SHERLOCK

SHERLOCK SECURITY REVIEW FOR

ßondProtocol

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 following contracts in this repository are in scope:

- bases
    - BondBaseCallback.sol
    - BondBaseSDA.sol
    - BondBaseTeller.sol
- interfaces
    - IBondAggregator.sol
    - IBondAuctioneer.sol
    - IBondCallback.sol
    - IBondFixedExpiryTeller.sol
    - IBondFixedTermTeller.sol
    - IBondSDA.sol
    - IBondTeller.sol
- BondAggregator.sol
- BondFixedExpirySDA.sol
- BondFixedExpiryTeller.sol
- BondFixedTermSDA.sol
- BondFixedTermTeller.sol
- BondSampleCallback.sol
- ERC20BondToken.sol

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.

SHERLOCK

- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
14	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

xiaoming90
obront
rvierdiiev

Zarf
8olidity
Bnke0x0

hansfriese

SHERLOCK

Issue H-1: Fixed Term Teller tokens can be created with an expiry in the past

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/34

Found by

obront

Summary

The Fixed Term Teller does not allow tokens to be created with a timestamp in the past. This is a fact that protocols using this feature will expect to hold and build their systems around. However, users can submit expiry timestamps slightly in the future, which correlate to tokenIds in the past, which allows them to bypass this check.

Vulnerability Detail

In `BondFixedTermTeller.sol`, the `create()` function allows protocols to trade their payout tokens directly for bond tokens. The expectation is that protocols will build their own mechanisms around this. It is explicitly required that they cannot do this for bond tokens that expire in the past, only those that have yet to expire:

```
if (expiry_ < block.timestamp) revert Teller_InvalidParams();
```

However, because tokenIds round timestamps down to the latest day, protocols are able to get around this check.

Here's an example:

- The most recently expired token has an expiration time of 1668524400 (correlates to 9am this morning)

- It is currently 1668546000 (3pm this afternoon)

- A protocol calls create() with an expiry of 1668546000 + 1

- This passes the check that `expiry_>=block.timestamp`

- When the expiry is passed to `getTokenId()` it rounds the time down to the latest day, which is the day corresponding with 9am this morning

- This expiry associated with this tokenId is 9am this morning, so they are able to redeem their tokens instantly

Impact

Protocols can bypass the check that all created tokens must have an expiry in the future, and mint tokens with a past expiry that can be redeemed immediately.

This may not cause a major problem for Bond Protocol itself, but protocols will be building on top of this feature without expecting this behavior.

Let's consider, for example, a protocol that builds a mechanism where users can stake some asset, and the protocol will trade payout tokens to create bond tokens for them at a discount, with the assumption that they will expire in the future. This issue could create an opening for a savvy user to stake, mint bond tokens, redeem and dump them immediately, buy more assets to stake, and continue this cycle to earn arbitrage returns and tank the protocol's token.

Because there are a number of situations like the one above where this issue could lead to a major loss of funds for a protocol building on top of Bond, I consider this a high severity.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L97-L105

Tool used

Manual Review

Recommendation

Before checking whether `expiry_<block.timestamp`, expiry should be rounded to the nearest day:

```
expiry = ((vesting_ + uint48(block.timestamp)) / uint48(1 days)) * uint48(1
↪  days);
```

Discussion

Evert0x

Message from sponsor

─────────────────────────────

Agree with this finding. We implemented rounding of the expiry before checking whether it is in the past to both the Fixed Term and Fixed Expiry Tellers (fixed expiry added since we made a change to round these to the nearest day as well).

SHERLOCK

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/54b6833a46b5ae4c3a3 ca183b9a55ca8c1266827

SHERLOCK

Issue M-1: Transferring Ownership Might Break The Market

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/41

Found by

xiaoming90

Summary

After the transfer of the market ownership, the market might stop working, and no one could purchase any bond token from the market leading to a loss of sale for the market makers.

Vulnerability Detail

The `callbackAuthorized` mapping contains a list of whitelisted market owners authorized to use the callback. When the users call the `purchaseBond` function, it will check at Line 390 if the current market owner is still authorized to use a callback. Otherwise, the function will revert.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L379

```
File: BondBaseSDA.sol
379:     function purchaseBond(
380:         uint256 id_,
381:         uint256 amount_,
382:         uint256 minAmountOut_
383:     ) external override returns (uint256 payout) {
384:         if (msg.sender != address(_teller)) revert
↪    Auctioneer_NotAuthorized();
385:
386:         BondMarket storage market = markets[id_];
387:         BondTerms memory term = terms[id_];
388:
389:         // If market uses a callback, check that owner is still callback
↪    authorized
390:         if (market.callbackAddr != address(0) &&
↪    !callbackAuthorized[market.owner])
391:             revert Auctioneer_NotAuthorized();
```

However, if the market owner transfers the market ownership to someone else. The market will stop working because the new market owner might not be on the list

of whitelisted market owners (`callbackAuthorized` mapping). As such, no one can purchase any bond token.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L336

```
File: BondBaseSDA.sol
336:     function pushOwnership(uint256 id_, address newOwner_) external override
↪   {
337:         if (msg.sender != markets[id_].owner) revert
↪   Auctioneer_OnlyMarketOwner();
338:         newOwners[id_] = newOwner_;
339:     }
```

Impact

After the transfer of the market ownership, the market might stop working, and no one could purchase any bond token from the market leading to a loss of sale for the market makers.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L379

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L336

Tool used

Manual Review

Recommendation

Before pushing the ownership, if the market uses a callback, implement an additional validation check to ensure that the new market owner has been whitelisted to use the callback. This will ensure that transferring the market ownership will not break the market due to the new market owner not being whitelisted.

```
function pushOwnership(uint256 id_, address newOwner_) external override {
    if (msg.sender != markets[id_].owner) revert Auctioneer_OnlyMarketOwner();
+   if (markets[id_].callbackAddr != address(0) &&
↪   !callbackAuthorized[newOwner_])
+       revert newOwnerNotAuthorizedToUseCallback();
    newOwners[id_] = newOwner_;
}
```

SHERLOCK

Discussion

Evert0x

Message from sponsor

Acknowledged. We added the check for the owner to be whitelisted for a callback on purchase to provide a shutdown mechanism in the event of a malicious callback. The ownership transfer functionality is meant to be used when a callback isn't being used to payout market purchases. E.g. create a market with an EOA from a script and transfer ownership to a multisig for payouts (would also require multisig to approve the teller for the capacity in payout tokens).

