# Moonwell

audited.xyz
by zack.eth

Date
**Feb 28, 2026 08:18 UTC**

Commit
**`17f7f0e`**
44 files, 6,551 lines

## Summary

The Moonwell Oracle codebase is generally well-structured with appropriate use of Chainlink oracle integration patterns, OEV capture mechanisms, and bounded composite oracles. The OEV wrappers implement proper round-delay mechanisms to capture MEV value for the protocol, and the bounded composite oracle provides sensible fallback logic for derivative asset pricing.

The most notable issue is the `ChainlinkCompositeOracle` returning `block.timestamp` instead of actual oracle update timestamps, which masks price staleness from downstream consumers. This could allow stale prices to be used for lending/borrowing decisions affecting derivative assets (wstETH, cbETH, rETH, weETH, LBTC, etc.). Additionally, there is an inconsistency in reentrancy protection between the core market and Morpho OEV wrappers, and missing stale price checks in the base `ChainlinkOracle`.

## Findings

4 issues identified

● 4 Low

L-1 LOW Open

### ChainlinkCompositeOracle masks oracle staleness by returning block.timestamp

`src/oracles/ChainlinkCompositeOracle.sol:56`

#### Description

The latestRoundData() function returns block.timestamp as the updatedAt value instead of the actual minimum timestamp from underlying Chainlink feeds. Any downstream staleness check will always see the price as fresh, even when underlying feeds are stale. This affects derivative assets including wstETH, cbETH, rETH, weETH, wrsETH, and LBTC.

#### Impact

Stale prices from underlying Chainlink feeds could be used for lending/borrowing decisions without detection, potentially enabling users to supply overvalued collateral, borrow against stale prices, or avoid proper liquidation.

#### Recommendation

Return the minimum updatedAt timestamp from all underlying oracle feeds instead of block.timestamp.

#### Fix

Returns the minimum updatedAt timestamp from all underlying oracle feeds instead of block.timestamp, allowing downstream consumers to accurately detect when any underlying feed is stale.

`src/oracles/ChainlinkCompositeOracle.sol`

```diff
-    function latestRoundData()
-        external
-        view
-        returns (
-            uint80, /// roundId always 0, value unused in ChainlinkOracle.sol
-            int256, /// the composite price
-            uint256, /// startedAt always 0, value unused in ChainlinkOracle.sol
-            uint256, /// always block.timestamp
-            uint80 /// answeredInRound always 0, value unused in ChainlinkOracle.sol
-        )
-    {
-        if (secondMultiplier == address(0)) {
-            /// if there is only one multiplier, just use that
-            return (
-                0,
-                /// fetch uint256, then cast back to int256, this cast to uint256 is a sanity check
-                /// that chainlink did not return a negative value
-                getDerivedPrice(base, multiplier, decimals).toInt256(),
-                0,
-                block.timestamp, /// return current block timestamp
-                0
-            );
-        }
-
-        /// if there is a second multiplier apply it
-        return (
-            0, /// unused
-            getDerivedPriceThreeOracles(
-                base,
-                multiplier,
-                secondMultiplier,
-                decimals
-            ).toInt256(),
-            0, /// unused
-            block.timestamp, /// return current block timestamp
-            0 /// unused
-        );
-    }
+    function latestRoundData()
+        external
+        view
+        returns (
+            uint80, /// roundId always 0, value unused in ChainlinkOracle.sol
+            int256, /// the composite price
+            uint256, /// startedAt always 0, value unused in ChainlinkOracle.sol
+            uint256, /// minimum updatedAt from underlying feeds
+            uint80 /// answeredInRound always 0, value unused in ChainlinkOracle.sol
+        )
+    {
+        if (secondMultiplier == address(0)) {
+            (, , , uint256 baseUpdatedAt, ) = AggregatorV3Interface(base).latestRoundData();
+            (, , , uint256 multUpdatedAt, ) = AggregatorV3Interface(multiplier).latestRoundData();
+            uint256 minTimestamp = baseUpdatedAt < multUpdatedAt ? baseUpdatedAt : multUpdatedAt;
+            /// if there is only one multiplier, just use that
+            return (
+                0,
+                /// fetch uint256, then cast back to int256, this cast to uint256 is a sanity check
+                /// that chainlink did not return a negative value
+                getDerivedPrice(base, multiplier, decimals).toInt256(),
+                0,
+                minTimestamp, /// return minimum updatedAt from underlying feeds
+                0
+            );
+        }
+
+        (, , , uint256 baseUpdatedAt, ) = AggregatorV3Interface(base).latestRoundData();
+        (, , , uint256 multUpdatedAt, ) = AggregatorV3Interface(multiplier).latestRoundData();
+        (, , , uint256 secMultUpdatedAt, ) = AggregatorV3Interface(secondMultiplier).latestRoundData();
+        uint256 minTimestamp = baseUpdatedAt < multUpdatedAt ? baseUpdatedAt : multUpdatedAt;
+        minTimestamp = minTimestamp < secMultUpdatedAt ? minTimestamp : secMultUpdatedAt;
+
+        /// if there is a second multiplier apply it
+        return (
+            0, /// unused
+            getDerivedPriceThreeOracles(
+                base,
+                multiplier,
+                secondMultiplier,
+                decimals
+            ).toInt256(),
+             0, /// unused
+             minTimestamp, /// return minimum updatedAt from underlying feeds
+             0 /// unused
+         );
+     }
```

