SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Bond
Prepared by:	Sherlock
Lead Security Expert:	xiaoming90
Dates Audited:	March 13 - March 20, 2023
Prepared on:	April 4, 2023

Introduction

Bond Protocol is the next evolution of bonds-as-a-service. Olympus-style bonds have revolutionized the way protocols approach acquiring assets.

Scope

The contracts in-scope for this audit are:

```
src
 bases
|    BondBaseOSDA.sol
|    BondBaseOFDA.sol
|    BondBaseFPA.sol
|    BondBaseOracle.sol
 interfaces
|    AggregatorV2V3Interface.sol
|    IBondOSDA.sol
|    IBondOFDA.sol
|    IBondFPA.sol
|    IBondOracle.sol
|    IBondBatchAuctionFactoryV1.sol
|    IBondBatchAuctionV1.sol
|    IGnosisEasyAuction.sol
 BondFixedExpiryOSDA.sol
 BondFixedTermOSDA.sol
 BondFixedExpiryOFDA.sol
 BondFixedTermOSDA.sol
 BondFixedExpiryFPA.sol
 BondFixedTermFPA.sol
 BondChainlinkOracle.sol
 BondBatchAuctionFactoryV1.sol
 BondBatchAuctionV1.sol
```

The in-scope contracts integrate with these previously audited contracts:

```
src
 bases
|    BondBaseTeller.sol
|    BondBaseCallback.sol
 interfaces
|    IBondTeller.sol
|    IBondFixedExpiryTeller.sol
|    IBondFixedTermTellers.sol
|    IBondAggregator.sol
|    IBondCallback.sol
|    IBondAuctioneer.sol
```

```
BondAggregator.sol
BondFixedTermTeller.sol
BondFixedExpiryTeller.sol
BondSampleCallback.sol
ERC20BondToken.sol
```

The following sections provide context and details about the different implementations.

Findings

Each issue has an assigned severity:

- 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.

Issues found

Medium	High
3	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

xiaoming90	whitehat
Avci	Bauer

Issue M-1: The createMarket transaction lack of expiration timestamp check

Source: https://github.com/sherlock-audit/2023-02-bond-judging/issues/60

Found by

whitehat

Summary

The createMarket transaction lack of expiration timestamp check

Vulnerability Detail

Let us look into the heavily forked Uniswap V2 contract addLiquidity function implementation

https://github.com/Uniswap/v2-periphery/blob/0335e8f7e1bd1e8d8329fd300aea2ef2f36dd19f/contracts/UniswapV2Router02.sol#L61

```
// **** ADD LIQUIDITY ****
function _addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin
) internal virtual returns (uint amountA, uint amountB) {
    // create the pair if it doesn't exist yet
    if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
        IUniswapV2Factory(factory).createPair(tokenA, tokenB);
    }
    (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory,
↪   tokenA, tokenB);
    if (reserveA == 0 && reserveB == 0) {
        (amountA, amountB) = (amountADesired, amountBDesired);
    } else {
        uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA,
↪   reserveB);
        if (amountBOptimal <= amountBDesired) {
            require(amountBOptimal >= amountBMin, 'UniswapV2Router:
↪   INSUFFICIENT_B_AMOUNT');
            (amountA, amountB) = (amountADesired, amountBOptimal);
        } else {
            uint amountAOptimal = UniswapV2Library.quote(amountBDesired,
↪   reserveB, reserveA);
            assert(amountAOptimal <= amountADesired);
            require(amountAOptimal >= amountAMin, 'UniswapV2Router:
↪   INSUFFICIENT_A_AMOUNT');
            (amountA, amountB) = (amountAOptimal, amountBDesired);
        }
    }
}

function addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external virtual override ensure(deadline) returns (uint amountA, uint
↪   amountB, uint liquidity) {
    (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired,
↪   amountBDesired, amountAMin, amountBMin);
    address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
    TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
    TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
    liquidity = IUniswapV2Pair(pair).mint(to);
}
```

the implementation has two point that worth noting,

the first point is the deadline check

```
modifier ensure(uint deadline) {
    require(deadline >= block.timestamp, 'UniswapV2Router: EXPIRED');
    _;
}
```

The transaction can be pending in mempool for a long time and can be executed in a long time after the user submit the transaction.