Issue M-2: Fixed Term Bond tokens can be minted with non-rounded expiry

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/32

Found by

obront

Summary

Fixed Term Tellers intend to mint tokens that expire once per day, to consolidate liquidity and create a uniform experience. However, this rounding is not enforced on the external `deploy()` function, which allows for tokens expiring at unexpected times.

Vulnerability Detail

In `BondFixedTermTeller.sol`, new tokenIds are deployed through the `_handlePayout()` function. The function calculates the expiry (rounded down to the nearest day), uses this expiry to create a tokenId, and — if that tokenId doesn't yet exist — deploys it.

```
...
expiry = ((vesting_ + uint48(block.timestamp)) / uint48(1 days)) * uint48(1
↪    days);

// Fixed-term user payout information is handled in BondTeller.
// Teller mints ERC-1155 bond tokens for user.
uint256 tokenId = getTokenId(payoutToken_, expiry);

// Create new bond token if it doesn't exist yet
if (!tokenMetadata[tokenId].active) {
    _deploy(tokenId, payoutToken_, expiry);
}
...
```

This successfully consolidates all liquidity into one daily tokenId, which expires (as expected) at the time included in the tokenId.

However, if the `deploy()` function is called directly, no such rounding occurs:

```
function deploy(ERC20 underlying_, uint48 expiry_)
    external
    override
    nonReentrant
    returns (uint256)
```

SHERLOCK

```
{
    uint256 tokenId = getTokenId(underlying_, expiry_);
    // Only creates token if it does not exist
    if (!tokenMetadata[tokenId].active) {
        _deploy(tokenId, underlying_, expiry_);
    }
    return tokenId;
}
```

This creates a mismatch between the tokenId time and the real expiry time, as tokenId is calculated by rounding the expiry down to the nearest day:

```
uint256 tokenId = uint256(
    keccak256(abi.encodePacked(underlying_, expiry_ / uint48(1 days)))
);
```

... while the `_deploy()` function saves the original expiry:

```
tokenMetadata[tokenId_] = TokenMetadata(
    true,
    underlying_,
    uint8(underlying_.decimals()),
    expiry_,
    0
);
```

Impact

The `deploy()` function causes a number of issues:

1) Tokens can be deployed that don't expire at the expected daily time, which may cause issues with your front end or break user's expectations

2) Tokens can expire at times that don't align with the time included in the tokenId

3) Malicious users can pre-deploy tokens at future timestamps to "take over" the token for a given day and lock it at a later time stamp, which then "locks in" that expiry time and can't be changed by the protocol

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L175-L187

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L243-L250

SHERLOCK

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L194-L212

Tool used

Manual Review

Recommendation

Include the same rounding process in `deploy()` as is included in `_handlePayout()`:

```solidity
function deploy(ERC20 underlying_, uint48 expiry_)
        external
        override
        nonReentrant
        returns (uint256)
    {
        expiry = ((vesting_ + uint48(block.timestamp)) / uint48(1 days)) *
↪    uint48(1 days);
        uint256 tokenId = getTokenId(underlying_, expiry_);
        ...
```

Discussion

Evert0x

Message from sponsor

———————————————————

Agree. We implemented rounding of the expiry value provided to `deploy` to match the calculation in `_handlePayouts`.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/54b6833a46b5ae4c3a3ca183b9a55ca8c1266827

SHERLOCK

Issue M-3: Read-only reentrancy in BondFixedTermTeller

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/23

Found by

Zarf

Summary

When minting new ERC1155 bonds in the `BondFixedTermTeller` contract, the total supply of this specific bond is updated after the new bonds are sent to the recipient, which introduces a reentrancy attack.

Vulnerability Detail

Whenever a new ERC1155 bond is minted in the `BondFixedTermTeller` contract, either through `_handlePayout()` or `create()`, the total supply is updated after the bond has been minted.

ERC1155 tokens will perform a callback to the recipient in case the recipient implements the `ERC1155TokenReceiver` interface. Therefore, the recipient (`msg.sender` in `create()` or `recipient_` in `_handlePayout()`) is able to perform a call to an arbitrary contract before the total supply of the bonds is updated.

While the recipient could enter the current `BondFixedTermTeller` contract to call any function, there is no interesting function which might result in financial loss in case it gets called in the callback. Alternatively, the recipient could enter a smart contract which uses the the public mapping `tokenMetadata` in `BondFixedTermTeller` to calculate the current bond price based on the supply. As the supply is not yet updated, but the tokens are minted, this might result in a miscalculation of the price.

Impact

While the `BondFixedTermTeller` contract itself is not at risk, any protocols integrating with `BondFixedTermTeller` and using the total supply of the ERC1155 bond token to calculate the price, might come at risk.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L218-L225

Tool used

Manual Review

Recommendation

Update the total supply and mint the tokens afterwards:

```
function _mintToken(
    address to_,
    uint256 tokenId_,
    uint256 amount_
) internal {
    tokenMetadata[tokenId_].supply += amount_;
    _mint(to_, tokenId_, amount_, bytes(""));
}
```

Discussion

Evert0x

Message from sponsor

Agree with this issue. We updated the `_mintToken()` and `_burnToken()` functions to update supply prior to minting/burning tokens to avoid the reentrancy issue.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/fafd81d04d685d15612cc 56af635513e11ddc626

SHERLOCK

Issue M-4: Existing Circuit Breaker Implementation Allow Faster Taker To Extract Payout Tokens From Market

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/21

Found by

hansfriese, xiaoming90

Summary

The current implementation of the circuit breaker is not optimal. Thus, the market maker will lose an excessive amount of payout tokens if a quoted token suddenly loses a large amount of value, even with a circuit breaker in place.

