SHERLOCK

SHERLOCK SECURITY REVIEW FOR

PERENNIAL

Prepared For:	Perennial
Prepared By:	Sherlock
Lead Security Expert:	WatchPug
Dates:	December 21st - 30th, 2022

Introduction

“Perennial is a cash-settled synthetic derivatives protocol. It allows developers to launch any synthetic market with just a few lines of code.”

This report is a follow-up security review for Perennial Protocol that was prepared by WatchPug from December 21st - 30th, 2022.

Scope

Scope: **202212 Sherlock Audit Items**where commits are frozen as of December 20th, 2022.

Protocol Info

Language: Solidity

Blockchain: Ethereum
L2s: None

Tokens used: USDC, DSU, Reward ERC20 tokens

Findings

Each issue has an assigned severity:

- Informational issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgement as to whether to address such issues.
- Low issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Informational	Low	Medium	High
4	0	5	0

Issue M-01

BalancedVault may unable to rebalance when makerLimit was decreased

Summary

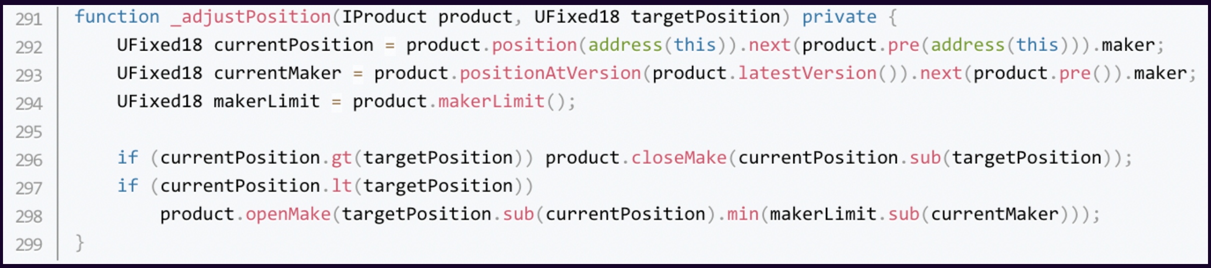
In _adjustPosition(), it is expected that makerLimit will always be greater than currentMaker. That should be the case in normal circumstances. However, the makerLimit can be changed by the productOwner at anytime. When the makerLimit is changed to a lower value, _adjustPosition will revert at L298 due to underflow.

Severity

Medium

Issue Detail

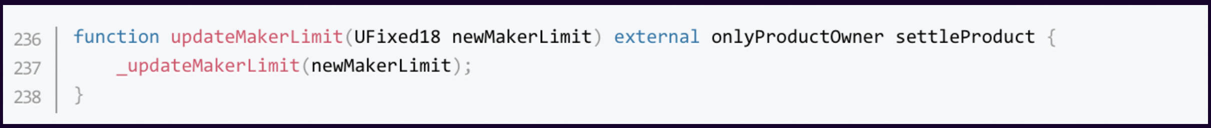
https://github.com/equilibria-xyz/perennial-mono/blob/3d2c5f16fb4f65f25ecf1f2cb2a5f89448415beb/packages/perennial-vaults/contracts/BalancedVault.sol#L291-L299



```
291  function _adjustPosition(IProduct product, UFixed18 targetPosition) private {
292      UFixed18 currentPosition = product.position(address(this)).next(product.pre(address(this))).maker;
293      UFixed18 currentMaker = product.positionAtVersion(product.latestVersion()).next(product.pre()).maker;
294      UFixed18 makerLimit = product.makerLimit();
295  
296      if (currentPosition.gt(targetPosition)) product.closeMake(currentPosition.sub(targetPosition));
297      if (currentPosition.lt(targetPosition))
298          product.openMake(targetPosition.sub(currentPosition).min(makerLimit.sub(currentMaker)));
299  }
```

The makerLimit can be changed by the productOwner at anytime:

https://github.com/equilibria-xyz/perennial-mono/blob/3d2c5f16fb4f65f25ecf1f2cb2a5f89448415beb/packages/perennial/contracts/product/UParamProvider.sol#L236-L238



```
236  function updateMakerLimit(UFixed18 newMakerLimit) external onlyProductOwner settleProduct {
237      _updateMakerLimit(newMakerLimit);
238  }
```

Tool used

Manual Review

Perennial Comment

We are not including the Vault in this release as it needs a rewrite mainly due to issue M2.

Issue M-02

BalancedVault sophisticated active users may deposit or withdraw at the last minute before the new oracle version to escape loss or snatch profits

Severity

Medium

Issue Detail

The PNL of a position is settled every time a new price is posted.

The BalancedVault is designed to be split 50/50 between long and short, but the actual effective position sizes can be different. Therefore, every new price comes with a profit or loss.