Problem is createMarket, which calculates the length and maxPayout by block.timestamp inside it.

```
// Calculate market length and check time bounds
uint48 length = uint48(params_.conclusion - block.timestamp); \
if (
    length < minMarketDuration ||
    params_.depositInterval < minDepositInterval ||
    params_.depositInterval > length
) revert Auctioneer_InvalidParams();

// Calculate the maximum payout amount for this market, determined by deposit
↪  interval
uint256 capacity = params_.capacityInQuote
    ? params_.capacity.mulDiv(scale, price)
    : params_.capacity;
market.maxPayout = capacity.mulDiv(uint256(params_.depositInterval),
↪  uint256(length));
```

After the market is created at wrong time, user can call purchase. At purchaseBond(),

```
// Payout for the deposit = amount / price
//
// where:
// payout = payout tokens out
// amount = quote tokens in
// price = quote tokens : payout token (i.e. 200 QUOTE : BASE), adjusted for
↪  scaling
payout = amount_.mulDiv(term.scale, price);

// Payout must be greater than user inputted minimum
if (payout < minAmountOut_) revert Auctioneer_AmountLessThanMinimum();

// Markets have a max payout amount, capping size because deposits
// do not experience slippage. max payout is recalculated upon tuning
if (payout > market.maxPayout) revert Auctioneer_MaxPayoutExceeded();
```

payout value is calculated by term.scale which the market owner has set assuming the market would be created at desired timestamp. Even, maxPayout is far bigger than expected, as it is calculated by very small length.

Impact

Even though the market owner close the market at any time, malicious user can attack the market before close and steal unexpectedly large amount of payout Tokens.

Code Snippet

Tool used

Manual Review

Recommendation

Use deadline, like uniswap

Discussion

Oighty

Agree with this finding. We have noticed some issues with shorter than expected durations for existing markets.

Our proposed fix is to have users specify a `start` timestamp and a `duration`, which will be used to calculate/set the conclusion. If `block.timestamp` is greater than the `start`, then the txn will revert. Therefore, users must create the market before the target start time. We may allow this to be bypassed by providing a start time of zero, which would then start the market at the `block.timestamp` for the provided `duration`.

hrishibhat

Given the pre-condition that the transaction needs to be in the mempool for a long time for it to have a significant impact, considering this issue as valid medium

Oighty

https://github.com/Bond-Protocol/bonds/pull/54

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/pull/54

Issue M-2: "Equilibrium price" is not used to compute the capacity (OSDA Only)

Source: https://github.com/sherlock-audit/2023-02-bond-judging/issues/18

Found by

xiaoming90, Bauer

Summary

"Equilibrium price" is not used to compute the capacity leading to a smaller-than-expected max payout.

Vulnerability Detail

In OFDA, it was observed that if the capacity is denominated in the quote token, the capacity will be calculated with the discounted price.

```
File: BondBaseOFDA.sol
    function _createMarket(MarketParams memory params_) internal returns (uint256) {
..SNIP..
        // Calculate the maximum payout amount for this market
        uint256 capacity = params_.capacityInQuote
            ? params_.capacity.mulDiv(
                scale,
                price.mulDivUp(
                    uint256(ONE_HUNDRED_PERCENT - params_.fixedDiscount),
                    uint256(ONE_HUNDRED_PERCENT)
                )
            )
            : params_.capacity;
        market.maxPayout = capacity.mulDiv(uint256(params_.depositInterval), uint256(length));
```

However, in OSDA, if the capacity is denominated in the quote token, the capacity will be calculated with the oracle price instead of the discounted price.

```
File: BondBaseOSDA.sol
    function _createMarket(MarketParams memory params_) internal returns (uint256) {
..SNIP..
        // Calculate the maximum payout amount for this market, determined by deposit interval
        uint256 capacity = params_.capacityInQuote
```

```
            ? params_.capacity.mulDiv(scale, price)
            : params_.capacity;
        market.maxPayout = capacity.mulDiv(uint256(params_.depositInterval), uint256(length));
```

In OSDA, it was also observed that the base discount is applied to the oracle price while calculating the price decimals because this will be the initial equilibrium price of the market. However, this "initial equilibrium price" is not used earlier when computing the capacity.

```
File: BondBaseOSDA.sol
    function _validateOracle(
        uint256 id_,
        IBondOracle oracle_,
        ERC20 quoteToken_,
        ERC20 payoutToken_,
        uint48 baseDiscount_
    )
..SNIP..
        // Get the price decimals for the current oracle price
        // Oracle price is in quote tokens per payout token
        // E.g. if quote token is $10 and payout token is $2000,
        // then the oracle price is 200 quote tokens per payout token.
        // If the oracle has 18 decimals, then it would return 200 * 10^18.
        // In this case, the price decimals would be 2 since 200 = 2 * 10^2.
        // We apply the base discount to the oracle price before calculating
        // since this will be the initial equilibrium price of the market.
        int8 priceDecimals = _getPriceDecimals(
            currentPrice.mulDivUp(
                uint256(ONE_HUNDRED_PERCENT - baseDiscount_),
                uint256(ONE_HUNDRED_PERCENT)
            ),
            oracleDecimals
        );
```