Vulnerability Detail

When the amount of the payout tokens purchased by the taker exceeds the `term.maxDebt`, the taker is still allowed to carry on with the transaction, and the market will only be closed after the current transaction is completed.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L427

```
File: BondBaseSDA.sol
426:            // Circuit breaker. If max debt is breached, the market is closed
427:            if (term.maxDebt < market.totalDebt) {
428:                _close(id_);
429:            } else {
430:                // If market will continue, the control variable is tuned to to
↪  expend remaining capacity over remaining market duration
431:                _tune(id_, currentTime, price);
432:            }
```

Assume that the state of the SDAM at T0 is as follows:

- `term.maxDebt` is 110 (debt buffer = 10%)

- `maxPayout` is 100

- `market.totalDebt` is 99

Assume that the quoted token suddenly loses a large amount of value (e.g. stablecoin depeg causing the quote token to drop to almost zero). Bob decided to purchase as many payout tokens as possible before reaching the `maxPayout` limit to maximize the value he could extract from the market. Assume that Bob is able to purchase

SHERLOCK

50 bond tokens at T1 before reaching the `maxPayout` limit. As such, the state of the SDAM at T1 will be as follows:

- `term.maxDebt` = 110

- `maxPayout` = 100

- `market.totalDebt` = 99 + 50 = 149

In the above scenario, Bob's purchase has already breached the `term.maxDebt` limit. However, he could still purchase the 50 bond tokens in the current transaction.

Impact

In the event that the price of the quote token falls to almost zero (e.g. 0.0001 dollars), then the fastest taker will be able to extract as many payout tokens as possible before reaching the `maxPayout` limit from the market. The extracted payout tokens are essentially free for the fastest taker. Taker gain is maker loss.

Additionally, in the event that a quoted token suddenly loses a large amount of value, the amount of payout tokens lost by the market marker is capped at the `maxPayout` limit instead of capping the loss at the `term.maxDebt` limit. This resulted in the market makers losing more payout tokens than expected, and their payout tokens being sold to the takers at a very low price (e.g. 0.0001 dollars).

The market makers will suffer more loss if the `maxPayout` limit of their markets is higher.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L427

Tool used

Manual Review

Recommendation

Considering only allowing takers to purchase bond tokens up to the `term.maxDebt` limit.

For instance, based on the earlier scenario, only allow Bob to purchase up to 11 bond tokens (term.maxDebt[110] - market.totalDebt[99]) instead of allowing him to purchase 50 bond tokens.

If Bob attempts to purchase 50 bond tokens, the market can proceed to purchase the 11 bond tokens for Bob, and the remaining quote tokens can be refunded back to Bob. After that, since the `term.maxDebt(110)==market.totalDebt(110)`, the market

SHERLOCK

can trigger the circuit breaker to close the market to protect the market from potential extreme market conditions.

This ensures that bond tokens beyond the `term.maxDebt` limit would not be sold to the taker during extreme market conditions.

Discussion

Evert0x

Message from sponsor

We acknowledge that this can be an issue in an edge case, but do not believe it is a High severity issue. Specifically, it requires a quote token to rapidly collapse in value AND a market to be configured with a large maxPayout (i.e. long deposit interval). Additionally, the potential updates (limiting purchase amount, reverting the transaction) have their own issues. Reverting the transaction will leave the market open, allow a retry transaction with a smaller amount, and have the price continue decaying (perhaps allowing additional buys) if the owner doesn't recognize the issue and close the market. Limiting the purchase amount requires a large amount of refactoring and would introduce additional gas costs for the checks required. We do not believe protecting from the edge case warrants the additional costs.

Evert0x

Downgrading to medium because of comment from protocol

Issue M-5: Market Price Lower Than Expected

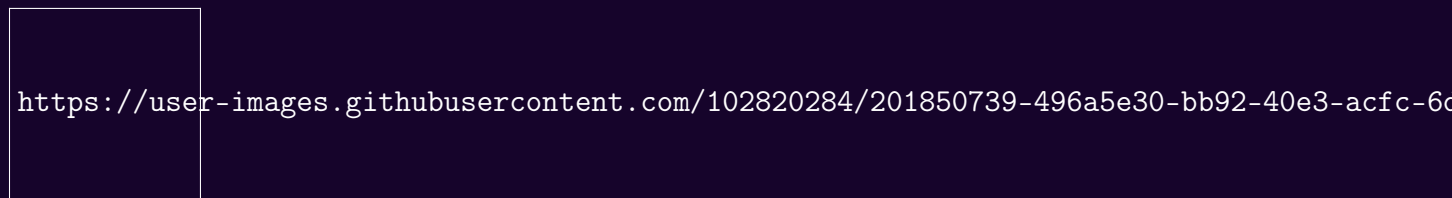Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/20

Found by

xiaoming90

Summary

The market price does not conform to the specification documented within the whitepaper. As a result, the computed market price is lower than expected.

Vulnerability Detail

The following definition of the market price is taken from the whitepaper. Taken from Page 13 of the whitepaper - Definition 25

```
https://user-images.githubusercontent.com/102820284/201850739-496a5e30-bb92-40e3-acfc-6c
```

The integer implementation of the market price must be rounded up per the whitepaper. This ensures that the integer implementation of the market price is greater than or equal to the real value of the market price so as to protect makers from selling tokens at a lower price than expected.

