage(CurvePoint memory p, CurvePointDelta memory dp) internal ...

struct CurvePoint      { uint256  z; int256  zeta; uint256  y; }
struct CurvePointDelta {  int256 dz; int256 dzeta;  int256 dy; }
```

And the updates for the curve points during the calculation or the final update to the storage should go through these functions so that the invariants are checked atomically.

In each, we would update the memory pointer `p` by `dp` and check the following invariants (for the storage version the storage parameters are also updated):

1. minimum threshold for the share reserves

$$z + \Delta z \geq z_{min}$$

2. minimum threshold for the effective share reserves

$$z + \Delta z - (\zeta + \Delta \zeta) \geq z_{min}$$

3. spot price being not greater than 1

$$\mu(z + \Delta z - (\zeta + \Delta \zeta)) \leq y + \Delta y$$

or

$$\frac{\mu(z + \Delta z - (\zeta + \Delta \zeta))}{y + \Delta y}$$

$t_s \leq 1$

4. Solvency check also we need to make sure the long exposure is updated before checking this:

$$(z + \Delta z) \geq \frac{L_{exp}}{c} + z_{min}$$

5. The net curve trade can be closed at this updated point `p + dp`.

DELV: Acknowledged. We don't think this is dangerous. Here is a fuzz test that I initially used to convince myself of this. We are not planning on fixing this if an exploit isn't found.

Spearbit: Acknowledged.

5.3.5 Fees are incorporated differently depending on the endpoint called

Severity: Low Risk

Context: HyperdriveLong.sol#L424-L431, HyperdriveLong.sol#L540-L550, HyperdriveShort.sol#L417-L422, HyperdriveShort.sol#L548-L558, HyperdriveBase.sol#L391-L394

Description: The focus of this issue is on the LP/curve fee and the related governance fees when one opens or closes long or short positions. The governance zombie fee is not discussed. All bonds are assumed to be freshly minted on the curve and are fully non-matured.

endpoint	unit	t_r	f_c	f_f	f_g,
L_o	$[y]$	1	$\phi_c \space\Big(\cfrac{1}{p_{spot}}- 1 \Big) \Big(c \space \Delta z \Big) $	0	ϕ_g
S_o	$[z]$	1	$\phi_c \Big(\cfrac{1}{p_{spot}}- 1 \Big)p_{spot} \Big(\cfrac{| t_r \Delta y |}{c} \Big) $	$\phi_f \frac{(1 - t_r)\|dy\|}{c}$	ϕ_g
L_c	$[z]$	t_r	$\phi_c \Big(\cfrac{1}{p_{spot}}- 1 \Big)p_{spot} \Big(\cfrac{| t_r \Delta y |}{c} \Big) $	$\phi_f \frac{(1 - t_r)\|dy\|}{c}$	ϕ_g
S_c	$[z]$	t_r	$\phi_c \Big(\cfrac{1}{p_{spot}}- 1 \Big)p_{spot} \Big(\cfrac{| t_r \Delta y |}{c} \Big) $	$\phi_f \frac{(1 - t_r)\|dy\|}{c}$	ϕ_g

In the above, we haven't included the scaling factor for L_c and S_c due to a drop in the price per vault share $\min(1, \frac{c_{close}}{c_{open}})$

1. Basically, when calculating f_c we take a ϕ_c portion of the fixed interest rate at the end of the position duration. These formulas are not so accurate since the the value of the position at the end of the term is known to be $t_r \Delta y$ and the principal paid for it is also known which is $c\Delta z$, so the fixed interest is:

$$t_r \Delta y - c \Delta z$$

So we could have had:

endpoint	unit	t_r	f_c
L_o	$[y]$	1	$\phi_c(t_r \Delta y - c \Delta z)$
S_o	$[z]$	1	$\phi_c(t_r \Delta y - c \Delta z)/c$
L_c	$[z]$	t_r	$\phi_c(t_r \Delta y - c \Delta z)/c$
S_c	$[z]$	t_r	$\phi_c(t_r \Delta y - c \Delta z)/c$

2.

endpoint	curve point after the trade	vector of LP fees added to the resultant point	vector of fees added to the resultant po
L_o	$(z_e + \Delta z, y - t_r \Delta y)$	$(0, f_c)$	$(-\frac{\phi_g f_c p_{spot}}{c}, (1-\phi_g) f_c)$
L_c	$(z_e - \Delta z, y + t_r \Delta y)$	$(f_c, 0)$	$(f_c, 0)$
S_o	$(z_e - \Delta z, y + t_r \Delta y)$	$(f_c, 0)$	$((1-\phi_g) f_c, 0)$
S_c	$(z_e + \Delta z, y - t_r \Delta y)$	$(f_c, 0)$	$((1-\phi_g) f_c, 0)$

Recommendation: To solve all the above inconsistencies, it would be best to calculate f_c for all the endpoints in the unit of $[z]$ share reserve with the formula:

$$f_c = \phi_c(\frac{t_r \Delta y}{c} - \Delta z)$$

and

endpoint	curve point after the trade	vector of LP fees added to the resultant point	vector of fees added to the resultant po
L_o	$(z_e + \Delta z, y - t_r \Delta y)$	$(f_c, 0)$ (changed)	$((1-\phi_g) f_c, 0)$ (changed)
L_c	$(z_e - \Delta z, y + t_r \Delta y)$	$(f_c, 0)$	$((1-\phi_g) f_c, 0)$ (changed)
S_o	$(z_e - \Delta z, y + t_r \Delta y)$	$(f_c, 0)$	$((1-\phi_g) f_c, 0)$
S_c	$(z_e + \Delta z, y - t_r \Delta y)$	$(f_c, 0)$	$((1-\phi_g) f_c, 0)$

Note that changing the fee vectors for L_o might require adjusting the ζ. Another option would have been to have the fee vectors for L_o:

endpoint	curve point after the trade	vector of LP fees added to the resultant point	vector of fees added to the resultant po
L_o	$(z_e + \Delta z, y - t_r \Delta y)$	$(0, f_c \cdot c)$	$(0, (1-\phi_g) f_c \cdot c)$

Another question that arises even after these fixes is that, how should the final fee vectors $(0, (1-\phi_g) f_c \cdot c)$ or $((1-\phi_g) f_c, 0)$ change the state. Currently these would just scale up the curve C to another curve λC. But one would expect that the LP/pool fees $(1-\phi_g) f_c$ should increase the present value PV by that amount:

$$PV \to PV + (1-\phi_g) f_c$$

So perhaps one should actually not use these simplified vectors as depending on the curve C and the current point after the curve trade one might end up with different scaling factors and also with different increase in present value ΔPV. One can approach the LP/pool fees differently by solving an optimisation problem where we would look for a scaling factor λ where when the curve C is scaled up to λC the present value is increased by $(1-\phi_g) f_c$.

DELV: Acknowledged.

I think that the issue of pricing is a design decision. We considered the calculation that you present, and it didn't have the properties we were looking for.

Regarding the point that our fees won't directly increase the present value, I think that this is a good callout. Considering that it actually does directly increase the present value when the net curve trade is zero and the impact increases as the pool has a larger net curve position, we don't think that this is that impactful in practice. We will most likely fix this in the next version of the protocol when we have more time to properly vet this kind of change.

Spearbit: Acknowledged.

5.3.6 After trading on the curve in `openLong` and applying the fees, spot price is not check to make sure it is still not greater than `1`

Severity: Low Risk

Context: HyperdriveLong.sol#L374, HyperdriveLong.sol#L435-L448, HyperdriveLong.sol#L470

Description: In `_calculateOpenLong` when one trades on the curve and applies the pool and governance fees, we don't check whether the spot price is still below 1. Previous reviewed revision had the following check after calculating the deltas:

```
// If the ending spot price is greater than or equal to 1, we are in the
// negative interest region of the trading function. The spot price is
// given by ((mu * z) / y) ** tau, so all that we need to check is that
// (mu * z) / y < 1 or, equivalently, that mu * z >= y.
if (
    _initialSharePrice.mulDown(
        _marketState.shareReserves + shareReservesDelta
    ) >= _marketState.bondReserves - bondReservesDelta
) {
    revert Errors.NegativeInterest();
}
```

The current version of `HypderdriveLong` only checks an upper bound for the spot price after trading on the curve (before applying the fees) to make sure the purchased bonds would not bear negative interest:

$$\frac{\mu(z - \zeta + \Delta z)}{y + \Delta y}$$

$$t_s \leq \frac{(1 - \phi_f)}{1 + \phi_c(\frac{1}{p_{spot}} - 1)(1 - \phi_f)} \leq 1$$

Note that the far right inequality is guaranteed if the spot price before trading on the curve $p_{spot} \leq 1$. But we also need to make sure the spot price stays not above 1 when fees are included:

$$\frac{\mu(z - \zeta + \Delta z - \frac{\phi_g p_{spot}}{c} f_c)}{y + \Delta y + (1 - \phi_g) f_c}$$

$$t_s \leq \frac{\mu(z - \zeta + \Delta z)}{y + \Delta y} t_s \leq 1$$

Thus the required inequality after applying the fees should be true, unless due to `pow` not being monotone, the left inequality above would not hold and the spot price after applying the fees would jump above 1.

Recommendation: It might be best to check for the spot price not being greater than 1 directly using the below inequality:

$$\mu(z - \zeta + \Delta z - \frac{\phi_g p_{spot}}{c} f_c) \leq y + \Delta y + (1 - \phi_g) f_c$$

or even check the more strict inequality without including the fees:

$$\mu(z - \zeta + \Delta z) \leq y + \Delta y$$

On the other hand, it might actually make more sense to check:

$$\frac{\mu(z - \zeta + \Delta z - \frac{\phi_g p_{spot}}{c} f_c)}{y + \Delta y + (1 - \phi_g) f_c}$$

$^{t_s} \leq 1$

aka the actual value of the calculated spot price since `pow` is not monotone in general. Or one should check/prove the following for all t_s and $x \leq 1$ we need to check using `pow`:

$$x^{t_s} \leq 1$$

DELV: Fixed in PR 789.

Spearbit: Verified.

5.3.7 The negative interest check for non-matured bonds in `_calculateCloseShort` is not entirely accurate

Severity: Low Risk

Context: HyperdriveShort.sol#L519-L538

Description: When closing a short the non-matured portion of the Δy, ie $t_r \Delta y$ gets minted for the pool and the trader would need to pay the pool the following amount of base:

$$\Delta x_{total} = \max(0, \frac{c_{close}}{c_{open}} + \phi_f$$

$\Delta y - (1 - t_r)(1 + \phi_f)\Delta y + c_1 \Delta z_{curve} + \phi_c(1 - p_{spot}) t_r \Delta y)\lambda$

where λ is:

$$\lambda = \min(1, \frac{c_{close}}{c_{open}}) \min(1, \frac{c_1 z_{zombie}}{x_{zombie}})$$

So ignoring λ and the max function we have:

$$\Delta x_{total} = \frac{c_{close}}{c_{open}} + \phi_f$$

$\Delta y - (1 - t_r)(1 + \phi_f)\Delta y + c_1 \Delta z_{curve} + \phi_c(1 - p_{spot}) t_r \Delta y$

So for closing the non-matured portion of the bond, the trader would need to pay back the pool the following amount of base which is why it is subtracted above:

$$c_1 \Delta z_{curve} + \phi_c(1 - p_{spot}) t_r \Delta y$$

assuming that this is like minting a new bond in the amount of $tr\Delta y$, at maturity it should have the base value of $t_r \Delta y$ so for this portion of the bond to be not bearing negative interests we need to have (below we are assuming no flat fees are taken from $t_r \Delta y$ at maturity since the trader happens from the LP pool to itself):

$$c_1 \Delta z_{curve} + \phi_c(1 - p_{spot}) t_r \Delta y \leq t_r \Delta y$$

Then:

$$\frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq 1 - \phi_c(1 - p_{spot})$$

Note that when closing a short first we trade on the curve the non-matured portion then we do the flat trade for the matured portion of the bonds.

Again like opening longs, since our curve is a convex curve and we have:

$$\frac{\mu(z - \zeta)}{y}$$

$$_{t_s} \leq \frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq \frac{\mu(z - \zeta + \Delta z_{curve})}{y - t_r \Delta y}_{t_s}$$

So checking:

$$\frac{\mu(z - \zeta + \Delta z_{curve})}{y - t_r \Delta y}$$

$$_{t_s} \leq 1 - \phi_c(1 - p_{spot})$$

implies:

$$\frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq 1 - \phi_c(1 - p_{spot})$$

There are a few issues:

1. The below inequality might not always be true due to errors in calculations of functions like `pow`:

$$\frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq \frac{\mu(z - \zeta + \Delta z_{curve})}{y - t_r \Delta y}$$

$_{t_s}$

2. In general, since at this stage λ can be calculated one should in total generality check:

$$\frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq \frac{1}{\lambda} - \phi_c(1 - p_{spot})$$

We have $\lambda \leq 1$ so the checked inequality without λ is actually more strict:

$$\frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq 1 - \phi_c(1 - p_{spot}) \leq \frac{1}{\lambda} - \phi_c(1 - p_{spot})$$

Recommendation: To address the issue of errors due to `pow` or other arithmetic errors it might be best to check the less strict but cheaper inequality:

$$\frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq 1 - \phi_c(1 - p_{spot})$$

and for 2. one can check less strict inequality involving λ:

$$\frac{c_1 \Delta z_{curve}}{t_r \Delta y} \leq \frac{1}{\lambda} - \phi_c(1 - p_{spot})$$

or just use the one without.

DELV: Acknowledged.

I acknowledge the issues with `pow`, but I don't think we'll apply the change that converts $\left(\frac{\mu \cdot (z-\zeta+\Delta z)}{y-\Delta y}\right)^{-t_s}$ to $\frac{c \cdot \Delta z}{\Delta y}$ since this would make the check significantly looser.

Consider the case when a trader brings the pool's spot price from 0.9 to 0.99. The ending spot price is 0.99, but their realized price would be closer to 0.94 or 0.95. The goal of this check is to ensure that the trader doesn't buy *any* bonds at a negative interest rate (even if the trade in aggregate was executed at a positive interest rate). For more details, you can refer to this issue which explains why we needed to make this change in the first place (hyperdrive issue 582).

We're not going to apply the second recommendation either. It will make the code quite a bit more complicated due to the issues with stack-too-deep, and it won't have a large impact on execution since we still verify that the ending spot price is less than or equal to 1. This latter point significantly limits the impact that the negative interest factor would have on the result.

Spearbit: Acknowledged.

5.3.8 The negative interest check for non-matured bonds in `_calculateOpenLong` is not entirely accurate

Severity: Low Risk

Context: HyperdriveLong.sol#L400-L420

Description: In this context, we actually want to enforce:

$$\frac{c_0 \Delta z}{-\Delta y} = \frac{\Delta x}{-\Delta y} \leq \frac{(1-\phi_f)\lambda}{1+\phi_c(\frac{1}{p_{spot}}-1)(1-\phi_f)\lambda}$$

where λ is:

$$\lambda = \min(1, \frac{c_{close}}{c_{open}}) \min(1, \frac{c_1 z_{zombie}}{x_{zombie}})$$

But since λ depends on some to-be-determined parameters in the future, one cannot use it for the check here. So we need to at least check assuming no negative interest on the vault side and also on the zombie interest side:

$$\frac{c_0 \Delta z}{-\Delta y} = \frac{\Delta x}{-\Delta y} \leq \frac{(1-\phi_f)}{1+\phi_c(\frac{1}{p_{spot}}-1)(1-\phi_f)}$$

But here we are instead checking the slope of the scaled perpendicular line at the resultant point $(z-\zeta+\Delta z, \Delta y)$. This is fine since our curve is a convex curve and we have:

$$\frac{\mu(z-\zeta)}{y}$$

$$t_s \leq \frac{\Delta x}{-\Delta y} \leq \frac{\mu(z-\zeta+\Delta z)}{y+\Delta y} t_s$$

Note that when opening a long our reserve point moves to the right of the diagram. So enforcing:

$$\frac{\mu(z-\zeta+\Delta z)}{y+\Delta y}$$

$$t_s \leq \frac{(1-\phi_f)}{1+\phi_c(\frac{1}{p_{spot}}-1)(1-\phi_f)}$$

would imply:

$$\frac{c_0 \Delta z}{-\Delta y} = \frac{\Delta x}{-\Delta y} \leq \frac{(1-\phi_f)}{1+\phi_c(\frac{1}{p_{spot}}-1)(1-\phi_f)}$$

Although this a great check to perform it would still not protect the user from negative interests stemming from the vault share price or zombie interest calculations λ.

But still, there is a small issue. Due to errors in the `pow` functions or other arithmetic errors in divisions the following inequality might not hold in general:

$$\frac{\Delta x}{-\Delta y} \leq \frac{\mu(z-\zeta+\Delta z)}{y+\Delta y}$$

t_s

Recommendation: It is cheaper and more concise to check the original inequality:

$$\frac{c_0 \Delta z}{-\Delta y} = \frac{\Delta x}{-\Delta y} \leq \frac{(1-\phi_f)}{1+\phi_c(\frac{1}{p_{spot}}-1)(1-\phi_f)}$$

DELV: Acknowledged. I acknowledge the issues with `pow`, but I don't think we'll apply the change that converts $\left(\frac{\mu \cdot (z-\zeta+\Delta z)}{y-\Delta y}\right)^{-t_s}$ to $\frac{c \cdot \Delta z}{\Delta y}$ since this would make the check significantly looser.

Consider the case when a trader brings the pool's spot price from 0.9 to 0.99. The ending spot price is 0.99, but their realized price would be closer to 0.94 or 0.95. The goal of this check is to ensure that the trader doesn't buy *any* bonds at a negative interest rate (even if the trade in aggregate was executed at a positive interest rate). For more details, you can refer to this issue which explains why we needed to make this change in the first place (hyperdrive issue 582).

Spearbit: Acknowledged.

5.3.9 Front-running when deploying an instance

Severity: Low Risk

Context: HyperdriveDeployerCoordinator.sol#L155, HyperdriveDeployerCoordinator.sol#L211, StETH-HyperdriveCoreDeployer.sol#L46, StETHTarget0Deployer.sol#L34, StETHTarget1Deployer.sol#L34, StETHTarget2Deployer.sol#L34, StETHTarget3Deployer.sol#L34, StETHTarget4Deployer.sol#L34

Description: The desired deployment process goes through several contracts: `Factory` → `Coordinator` → `InstanceCoreDeployer`. It's important that the transaction can't be made to fail at any of these levels, as it would cause a failed deployment in the upper levels. They can be made to fail if the subset of parameters that are used for the deployment call can be re-used by a different party:

1. Front-running any `StETHTarget01234Deployer.deploy(config, extraData, salt)`: Anyone can front-run this by choosing the exact same parameters. The victim transaction will fail.

   We can fix this by binding one of the parameters used in `create2` to the `msg.sender`. `_salt` is the most natural one: `_salt = keccak256(salt, msg.sender)`.

```
new StETHTarget0{ salt: _salt }(_config, lido)
```

2. Front-running any `StETHHyperdriveCoreDeployer.deploy(config, extraData, target0, target1, target2, target3, target4, salt)`: Same case as 1): Anyone can fron-trun this by choosing the exact same parameters. The victim transaction will fail.

   We can fix this by binding one of the parameters used in create2 to the `msg.sender`. `_salt` is the most natural one: `_salt = keccak256(salt, msg.sender)`.

```
new StETHHyperdrive{ salt: _salt }(
    _config,
    target0,
    target1,
    target2,
    target3,
    target4,
    lido
)
```

3. Front-running any `HyperdriveDeployerCoordinator.deployTarget(deploymentId, deployConfig, extraData, targetIndex, salt)`: Anyone can front-run this by choosing `deploymentId' != deploymentId` but otherwise the exact same parameters. Because the deployment-relevant call ignores the `deploymentId`:

```
target = IHyperdriveTargetDeployer(target0Deployer).deploy(
    config_,
    _extraData,
    _salt
);
```

   We can fix this by binding one of these parameters to the `msg.sender`. We also need to bind it to `deploymentId` to make it part of the deployment-relevant parameters (important for preventing factory deployment collisions). `_salt` is the most natural one: `_salt = keccak256(salt, msg.sender, deploymentId)`.

Recommendation: The recommendation is in 2 steps:

- Consider hashing the `deploymentId` and `msg.sender` into the `salt` at the coordinator level. HyperdriveDeployerCoordinator.sol#L155: this line should be `keccak256(abi.encode(msg.sender, _deploymentId, _salt)` The same change should be applied for the `deployTarget` within the `HyperdriveDeployerCoordinator`.
- Consider hashing the `msg.sender` at the Core Deployer level. The following example is just for `StETHHyperdriveCoreDeployer` but it should be applied for all the StETH targets and ERC4626 targets and core. StETHHyperdriveCoreDeployer.sol#L46: this line should be `keccak256(abi.encode(msg.sender, _salt)`

DELV: Fixed in PR 780.

Spearbit: Verified.

5.3.10 `calculateSharesOutGivenBondsInDownSafe` succeeds even if it should fail

Severity: Low Risk

Context: YieldSpaceMath.sol#L308-L311

Description: In the last step of `calculateSharesOutGivenBondsInDownSafe`, `ze - z` is computed. The function should fail if `ze < z`. However, it still succeeds with a return value of `0`.

```
if (ze > _z) {
    result = ze - _z;
}
success = true;
```

Recommendation: Consider using explicit returns.