When there is a significant amount of profit added to the holdings by the update, a sophisticated user can monitor the price update very closely, maybe even frontrun the price update transaction and deposit to the BalancedVault, and exit right after the update.

By doing so, the user would be able to take a portion of the profit.

Vice versa, if there is an upcoming price movement that causes a loss to the BalancedVault, the user would be able to escape the loss.

The root cause for this issue is that the strategy is asynchronous, but the vault is synchronous.

The pending PNL of the asynchronous strategy is predictable, which creates an opportunity for active users to use this knowledge against inactive users.

Perennial Comment

We are not including the Vault in this release as it needs a rewrite mainly due to issue M2.

Issue M-03 (previously M-05)

ChainlinkFeedOracle.sol#sync() will revert when there are multiple phases

Severity

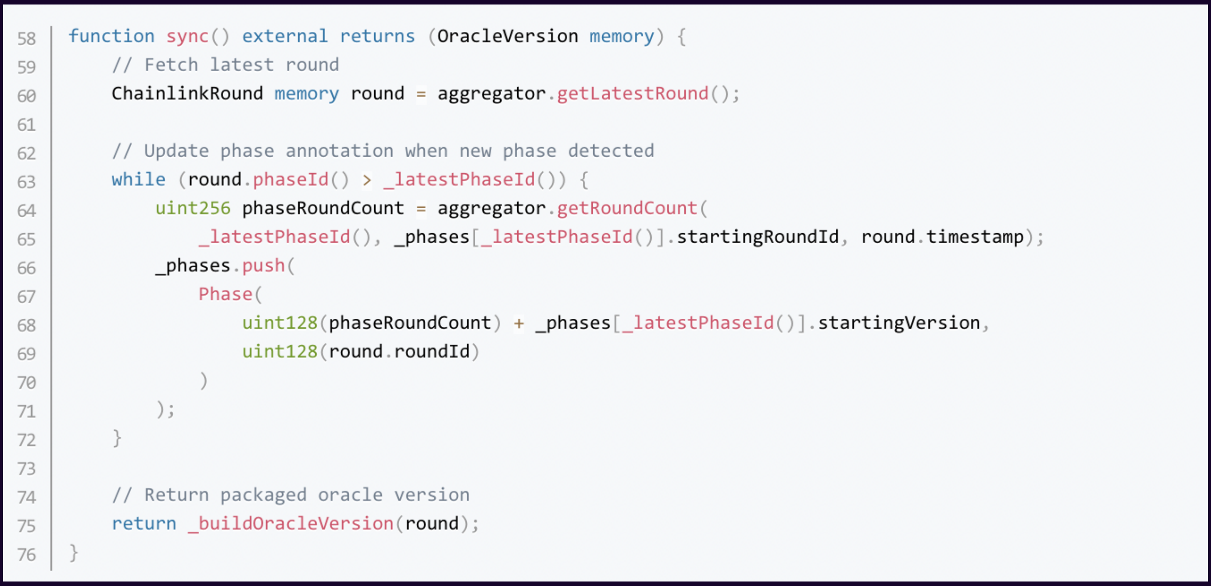
Medium

Issue Detail

When there are multiple new phases at L63, they will all start with round.roundId.

As a result, sync() → getRoundCount() will revert, as the startingRoundId of the last phase for the next phase can be larger than its latestRoundId.

https://github.com/equilibria-xyz/perennial-mono/blob/e801b6eecae6ca609597710d2980bd26184a2ef8/packages/perennial-oracle/contracts/ChainlinkFeedOracle.sol#L58-L76



```
58  function sync() external returns (OracleVersion memory) {
59      // Fetch latest round
60      ChainlinkRound memory round = aggregator.getLatestRound();
61
62      // Update phase annotation when new phase detected
63      while (round.phaseId() > _latestPhaseId()) {
64          uint256 phaseRoundCount = aggregator.getRoundCount(
65              _latestPhaseId(), _phases[_latestPhaseId()].startingRoundId, round.timestamp);
66          _phases.push(
67              Phase(
68                  uint128(phaseRoundCount) + _phases[_latestPhaseId()].startingVersion,
69                  uint128(round.roundId)
70              )
71          );
72      }
73
74      // Return packaged oracle version
75      return _buildOracleVersion(round);
76  }
```

https://github.com/equilibria-xyz/perennial-mono/blob/e801b6eecae6ca609597710d2980bd26184a2ef8/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L59-L76

```solidity
function getRoundCount(ChainlinkAggregator self, uint16 phaseId, uint256 startingRoundId, uint256 maxTimestamp)
internal view returns (uint256) {
    AggregatorProxyInterface proxy = AggregatorProxyInterface(ChainlinkAggregator.unwrap(self));
    AggregatorV2V3Interface agg = AggregatorV2V3Interface(proxy.phaseAggregators(phaseId));

    (uint80 aggRoundId,,,uint256 updatedAt,) = agg.latestRoundData();

    // If the latest round for the aggregator is after maxTimestamp, walk back until we find the
    // correct round
    while (updatedAt > maxTimestamp) {
        aggRoundId--;
        (,,,updatedAt,) = agg.getRoundData(aggRoundId);
    }

    // Convert the aggregator round to a proxy round
    uint256 latestRoundId = _aggregatorRoundIdToProxyRoundId(phaseId, aggRoundId);
    return uint256(latestRoundId - startingRoundId + 1);
}
```