Within the `BondBaseSDA.marketPrice` function, the computation of the market price is rounded up in Line 688, which conforms to the specification.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L687

```
File: BondBaseSDA.sol
687:     function marketPrice(uint256 id_) public view override returns (uint256)
↪   {
688:         uint256 price =
↪   currentControlVariable(id_).mulDivUp(currentDebt(id_), markets[id_].scale);
689:
690:         return (price > markets[id_].minPrice) ? price :
↪   markets[id_].minPrice;
691:     }
```

However, within the `BondBaseSDA._currentMarketPrice` function, the market price is rounded down, resulting in the makers selling tokens at a lower price than expected.

SHERLOCK

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L631

```
File: BondBaseSDA.sol
631:    function _currentMarketPrice(uint256 id_) internal view returns
↪   (uint256) {
632:        BondMarket memory market = markets[id_];
633:        return terms[id_].controlVariable.mulDiv(market.totalDebt,
↪   market.scale);
634:    }
```

Impact

Loss for the makers as their tokens are sold at a lower price than expected.

Additionally, the affected `BondBaseSDA._currentMarketPrice` function is used within the `BondBaseSDA._decayAndGetPrice` function to derive the market price. Since a lower market price will be returned, this will lead to a higher amount of payout tokens. Subsequently, the `lastDecayIncrement` will be higher than expected, which will lead to a lower `totalDebt`. Lower debt means a lower market price will be computed later.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L687

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L631

Tool used

Manual Review

Recommendation

Ensure the market price is rounded up so that the desired property can be achieved and the makers will not be selling tokens at a lower price than expected.

```
function _currentMarketPrice(uint256 id_) internal view returns (uint256) {
    BondMarket memory market = markets[id_];
-   return terms[id_].controlVariable.mulDiv(market.totalDebt, market.scale);
+   return terms[id_].controlVariable.mulDivUp(market.totalDebt, market.scale);
}
```

SHERLOCK

Discussion

Evert0x

Message from sponsor

Agree with this issue. We have updated the price calculation in `_currentMarketPrice()` to round up to match the specification.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/a77f0150dd4bf401a1b30c16eca4865bb69c59d3

Issue M-6: Teller Cannot Be Removed From Callback Contract

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/18

Found by

xiaoming90

Summary

If a vulnerable Teller is being exploited by an attacker, there is no way for the owner of the Callback Contract to remove the vulnerable Teller from their Callback Contract.

Vulnerability Detail

The Callback Contract is missing the feature to remove a Teller. Once a Teller has been added to the whitelist (`approvedMarkets` mapping), it is not possible to remove the Teller from the whitelist.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseCallback.sol#L59

```
File: BondBaseCallback.sol
56:     /* ========== WHITELISTING ========== */
57:
58:     /// @inheritdoc IBondCallback
59:     function whitelist(address teller_, uint256 id_) external override
↪  onlyOwner {
60:         // Check that the market id is a valid, live market on the aggregator
61:         try _aggregator.isLive(id_) returns (bool live) {
62:             if (!live) revert Callback_MarketNotSupported(id_);
63:         } catch {
64:             revert Callback_MarketNotSupported(id_);
65:         }
66:
67:         // Check that the provided teller is the teller for the market ID on
↪  the stored aggregator
68:         // We could pull the teller from the aggregator, but requiring the
↪  teller to be passed in
69:         // is more explicit about which contract is being whitelisted
70:         if (teller_ != address(_aggregator.getTeller(id_))) revert
↪  Callback_TellerMismatch();
71:
72:         approvedMarkets[teller_][id_] = true;
73:     }
```

SHERLOCK

Impact

In the event that a whitelisted Teller is found to be vulnerable and has been actively exploited by an attacker in the wild, the owner of the Callback Contract needs to mitigate the issue swiftly by removing the vulnerable Teller from the Callback Contract to stop it from draining the asset within the Callback Contract. However, the mitigation effort will be hindered by the fact there is no way to remove a Teller within the Callback Contract once it has been whitelisted. Thus, it might not be possible to stop the attacker from exploiting the vulnerable Teller to drain assets within the Callback Contract. The Callback Contract owners would need to find a workaround to block the attack, which will introduce an unnecessary delay to the recovery process where every second counts.

Additionally, if the owner accidentally whitelisted the wrong Teller, there is no way to remove it.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseCallback.sol#L59

Tool used

Manual Review

Recommendation

Consider implementing an additional function to allow the removal of a Teller from the whitelist (`approvedMarkets` mapping), so that a vulnerable Teller can be removed swiftly if needed.

```solidity
function removeFromWhitelist(address teller_, uint256 id_) external override
↪   onlyOwner {
    approvedMarkets[teller_][id_] = false;
}
```

Note: Although the owner of the Callback Contract can DOS its own market by abusing the `removeFromWhitelist` function, no sensible owner would do so.

Discussion

Evert0x

Message from sponsor

SHERLOCK

Agree with this issue. We implemented a `blacklist()` function on the BondBaseCall-back.sol contract to allow removing a teller and market ID combination from using the callback.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/368e6c5b120d9fc44c59cc21d33ee51728728067

Issue M-7: Create Fee Discount Feature Is Broken

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/16

Found by

8olidity, xiaoming90

Summary

The create fee discount feature is found to be broken within the protocol.