```
- if (ze > _z) {
-     result = ze - _z;
- }
- success = true;
+ if (ze < _z) {
+     return (0, false);
+ }
+ return (ze - z, true);
```

DELV: Fixed in PR 810. I left some comments on the drawbacks of this change and the test that highlights the issue. I think it's probably still worth accepting, but it would be good if you could consider the drawbacks.

Spearbit: Fixed. The PR is now more explicit about what is allowed to fail and what isn't, which is good. The cutoff threshold for considering the net curve trade failed is set at `minimumTransactionAmount`, which is a bit arbitrary, I can imagine you could also come up with tests that fail given this threshold. But I think it's ok to do it like this.

5.3.11 Unsafe type casts

Severity: Low Risk

Context: See below.

Description: When type-casting from a type with a larger range to a type with a smaller range, Solidity truncates any bits that exceed the new range instead of reverting. This silent truncation can lead to unexpected errors. The following code performs truncated type-casts:

- There are 56 `int256` casts from types with a higher input range throughout the code base in `HyperdriveBase`, `HyperdriveCheckpoint`, `HyperdriveLong`, `HyperdriveLP`, `HyperdriveShort`, `FixedPointMath`, `HyperdriveMath`, `LPMath`.
- Two `int128` casts from types with a higher input range: HyperdriveBase.sol#L277, HyperdriveBase.sol#L402.

Recommendation: Consider using a `SafeCast` library as the default way to perform type casts.

DELV: Fixed in PR 812. I didn't use safe cast within the `exp` implementation which is consistent with identical (and audited) implementations. Additionally, I didn't add a safe cast in `_isSolvent`, since the casting was removed in PR 779.

Spearbit: Verified.

5.3.12 Wrong conditional in `calculateSharesInGivenBondsOutDerivativeSafe`

Severity: Low Risk

Context: LPMath.sol#L1437

Description: The `calculateSharesInGivenBondsOutDerivativeSafe` function checks `if (inner >= 0)` but `inner` is an unsigned integer, therefore this conditional is always true.

Recommendation: Consider changing the conditional to `if (inner >= ONE)`.

DELV: Fixed in PR 785.

Spearbit: Verified.

5.3.13 Non-matued shorts might no be able to close because of the netting feature

Severity: Low Risk

Context: HyperdriveShort.sol#L190-L196

Description: A correctly implemented long ⇔ shorts netting feature per checkpoint will check that the protocol is still solvent when closing netted, non-matured shorts. This can lead to a DoS for non-matured `closeShort` calls by frontrunning it by taking out the max long. Even if not malicious, the protocol can be in a state of low excess idle reserves as idle reserves are regularly removed through the withdrawal shares conversion process.

Recommendation: Consider ways to mitigate the impact. For example, by adding a short buffer when computing the idle that can be distributed to withdrawal shares.

DELV: We acknowledge this issue. Liquidity is not guaranteed for positions to be closed before maturity. In most cases, the economics will result in some liquidity being available for traders to close their positions, and it is expensive and unprofitable to DoS users who are trying to close their positions; however, in the worst case, traders may need to wait until maturity to close their positions. Solvency is guaranteed for these positions, so even if a position can't be closed part-way through a term, the system maintains solvency to close the position at maturity.

Spearbit: Acknowledged.

5.3.14 `_minOutputPerShare` compared against both base and vault share amounts

Severity: Low Risk

Context: HyperdriveLP.sol#L425-L427

Description: The `_minOutputPerShare` parameter in `_redeemWithdrawalSharesInternal` is defined as:

> `_minOutputPerShare` The minimum amount of base the LP expects to receive for each withdrawal share that is burned.

It is compared against the withdrawn proceeds in `_minOutputPerShare.mulUp(withdrawalSharesRedeemed) > proceeds`. However, `proceeds` can be either a base or vault shares amount, depending on the withdrawal `_options`.

Recommendation: Consider either updating `_minOutputPerShare` to be used as either the minimum base *or* shares amount, or convert `proceeds` to base proceeds first.

DELV: Fixed in PR 782 by updating the documentation.

Spearbit: Verified.

5.3.15 `addLiquidity` with `minApr`/`maxApr` can be DoS'd

Severity: Low Risk

Context: HyperdriveLP.sol#L149-L151

Description: The `_addLiquidity` function reverts if the APR is out of the `_minApr`/`_maxApr` bounds. It's possible for an attacker to sandwich the `addLiquidity` transaction with trades that move the APR out of this range (and back-run to undo this trade), making the liquidity provisioning revert.

Recommendation: We don't believe this attack has a high likelihood as it would cost gas and trading fees and there's no incentive to perform it. Furthermore, the victim can either adjust the slippage parameters or send the transaction without broadcasting it to a public mempool.

DELV: Acknowledged.

Spearbit: Acknowledged.

5.3.16 Fixed `_minimumTransactionAmount` is used for shares that grow in value, pricing out users

Severity: Low Risk

Context: HyperdriveCheckpoint.sol#L32

Description: The `_minimumTransactionAmount` is compared against amounts of all types: base, vault shares, LP shares, bonds. The vault shares and LP shares will usually grow in value so the minimum transaction amount for these will also grow in value over time. Users might be priced out as the minimum transaction amount's value increases.

Recommendation: Under normal circumstances, the share growth should be limited but an attacker can inflate the share prices either by farming fees for LPs or other ways for the vault. However, these attacks come with a cost and we consider them unlikely to happen in practice.

DELV: Acknowledged. Agreed that this is unlikely to happen. Worst case scenario, a new pool could be deployed with a different minimum transaction amount. We're not going to fix this.

Spearbit: Acknowledged.

5.3.17 Wrong `minimumTransactionAmount` verification

Severity: Low Risk

Context: ERC4626HyperdriveDeployerCoordinator.sol#L67-L73

Description: The `ERC4626HyperdriveDeployerCoordinator._checkPoolConfig` function checks the `minimumShareReserves` twice.

Recommendation: The second check should be on the `_deployConfig.minimumTransactionAmount` amount instead.

DELV: Fixed in PR 772.

Spearbit: Verified.

5.3.18 Factory's deployment config parameters should be explicit about its values

Severity: Low Risk

Context: HyperdriveFactory.sol#L888-L928

Description: The `deployAndInitialize` and `deployTarget` functions accept a `_config` parameter. Its `linkerFactory`, `linkerCodeHash`, `feeCollector`, `governance` values are taken from the contract's current storage values. It can happen that a change to one of these contracts happens before the `deploy*` transaction and the deployer ends up using unexpected values.

Recommendation: Consider setting the `_config` values to the expected values and verify that they match the current storage variables of the contract instead of defaulting to them.

DELV: Fixed in PR 776.

Spearbit: Verified.

5.3.19 `deployAndInitialize` flow decision is incorrectly made using `msg.value` rather than `baseToken`

Severity: Low Risk

Context: HyperdriveFactory.sol#L634-649

Description: When calling `deployAndInitialize`, the logic to determine the use of ETH versus ERC20 tokens is currently based on the following check:

```
if (msg.value >= _contribution) {
```

This piece of code is comparing `msg.value` included within the call with the `contribution` amount. This approach can lead to some reverts and unexpected behavior, as when `msg.value` is greater or equal to the `contribution`, the contract assumes payment in ETH regardless of the `baseToken` setting. Conversely, if the `baseToken` is ETH (`0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`), but `msg.value` is less than the `contribution`, the contract attempts to process the payment as an ERC20 transaction, which will fail.

Recommendation: Change the logic to first check `baseToken` against the ETH placeholder (`0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`) to determine the method of payment rather than regarding `msg.value`. The contract should first verify if the `baseToken` is ETH and confirm that `msg.value` matches the `contribution` amount for ETH transactions.

DELV: Fixed in PR 773.

Spearbit: Verified.

5.4 Gas Optimization

5.4.1 `toString` can use an auxiliary function to avoid manual repetition

Severity: Gas Optimization

Context: AssetId.sol#L107-L139

Description: `toString` some sort of do-while mechanism, that can be refactored into an auxiliary function for clarity. Additionally, it would improve gas usage:

- `Snapshots result: Overall gas change: -49959657 (-0.417%)`

Recommendation: Consider implementing the following change:

```
 /// @dev Converts an unsigned integer to a string.
 /// @param _num The integer to be converted.
 /// @return result The stringified integer.
 function toString(
     uint256 _num
 ) internal pure returns (string memory result) {
     // We overallocate memory for the string. The maximum number of digits
     // that a uint256 can hold is log_10(2 ^ 255) which is approximately
     // 76.
     uint256 maxStringLength = 77;
     bytes memory rawResult = new bytes(maxStringLength);

     // Loop through the integer and add each digit to the raw result,
     // starting at the end of the string and working towards the beginning.
-     rawResult[maxStringLength - 1] = bytes1(
-         uint8(uint256((_num % 10) + 48))
-     );
-     _num /= 10;
-     uint256 digits = 1;
-     while (_num != 0) {
-         rawResult[maxStringLength - digits - 1] = bytes1(
-             uint8(uint256((_num % 10) + 48))
-         );
```

```
-           _num /= 10;
-           digits++;
-       }

+    uint256 digits; // = 0 notice before was set to 1

+     _num = _auxOperation(maxStringLength, digits++, _num, rawResult);

+     while (_num != 0) {
+        _num = _auxOperation(maxStringLength, digits++, _num, rawResult);
+     }

      // Point the string result to the beginning of the stringified integer
      // and update the length.
      assembly {
          result := add(rawResult, sub(maxStringLength, digits))
          mstore(result, digits)
      }
      return result;
  }