Proof of Concept

Given:

Last phase: 3

Last phase startingRoundId: (uint256(3) << 64) + 1

Last phase latestRoundId: (uint256(3) << 64) + 3

Last phase startingVersion: 10

Current phase: 5

Current aggregator roundId: 1

Current proxy roundId: (uint256(5) << 64) + 1

L63, the first iterate:

```
Phase[4] = (
    13, // 3 + 10
    (uint256(5) << 64) + 1
)
```

The second iterate, L65 startingRoundId: (uint256(5) << 64) + 1;

ChainlinkAggregator.sol#L74, latestRoundId: (uint256(4) << 64) + 1;

ChainlinkAggregator.sol#L75, latestRoundId - startingRoundId will revert.

Recommendation

Consider adding a parameter startingRoundIds and validate them:

```
uint256 currentPhaseId = _latestPhaseId() + 1;
require(startingRoundIds[i] >> 64 == currentPhaseId);
// If empty phase, latestAggregatorRoundId will be 0
if (startingRoundIds[i] << 64 == 0) {
    AggregatorV2V3Interface agg = AggregatorV2V3Interface(aggregator.phaseAggregators(currentPhaseId));
    (uint80 aggRoundId,,,,) = agg.latestRoundData();
    require(aggRoundId == 0);
} else {
    (, , uint256 previousOracleUpdateTimestamp, , ) = aggregator.getRoundData(startingRoundIds[i]);
    require(previousOracleUpdateTimestamp > 0);
    // Previous round data must not exist
    (, , uint256 previousOracleUpdateTimestamp, , ) = aggregator.getRoundData(startingRoundIds[i] - 1);
    require(previousOracleUpdateTimestamp == 0);
}
```

Perennial Comment
Acknowledged.

Partial Fix: https://github.com/equilibria-xyz/perennial-mono/pull/110

Our fix here is for us to explicitly revert if trying to sync more than 1 phase in a `sync`. Identifying the starting round ID for intermediary phases is something we will try to fix in our next oracle update, but for now we will rely on off-chain mechanisms to update the rounds soon after the Chainlink update occurs.

WatchPug Comment
Let's mark [original] M5 and S1 as acknowledged, and we can sign off on this audit.

Issue M-04 (previously M-06)

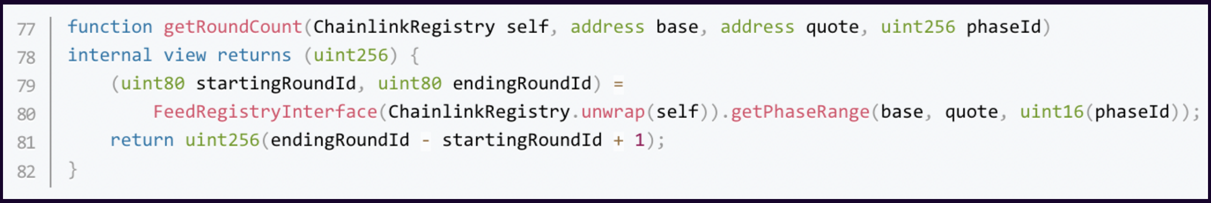
Improper handling of empty phases

Severity
Medium

Issue Detail
For an empty phase, both the endingRoundId and startingRoundId will be 0.

And the roundCount should be 0 instead of 1.

https://github.com/equilibria-xyz/perennial-mono/blob/e801b6eecae6ca609597710d2980bd26184a2ef8/packages/perennial-oracle/contracts/types/ChainlinkRegistry.sol#L77-L82



```
77  function getRoundCount(ChainlinkRegistry self, address base, address quote, uint256 phaseId)
78  internal view returns (uint256) {
79      (uint80 startingRoundId, uint80 endingRoundId) =
80          FeedRegistryInterface(ChainlinkRegistry.unwrap(self)).getPhaseRange(base, quote, uint16(phaseId));
81      return uint256(endingRoundId - startingRoundId + 1);
82  }
```

When empty phases are involved, the result of getRoundCount() will be incorrect, and therefore OracleVersion will also be incorrect.