L-2 LOW Open

### Incomplete oracle validation in ChainlinkOEVMorphoWrapper._getLoanTokenPrice

src/oracles/ChainlinkOEVMorphoWrapper.sol:388

#### Description

The _getLoanTokenPrice function only checks loanAnswer > 0 but does not validate updatedAt != 0 or answeredInRound >= roundId. The equivalent function in ChainlinkOEVWrapper properly calls _validateRoundData with all checks. A stale loan token price affects the collateral split calculation in updatePriceEarlyAndLiquidate.

#### Impact

A stale loan token price could cause incorrect collateral split between liquidator and protocol fee recipient during OEV liquidations through the Morpho wrapper.

#### Recommendation

Add full round data validation using _validateRoundData, matching the pattern in ChainlinkOEVWrapper._getLoanTokenPrice.

#### Fix

Replaces the partial validation (only answer > 0) with the full _validateRoundData call that also checks updatedAt != 0 and answeredInRound >= roundId, matching the validation standard used in ChainlinkOEVWrapper.

src/oracles/ChainlinkOEVMorphoWrapper.sol

```
-     /// @notice Get the loan token price from ChainlinkOracle
-     /// @dev Gets the feed for the loan token and scales the price similar to ChainlinkOracle
-     /// @param loanToken The loan token interface
-     /// @return The price scaled to 1e18 and adjusted for token decimals
-     function _getLoanTokenPrice(
-         EIP20Interface loanToken
-     ) private view returns (uint256) {
-         // Get the price feed for the loan token
-         AggregatorV3Interface loanFeed = chainlinkOracle.getFeed(
-             loanToken.symbol()
-         );
-
-         // Get the latest price from the feed
-         (, int256 loanAnswer, , , ) = loanFeed.latestRoundData();
-         require(
-             loanAnswer > 0,
-             "ChainlinkOEVMorphoWrapper: invalid loan token price"
```

```diff
-        );
+    /// @notice Get the loan token price from ChainlinkOracle
+    /// @dev Gets the feed for the loan token and scales the price similar to ChainlinkOracle
+    /// @param loanToken The loan token interface
+    /// @return The price scaled to 1e18 and adjusted for token decimals
+    function _getLoanTokenPrice(
+        EIP20Interface loanToken
+    ) private view returns (uint256) {
+        // Get the price feed for the loan token
+        AggregatorV3Interface loanFeed = chainlinkOracle.getFeed(
+            loanToken.symbol()
+        );
+
+        // Get the latest price from the feed and validate
+        (
+            uint80 roundId,
+            int256 loanAnswer,
+            ,
+            uint256 updatedAt,
+            uint80 answeredInRound
+        ) = loanFeed.latestRoundData();
+        _validateRoundData(roundId, loanAnswer, updatedAt, answeredInRound);
```

L-3 LOW Open

## _validateRoundData inside try block causes uncatchable revert in ChainlinkFeedOEVWrapper

`src/oracles/ChainlinkFeedOEVWrapper.sol:121`

### Description

In latestRoundData(), _validateRoundData is called inside the try block's success handler. The catch block only catches reverts from the external call originalFeed.getRoundData(). If the external call succeeds but the returned data fails validation, the revert propagates up uncaught, causing the entire latestRoundData() to revert. This differs from ChainlinkOEVWrapper which correctly validates only after the loop.

### Impact

If any previous round has data that fails validation (e.g., incomplete round with updatedAt==0), the oracle becomes temporarily unusable, freezing all market operations (supply, borrow, redeem, liquidation) until the maxRoundDelay passes.

### Recommendation

Use soft validation (if-check) inside the loop and reserve the hard revert for the final fallback, matching ChainlinkOEVWrapper's pattern.