+   /// @dev some clearer name of the goal of the function may be nice
+   /// @param maxStringLength size of rawResult
+   /// @param digits counter where first call is 0
+   /// @param _num initial _num or it's result over iterations
+   /// @param rawResult the raw result we are writing
+   /// @return _num result of dividing input _num by 10
+   function _auxOperation (
+       uint maxStringLength,
+       uint digits,
+       uint _num,
+       bytes memory rawResult
+   ) pure internal returns(uint) {
+       rawResult[maxStringLength - digits - 1] = bytes1(
+           uint8((_num % 10) + 48)
+       );
+       return _num / 10;
+   }
```

DELV: Fixed in PR 825.

Spearbit: Verified.

5.4.2 Deriving `maxShareReservesDelta` in `calculateDistributeExcessIdle` can be simplified using a scaling trick

Severity: Gas Optimization

Context: LPMath.sol#L436-L439

Description: Assume the pool is net short since in the net long case we would just return the idle share amount. Let:

$$F(z) = dy_{max}(z) = y - \frac{K}{\frac{c}{\mu} + 1}$$

$$\frac{1}{1 - t_s - y^{net}_{curve}}$$

where K is:

$$K = \frac{c}{\mu}(\mu(z - \zeta))^{1-t_s} + y^{1-t_s}$$

$F(z)$ is the function that calculates the maximum amount of bonds that can be taken away from the pool's curve while still guaranteeing that the net shorts can be closed. This is the function used in the optimization problem using Newton's method to find the maximum amount of share reserves that can be taken from the pool while still being able to close the net shorts on the updated curve.

So we have:

$$\frac{dF}{dz} = \frac{dy}{dz} - \frac{1}{1 - t_s} \frac{K}{\frac{c}{\mu} + 1}$$

$$\frac{t_s}{1 - t_s} \frac{\frac{dK}{dz}}{\frac{c}{\mu} + 1}$$

$$\frac{dK}{dz} = (1 - t_s) \cdot c(\mu(z - \zeta))^{-t_s}(1 - \frac{d\zeta}{dz}) + y^{-t_s}\frac{dy}{dz}$$

and so we have:

$$\frac{dF}{dz} = \frac{dy}{dz} - \frac{K}{\frac{c}{\mu} + 1}$$

$$\frac{t_s}{1 - t_s} \frac{c(\mu(z - \zeta))^{-t_s}(1 - \frac{d\zeta}{dz}) + y^{-t_s}\frac{dy}{dz}}{\frac{c}{\mu} + 1}$$

or

$$\frac{dF}{dz} = \frac{dy}{dz} - \frac{K^{t_s}}{\frac{c}{\mu} + 1}$$

$$\frac{1}{1 - t_s} c(\mu(z - \zeta))^{-t_s}(1 - \frac{d\zeta}{dz}) + y^{-t_s}\frac{dy}{dz}$$

Now note that before calling `calculateMaxBuyBondsOutDerivativeSafe` one calls:

1. `calculateUpdateLiquidity` which scales our points (in our calculations dz has a negative value, in the implementation the absolute value of it is used):

$$(z_{og}, \zeta_{og}, y_{og}) \to \frac{z_{og} + dz}{z_{og}}(z_{og}, \zeta_{og}, y_{og})$$

and so we get:

$$\frac{d\zeta}{dz} = \frac{\zeta_{og}}{z_{og}}$$

$$\frac{dy}{dz} = \frac{y_{og}}{z_{og}}$$

The above are just rough discrete approximations of the actual derivatives.

2. `calculateMaxBuyBondsOut` and `calculateMaxBuyBondsOutDerivativeSafe` use these scaled values for (z, ζ, y).

So replacing these approximate derivates above we get:

$$\frac{dF}{dz} = \frac{y_{og}}{z_{og}} - \frac{K^{t_s}}{\frac{c}{\mu} + 1}$$

$$\frac{1}{1 - t_s} c(\mu(z - \zeta))^{-t_s}(1 - \frac{\zeta_{og}}{z_{og}}) + y^{-t_s}\frac{y_{og}}{z_{og}}$$

If we simplify the calculation in `calculateMaxBuyBondsOutDerivativeSafe` we would also get the same result up to rounding errors and the `delta` underflow check.

All this is done so that one can use the Newton method to approximate the zero of $F(z)$:

$$z_{i+1} = z_i - \frac{F(z)}{F'(z)}$$

Scaling

Since approximate derivation calculation is performed by scaling our points in `calculateUpdateLiquidity` , one can use a different approach:

$$(z_{og}, \zeta_{og}, y_{og}) \to \lambda(z_{og}, \zeta_{og}, y_{og})$$

ie, the derivation is not performed in the direction of z but the direction of the ray connecting (z, ζ, y) to the origin.

Define:

$$G(z, \zeta, y) = y - \frac{K}{\frac{c}{\mu} + 1}$$

$$\frac{1}{1 - t_s}$$

and note that F is actually a function of 3 dependant parameters $F(z, \zeta, y)$ and so we have:

$$F(z, \zeta, y) = G(z, \zeta, y) - y_{curve}^{net}$$

and

$$F(\lambda z, \lambda\zeta, \lambda y) = \lambda G(z, \zeta, y) - y_{curve}^{net}$$

and so let's define:

$$f(\lambda) = F(\lambda z_{og}, \lambda \zeta_{og}, \lambda y_{og})$$

then we have:

$$f'(\lambda) = G(z_{og}, \zeta_{og}, y_{og})$$

and so by Newton's formula, we get:

$$\lambda_{i+1} = \lambda_i - \frac{f(\lambda_i)}{f'(\lambda)} = \frac{y^{net}_{curve}}{G(z_{og}, \zeta_{og}, y_{og})}$$

Which was also obvious from the scaling properties of G one could have solved for the root of f directly, the root being:

$$\lambda_* = \frac{y^{net}_{curve}}{G(z_{og}, \zeta_{og}, y_{og})}$$

Therefore Newton's method is actually not needed one can first derive the scaling factor λ_* and use that to find:

$$dz_{max} = (\lambda_* - 1) z_{og}$$

Note that in the implementation you would use the negated value of the above formula since you use work with the absolute values.

Recommendation: Use the scaling approach to simplify the implementation of `calculateMaxShareReserves-Delta` in the case where the pool is net short.

DELV: Fixed in PR 771.

Spearbit: Verified.

5.4.3 The `rhs` scaling in `calculateMaxShareReservesDeltaInitialGuess` can be simplified

Severity: Gas Optimization

Context: LPMath.sol#L1183-L1206

Description: Not considering the different rounding errors the current computation in this context is:

$$\frac{r}{\frac{y_{og}(1 - \zeta_{og}/z_{og})}{z_{og} - \zeta_{og}}} = \frac{r \cdot z_{og}}{y_{og}}$$

Recommendation: Thus it can be simplified to:

```
rhs = rhs.mulDivUp(_params.originalShareReserves, _params.originalBondReserves)
```

DELV: The blocks of code in this context have been removed in PR 771.

Spearbit: Verified.

5.4.4 Duplicate condition check in `shouldShortCircuitDistributeExcessIdleShareProceeds`

Severity: Gas Optimization

Context: LPMath.sol#L896-L903

Description: The `shouldShortCircuitDistributeExcessIdleShareProceeds` function performs the following checks:

```
if (lpSharePriceAfter < lpSharePriceBefore) {
    return false;
}
return
    lpSharePriceAfter >= lpSharePriceBefore &&
    lpSharePriceAfter <=
    // NOTE: Round down to make the check stricter.
    lpSharePriceBefore.mulDown(ONE + SHARE_PROCEEDS_MIN_TOLERANCE);
```

However, the `lpSharePriceAfter < lpSharePriceBefore` condition & return `false` is already covered by the `lpSharePriceAfter >= lpSharePriceBefore && ...` that follows.

Recommendation: Consider removing the first check:

```
- if (lpSharePriceAfter < lpSharePriceBefore) {
-     return false;
- }
```

DELV: Fixed in PR 784.

Spearbit: Verified.

5.4.5 Calculation can be omitted in case of `ONE == derivative`

Severity: Gas Optimization

Context: LPMath.sol#L1322-L1330, LPMath.sol#L1468-L1475

Description: We can see how the code includes an `if/else` block, where in the case `(ONE >= derivative)` it calculates the derivative, otherwise it fast returns `(0, true)`. However, this approach is redundant for `ONE == derivative`, resulting in unnecessary operations since `derivative = ONE - ONE` simplifies to `0`, and subsequent calculations yield the same outcome.

```
// derivative = 1 - derivative
if (ONE >= derivative) {
    derivative = ONE - derivative;
} else {
    // NOTE: Small rounding errors can result in the derivative being
    // slightly (on the order of a few wei) greater than 1. In this case,
    // we return 0 since we should proceed with Newton's method.
    return (0, true);
}

// ...

// NOTE: Round down to round the final result down.
//
/// derivative = derivative * (1 - (zeta / z))
// ...
return (derivative, true); //@audit so in that case is performed a multiplication by 0 and returned (0,
↪  true)
```

Recommendation: Modify the conditional check from `if (ONE >= derivative)` to `if (ONE > derivative)` to exclude the equality case, which leads to redundant calculations. This adjustment will ensure that the unnecessary

subtraction and multiplication by zero are avoided, simplifying the logic and potentially improving the performance of the function.

DELV: Resolved in PR 786.

Spearbit: Fixed.

5.4.6 Unused name returns affect readability and gas usage

Severity: Gas Optimization

Context: HyperdriveTarget0.sol#L47, HyperdriveTarget1.sol#L44, HyperdriveTarget1.sol#L65, HyperdriveTarget3.sol#L49, HyperdriveTarget4.sol#L49, StETHBase.sol#L84, StETHBase.sol#L138, StETHBase.sol#L136, HyperdriveBase.sol#L430, HyperdriveBase.sol#L451, HyperdriveLong.sol#L128, HyperdriveLong.sol#L472-L477, HyperdriveLP.sol#L114, HyperdriveLP.sol#L314, HyperdriveLP.sol#L366, HyperdriveLP.sol#L429, HyperdriveMath.sol#L116, HyperdriveShort.sol#L461, AssetId.sol#L137, HyperdriveMath.sol#L179, HyperdriveMath.sol#L266, HyperdriveMath.sol#L328, LPMath.sol#L1119

Description: Through the codebae, there are multiple instances of named returns not being used, i.e. being declared but later explicitly returning those variables or a mixed statement where, for example, 3 named return variables are declared, but later explicitly only 2 of the 3 are returned.

Removing unused named return variables can reduce gas usage and improve code clarity.

Recommendation: Remove unused named return variables or use them consistently

DELV: Partially resolved. In the cases where there is a named return that is explicitly returned, there was less of a reason to remove since the return helps to keep the code explicit (see PR 824).

Spearbit: Verified.

5.4.7 Caching storage variables to save `SLOAD`s

Severity: Gas Optimization

Context: HyperdriveDeployerCoordinator.sol#L178, HyperdriveFactory#L489-L492, HyperdriveFactory#L510-L513, HyperdriveFactory#L873-L880

Description: When a storage variable is used several times within the same scope, it is more efficient to cache it for the subsequent reads rather than directly reading from storage, as this implies unnecessary executions of the `SLOAD` instruction.

Recommendation: Consider caching the storage variables that are used several times within the same scope. See below for several instances within the codebase where this optimization may be applied:

- HyperdriveDeployerCoordinator.sol#L178. Cache into memory deployments and then use it on reads. Overall gas change: -45983371 (-0.383%)

```
    ) external returns (address target) {
+       Deployment memory deployment = _deployments[msg.sender][_deploymentId];
```

- HyperdriveFactory#L489-L492, HyperdriveFactory#L510-L513, HyperdriveFactory#L873-L880.Cache into memory deployments and then use it on reads. Overall gas change: -39212163 (-0.326%)