Recommendation
The case of endingRoundId == 0 should be specially treated, because when a past phase's endingRoundId is 0, it means that it is an empty phase.

Perennial Comment
Acknowledged and fixed (https://github.com/equilibria-xyz/perennial-mono/pull/112).

WatchPug Comment
Fix confirmed.

Issue M-05 (previously M-02 in fix review)

ChainlinkAggregator.sol#getRoundCount() should return 0 for empty phases

Severity

Medium

Issue Detail

https://github.com/equilibria-xyz/perennial-mono/blob/e801b6eecae6ca609597710d2980bd26184a2ef8/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L50-L76

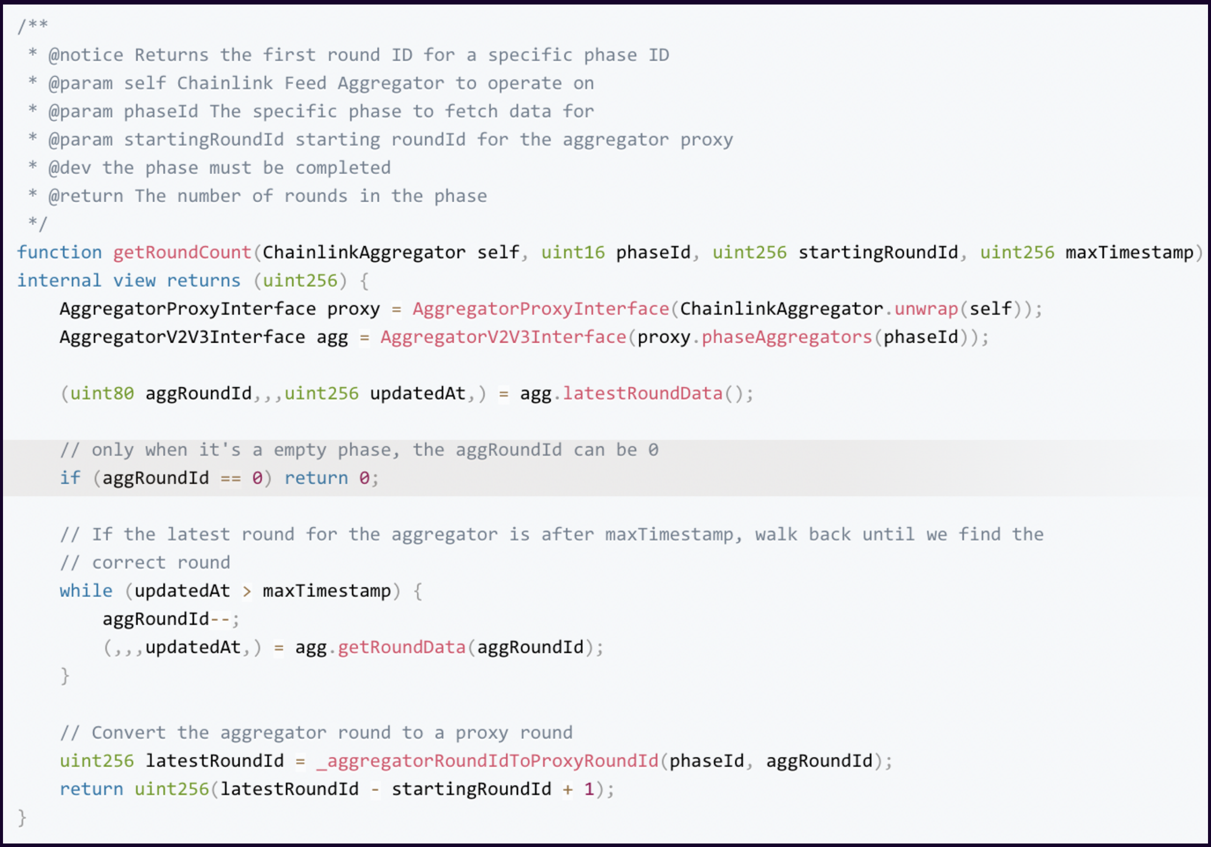
```
/**
 * @notice Returns the first round ID for a specific phase ID
 * @param self Chainlink Feed Aggregator to operate on
 * @param phaseId The specific phase to fetch data for
 * @param startingRoundId starting roundId for the aggregator proxy
 * @param maxTimestamp maximum timestamp allowed for the last round of the phase
 * @dev Assumes the phase ends at the aggregators latestRound or earlier
 * @return The number of rounds in the phase
 */
function getRoundCount(ChainlinkAggregator self, uint16 phaseId, uint256 startingRoundId, uint256 maxTimestamp)
internal view returns (uint256) {
    AggregatorProxyInterface proxy = AggregatorProxyInterface(ChainlinkAggregator.unwrap(self));
    AggregatorV2V3Interface agg = AggregatorV2V3Interface(proxy.phaseAggregators(phaseId));

    (uint80 aggRoundId,,,uint256 updatedAt,) = agg.latestRoundData();

    // If the latest round for the aggregator is after maxTimestamp, walk back until we find the
    // correct round
    while (updatedAt > maxTimestamp) {
        aggRoundId--;
        (,,,updatedAt,) = agg.getRoundData(aggRoundId);
    }

    // Convert the aggregator round to a proxy round
    uint256 latestRoundId = _aggregatorRoundIdToProxyRoundId(phaseId, aggRoundId);
    return uint256(latestRoundId - startingRoundId + 1);
}
```