Vulnerability Detail

The create fee discount feature relies on the `createFeeDiscount` state variable to determine the fee to be discounted from the protocol fee. However, it was observed that there is no way to initialize the `createFeeDiscount` state variable. As a result, the `createFeeDiscount` state variable will always be zero.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedExpiryTeller.sol#L118

```
File: BondFixedExpiryTeller.sol
118:            // If fee is greater than the create discount, then calculate the
 ↪   fee and store it
119:            // Otherwise, fee is zero.
120:            if (protocolFee > createFeeDiscount) {
121:                // Calculate fee amount
122:                uint256 feeAmount = amount_.mulDiv(protocolFee -
 ↪   createFeeDiscount, FEE_DECIMALS);
123:                rewards[_protocol][underlying_] += feeAmount;
124:
125:                // Mint new bond tokens
126:                bondToken.mint(msg.sender, amount_ - feeAmount);
127:
128:                return (bondToken, amount_ - feeAmount);
129:            } else {
130:                // Mint new bond tokens
131:                bondToken.mint(msg.sender, amount_);
132:
133:                return (bondToken, amount_);
134:            }
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L118

SHERLOCK

```
File: BondFixedTermTeller.sol
118:            // If fee is greater than the create discount, then calculate the
    ↪   fee and store it
119:            // Otherwise, fee is zero.
120:            if (protocolFee > createFeeDiscount) {
121:                // Calculate fee amount
122:                uint256 feeAmount = amount_.mulDiv(protocolFee -
    ↪   createFeeDiscount, FEE_DECIMALS);
123:                rewards[_protocol][underlying_] += feeAmount;
124:
125:                // Mint new bond tokens
126:                _mintToken(msg.sender, tokenId, amount_ - feeAmount);
127:
128:                return (tokenId, amount_ - feeAmount);
129:            } else {
130:                // Mint new bond tokens
131:                _mintToken(msg.sender, tokenId, amount_);
132:
133:                return (tokenId, amount_);
134:            }
```

Impact

The create fee discount feature is broken within the protocol. There is no way for the protocol team to configure a discount for the users of the `BondFixedExpiryTeller.create` and `BondFixedTermTeller.create` functions. As such, the users will not obtain any discount from the protocol when using the create function.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedExpiryTeller.sol#L118

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L118

Tool used

Manual Review

Recommendation

Implement a setter method for the `createFeeDiscount` state variable and the necessary verification checks.

SHERLOCK

```
function setCreateFeeDiscount(uint48 createFeeDiscount_) external requiresAuth {
    if (createFeeDiscount_ > protocolFee)  revert Teller_InvalidParams();
    if (createFeeDiscount_ > 5e3) revert Teller_InvalidParams();
    createFeeDiscount = createFeeDiscount_;
}
```

Discussion

Evert0x

Message from sponsor

Agree. We implemented a `setCreateFeeDiscount` function on the BondBaseTeller to allow updating the create fee discount.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/570eb0b74b2401c7b6d0 7a30f8dd452bf7f225f9

SHERLOCK

Issue M-8: `BondAggregator.findMarketFor` Function Will Break In Certain Conditions

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/14

Found by

xiaoming90

Summary

`BondAggregator.findMarketFor` function will break when the `BondBaseSDA.payoutFor` function within the for-loop reverts under certain conditions.

Vulnerability Detail

The `BondBaseSDA.payoutFor` function will revert if the computed payout is larger than the market's max payout. Refer to Line 711 below.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L699

```
File: BondBaseSDA.sol
699:     function payoutFor(
700:         uint256 amount_,
701:         uint256 id_,
702:         address referrer_
703:     ) public view override returns (uint256) {
704:         // Calculate the payout for the given amount of tokens
705:         uint256 fee = amount_.mulDiv(_teller.getFee(referrer_), 1e5);
706:         uint256 payout = (amount_ - fee).mulDiv(markets[id_].scale,
      marketPrice(id_));
707:
708:         // Check that the payout is less than or equal to the maximum
      payout,
709:         // Revert if not, otherwise return the payout
710:         if (payout > markets[id_].maxPayout) {
711:             revert Auctioneer_MaxPayoutExceeded();
712:         } else {
713:             return payout;
714:         }
715:     }
```

The `BondAggregator.findMarketFor` function will call the `BondBaseSDA.payoutFor` function at Line 245. The `BondBaseSDA.payoutFor` function will revert if the final com-

SHERLOCK

puted payout is larger than the `markets[id_].maxPayout` as mentioned earlier. This will cause the entire for-loop to "break" and the transaction to revert.

Assume that the user configures the `minAmountOut_` to be `0`, then the condition `minAmountOut_<=maxPayout` Line 244 will always be true. The `amountIn_` will always be passed to the `payoutFor` function. In some markets where the computed payout is larger than the market's max payout, the `BondAggregator.findMarketFor` function will revert.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondAggregator.sol#L221

```
File: BondAggregator.sol
220:       /// @inheritdoc IBondAggregator
221:       function findMarketFor(
222:           address payout_,
223:           address quote_,
224:           uint256 amountIn_,
225:           uint256 minAmountOut_,
226:           uint256 maxExpiry_
227:       ) external view returns (uint256) {
228:           uint256[] memory ids = marketsFor(payout_, quote_);
229:           uint256 len = ids.length;
230:           uint256[] memory payouts = new uint256[](len);
231:
232:           uint256 highestOut;
233:           uint256 id = type(uint256).max; // set to max so an empty set
     ↪   doesn't return 0, the first index
234:           uint48 vesting;
235:           uint256 maxPayout;
236:           IBondAuctioneer auctioneer;
237:           for (uint256 i; i < len; ++i) {
238:               auctioneer = marketsToAuctioneers[ids[i]];
239:               (, , , , vesting, maxPayout) =
     ↪   auctioneer.getMarketInfoForPurchase(ids[i]);
240:
241:               uint256 expiry = (vesting <= MAX_FIXED_TERM) ? block.timestamp +
     ↪   vesting : vesting;
242:
243:               if (expiry <= maxExpiry_) {
244:                   payouts[i] = minAmountOut_ <= maxPayout
245:                       ? payoutFor(amountIn_, ids[i], address(0))
246:                       : 0;
247:
248:                   if (payouts[i] > highestOut) {
249:                       highestOut = payouts[i];
250:                       id = ids[i];
251:                   }
```

SHERLOCK