```
+ IHyperdrive.Fees memory maxFees = _maxFees;
+ IHyperdrive.Fees memory minFees = _minFees;
```

DELV: Fixed in PR 818.

Spearbit: Verified.

5.4.8 Unchecked arithmetic can be used for gas optimizations

Severity: Gas Optimization

Context: HyperdriveFactory.sol#L637, HyperdriveFactory.sol#L794, HyperdriveFactory.sol#L834, HyperdriveLP.sol#L69, HyperdriveMultiToken.sol#L161, HyperdriveLong.sol#L311, HyperdriveBase.sol#L330, HyperdriveBase.sol#L337, YieldSpaceMath.sol#L75, YieldSpaceMath.sol#L92, YieldSpaceMath.sol#L184, YieldSpaceMath.sol#L194, YieldSpaceMath.sol#L222, YieldSpaceMath.sol#L289, YieldSpaceMath.sol#L309, YieldSpaceMath.sol#L442, LPMath.sol#L759, LPMath.sol#L790, LPMath.sol#L695, LPMath.sol#L650, LPMath.sol#L1050, LPMath.sol#L1072, LPMath.sol#L1214, LPMath.sol#L1297, LPMath.sol#L1324, LPMath.sol#L1434, LPMath.sol#L1469, LPMath.sol#L1603, HyperdriveMath.sol#L263, HyperdriveMath.sol#L325, StETHBase.sol#L54, HyperdriveShort.sol#L258, HyperdriveMultiToken.sol#L231

Description: Unchecked operations may be applied in certain areas where it's known not to overflow/underflow. Therefore, if already exists a check that `a > b` or `a >= b`, it is safe to assume than `a - b` would be safe. Additionally, uint256 counters are likely to never reach the maximum value.

Recommendation: Consider applying unchecked operations for gas savings where is assumed to be safe to do the operation

DELV: Fixed in PR 819.

Spearbit: Verified.

5.5 Informational

5.5.1 `calculateMaxShareReservesDelta` can return early if `calculateMaxShareReservesDeltaInitialGuess` is `0`

Severity: Informational

Context: LPMath.sol#L964

Description/Recommendation: In `calculateDistributeExcessIdle`, if the pool is net short and if we end up calculating `calculateMaxShareReservesDeltaInitialGuess`:

$$\Delta z_{guess} = \frac{z_{og}}{y_{og}} \max(0, y_{og} - (\Delta y_{net} + y_{optimal}))$$

If $\Delta z_{guess} = 0$, it basically means we cannot remove any liquidity from the curve and $F(z) = y_{og} - (\Delta y_{net} + y_{optimal})$ is already negative and so one can return `0` early and skip the main loop in `calculateMaxShareReservesDelta`.

Note that if we use the suggested scaling solution the initial guess and the loop would not be needed, see the issue "Deriving `maxShareReservesDelta` in `calculateDistributeExcessIdle` can be simplified using a scaling trick".

DELV: `calculateMaxShareReservesDeltaInitialGuess` has been removed in PR 771. The issue is not relevant anymore.

Spearbit: Verified.

5.5.2 Postfix function names with `Down` or `Up` if they are overestimating or underestimating the returned value

Severity: Informational

Context: See Description.

Description: Some utility functions return overestimated or underestimated values.

Recommendation: Postfix function names with `Down` or `Up` if they are overestimating or underestimating the returned value. This would help catch potential rounding bugs easier and it would make it easier to reason about the rounding directions.

- `down` suffix:
  - `_calculateTimeRemaining`
  - `_calculateTimeRemainingScaled`
  - `_calculateLPSharePrice`
  - `_calculateFeesGivenShares`
  - `_calculateFeesGivenBonds`
  - `_convertToBaseFromOption`
  - `_convertToOptionFromBase`
  - `_calculateMaturedProceeds`
  - `calculateOpenLongMaxSpotPrice`
  - `calculateCloseShortMaxSpotPrice`
  - `calculateOpenLong`
  - `calculateCloseLong`
  - `calculateOpenShort`
  - `calculateNegativeInterestOnClose`
  - `calculateUpdateLiquidity`
  - `calculateMaxBuySharesIn`
  - `calculateMaxBuyBondsOut`
  - `calculateMaxSellBondsInSafe`
  - `calculateNetFlatTrade`
  - `calculateDistributeExcessIdleShareProceedsNetLongEdgeCaseSafe`
  - `calculateSharesOutGivenBondsInDerivativeSafe`
  - `calculateSharesInGivenBondsOutDerivativeSafe`
- `up` suffix:
  - `calculateCloseShort` **
  - `calculateMaxBuyBondsOutDerivativeSafe`

***Note**: Most of these utility functions use `pow` so their rounding direction might not be entirely accurate.*

DELV: Acknowledged.

Spearbit: Acknowledged.

5.5.3 The use of `_convertToBaseFromOption` may be unnecesary and needs more testing

Severity: Informational

Context: HyperdriveAdmin.sol#L18, HyperdriveBase.sol#L589

Description: `_convertToBaseFromOption` is a helper function that depending on `options.asBase`, will return the same amount or `mulDown` by `_vaultSharePrice`. The comment in `_collectGovernanceFee` regarding the return value, `/// @return proceeds The amount collected in units specified by _options.`, may seem to imply that the return value, before being returned, is converted through `_convertToBaseFromOption` to accomplish it. However, the return value doesn't include any call to `_convertToBaseFromOption` while the emit in the same function does.

Actually, removing all the instances of `_convertToBaseFromOption` from the code, or adding `_convertToBaseFromOption` call for the return value in `_collectGovernanceFee`, didn't provide any change in the foundry tests (all the tests worked), which points towards:

- There is a gap in the tests / code as it's not properly validating that the output of functions where `_convertToBaseFromOption` is used in the return value is converted or keep at it is.

and / or:

- The function may be redundant in the current code, as the conversion it's already being done in other intermediary functions.

Recommendation: Add more testing on the use of `_convertToBaseFromOption` and remove if it's unnecesary, as it will provide gas optimizations: `(Overall gas change: -63549830 (-0.530%))`

DELV: Acknowledged. The point of the function is to convert a unit that is either in base or shares to base using the options.asBase value. With this in mind, it's not redundant, and we need to use it to convert quantities like proceeds, which have ambiguous units to base for events. We acknowledge the issue with the test coverage (this is realistically covered in the testing done on the frontend and in the fuzzing/simulation infrastructure, but our unit tests could be better), but we're not going remove the function.

Spearbit: Acknowledged.

5.5.4 `isInstances` can be spammed

Severity: Informational

Context: HyperdriveFactory.sol#L584

Description: `deployAndInitialize` can be callable by anyone. The function pushes to the `isInstances` array, and there is no way of removing or reordering any of the elements. For example, as a correct deployment happens, it can be followed by tons of dummy deployments between legitimate calls. In fact, a couple of bad UX situations may arise due to:

- `getInstancesInRange` may return a lot of dummy information that may be need to filtered later.
- If the amount pushed between elements to obtain is high enough, it can deny on-chain integrations.

Recommendation: Consider adding access control or validation and documenting this in `getInstancesInRange`. Another option would be to avoid using ranges and use a batch version with the indexes of the elements to obtain as the input.

DELV: Acknowledged. It costs around ~15 million gas to make a full deployment. At a gas price of 25 gwei and the current ETH price, a full deployment will cost around $1000. With this in mind, we're not going to make a fix for this.

Spearbit: Acknowledged.

5.5.5 Rounding errors

Severity: Informational

Context: HyperdriveBase.sol#L105, HyperdriveBase.sol#L488, HyperdriveBase.sol#L564

Description: Throughout the contracts we've noticed various rounding errors.

- LPMath.sol#L84-L102: In case we want to err on the side of slightly higher effective share reserves, the positive `shareAdjustment` should round down and the negative up.
- HyperdriveBase.sol#L105: The `timeRemaining` is rounding down in all the instances but in fact it depends for the portion used when we are trading on the flat curve or the curve. When we trade on the flat we should round up and when we trade on the curve we should round down. Consider creating 2 functions `_calculateTimeRemainingDown` and `_calculateTimeRemainingUp` and use them accordingly. The same would apply for the `_calculateTimeRemainingScaled`.
- HyperdriveBase.sol#L488, HyperdriveBase.sol#L564: We should overestimate the `curveFee` and the `flatFee`, the governance fee rounding down is acceptable as the governance wants to capture the lower bound but the curve/flat fees should round up to not be in user's favor.

Recommendation: Consider fixing the aforementioned rounding errors.

DELV: Fixed in PR 815. We excluded the fixes to `calculateTimeRemaining` since there are some drawbacks to doing the rounding discussed in the issue. We'll revisit this.

Spearbit: Verified.

5.5.6 Improper documentation for `Zombie Interest`

Severity: Informational

Context: Documentation

Description: The team has chosen to implement a fix for any interest that will accrue for long/short positions that aren't closed after maturity and that accrue interest until they are fully closed. The fix is called `Zombie Interest` and is not properly documented within the whitepaper/code or official documentation. The only documentation we could find was in a pull request (see PR 721).

Recommendation: Consider properly document this feature.

DELV: Acknowledged.

Spearbit: Acknowledged.

5.5.7 Change the direction of roundings in `_isSolvent` to impose a stricter inequality

Severity: Informational

Context: HyperdriveBase.sol#L270-L279

Description/Recommendation: Change the direction of roundings in `_isSolvent` to impose a stricter inequality:

```
int256((uint256(_marketState.shareReserves).mulDown(_vaultSharePrice))) -
↪  int128(_marketState.longExposure)
>=
int256(_minimumShareReserves.mulUp(_vaultSharePrice));
```

DELV: Fixed in pr 779.

Spearbit: Verified.

5.5.8 `calculateTimeStretch` can be simplified

Severity: Informational

Context: HyperdriveMath.sol#L29

Description: In calculateTimeStretch when one wants to calculate the `timeStretch` for position durations other than 1 year, one uses the following formula:

Below $A = \frac{\mu z_e}{y}$

$$(1 + at_1)A^{t_{s,0}} = 1$$

$$(1 + at_2)A^{t_{s,1}} = 1$$

$$t_{s,1} = \frac{\ln(\frac{1}{1+at_2})}{\ln((\frac{1}{1+at_1})^{1/t_{s,0}})} = \left(\frac{ln(1+at_2)}{ln(1+at_1)}\right.$$

$t_{s,0}$

parameter	description
$t_{s,0}$	initial `timeStretch` for the position duration t_1
$t_{s,1}$	time stretch for a custom `_positionDuration` equal to t_2

In the above derivation we need to make sure when the liquidity pool is initialised the scaled ratio of the reserves would equal to A.

Recommendation: Use the more simplified formula to calculate $t_{s,1}$:

$$\left(\frac{ln(1+at_2)}{ln(1+at_1)}\right.$$

$t_{s,0}$

DELV: Fixed in PR 795.

Spearbit: Verified.

5.5.9 In `calculateDistributeExcessIdleShareProceedsNetLongEdgeCaseSafe` the `presentValueParams` curve data are updated in memory

Severity: Informational

Context: LPMath.sol#L833-L838

Description: In `calculateDistributeExcessIdleShareProceedsNetLongEdgeCaseSafe` the `presentValueParams` curve data are updated in memory (z, ζ, y) to $(z_{og}, \zeta_{og}, y_{og})$, although it is not needed in `calculateNetFlatTrade`. The effect is that these memory positions are updated but not read anymore during the execution since after the call to `calculateDistributeExcessIdleShareProceedsNetLongEdgeCaseSafe`, `calculateDistributeExcessIdleShareProceeds` returns which in turn `calculateDistributeExcessIdle` returns:

```
// Calculate the amount of withdrawal shares that should be redeemed
// and their share proceeds.
(uint256 withdrawalSharesRedeemed, uint256 shareProceeds) = LPMath
    .calculateDistributeExcessIdle(
        _getDistributeExcessIdleParams(
            idle,
            withdrawalSharesTotalSupply,
            _vaultSharePrice
        )
    );

// Update the withdrawal pool's state.
_withdrawPool.readyToWithdraw += withdrawalSharesRedeemed.toUint128();
_withdrawPool.proceeds += shareProceeds.toUint128();

// Remove the withdrawal pool proceeds from the reserves.
_updateLiquidity(-int256(shareProceeds));
```

the generated `_getDistributeExcessIdleParams(...)` not used afterwards.

Recommendation: It would be best to leave a note that these memory updates are not needed or remove them if their update would not affect any future plans/modifications:

```
- _params.presentValueParams.shareReserves = _params
-     .originalShareReserves;
- _params.presentValueParams.shareAdjustment = _params
-     .originalShareAdjustment;
- _params.presentValueParams.bondReserves = _params.originalBondReserves;
```

DELV: Fixed in PR 794.

Spearbit: Verified.

5.5.10 `calculateShares...GivenBonds...DerivativeSafe` can be refactored

Severity: Informational

Context: LPMath.sol#L1239, LPMath.sol#L1376

Description: Both:

- `calculateSharesOutGivenBondsInDerivativeSafe` and
- `calculateSharesInGivenBondsOutDerivativeSafe`

use the same formula except that one adds `_bondAmount` to `bondReserves` and the other subtracts in the calculations.

$$\frac{d\beta(z,\zeta,y)}{dz} = \left(1 - \frac{\zeta_{og}}{z_{og}}\right) 1 - \frac{1}{c}\frac{\mu}{c}(K - (y+\Delta y)^{1-t_s})$$

$$\frac{t_s}{1-t_s} c(\mu(z-\zeta))^{-t_s} + y^{-t_s} \frac{y_{og}}{z_{og}-\zeta_{og}}\Bigg) - (y+\Delta y)^{-t_s} \frac{y_{og}}{z_{og}-\zeta_{og}}\Bigg)$$

where K is:

$$K = \frac{c}{\mu}(\mu(z-\zeta))^{1-t_s} + y^{1-t_s}$$

and Δy can also be negative.

Recommendation: To avoid potential mistakes in the implementations it would be best to combine these two functions into one `calculateDeltaSharesGivenDeltaBondsDerivativeSafe` of the following forms:

- let `_bondAmount` be an `int256` and update `bondReserves` accordingly or
- pass an operation as one of the inputs to the function so that we would know how to update `bondReserves` based on the provided `_bondAmount`:

```
function calculateDeltaSharesGivenDeltaBondsDerivativeSafe(
    DistributeExcessIdleParams memory _params,
    uint256 _originalEffectiveShareReserves,
    uint256 _bondAmount,
    function (uint256, uint256) internal pure returns (uint256) addOrSubtract
) internal pure returns (uint256, bool) {
```

DELV: This is a fantastic suggestion! See the fix in PR 822.

Spearbit: Verified.

5.5.11 In `calculateNetCurveTradeSafe` round in the direction that would result in returning a smaller value

Severity: Informational

Context: LPMath.sol#L320-L331, LPMath.sol#L260-L271

Description/Recommendation: In `calculateNetCurveTradeSafe` when the `netCurvePosition` is negative or the pool is net short, we would want to use `calculateSharesInGivenBondsOutDown` instead of `calculateSharesInGivenBondsOutUp` when calculating the `netCurveTrade`:

```
- .calculateSharesInGivenBondsOutUp(
+ .calculateSharesInGivenBondsOutDown(
```

to underestimate its value, so ultimately we would underestimate the present value PV.

Similarly, when the pool is net long we should use `calculateSharesOutGivenBondsInUp` instead of `calculateSharesOutGivenBondsInDown` since we are returning a negated value:

```
- calculateSharesOutGivenBondsInDown
+ calculateSharesOutGivenBondsInUp
```

DELV: Acknowledged. We won't apply the fixes since the intention is for calculateNetCurveTrade to simulate closing all of the trades as accurately as possible. Changing the rounding behavior to be more conservative would break this assumption since we'd be using different rounding to close trades in practice than we do with the present value calculation. The documentation was updated to reflect our intentions in PR 793.

Spearbit: Acknowledged.

5.5.12 Use `divUp` in `shouldShortCircuitDistributeExcessIdleShareProceeds` when calculating `lpSharePriceBefore`

Severity: Informational

Context: LPMath.sol#L890-L892

Description/Recommendation:

It would be best to use `divUp` in this context when calculating the unadjusted price of LP shares since we are comparing:

$$\frac{PV_0}{s_{LP} + s_w} \leq \frac{PV_1}{s_{LP}} \leq \frac{PV_0}{s_{LP} + s_w}(1 + \epsilon)$$

To be more conservative one can actually do:

1. For the first inequality uses `divUp` for `lpSharePriceBefore` and `divDown` for `lpSharePriceAfter`.
2. For the second inequality use `{div, mul}Down` for all the operations to calculate $\frac{PV_0}{s_{LP} + s_w}(1 + \epsilon)$ and `divUp` for `lpSharePriceAfter`. This part might not be required as it can kind of her incorporated into the `SHARE_PROCEEDS_MIN_TOLERANCE` (ϵ).

DELV: Fixed in PR 792 using both recommendations.

Spearbit: Verified.

5.5.13 The implementation of `exp` and `ln` are not verified

Severity: Informational

Context: FixedPointMath.sol#L134, FixedPointMath.sol#L204

Description: The implementations of `exp` and `ln` are not verified due to a lack of time for the engagement. Specifically, the derivation of the rational approximations used in these functions has not been checked.

Recommendation: It would be best to verify the implementation of these functions to make sure that they behave as expected.

DELV: This implementation is copied from Remco's SolExp library and almost identical to the calculation used by Solmate in their fixed point math library. This code has been audited several times, and we are comfortable with the results given our testing.

Spearbit: Acknowledged.

5.5.14 The functions `pow`, `exp` and `ln` are not monotonic

Severity: Informational

Context: FixedPointMath.sol#L101, FixedPointMath.sol#L134, FixedPointMath.sol#L204

Description: The functions `pow`, `exp`, and `ln` are not monotonic.

For example, basically $\ln(x)$ is approximated as:

$$\left\lfloor \frac{A \cdot \operatorname{sgn}(\frac{p(\bar{x})}{q(\bar{x})}) \left\lfloor \left| \frac{p(\bar{x})}{q(\bar{x})} \right| \right\rfloor + B(\lfloor \log_2(x) \rfloor) + C}{D} \right\rfloor$$

where $A, B, C, D \in \mathbb{N}$ and $p, q \in \mathbb{Z}[x]$ and $\bar{x}$ is the normalised x.

$$\bar{x} = \left\lfloor \frac{x \cdot 2^{96}}{2^{\lfloor \log_2(x) \rfloor}} \right\rfloor$$

1. $\bar{x}$ as a function of x is not monotone (not order-preserving).
2. The rational approximation term which is multiples to A is not necessarily monotone.

Thus in general one **cannot** expect that if $x_1 \leq x_2$ we would have $f(x_1) \leq f(x_2)$ (or the other way around for non-increasing functions).

The codebase has multiple `if` / `else` branching logic when it comes to using `pow` and it is assumed / commented that `pow` is a non-decreasing function:

```
if (inner >= ONE) {
    // Rounding down the exponent results in a smaller result. <--- not true in general
    inner = inner.pow(X.divDown(Y));
} else {
    // Rounding up the exponent results in a smaller result. <--- not true in general
    inner = inner.pow(X.divUp(Y));
}
```

Recommendation: Special care needs to be taken when one uses these functions or try to reason about them in different inequalities as the direction of the inequality might not always be the direction one expects. One can also try to come up with different implementations that are order-preserving.

DELV: We're aware of this limitation and don't find that it imposes a significant risk (outside of any higher severity issues mentioned). We aren't going to fix this.

Spearbit: Acknowledged.

5.5.15 `toString` fails for large numbers

Severity: Informational

Context: AssetId.sol#L110-L113

Description: The max string length for a `uint256` should be: `ceil(log_10(2 ^ 256 - 1)) = ceil(77.06) = 78`. But the contract uses `maxStringLength = 77`. Calling `toString` with high values will fail.

Recommendation: Consider using `maxStringLength = 78`.

DELV: Fixed in PR 801.

Spearbit: Verified.

5.5.16 Net-short case always uses `calculateSharesInGivenBondsOut` derivative

Severity: Informational

Context: LPMath.sol#L742-L745

Description: In the net-short case of `calculateDistributeExcessIdleShareProceeds`, Newton's method is used with the $net_c^{s'}(x) = z'_{in}(x, y_s \cdot t_s - y_l \cdot t_l)$ derivative. However, as mentioned in the documentation and code, this derivative may only be used if $y_s \cdot t_s - y_l \cdot t_l \leq y_{out}^{max}(x)$. Otherwise, $z_{in}^{max'}(x) - y_{out}^{max'}(x)$ should be used.

Recommendation: Document why $y_s \cdot t_s - y_l \cdot t_l \leq y_{out}^{max}(x)$ is always true in this scenario, or if it's not, consider using the other case instead.

DELV: Fixed in PR 806.

Spearbit: Verified. The code now restricts `dz` to `dz_max` which by definition ensures that the net short curve trade can be closed after removing `dz_max` liquidity. I don't see any new issues arising from capping the newton iterations at `dz_max`. Worst case, it might not terminate (with the given max iteration count) but we always have this issue.

5.5.17 Properly document the return values of the derivative functions

Severity: Informational

Context: LPMath.sol#L1239, LPMath.sol#L1376

Description: The derivation for the calculations is given here. Referring to this notation:

- Currently, `calculateSharesOutGivenBondsInDerivativeSafe` returns $net_c^{l'}(x) = -z'_{out}(x, y_l \cdot t_l - y_s \cdot t_s)$, instead of just z'_{out} as the name would suggest.
  - The function's Natspec and the returned value should match. Consider changing the function to return the negated value so it matches the current Natspec (`-(1 - zeta / z) * ...`) and z'_{out}. Alternatively, rename the function and change the Natspec to `(1 - zeta / z) * ...` such that it matches the return value.
  - (the derivative we use is 1 minus this derivative so this rounds the derivative up).

    The above comment refers to many different "derivatives" and it's unclear what they refer to. It could be `z_out'`, the return value of this function, or the derivative `F'(x)` that the caller uses for the Newton method. Consider being more explicit about what is being referred to.
- Document that `calculateSharesInGivenBondsOutDerivativeSafe` should return $net_c^{s'}(x) = z'_{in}(x, y_s \cdot t_s - y_l \cdot t_l)$. The Natspec and returned value currently match.
  - (the derivative we use is 1 minus this derivative so this rounds the derivative up).

    The above comment refers to many different "derivatives" and it's unclear what they refer to. It could be `z_in'`, or the derivative `F'(x)` that the caller uses for the Newton method. Consider being more explicit about what is being referred to.

Recommendation: Consider implementing the mentioned changes.

DELV: Fixed in PR 811.

Spearbit: Verified.

5.5.18 Integrating new ERC4626 vaults should check for read-only reentrancy issues

Severity: Informational

Context: ERC4626Base.sol#L139-L141

Description: Hyperdrive supports generic ERC4626 vaults with the `ERC4626Hyperdrive` contract deployments. The vault share price is read in its `_pricePerVaultShare()` function. It might be possible that this value can be temporarily inflated in the vault, for example, through read-only reentrancy or adjacent issues where the vault grants a callback in the middle of some function and it leaves the protocol in a state that temporarily overestimates the price until the end of its execution. One could, for example, create a checkpoint at an inflated price.

Recommendation: There is no generic solution to prevent against this as it's an issue of the underlying vault. One should check for these issues when adding support for a new ERC4626. A common fix against these issues is checking that the vault's reentrancy guards are not set in `_pricePerVaultShare()`.

Furthermore, the `ERC4626` Hyperdrive supports only standard `ERC4626` implementations. If we think it from ERC20 perspective, there are non-standard ERC20s that always cause issues, the same should apply to the ERC4626 as for example a transfer tax implementation would not work as expected. Consider documenting the fact that the ERC4626 Hyperdrive supports only standard ERC4626 implementations.

DELV: PR 777 contains the requested documentation.

Spearbit: Verified.

5.5.19 Signature replay risk in case of a hard fork

Severity: Informational

Context: Hyperdrive.sol#L126-L135, ERC20Forwarder.sol#L54-L65

Description: Using an immutable `DOMAIN_SEPARATOR` creates a vulnerability for replay attacks of all future signatures from the root chain being replayable in the forked chain due to the static `chainId` stored in `DOMAIN_SEPARATOR`.

Recommendation: To mitigate this case, one can use the battle-tested (or a similar implementation of) OpenZeppelin EIP712.sol contract. This option fixes this by dynamically generating the `DOMAIN_SEPARATOR` in case the current `chainId` or `address(this)` are not the same as the cached ones. This change would prevent replay attacks by ensuring the `DOMAIN_SEPARATOR` reflects the actual chain context.

DELV: Fixed in PR 804.

Spearbit: Verified.

5.5.20 Governance can block any ongoing deployment by changing storage parameters

Severity: Informational

Context: HyperdriveFactory#L924-L926

Description: Deployments done in `HyperdriveFactory` are done via a multi-step process. If an admin changes any of the related storage variables during a deployment, it will revert. This is due to the check regarding the config hash being the same in all stages. Therefore, an admin can DoS any deployment.

Recommendation: Consider commenting this case and maybe applying some solution, like adding some sort of mapping that stores in-progress deployments and therefore avoiding this DoS case.

DELV: Acknowledged. We are aware of this and the idea is just to document it. In practice, Governance is timelocked, so users will have the opportunity to see these parameter changes coming for several days and can schedule around them.

Spearbit: Acknowledged.

5.5.21 Minting destination can be `address(0)`, locking assets in the process

Severity: Informational

Context: HyperdriveMultitoken.sol#L137

Description: Unlike `_transferFrom` and the batch version of `_transferFrom`, `_mint` doesn't check if the destination is `address(0)`, therefore it could accidentally mint these tokens to an unreachable address by the user in case `options.destination` isn't set properly. Actually, the code mints initially to zero address by decision:

```
// Mint the minimum share reserves to the zero address as a buffer that
// ensures that the total LP supply is always greater than or equal to
// the minimum share reserves. The initializer will receive slightly
// less shares than they contributed to cover the shares set aside as a
// buffer on the share reserves and the shares set aside for the zero
// address, but this is a small price to pay for the added security
// in practice.
```

While this is indeed good for a part of the security, this initial amount can also be sent to another burn address, as `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`, to ensure the invariant that "*LP supply is always greater or equal to the minimum share reserves*" and therefore, being able to add a safety check for default zero address.

Luckily, `ERC20Forwarder` can use their superpowers and transfer from any address on desire, rescuing the assets. However, without taking into account this exceptional case, this would lead to a temporal lock of funds and the need of the `ERC20Forwarder`'s intervention each time a user makes this mistake.

Recommendation: Consider adding safety checks for zero address on value transfer and send the initial mint to `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE`.

DELV: Fixed in PR 816.

Spearbit: Verified.

5.5.22 Following CEI pattern when posible aligns with best security practices

Severity: Informational

Context: HyperdriveFactory.sol#L678-L683

Description: `HyperdriveFactory.sol` performs a refund call of the excess value that could be performed at the very end of the function, avoiding possible reentrancy issues. While there haven't been any direct reentrancy exploits identified, adhering to the Checks-Effects-Interactions (CEI) pattern is a good practice.

Recommendation: Move the refund logic to the very end of the contract.

DELV: Fixed in PR 805.

Spearbit: Verified.

5.5.23 Use of OpenZeppelin's `EnumerableSet` can simplifiy common logic

Severity: Informational

Context: HyperdriveFactory.sol#L123-L127, HyperdriveFactory.sol#L136-L139

Description: The current implementation for managing `_deployerCoordinators` and `_instances`, along with their respective checks (`isDeployerCoordinator` and `isInstance`), relies on manual array management and mapping for existence checks. This design can be simplified using a battle-tested mechanism like OpenZeppelin's `EnumerableSet`.

Recommendation: Consider refactoring the `_deployerCoordinators` and `_instances` variables to use OpenZeppelin's `EnumerableSet`. This library offers functions for adding, removing, and checking the existence of elements in a set, alongside iteration capabilities.

DELV: Acknowledged. The team has decided to not implement this due to time-constraints and low impact of the change.

Spearbit: Acknowledged.

5.5.24 `CollectGovernanceFee` event misses useful information

Severity: Informational

Context: HyperdriveAdmin.sol#L41-44

Description: `CollectGovernanceFee` event tracks the destination address of the fee and the amount of proceeds withdrawn. However, the current implementation includes the following variable, but not who initiated the collection (there are 3 role types, one an array of addresses), which can be useful for monitoring in case of issues.

Recommendation: Consider adding a new initiator parameter and renaming `collector` into `recipient` (which is less ambiguous):