Recommendation



```solidity
/**
 * @notice Returns the first round ID for a specific phase ID
 * @param self Chainlink Feed Aggregator to operate on
 * @param phaseId The specific phase to fetch data for
 * @param startingRoundId starting roundId for the aggregator proxy
 * @dev the phase must be completed
 * @return The number of rounds in the phase
 */
function getRoundCount(ChainlinkAggregator self, uint16 phaseId, uint256 startingRoundId, uint256 maxTimestamp)
internal view returns (uint256) {
    AggregatorProxyInterface proxy = AggregatorProxyInterface(ChainlinkAggregator.unwrap(self));
    AggregatorV2V3Interface agg = AggregatorV2V3Interface(proxy.phaseAggregators(phaseId));

    (uint80 aggRoundId,,,uint256 updatedAt,) = agg.latestRoundData();

    // only when it's a empty phase, the aggRoundId can be 0
    if (aggRoundId == 0) return 0;

    // If the latest round for the aggregator is after maxTimestamp, walk back until we find the
    // correct round
    while (updatedAt > maxTimestamp) {
        aggRoundId--;
        (,,,updatedAt,) = agg.getRoundData(aggRoundId);
    }

    // Convert the aggregator round to a proxy round
    uint256 latestRoundId = _aggregatorRoundIdToProxyRoundId(phaseId, aggRoundId);
    return uint256(latestRoundId - startingRoundId + 1);
}
```

Perennial Comment

Fixed: https://github.com/equilibria-xyz/perennial-mono/pull/115

Although this appears to be an unreachable state since in the current implementation, startingRoundId cannot be 0 and therefore the phase can't be empty.

WatchPug Comment

Fix confirmed.

Issue I-01 (previously I-07)

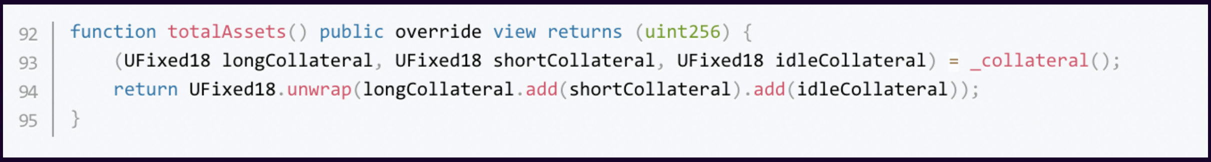
BalancedVault.sol unsettled PNL can cause inaccuracy in many functions

Severity

Informational

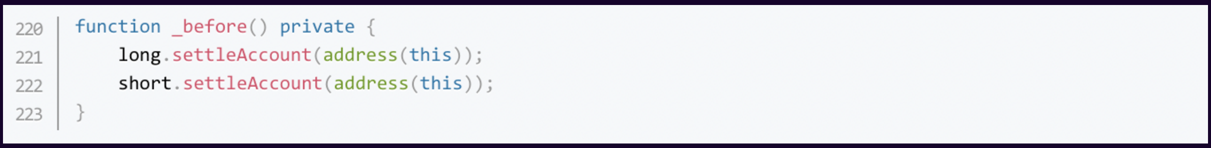
Issue Detail

https://github.com/equilibria-xyz/perennial-mono/blob/3d2c5f16fb4f65f25ecf1f2cb2a5f89448415beb/packages/perennial-vaults/contracts/BalancedVault.sol#L92-L95



```
function totalAssets() public override view returns (uint256) {
    (UFixed18 longCollateral, UFixed18 shortCollateral, UFixed18 idleCollateral) = _collateral();
    return UFixed18.unwrap(longCollateral.add(shortCollateral).add(idleCollateral));
}
```

https://github.com/equilibria-xyz/perennial-mono/blob/3d2c5f16fb4f65f25ecf1f2cb2a5f89448415beb/packages/perennial-vaults/contracts/BalancedVault.sol#L220-L223



```
function _before() private {
    long.settleAccount(address(this));
    short.settleAccount(address(this));
}
```

If there is unsettled PNL, BalancedVault.sol#totalAssets() can deviate from the actual total amount of the underlying asset.