### Fix

Replaces the hard-reverting _validateRoundData call inside the loop with a soft if-check. Invalid previous rounds are now skipped instead of causing an uncatchable revert. The hard validation is preserved for the final fallback to the latest round data.

src/oracles/ChainlinkFeedOEVWrapper.sol

```diff
-        for (uint256 i = 0; i < maxDecrements && --startRoundId > 0; i++) {
-            try originalFeed.getRoundData(uint80(startRoundId)) returns (
-                uint80 r,
-                int256 a,
-                uint256 s,
-                uint256 u,
-                uint80 ar
-            ) {
-                _validateRoundData(r, a, u, ar);
-
-                roundId = r;
-                answer = a;
-                startedAt = s;
-                updatedAt = u;
-                answeredInRound = ar;
-                return (roundId, answer, startedAt, updatedAt, answeredInRound);
-            } catch {}
-        }
-
-        // Validate round data if we fall back to the latest price
-        _validateRoundData(roundId, answer, updatedAt, answeredInRound);
+        for (uint256 i = 0; i < maxDecrements && --startRoundId > 0; i++) {
+            try originalFeed.getRoundData(uint80(startRoundId)) returns (
+                uint80 r,
+                int256 a,
+                uint256 s,
+                uint256 u,
+                uint80 ar
+            ) {
+                // Use soft validation inside loop to skip invalid rounds
+                // rather than reverting the entire call
+                if (a > 0 && u != 0 && ar >= r) {
+                    roundId = r;
+                    answer = a;
+                    startedAt = s;
+                    updatedAt = u;
+                    answeredInRound = ar;
+                    return (roundId, answer, startedAt, updatedAt, answeredInRound);
+                }
+            } catch {}
+        }
+
+        // Validate round data if we fall back to the latest price
+        _validateRoundData(roundId, answer, updatedAt, answeredInRound);
```

L-4 LOW Open

### Missing heartbeat/staleness duration check on Chainlink price feeds

</> src/oracles/ChainlinkOracle.sol:89-100

## Description

The `getChainlinkPrice` function in ChainlinkOracle.sol checks that `updatedAt != 0` and `answer > 0`, but does not validate that the price data is recent by comparing `updatedAt` against a maximum staleness threshold (heartbeat). Additionally, it is missing the `answeredInRound >= roundId` check that the OEV wrappers include. The same missing heartbeat check also applies to `ChainlinkCompositeOracle.sol` (`getPriceAndDecimals` at line 167) and `ChainlinkBoundedCompositeOracle.sol` (`_getValidatedOracleData` at line 213). Without a heartbeat check, if a Chainlink feed stops updating (e.g., during network congestion or feed deprecation), the protocol will continue using an arbitrarily stale price, which could allow undercollateralized borrowing or prevent valid liquidations.

## Impact

An attacker could exploit a stale oracle price to borrow against inflated collateral values or avoid liquidation. If a Chainlink feed goes stale during a price crash, positions that should be liquidated will remain open, leading to bad debt accrual for the protocol. Conversely, if the feed goes stale during a price pump, users could be unfairly liquidated based on an outdated lower price.

## Recommendation

Add a configurable heartbeat duration check to all oracle price validation functions. Each feed should have a maximum allowed staleness period (matching the Chainlink feed's heartbeat, e.g., 3600s for most feeds, 86400s for some). Also add the missing `answeredInRound >= roundId` check to `ChainlinkOracle.getChainlinkPrice`.

### Fix

```diff
-    function getChainlinkPrice(
-        AggregatorV3Interface feed
-    ) internal view returns (uint256) {
-        (, int256 answer, , uint256 updatedAt, ) = AggregatorV3Interface(feed)
-            .latestRoundData();
-        require(answer > 0, "Chainlink price cannot be lower than 0");
-        require(updatedAt != 0, "Round is in incompleted state");
+    /// @notice Maximum staleness allowed for a Chainlink price feed
+    uint256 public maxStalePeriod = 86400;
+
+    /// @notice Set the maximum stale period for price feeds
+    /// @param _maxStalePeriod The new max stale period in seconds
+    function setMaxStalePeriod(uint256 _maxStalePeriod) external onlyAdmin {
+        maxStalePeriod = _maxStalePeriod;
+    }
+
+    function getChainlinkPrice(
+        AggregatorV3Interface feed
+    ) internal view returns (uint256) {
+        (uint80 roundId, int256 answer, , uint256 updatedAt, uint80 answeredInRound) = AggregatorV3Interface(feed)
+            .latestRoundData();
+        require(answer > 0, "Chainlink price cannot be lower than 0");
+        require(updatedAt != 0, "Round is in incompleted state");
+        require(answeredInRound >= roundId, "Stale price");
+         require(block.timestamp - updatedAt <= maxStalePeriod, "Chainlink price is stale");
```

## Conclusion

The Moonwell Oracle codebase demonstrates solid security practices overall, with proper Chainlink integration patterns, OEV capture mechanisms, and appropriate access controls. The bounded composite oracle implements sensible fallback logic, and the OEV wrappers correctly delay price updates to capture protocol value.

The primary concern is the `ChainlinkCompositeOracle` masking oracle staleness by returning `block.timestamp`, which affects multiple derivative asset markets. While the direct exploitation path requires a Chainlink feed outage (an unlikely but not unprecedented event), the fix is straightforward and recommended before any increase in TVL for affected markets.

The remaining findings are Low severity and reflect inconsistencies between similar contracts rather than fundamental design flaws. The codebase appears suitable for mainnet deployment after addressing the Medium-severity timestamp masking issue and the identified inconsistencies between the OEV wrapper implementations.