```
252:                }
253:            }
254:
255:            return id;
256:        }
```

Impact

The find market feature within the protocol is broken under certain conditions. As such, users would not be able to obtain the list of markets that meet their requirements. The market makers affected by this issue will lose the opportunity to sell their bond tokens.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L699

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondAggregator.sol#L221

Tool used

Manual Review

Recommendation

Consider using try-catch or address.call to handle the revert of the `BondBaseSDA.payoutFor` function within the for-loop gracefully. This ensures that a single revert of the `BondBaseSDA.payoutFor` function will not affect the entire for-loop within the `BondAggregator.findMarketFor` function.

Discussion

Evert0x

Message from sponsor

Agree with this finding. We implemented the try-catch suggestion without the `findMarketFor` for loop to silently handle these errors while still preserving the error for other uses.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/56a11260c40785509215b8c81830f6270b46d15f

SHERLOCK

Issue M-9: Auctioneer Cannot Be Removed From The Protocol

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/13

Found by

xiaoming90

Summary

If a vulnerable Auctioneer is being exploited by an attacker, there is no way to remove the vulnerable Auctioneer from the protocol.

Vulnerability Detail

The protocol is missing the feature to remove an auctioneer. Once an auctioneer has been added to the whitelist, it is not possible to remove the auctioneer from the whitelist.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondAggregator.sol#L62

```
File: BondAggregator.sol
62:     function registerAuctioneer(IBondAuctioneer auctioneer_) external
→  requiresAuth {
63:         // Restricted to authorized addresses
64:
65:         // Check that the auctioneer is not already registered
66:         if (_whitelist[address(auctioneer_)])
67:             revert Aggregator_AlreadyRegistered(address(auctioneer_));
68:
69:         // Add the auctioneer to the whitelist
70:         auctioneers.push(auctioneer_);
71:         _whitelist[address(auctioneer_)] = true;
72:     }
```

Impact

In the event that a whitelisted Auctioneer is found to be vulnerable and has been actively exploited by an attacker in the wild, the protocol needs to mitigate the issue swiftly by removing the vulnerable Auctioneer from the protocol. However, the mitigation effort will be hindered by the fact there is no way to remove an Auctioneer within the protocol once it has been whitelisted. Thus, it might not be possible to stop the attacker from exploiting the vulnerable Auctioneer. The protocol team would

need to find a workaround to block the attack, which will introduce an unnecessary delay to the recovery process where every second counts.

Additionally, if the admin accidentally whitelisted the wrong Auctioneer, there is no way to remove it.

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondAggregator.sol#L62

Tool used

Manual Review

Recommendation

Consider implementing an additional function to allow the removal of an Auctioneer from the whitelist, so that vulnerable Auctioneer can be removed swiftly if needed.

```
function deregisterAuctioneer(IBondAuctioneer auctioneer_) external requiresAuth
↪ {
    // Remove the auctioneer from the whitelist
    _whitelist[address(auctioneer_)] = false;
}
```

Discussion

Evert0x

Message from sponsor

We acknowledge that a whitelisted Auctioneer cannot be removed from the Aggregator. Auctioneers can be sunset by disallowing new markets to be created. Adding a function to remove a whitelisted auctioneer would effectively give the protocol the ability to shutdown live, user-created markets, which goes against the goal of a permissionless system. Users have the ability to close their own markets if needed.

SHERLOCK

Issue M-10: Debt Decay Faster Than Expected

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/12

Found by

xiaoming90

Summary

The debt decay at a rate faster than expected, causing market makers to sell bond tokens at a lower price than expected.

Vulnerability Detail

The following definition of the debt decay reference time following any purchases at time t taken from the whitepaper. The second variable, which is the delay increment, is rounded up. Following is taken from Page 15 of the whitepaper - Definition 27

```
https://user-images.githubusercontent.com/102820284/201844416-023c6d4f-893d-40ab-b6cb-6e
```

However, the actual implementation in the codebase differs from the specification. At Line 514, the delay increment is rounded down instead.

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L514

```
File: BondBaseSDA.sol
513:            // Set last decay timestamp based on size of purchase to linearize
↪   decay
514:            uint256 lastDecayIncrement = debtDecayInterval.mulDiv(payout_,
↪   lastTuneDebt);
515:            metadata[id_].lastDecay += uint48(lastDecayIncrement);
```

Impact

When the delay increment (TD) is rounded down, the debt decay reference time increment will be smaller than expected. The debt component will then decay at a faster rate. As a result, the market price will not be adjusted in an optimized manner, and the market price will fall faster than expected, causing market makers to sell bond tokens at a lower price than expected.

SHERLOCK

Following is taken from Page 8 of the whitepaper - Definition 8

https://user-images.githubusercontent.com/102820284/201844554-bdb7c975-ec4c-417f-a83e-5

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSD
A.sol#L514

Tool used

Manual Review

Recommendation

When computing the `lastDecayIncrement`, the result should be rounded up.