Other functions including previewDeposit(), previewMint(), previewWithdraw(), and previewRedeem() will be inaccurate due to unsettled PNL as well.

Perennial Comment

We are not including the Vault in this release as it needs a rewrite mainly due to issue M2.

Issue I-02 (previously G-08)

In BalancedVault.sol#_adjustPosition() avoiding an unnecessary external call can save gas

Severity

Informational

Issue Detail

https://github.com/equilibria-xyz/perennial-mono/blob/3d2c5f16fb4f65f25ecf1f2cb2a5f89448415beb/packages/perennial-vaults/contracts/BalancedVault.sol#L286-L299

```
/**
 * @notice Adjusts the position on `product` to `targetPosition`
 * @param product The product to adjust the vault's position on
 * @param targetPosition The new position to target
 */
function _adjustPosition(IProduct product, UFixed18 targetPosition) private {
    UFixed18 currentPosition = product.position(address(this)).next(product.pre(address(this))).maker;
    UFixed18 currentMaker = product.positionAtVersion(product.latestVersion()).next(product.pre()).maker;
    UFixed18 makerLimit = product.makerLimit();

    if (currentPosition.gt(targetPosition)) product.closeMake(currentPosition.sub(targetPosition));
    if (currentPosition.lt(targetPosition))
        product.openMake(targetPosition.sub(currentPosition).min(makerLimit.sub(currentMaker)));
}
```

makerLimit is only needed when L297 is true. Therefore, it should be moved below L297.

```
/**
 * @notice Adjusts the position on `product` to `targetPosition`
 * @param product The product to adjust the vault's position on
 * @param targetPosition The new position to target
 */
function _adjustPosition(IProduct product, UFixed18 targetPosition) private {
    UFixed18 currentPosition = product.position(address(this)).next(product.pre(address(this))).maker;
    UFixed18 currentMaker = product.positionAtVersion(product.latestVersion()).next(product.pre()).maker;

    if (currentPosition.gt(targetPosition)) product.closeMake(currentPosition.sub(targetPosition));
    if (currentPosition.lt(targetPosition)) {
        UFixed18 makerLimit = product.makerLimit();
        product.openMake(targetPosition.sub(currentPosition).min(makerLimit.sub(currentMaker)));
    }
}
```

Perennial Comment

We are not including the Vault in this release as it needs a rewrite mainly due to issue M2.

Issue I-03 (previously N-09)