Impact

As the discount is not applied to the price when computing the capacity, the price will be higher which leads to a smaller capacity. A smaller capacity will in turn result in a smaller max payout. A smaller-than-expected max payout reduces the maximum number of payout tokens a user can purchase at any single point in time, which might reduce the efficiency of a Bond market.

Users who want to purchase a large number of bond tokens have to break their trade into smaller chunks to overcome the smaller-than-expected max payout, leading to unnecessary delay and additional gas fees.

Code Snippet

https://github.com/sherlock-audit/2023-02-bond/blob/main/bonds/src/bases/BondBaseOSDA.sol#L122

Tool used

Manual Review

Recommendation

Applied the discount to obtain the "equilibrium price" before computing the capacity.

```
// Calculate the maximum payout amount for this market, determined by deposit
↪  interval
uint256 capacity = params_.capacityInQuote
-    ? params_.capacity.mulDiv(scale, price)
+    ? params_.capacity.mulDiv(scale, price.mulDivUp(
+            uint256(ONE_HUNDRED_PERCENT - params_.baseDiscount),
+            uint256(ONE_HUNDRED_PERCENT)
+        )
+    )
    : params_.capacity;
market.maxPayout = capacity.mulDiv(uint256(params_.depositInterval),
↪  uint256(length));
```

Discussion

UsmannK

Escalate for 10 USDC.

This issue should be Medium, not High. The issue identified is that `BondBaseOFDA` Auctions, in only some cases (`if capacityInQuote=true`), do not take into account the `fixedDiscount` parameter when calculating the auction's `capacity`.

The capacity in these cases may be **under-calculated**. If there is a situation as:

Payout token: DAI Quote token: UNI Exchange rate: 5 UNI : 1 DAI Fixed discount: 10% `capacityInQuote`: true `params_.capacity`: 10

Then the final `capacity` will be calculated as 10/5=`2` instead of 10/(4.5)=`2.222`.

AKA the auction will sell slightly **fewer** tokens than intended, and end early. This is equivalent to a very minor denial of service and is at best a medium issue by the following criteria, taken from the Sherlock docs:

```
Medium: There is a viable scenario (even if unlikely) that could cause the
↪  protocol to enter a state where a material amount of funds can be lost.

High: This vulnerability would result in a material loss of funds,
```

If the protocol made this mistake in reverse and actually took slightly extra tokens, it may be closer to a High. But that is not the case. Actually the protocol just sells slightly **fewer** of the owner's tokens than intended, **but sells all of those at the correct price**.

sherlock-admin

> Escalate for 10 USDC.
>
> This issue should be Medium, not High. The issue identified is that `BondBaseOFDA` Auctions, in only some cases (`if capacityInQuote=true`), do not take into account the `fixedDiscount` parameter when calculating the auction's `capacity`.
>
> The capacity in these cases may be **under-calculated**. If there is a situation as:
>
> Payout token: DAI Quote token: UNI Exchange rate: 5 UNI : 1 DAI Fixed discount: 10% `capacityInQuote`: true `params_.capacity`: 10
>
> Then the final `capacity` will be calculated as 10/5=`2` instead of 10/(4.5)=`2.222`.
>
> AKA the auction will sell slightly **fewer** tokens than intended, and end early. This is equivalent to a very minor denial of service and is at best a medium issue by the following criteria, taken from the Sherlock docs:
>
> ```
> Medium: There is a viable scenario (even if unlikely) that could cause the
> ↪  protocol to enter a state where a material amount of funds can be lost.
>
> High: This vulnerability would result in a material loss of funds,
> ```
>
> If the protocol made this mistake in reverse and actually took slightly extra tokens, it may be closer to a High. But that is not the case. Actually the protocol just sells slightly **fewer** of the owner's tokens than intended, **but sells all of those at the correct price**.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Oighty

Issue fixed here: https://github.com/Bond-Protocol/bonds/pull/47

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/pull/47

hrishibhat

Escalation accepted

Issue is a valid medium Given the preconditions and the impact of the incorrect calculation, considering this issue as a valid medium

sherlock-admin

> Escalation accepted
>
> Issue is a valid medium Given the preconditions and the impact of the incorrect calculation, considering this issue as a valid medium

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue M-3: _validateAndGetPrice() doesn't check If Arbitrum sequencer is down in Chainlink feeds

Source: https://github.com/sherlock-audit/2023-02-bond-judging/issues/1

Found by

Avci

Summary

When utilizing Chainlink in L2 chains like Arbitrum, it's important to ensure that the prices provided are not falsely perceived as fresh, even when the sequencer is down. This vulnerability could potentially be exploited by malicious actors to gain an unfair advantage.