```
- event CollectGovernanceFee(address indexed collector, uint256 fees);
+ event CollectGovernanceFee(address indexed initiator, address indexed recipient, uint256 fees);
```

DELV: Acknowledged. We aren't going to address it since we don't believe the initiator is a necessary field of this event.

Spearbit: Acknowledged.

5.5.25 `pauser` role can `collectFees` and may lead to unexpected behaviors

Severity: Informational

Context: HyperdriveAdmin.sol#L29

Description: In the current implementation, the pauser role has the ability to collect fees, a function that is not really aligned with a common pauser role. This mix of powers can lead to potential security risks and unexpected behavior.

Recommendation: Consider separating the fee collection and pausing functionalities into distinct roles to mitigate the risks associated with role concentration and to better align with the principle of least privilege.

DELV: Acknowledged. We aren't planning on fixing it. This is an intentional design choice, and as you say, the fees are always transferred to the fee collector.

Spearbit: Acknowledged.

5.5.26 Common and duplicated logic can be simplified to improve maintainability and readability

Severity: Informational

Context: HyperdriveMultiToken.sol#L51-L53, HyperdriveMultiToken.sol#L82-L85, StETHBase.sol#L63, StETHBase.sol#L73

Description: In general, some new features from the Hyperdrive contracts contain instances where similar logic or patterns are replicated, leading to code duplication. Additionally, some contracts exhibit complex implementations that could benefit from simplification. Code duplication not only increases the maintenance effort but also raises the risk of inconsistencies and bugs during updates or modifications.

Recommendation: Implement code de-duplication and simplification strategies. This can be achieved by identifying common functionalities and extracting them into internal functions, shared libraries or base contracts, which can then be reused across the codebase. Here are some examples of changes that can be performed:

- HyperdriveMultiToken.sol#L51-L53, HyperdriveMultiToken.sol#L82-L85: `if (from == address(0) || to == address(0))` can be an internal function or modifier.
- StETHBase.sol#L63, StETHBase.sol#L73: `vaultSharePrice = _pricePerVaultShare();` can be moved outside of the `if`/`else` block as it is being performed at the very end of both with no difference.

DELV: The second recommendation was followed in PR 803. The first recommendation was acknowledged.

Spearbit: Acknowledged. Partially fixed.

5.5.27 Naming can be more consistent

Severity: Informational

Context: StETHTarget0.sol#L28, StETHTarget1.sol#L21, StETHTarget2.sol#L21, StETHTarget3.sol#L21, StETHTarget4.sol#L21, LPMath.sol#L156, HyperdriveBase.sol#L156

Description: Various naming of parameters/functions can be more clear or consistent:

- In ERC4626Target0.sol#L28: there is a declaration of `__vault` rather than `_vault` to avoid shadowing the Base contract `_vault` definition, however `StETHTarget` contracts use the `_lido` variable for both the Base and Target contracts, leading to (a harmless) shadowing. Rename `_lido` instances to `__lido` to avoid shadowing and keep naming consistency.
- LPMath.sol#L156: the custom error name is not accurate as `success` can be false due to `calculateNetCurveTradeSafe` returning `success == false`.
- HyperdriveBase.sol#L156: the long exposure usually represents the bonds, here it is transformed into shares, so it would be better to rename this variable to `longExposureShares` to avoid any confusion.

Recommendation: Consider fixing the aforementioned naming inconsistencies.

DELV: Fixed in PR 821. Last suggestion was not accepted as `longExposure` is more readable and that the units are clear in context.

Spearbit: Verified.

5.5.28 Index can go out of range

Severity: Informational

Context: HyperdriveFactory.sol#L827, HyperdriveFactory.sol#L787

Description: The functions `getDeployerCoordinatorsInRange` and `getInstancesInRange` have some checks to verify `endIndex` is within the bounds to iterate over `_instances` and `_deployerCoordinators`. However, the actual check allows `endIndex` to be equal to the array length, and if this happens, it will result in an out-of-bounds access as indices are 0-based and the last valid index is `length - 1`.

Recommendation: Modify the conditional checks to ensure `endIndex` does not exceed array length - 1. For example, implementing `endIndex >= _instances.length` as the condition will prevent out-of-bounds errors and ensure safe access to array elements.

Element Fixed in PR 791.

Spearbit Verified.

5.5.29 A common EIP712 interface can be declared to avoid duplication of code

Severity: Informational

Context: IMultiTokenMetadata.sol#L6-L9, IERC20Forwarder.sol#L34-L38

Description: `DOMAIN_SEPARATOR()` and `PERMIT_TYPEHASH()` are both part of `IMultiTokenMetadata` and `IERC20Forwarder` interfaces. This implies code duplication of interfaces and can affect maintainability, as both can inherit from a common EIP712 interface and any change or comment would only affect a single file.

Recommendation: Consider creating some interface for EIP712 commons that encapsulates the `DOMAIN_SEPARATOR()` and `PERMIT_TYPEHASH()` functions. Both `IMultiTokenMetadata` and `IERC20Forwarder` can inherit from it to enhance maintainability.

DELV: Acknowledged.

Spearbit: Acknowledged.

5.5.30 Single-step `governance` transfer can be risky

Severity: Informational

Context: HyperdriveAdmin.sol#L69, HyperdriveRegistry.sol#L31

Description: The code introduces a function to set the governance address in a single step and without default-value checks. Single-step governance transfers add the risk of setting an unwanted governance by accident if the governance transfer is not done with excessive care it can be lost forever.

Recommendation: Consider employing a 2 step admin/governance transfer mechanisms for this critical ownership change where first, a pending admin/governance address is declared, and if that address doesn't accept the role you can choose another one or cancel it. This would reduce the risk significantly. Some examples are Open Zeppelin's `Ownable2Step` or Synthetic's `Owned`.

DELV: Acknowledged. Hyperdrive is intended to be used with a governance system like Council. With this in mind, these governance systems already include several sanity checks including a long voting period, timelocks, and veto power by a governance council. With all of this in mind, I would disagree that this is a low-severity issue. It seems more like an informational issue since it would primarily impact Hyperdrive systems that are improperly configured.

Spearbit: Acknowledged.

5.5.31 Events lacking useful information affect monitoring

Severity: Informational

Context: HyperdriveAdmin.sol#L84, HyperdriveLP.sol#L291

Description: Events are key for external applications and monitoring the use of the system. In some cases, key variables are missing. For example, when updating `_pauser[who]` status, no event is emitted regarding which status it is updated to, therefore unexpected scenarios may arise.

Also, if we redeem shares in the `_removeLiquidity`, there isn't a `RedeemWithdrawalShares` event emitted. It is advisble to be consistent with the events emitted under certain flows. One can do `lpAmount - withdrawalShareAmount` to get the actual redeemed shares though.

Recommendation: Consider the following:

- Add the status which the pauser is updated to.
- Add an explicit event for the `RedeemWithdrawalShares` within the remove liquidity function.

DELV: Here is the fix PR for the first recommendation: PR 802.

We aren't going to address the second recommendation since the information is already contained within the `RemoveLiquidity` event.

Spearbit: Acknowledged.

5.5.32 Missing/wrong comments and typos affect readability

Severity: Informational

Context: FixedPointMath.sol#L201, FixedPointMath.sol#L216-L225, FixedPointMath.sol#L227, LPMath.sol#L407, LPMath#L1050, HyperdriveBase.sol#L242, HyperdriveBase#L382, HyperdriveLP#L202, HyperdriveLong#L55, HyperdriveLong#L209, StETHHyperdriveCoreDeployer.sol#L19, StETHTarget3Deployer.sol#L19, StETHTarget1Deployer.sol#L19, StETHTarget2Deployer.sol#L19, StETHTarget0Deployer.sol#L19, StETHTarget4Deployer.sol#L19, ERC4626Base.sol#L138, StETHBase.sol#L59, StETHTarget0.sol#L46, ERC4626Target1.sol#L11, Hyperdrive.sol#L86-87, HyperdriveTarget2.sol#L35-36, HyperdriveTarget4.sol#L38-39, HyperdriveFactory.sol#L130, IHyperdriveGovernedRegistry.sol#L18, IMultiTokenCore.sol#L22, IMultiTokenRead.sol#L5, AssetId.sol#L85-L90, HyperdriveBase#L496 , YieldSpaceMath#322, LPMath.sol#528, HyperdriveShort.sol#L453, HyperdriveBase.sol#L524, ERC4626Target0.sol#L37

Description: Comments help to provide context and documentation on what different functions, contracts and variables do. Providing clear and precise comments is key to a clean and maintainable codebase. See below a list of related nitpicks:

- **Missing comments:**
  - FixedPointMath.sol#L216-L225: A reference to ilog2 implementation should be included.
  - YieldSpaceMath#322: The `calculateMaxBuyBondsOut` has a comment about rounding, consider adding one for `calculateMaxBuySharesIn` as well.
  - HyperdriveBase.sol#L524: NatSpec is missing for `totalGovernanceFee`.
- **Typos:**
  - LPMath.sol#L407: `algorith` should be `algorithm`.
  - HyperdriveBase#L382: `zombhie` should be `zombie`.
  - HyperdriveLP#L202: `.T` should be `. T`.
  - HyperdriveLong#L55: `mininum` should be `minimum`.
  - HyperdriveLong#L209: `be` should be `by`.

  - StETHHyperdriveCoreDeployer.sol#L19, StETHTarget3Deployer.sol#L19, StETHTarget1Deployer.sol#L19, StETHTarget2Deployer.sol#L19, StETHTarget0Deployer.sol#L19, StETHTarget4Deployer.sol#L19: `Instanstiates` should be `Instantiates`.
  - StETHTarget0.sol#L46: `teth` should be `the`.
  - Hyperdrive.sol#L86-87: `contains all /// some stateful` should be `contains stateful`.
  - HyperdriveFactory.sol#L130: `coordintor` should be `coordinator`.
  - LPMath.sol#L528: the comment states that the ending present value should be greater but it's actually greater or equal.
- **Wrong comment:**
  - FixedPointMath.sol#L227, it reduces the range to `[1, 2) * 2**96`. Note the inclusion of `1 * 2**96`.
  - LPMath#L1050: is a copy pasted comment from if block, should be less rather than greater.
  - StETHBase.sol#L59: Mentions another token than the one used; WETH should be ETH.
  - ERC4626Base.sol#L138: Data provider is no longer a component.
  - HyperdriveTarget2.sol#L35-36: `short` should be `long`.
  - HyperdriveTarget4.sol#L38-39: `long` should be `short`.
  - AssetId.sol#L85-L90: comment and function name suggest the naming of the variable is copy pasted from previous function, should be `_symbol` not `_name`.
  - HyperdriveBase#L496: comment stays `curve_fee * p * phi_gov` but the calculation doesn't include spot price `governanceCurveFee = curve_fee * phi_gov`.
  - LPMath.sol#L1507: NatSpec says `... k(x) / ((c / mu) + 1) ...` but should just be `k(x)`.
  - HyperdriveShort.sol#L453: the comment should be `We remove the governance fee from the share reservers`.
  - ERC4626Target0.sol#L37: mentions `transferFrom` selector with no instances of `transferFrom` but transfer in the function.
- **Incomplete comment:**
  - HyperdriveBase.sol#L242: `were purchased above price of 1 base per bonds.` should be `were purchased above the price of 1 base per bond`.
  - FixedPointMath.sol#L201: should include `or zero`.
  - ERC4626Target1.sol#L11: misses `This contract contains several stateful functions that couldn't fit into the Hyperdrive contract`.
  - IHyperdriveGovernedRegistry.sol#L18: `set` should be `to set`.
- **NatSpec:**
  - IMultiTokenCore.sol#L22: Missing NatSpec.
  - IMultiTokenRead.sol#L5: Missing 3 `@params`.

Recommendation: Improve comments all over the code, correct typos, remove stale comments and fix incorrect comments where posible.

DELV: Fixed in PR 820.

Spearbit: Verified.

5.5.33 Favor constants over "magic numbers"

Severity: Informational

Context: FixedPointMath.sol#L46, FixedPointMath.sol#L87, FixedPointMath.sol#L94, HyperdriveDeployerCoordinator.sol#L345 L348

Description: While in certain parts of the code constants are used to represent some values as `1e18`, in others these constants although declared, are unused.

Recommendation: Define or import constant variables as needed rather than using magic numbers in order to enhance code clarity and maintainability.

DELV: Fixed in PR 778.

Spearbit: Verified.

5.5.34 Use custom errors consistently

Severity: Informational

Context: HyperdriveMath.sol#L153

Description: The current error handling in the smart contracts utilizes custom errors, except at this instance:

```
require(effectiveShareReserves >= 0);
```

Which lacks an error string, which would aid monitoring.

Recommendation: Add a proper error for this instance and use custom errors consistently when reverting.

DELV: Fixed in PR 787.

Spearbit: Verified.

5.5.35 Unnecessary type casts can be avoided

Severity: Informational

Context: HyperdriveLP.sol#L409-L410, HyperdriveBase.sol#L288, LPMath.sol#L88, AssetId.sol#L119, AssetId.sol#L125, AssetId.sol#L81, AssetId.sol#L100, HyperdriveBase.sol#L253

Description: Casting a data type to the same type (for example, a `uint256` into a `uint256`) or a compatible one (in some cases) is unnecessary and only makes the code less clear.

In other cases like HyperdriveLong.sol#L253, it's first defined as `uint128` and then casted to `uint256`, which can be done in a single step as in HyperdriveLong.sol#L338.

Recommendation: Remove unnecessary casting where relevant.

DELV: I'm not sure what HyperdriveBase.sol#L253 was supposed to be linked to. Let me know if I missed something there, but it just links to a comment. Casting to `uint128` around HyperdriveLong.sol#L253 is actually what we want to do since we later set the state using this value.

Aside from those comments, I addressed all of the other unnecessary casts that I found in PR 781.

Spearbit: Verified.

5.5.36 Unused code increases deployment cost and decreases maintainability

Severity: Informational

Context: HyperdriveBase.sol#L4, HyperdriveStorage.sol#L8, ERC4626Base.sol#L8, ERC4626Hyperdrive.sol#L10-L11, Hyperdrive.sol#L8, HyperdriveRegistry.sol#L4

Description: There are several libraries that were left behind after some updates and are not used anymore. Unused code increases the overall complexity of the codebase, making it harder to maintain and read while also having potential implications on gas costs, as it contributes to larger contract sizes which affect deployment costs.

Recommendation: Remove unnecessary code to improve readability and deployment gas costs.

DELV: Fixed in PR 823.

Spearbit: Verified.