Wrong/Misleading Natspec comments

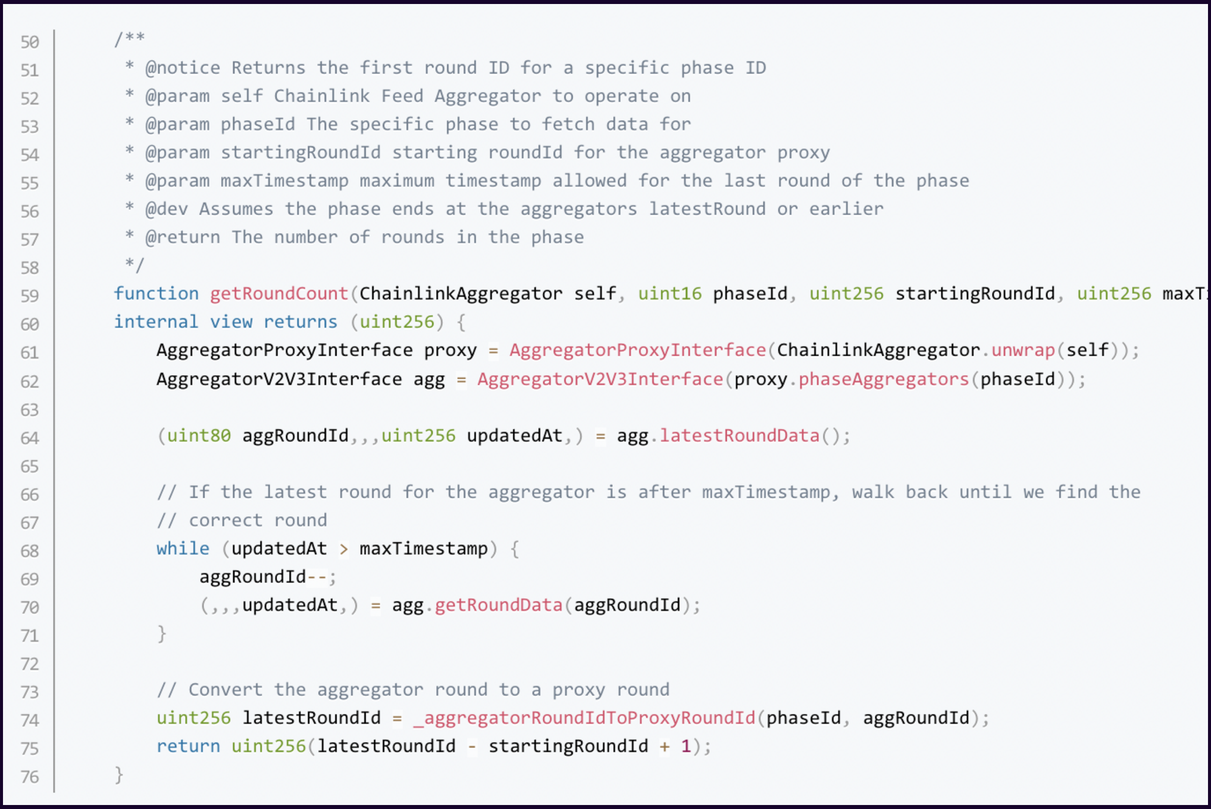
Severity

Informational

Issue Detail

Returns the first round ID for a specific phase ID → Returns the quantity of rounds for a specific phase ID

https://github.com/equilibria-xyz/perennial-mono/blob/e801b6eecae6ca609597710d2980bd26184a2ef8/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L50-L76



```
    /**
     * @notice Returns the first round ID for a specific phase ID
     * @param self Chainlink Feed Aggregator to operate on
     * @param phaseId The specific phase to fetch data for
     * @param startingRoundId starting roundId for the aggregator proxy
     * @param maxTimestamp maximum timestamp allowed for the last round of the phase
     * @dev Assumes the phase ends at the aggregators latestRound or earlier
     * @return The number of rounds in the phase
     */
    function getRoundCount(ChainlinkAggregator self, uint16 phaseId, uint256 startingRoundId, uint256 maxT
    internal view returns (uint256) {
        AggregatorProxyInterface proxy = AggregatorProxyInterface(ChainlinkAggregator.unwrap(self));
        AggregatorV2V3Interface agg = AggregatorV2V3Interface(proxy.phaseAggregators(phaseId));

        (uint80 aggRoundId,,,uint256 updatedAt,) = agg.latestRoundData();

        // If the latest round for the aggregator is after maxTimestamp, walk back until we find the
        // correct round
        while (updatedAt > maxTimestamp) {
            aggRoundId--;
            (,,,updatedAt,) = agg.getRoundData(aggRoundId);
        }

        // Convert the aggregator round to a proxy round
        uint256 latestRoundId = _aggregatorRoundIdToProxyRoundId(phaseId, aggRoundId);
        return uint256(latestRoundId - startingRoundId + 1);
    }
```

Perennial Comment

Fixed: https://github.com/equilibria-xyz/perennial-mono/pull/111

Issue I-04 (previously S-01)

An alternative implementation of ChainlinkFeedOracle.sol#sync() to handle multiple phases gracefully

Severity

Informational (Auditor suggestion)

Issue Detail

Given that it is possible for the actual startingRoundId of a phase to be larger than 1 (when the first new aggregator rounds were posted before the aggregator was set as the new aggregator on the proxy).

And the actual endingRoundId on the proxy can be less than the latestRoundId on the phase aggregator (when the last new aggregator rounds were posted after the aggregator was overridden by a new aggregator on the proxy).

We believe it is possible to accurately get the actual startingRoundId and endingRoundId in such cases.

The current implementation relies on the sync() function to be called shortly after the phase change.

If the sync() function is called after a few rounds in the new phase, those rounds will be discarded.