```
// Set last decay timestamp based on size of purchase to linearize decay
- uint256 lastDecayIncrement = debtDecayInterval.mulDiv(payout_, lastTuneDebt);
+ uint256 lastDecayIncrement = debtDecayInterval.mulDivUp(payout_, lastTuneDebt);
metadata[id_].lastDecay += uint48(lastDecayIncrement);
```

Discussion

Evert0x

Message from sponsor

_____

Agree that the rounding should be to match the specification. This was inadvertently
changed when another change was implemented. Good catch.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/071d2a450779dd341322
4831934727dcb77e3045

Issue M-11: BondBaseSDA.setDefaults doesn't validate inputs

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/11

Found by

rvierdiiev

Summary

BondBaseSDA.setDefaults doesn't validate inputs which can lead to initializing new markets incorrectly

Vulnerability Detail

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L348-L356

```
function setDefaults(uint32[6] memory defaults_) external override requiresAuth {
    // Restricted to authorized addresses
    defaultTuneInterval = defaults_[0];
    defaultTuneAdjustment = defaults_[1];
    minDebtDecayInterval = defaults_[2];
    minDepositInterval = defaults_[3];
    minMarketDuration = defaults_[4];
    minDebtBuffer = defaults_[5];
}
```

Function BondBaseSDA.setDefaults doesn't do any checkings, as you can see. Because of that it's possible to provide values that will break market functionality.

For example you can set `minDepositInterval` to be bigger than `minMarketDuration` and it will be not possible to create new market.

Or you can provide `minDebtBuffer` to be 100% ot 0% that will break logic of market closing.

Impact

Can't create new market or market logic will be not working as designed.

Code Snippet

Provided above

SHERLOCK

Tool used

Manual Review

Recommendation

Add input validation.

Discussion

Evert0x

Message from sponsor

Agree. We added the following validation checks to `setDefaults`:

- defaultTuneInterval >= defaultTuneAdjustment
- defaultTuneInterval >= minDepositInterval
- minMarketDuration >= minDepositInterval
- minDebyDecayInterval >= 5 * minDepositInterval

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/141c286cccb7797f8cca68edaf9b886f12897405

SHERLOCK

Issue M-12: BondAggregator.liveMarketsBy eventually will revert because of block gas limit

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/10

Found by

rvierdiiev

Summary

BondAggregator.liveMarketsBy eventually will revert because of block gas limit

Vulnerability Detail

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondAggregator.sol#L259-L280

```
function liveMarketsBy(address owner_) external view returns (uint256[] memory) {
    uint256 count;
    IBondAuctioneer auctioneer;
    for (uint256 i; i < marketCounter; ++i) {
        auctioneer = marketsToAuctioneers[i];
        if (auctioneer.isLive(i) && auctioneer.ownerOf(i) == owner_) {
            ++count;
        }
    }


    uint256[] memory ids = new uint256[](count);
    count = 0;
    for (uint256 i; i < marketCounter; ++i) {
        auctioneer = marketsToAuctioneers[i];
        if (auctioneer.isLive(i) && auctioneer.ownerOf(i) == owner_) {
            ids[count] = i;
            ++count;
        }
    }


    return ids;
}
```

BondAggregator.liveMarketsBy function is looping through all markets and does at least `marketCounter` amount of external calls(when all markets are not live) and at most 4 * `marketCounter` external calls(when all markets are live and owner matches.

SHERLOCK

This all consumes a lot of gas, even that is called from view function. And each new market increases loop size.

That means that after some time `marketsToAuctioneers` mapping will be big enough that the gas amount sent for view/pure function will be not enough to retrieve all data(50 million gas according to this). So the function will revert.

Also similar problem is with `findMarketFor`, `marketsFor` and `liveMarketsFor` functions.

Impact

Functions will always revert and whoever depends on it will not be able to get information.

Code Snippet

Provided above

Tool used

Manual Review

Recommendation

Remove not active markets or some start and end indices to functions.

Discussion

Evert0x

Message from sponsor

Agree. We added start and stop indices to the BondAggregator.liveMarketsBy function to allow pagination through the bond markets and avoid the block gas limit.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/5a2a9a982f3bdfc31d22f72d270bf2d556096281

Issue M-13: meta.tuneBelowCapacity param is not updated when BondBaseSDA.setIntervals is called

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/9

Found by

rvierdiiev

Summary

When BondBaseSDA.setIntervals function is called then meta.tuneBelowCapacity param is not updated which has impact on price tuning.

Vulnerability Detail

BondBaseSDA.setIntervals function allows for market owner to change some market interval. One of them is `meta.tuneInterval`. https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L303-L333

```
function setIntervals(uint256 id_, uint32[3] calldata intervals_) external
↪  override {
    // Check that the market is live
    if (!isLive(id_)) revert Auctioneer_InvalidParams();


    // Check that the intervals are non-zero
    if (intervals_[0] == 0 || intervals_[1] == 0 || intervals_[2] == 0)
        revert Auctioneer_InvalidParams();


    // Check that tuneInterval >= tuneAdjustmentDelay
    if (intervals_[0] < intervals_[1]) revert Auctioneer_InvalidParams();


    BondMetadata storage meta = metadata[id_];
    // Check that tuneInterval >= depositInterval
    if (intervals_[0] < meta.depositInterval) revert Auctioneer_InvalidParams();


    // Check that debtDecayInterval >= minDebtDecayInterval
    if (intervals_[2] < minDebtDecayInterval) revert Auctioneer_InvalidParams();


    // Check that sender is market owner
    BondMarket memory market = markets[id_];
```

SHERLOCK

```
        if (msg.sender != market.owner) revert Auctioneer_OnlyMarketOwner();


        // Update intervals
        meta.tuneInterval = intervals_[0];
        meta.tuneIntervalCapacity = market.capacity.mulDiv(
            uint256(intervals_[0]),
            uint256(terms[id_].conclusion) - block.timestamp
        ); // don't have a stored value for market duration, this will update
 ↪   tuneIntervalCapacity based on time remaining
        meta.tuneAdjustmentDelay = intervals_[1];
        meta.debtDecayInterval = intervals_[2];
}
```

`meta.tuneInterval` has impact on `meta.tuneIntervalCapacity`. That means that when you change tuning interval you also change the capacity that is operated during tuning. There is also one more param that depends on this, but is not counted here.

This is `meta.tuneBelowCapacity` param and it is needed to say if the market has over-sold tokens. In another words it says if `meta.tuneIntervalCapacity` is already sold. This param is checked while tuning and then is updated after the tuning. https://githu b.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseSDA.sol#L576-L621

