Zellic

Bond Protocol

Smart Contract Security Assessment

October 18, 2022

Prepared for:

Bond Labs

Prepared by:

Vlad Toie and Katerina Belotskaia

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1 Executive Summary

Zellic conducted an audit for Bond Labs from October 10th to October 14th, 2022.

Our general overview of the code is that it was very well-organized and structured. The code coverage is high, and tests are included for the majority of the functions. The documentation was adequate, although it could be improved. The code was easy to comprehend, and in most cases, intuitive.

We applaud Bond Labs for their attention to detail and diligence in maintaining incredibly high code quality standards in the development of Bond Protocol.

Zellic thoroughly reviewed the Bond Protocol codebase to find protocol-breaking bugs as defined by the documentation and to find any technical issues outlined in the Methodology section (2.2) of this document.

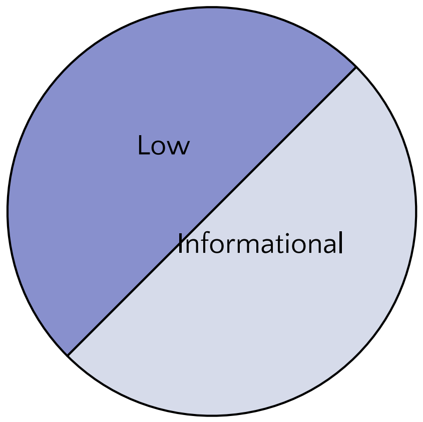
Specifically, taking into account Bond Protocol's threat model, we focused heavily on issues that would break core invariants such as the access control over the creation of bond markets, edge cases that would bypass or break the buying of bonds process, and any other issues involving the pricing of the bonds.

During our assessment on the scoped Bond Protocol contracts, we discovered two findings. Fortunately, no critical issues were found. Of the two findings, one was of low severity, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the audit for Bond Labs's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	1



2 Introduction

2.1 About Bond Protocol

Bond Protocol is a system to create Olympus-style bond markets for any token pair. The markets do not require maintenance and will manage bond prices based on activity. Bond issuers create BondMarkets that pay out a payout token in exchange for deposited quote tokens. Users can purchase future-dated payout tokens with quote tokens at the current market price and receive bond tokens to represent their position while their bond vests. Once the bond tokens vest, they can redeem it for the quote tokens.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of open-source tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. We analyze the scoped smart contract code using automated tools to quickly sieve out and catch these shallow bugs. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so forth as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We manually review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents. We also thoroughly examine the specifications and designs themselves for inconsistencies, flaws, and vulnerabilities. This involves use cases that open the opportunity for abuse, such as flawed tokenomics or share pricing, arbitrage opportunities, and so forth.

Complex integration risks. Several high-profile exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. We perform a meticulous review of all of the contract's possible external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so forth.

Code maturity. We review for possible improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so forth.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact; we assign it on a case-by-case basis based on our professional judgment and experience. As one would expect, both the severity and likelihood of an issue affect its impact; for instance, a highly severe issue's impact may be attenuated by a very low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Similarly, Zellic organizes its reports such that the most important findings come first in the document rather than being ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their importance may differ. This varies based on numerous soft factors, such as our clients' threat models, their business needs, their project timelines, and so forth. We aim to provide useful and actionable advice to our partners that consider their long-term goals rather than simply provide a list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

Bond Protocol Contracts

Repository	https://github.com/OlympusDAO/bonds
Versions	`2d356c32eebe975ffd5061a38683c47e1f639425`
Programs	• BondAggregator • ERC20BondToken • BondFixedTermSDA • BondFixedExpiryTeller • BondFixedExpirySDA • BondBaseTeller • BondBaseCallback • BondBaseSDA • BondFixedTermTeller • BondSampleCallback
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-weeks. The assessment was conducted over the course of one calendar week.

Contact Information

The following project managers were associated with the engagement:

Jasraj Bedi, Co-founder
jazzy@zellic.io

Stephen Tong, Co-founder
stephen@zellic.io

The following consultants were engaged to conduct the assessment:

Vlad Toie, Engineer
vlad@zellic.io

Katerina Belotskaia, Engineer
kate@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

October 10, 2022	Start of primary review period
October 14, 2022	End of primary review period

3 Detailed Findings

3.1 Bond purchases are theoretically front-runnable

- **Target**: BondBaseTeller
- **Category**: Business Logic
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The referrers can arbitrarily change their fees. These fees are applied upon the purchase of bonds by the end users.

```
function purchase(
        address recipient_,
        address referrer_,
        uint256 id_,
        uint256 amount_,
        uint256 minAmountOut_
    ) external virtual nonReentrant returns (uint256, uint48) {
        // ...

        // Calculate fees for purchase
        // 1. Calculate referrer fee
        // 2. Calculate protocol fee as the total expected fee amount
    minus the referrer fee
        // to avoid issues with rounding from separate fee calculations
        uint256 toReferrer = amount_.mulDiv(referrerFees[referrer_],
    FEE_DECIMALS);
        uint256 toProtocol = amount_.mulDiv(protocolFee + referrerFees[
    referrer_], FEE_DECIMALS) -
            toReferrer;

        {
            IBondAuctioneer auctioneer = _aggregator.getAuctioneer(id_);
            address owner;
            (owner, , payoutToken, quoteToken, vesting, ) = auctioneer.
    getMarketInfoForPurchase(
                id_
            );

            // Auctioneer handles bond pricing, capacity, and duration
            uint256 amountLessFee = amount_ - toReferrer - toProtocol;

            payout = auctioneer.purchaseBond(id_, amountLessFee,
     minAmountOut_);
        }
        // ...
```

Impact

The `minAmountOut_` parameter limits the amount of slippage that is allowed; thus, the theoretical chances of a front-run attack are relatively low. Even so, in the case that the `minAmountOut_` is set to a really low threshold or not even set at all, the user could face a loss of funds.