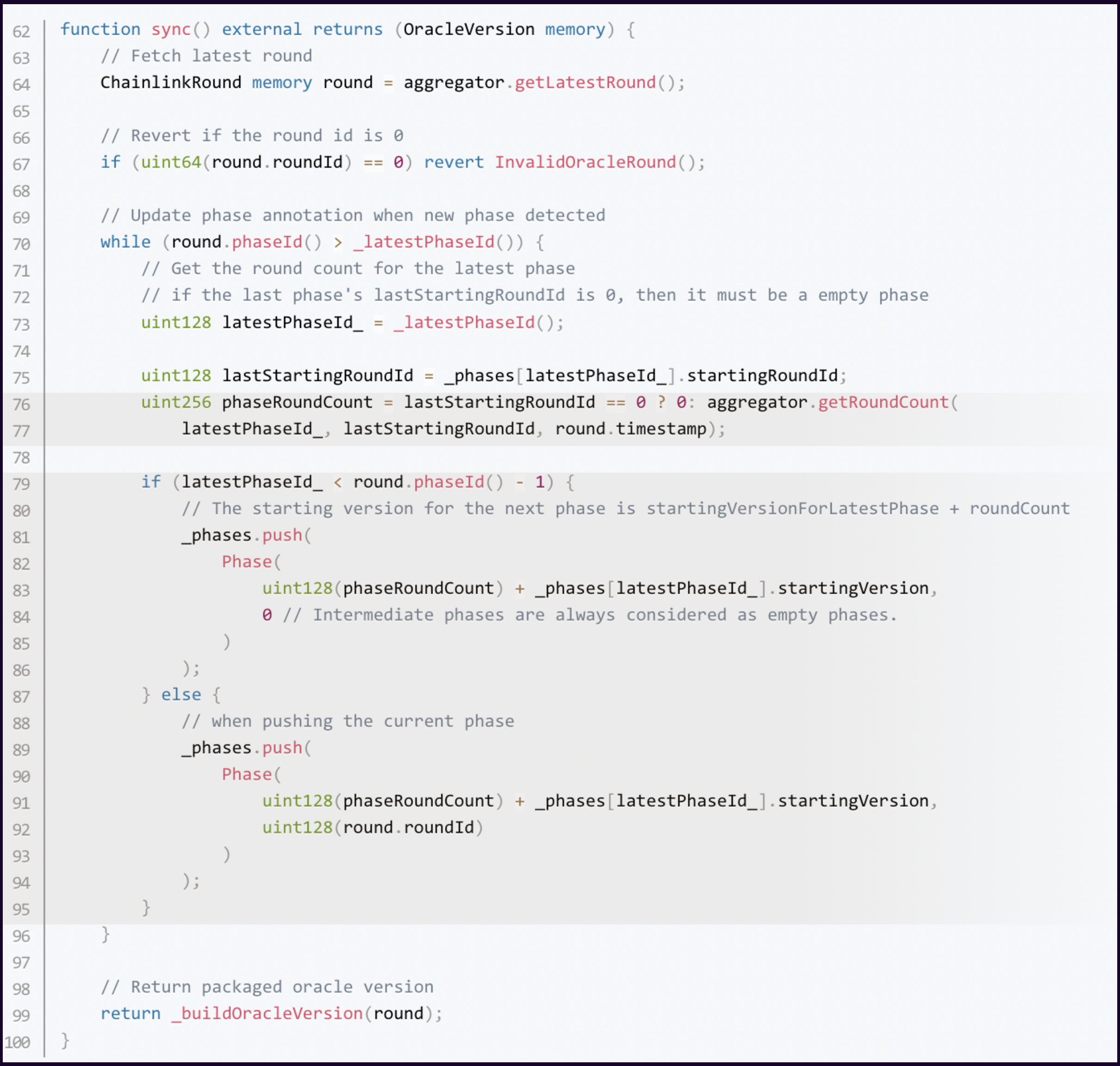
Therefore, we recommend that you consider the following alternative implementation. It will consider all missed phases as empty phases and discard all rounds if they exist.

The advantages of this implementation are that it will remain effective even if there are multiple phases between the current phase and the last known phase.

The current implementation will revert, and it cannot sync() until the oracle address is updated.

This effectively freezes everyone's funds in the contract, as it can not _settle() anymore.

Recommendation



```
function sync() external returns (OracleVersion memory) {
    // Fetch latest round
    ChainlinkRound memory round = aggregator.getLatestRound();

    // Revert if the round id is 0
    if (uint64(round.roundId) == 0) revert InvalidOracleRound();

    // Update phase annotation when new phase detected
    while (round.phaseId() > _latestPhaseId()) {
        // Get the round count for the latest phase
        // if the last phase's lastStartingRoundId is 0, then it must be a empty phase
        uint128 latestPhaseId_ = _latestPhaseId();

        uint128 lastStartingRoundId = _phases[latestPhaseId_].startingRoundId;
        uint256 phaseRoundCount = lastStartingRoundId == 0 ? 0: aggregator.getRoundCount(
            latestPhaseId_, lastStartingRoundId, round.timestamp);

        if (latestPhaseId_ < round.phaseId() - 1) {
            // The starting version for the next phase is startingVersionForLatestPhase + roundCount
            _phases.push(
                Phase(
                    uint128(phaseRoundCount) + _phases[latestPhaseId_].startingVersion,
                    0 // Intermediate phases are always considered as empty phases.
                )
            );
        } else {
            // when pushing the current phase
            _phases.push(
                Phase(
                    uint128(phaseRoundCount) + _phases[latestPhaseId_].startingVersion,
                    uint128(round.roundId)
                )
            );
        }
    }

    // Return packaged oracle version
    return _buildOracleVersion(round);
}
```

Perennial Comment

Even with this solution there is an edge case where the last synced version for a phase is the last round in that phase. If we then miss a phase, the next synced version will be incorrectly marked as lastSyncedVersion + 1 (even though there were intermediary rounds). Instead of trying to introduce extra logic to handle these edge cases we'd like to solve the general case in a new algorithm to deploy as part of the next audit.