```
if (
    (market.capacity < meta.tuneBelowCapacity && timeNeutralCapacity <
 ↪   initialCapacity) ||
    (time_ >= meta.lastTune + meta.tuneInterval && timeNeutralCapacity >
 ↪   initialCapacity)
) {
    // Calculate the correct payout to complete on time assuming each bond
    // will be max size in the desired deposit interval for the remaining time
    //
    // i.e. market has 10 days remaining. deposit interval is 1 day. capacity
    // is 10,000 TOKEN. max payout would be 1,000 TOKEN (10,000 * 1 / 10).
    markets[id_].maxPayout = capacity.mulDiv(uint256(meta.depositInterval),
 ↪   timeRemaining);


    // Calculate ideal target debt to satisty capacity in the remaining time
    // The target debt is based on whether the market is under or oversold at
 ↪   this point in time
    // This target debt will ensure price is reactive while ensuring the
 ↪   magnitude of being over/undersold
    // doesn't cause larger fluctuations towards the end of the market.
    //
    // Calculate target debt from the timeNeutralCapacity and the ratio of debt
 ↪   decay interval and the length of the market
```

```
    uint256 targetDebt = timeNeutralCapacity.mulDiv(
        uint256(meta.debtDecayInterval),
        uint256(meta.length)
    );



    // Derive a new control variable from the target debt
    uint256 controlVariable = terms[id_].controlVariable;
    uint256 newControlVariable = price_.mulDivUp(market.scale, targetDebt);



    emit Tuned(id_, controlVariable, newControlVariable);



    if (newControlVariable < controlVariable) {
        // If decrease, control variable change will be carried out over the tune
↪   interval
        // this is because price will be lowered
        uint256 change = controlVariable - newControlVariable;
        adjustments[id_] = Adjustment(change, time_, meta.tuneAdjustmentDelay,
↪   true);
    } else {
        // Tune up immediately
        terms[id_].controlVariable = newControlVariable;
        // Set current adjustment to inactive (e.g. if we are re-tuning early)
        adjustments[id_].active = false;
    }



    metadata[id_].lastTune = time_;
    metadata[id_].tuneBelowCapacity = market.capacity > meta.tuneIntervalCapacity
        ? market.capacity - meta.tuneIntervalCapacity
        : 0;
    metadata[id_].lastTuneDebt = targetDebt;
}
```

If you don't update `meta.tuneBelowCapacity` when changing intervals you have a risk, that price will not be tuned when tuneIntervalCapacity was decreased or it will be still tuned when tuneIntervalCapacity was increased.

As a result tuning will not be completed when needed.

Impact

Tuning logic will not be completed when needed.

SHERLOCK

Code Snippet

Provided above

Tool used

Manual Review

Recommendation

Update meta.tuneBelowCapacity in BondBaseSDA.setIntervals function.

Discussion

Evert0x

Message from sponsor

Agree. We added a line to set meta.tuneBelowCapacity in the setIntervals function to fix this.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/bfd9eea0cc035b3ef1cca4072356f83695f960eb

SHERLOCK

Issue M-14: Solmate safetransfer and safetransferfrom does not check the code size of the token address, which may lead to funding loss

Source: https://github.com/sherlock-audit/2022-11-bond-judging/issues/8

Found by

8olidity, Bnke0x0

Summary

Vulnerability Detail

Impact

the safetransfer and safetransferfrom don't check the existence of code at the token address. This is a known issue while using solmate's libraries. Hence this may lead to miscalculation of funds and may lead to loss of funds, because if safetransfer() and safetransferfrom() are called on a token address that doesn't have a contract in it, it will always return success, bypassing the return value check. Due to this protocol will think that funds have been transferred successfully, and records will be accordingly calculated, but in reality, funds were never transferred. So this will lead to miscalculation and possibly loss of funds

Code Snippet

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseCallback.sol#L143

```
'token_.safeTransfer(to_, amount_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseCallback.sol#L152

```
'token_.safeTransferFrom(msg.sender, address(this), amount_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseTeller.sol#L108

```
'token.safeTransfer(to_, send);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseTeller.sol#L187

SHERLOCK

```
'quoteToken.safeTransferFrom(msg.sender, address(this), amount_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseTeller.sol#L195

```
'quoteToken.safeTransfer(callbackAddr, amountLessFee);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseTeller.sol#L210

```
'payoutToken.safeTransferFrom(owner, address(this), payout_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/bases/BondBaseTeller.sol#L214

```
'quoteToken.safeTransfer(owner, amountLessFee);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedExpiryTeller.sol#L89

```
'underlying_.safeTransfer(recipient_, payout_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedExpiryTeller.sol#L114

```
'underlying_.safeTransferFrom(msg.sender, address(this), amount_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedExpiryTeller.sol#L152

```
'underlying.safeTransfer(msg.sender, amount_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L90

```
'payoutToken_.safeTransfer(recipient_, payout_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L114

```
'underlying_.safeTransferFrom(msg.sender, address(this), amount_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondFixedTermTeller.sol#L151

SHERLOCK

```
'meta.underlying.safeTransfer(msg.sender, amount_);'
```

https://github.com/sherlock-audit/2022-11-bond/blob/main/src/BondSampleCallback.sol#L42

```
'payoutToken_.safeTransfer(msg.sender, outputAmount_);'
```

Tool used

Manual Review

Recommendation

Use openzeppelin's safeERC20 or implement a code existence check

Discussion

Evert0x

Message from sponsor

———————————————————————

Agree. We implemented a code size check in the TransferHelper.sol library to fix this.

xiaoming9090

Fixed in https://github.com/Bond-Protocol/bonds/commit/a247783a240cb7fe6fb25fa19ab9385c025f8e4f

SHERLOCK