Vulnerability Detail

There is no check:

Impact

could potentially be exploited by malicious actors to gain an unfair advantage.

Code Snippet

https://github.com/sherlock-audit/2023-02-bond-0xdanial/blob/0d6f979c9f361bc1101f429b3bb09264577b9a71/bonds/src/BondChainlinkOracle.sol#L129

Tool used

Manual Review

Recommendation

code example of Chainlink:
https://docs.chain.link/data-feeds/l2-sequencer-feeds#example-code

Discussion

Oighty

Agree this should be fixed for using the Chainlink Oracle Contract on L2s. I think the best way to handle is to have a mainnet version of the contract (as is) and L2 version of the contract which implements the sequencer feed check.

UsmannK

Escalate for 10 USDC.

Watson states that the arbitrum sequencer may temporarily go down and cause stale prices to be read from the oracle. This is incorrect; the arbitrum sequencer going down cannot result in stale prices to be accepted.

Stale prices will have an old `updatedAt` timestamp and be rejected by the following code: https://github.com/sherlock-audit/2023-02-bond/blob/8a326a4b39fdaf9eaf5911cfd3e9676a83c24a58/bonds/src/BondChainlinkOracle.sol#L141-L146

```
// Validate chainlink price feed data
// 1. Answer should be greater than zero
// 2. Updated at timestamp should be within the update threshold
// 3. Answered in round ID should be the same as the round ID
if (
    priceInt <= 0 ||
    updatedAt < block.timestamp - uint256(updateThreshold_) ||
    answeredInRound != roundId
) revert BondOracle_BadFeed(address(feed_));
```

The watson's link (https://docs.chain.link/data-feeds/l2-sequencer-feeds#arbitrum) is actually a metadata feed about historical uptime/downtime data that is not related to the supposed issue.

sherlock-admin

> Escalate for 10 USDC.
>
> Watson states that the arbitrum sequencer may temporarily go down and cause stale prices to be read from the oracle. This is incorrect; the arbitrum sequencer going down cannot result in stale prices to be accepted.
>
> Stale prices will have an old `updatedAt` timestamp and be rejected by the following code: https://github.com/sherlock-audit/2023-02-bond/blob/8a326a4b39fdaf9eaf5911cfd3e9676a83c24a58/bonds/src/BondChainlinkOracle.sol#L141-L146
>
> ```
> // Validate chainlink price feed data
> // 1. Answer should be greater than zero
> // 2. Updated at timestamp should be within the update threshold
> // 3. Answered in round ID should be the same as the round ID
> if (
>     priceInt <= 0 ||
>     updatedAt < block.timestamp - uint256(updateThreshold_) ||
>     answeredInRound != roundId
> ) revert BondOracle_BadFeed(address(feed_));
> ```

> The watson's link (https://docs.chain.link/data-feeds/l2-sequencer-feeds#arbitrum) is actually a metadata feed about historical uptime/downtime data that is not related to the supposed issue.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

Oighty

Issue fixed here: https://github.com/Bond-Protocol/bonds/pull/53

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/pull/53

hrishibhat

Escalation rejected

Updating the escalation resolution.
Considering this issue as a valid medium, additional sponsor comments:

> If it updates again within the update threshold. The feeds typically can update several times within a threshold period if the price is moving a lot when the sequencer is down, the new price won't be reported to the chain. the feed on the L2 will return the value it had when it went down

sherlock-admin

> Escalation rejected
>
> Updating the escalation resolution.
> Considering this issue as a valid medium, additional sponsor comments:
>
> > If it updates again within the update threshold. The feeds typically can update several times within a threshold period if the price is moving a lot when the sequencer is down, the new price won't be reported to the chain. the feed on the L2 will return the value it had when it went down

This issue's escalations have been rejected!

Watsons who escalated this issue will have their escalation amount deducted from their next payout.