Recommendations

We recommend limiting the amount of fees that the referrers can enforce, thus further decreasing the likelihood of any profitable front-run attacks.

Remediation

The finding has been acknowledged by Bond Labs. Their official response is reproduced below:

> Bond Labs acknowledges the finding but doesn't believe it has security implications. However, we may deploy a bug fix to address it.

3.2 Disallow reclosing a market

- **Target**: BondBaseSDA
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

A market can be closed by its owner through the `closeMarket()` function, which effectively sets its conclusion to the current block timestamp and its capacity to 0, disallowing any further operations on that market.

```
function closeMarket(uint256 id_) external override {
    if (msg.sender != markets[id_].owner) revert
    Auctioneer_OnlyMarketOwner();
    _close(id_);
}

// ...

function _close(uint256 id_) internal {
    terms[id_].conclusion = uint48(block.timestamp);
    markets[id_].capacity = 0;

    emit MarketClosed(id_);
}
```

Impact

There are no checks in place on whether the market has been previously closed or not. The market owner could set the `terms[id].conclusion` to any timestamp by re-calling the `closeMarket` function.

Recommendations

As currently implemented, there are no security implications attached to this issue, however, we do recommend making sure that the market has not been previously closed when calling this function.

```
function closeMarket(uint256 id_) external override {
    if (msg.sender != markets[id_].owner) revert
    Auctioneer_OnlyMarketOwner();
    require(markets[id_].capacity ≠0, “market already closed”);
    _close(id_);
}
```

Remediation

The finding has been acknowledged by Bond Labs. Their official response is reproduced below:

> Bond Labs acknowledges the finding but doesn’t believe it has security implications. However, we may deploy a bug fix to address it.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

4.1 Gas optimization

This section contains our recommendations for gas optimizations.

Repeated external call

Inside the `purchase` function, there are `getAuctioneer` and `getMarketInfoForPurchase` external calls and also internal `_handleTransfers` function call. The `_handleTransfers` function also calls the same functions and gets almost the same values from `getMarketInfoForPurchase`. Since the `_handleTransfers` function is called only from `purchase` to reduce the amount of gas usage due to repeated external calls, you can pass the necessary arguments to the `_handleTransfers` function.

Repeated calculations

Inside the `_createMarket` function, the `lastTuneDebt` and the `capacity` variables are calculated in the same way. Since the `capacity` value is already stored in a local variable, then, to reduce the amount of gas consumed, this variable can be used for the `lastTuneDebt` instead of calculations.

4.2 Error handling

Some functions lack adequate error handling. As an example, the `deploy()` in `BondFixedExpiryTeller` silently fails when a bond token of the `underlying_, expiry` pair has already been deployed. For that reason, we recommend implementing adequate events/error handling wherever the case, such that users are aware of the consequence of their actions after any function interaction.

4.3 The `setIntervals` function

The checks inside the `setIntervals` function do not correspond to the `_createMarket` checks; this can lead to incorrect calculations of the `tuneIntervalCapacity` value.

Inside the `_createMarket` function, the `tuneIntervalCapacity` is calculated using `defau` ltTuneInterval in case that param_.depositInterval is less than defaultTuneInterval.

```
uint256 tuneIntervalCapacity = params_.capacity.mulDiv(
    uint256(
        params_.depositInterval > defaultTuneInterval
            ? params_.depositInterval
            : defaultTuneInterval
    ),
    uint256(secondsToConclusion)
);
```

However, the setIntervals function allows to recalculate the tuneIntervalCapacity using the intervals_[0] value, which can be less than defaultTuneInterval since there is a check only that intervals_[0] is not less than the meta.depositInterval.

```
function setIntervals(uint256 id_, uint32[3] calldata
    intervals_) external override {
        ...
        if (intervals_[0] < intervals_[1]) revert
    Auctioneer_InvalidParams();
        ...
        if (intervals_[0] < meta.depositInterval) revert
    Auctioneer_InvalidParams();
        ...
        meta.tuneIntervalCapacity = market.capacity.mulDiv(
            uint256(intervals_[0]), // 1
            uint256(terms[id_].conclusion) - block.timestamp
        );
        ...
    }
```

To avoid unexpected market behavior, we recommend to check that intervals_[0] is greater than or equal to defaultTuneInterval. Adding such a check will also result in the correct validation of the new tuneAdjustmentDelay value, which should also not be set less than the defaultTuneInterval. Moreover, it is more preferable to store and use the initial secondsToConclusion value for tuneIntervalCapacity calculation.

4.4 Changing market owner affects usage of callback contract

The owner change functionality consists of the two functions pushOwnership and pullOwnership and is allowed for a market that uses a callback contract. Unlike the market that doesn't use a callback, part of the functionality will still be available to the previous owner, such as withdrawing and depositing funds from the `callback` contract, since the previous owner of the market will still be the owner of the callback contract, keeping their full capabilities.

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

/// @inheritdoc IBondAuctioneer
function pullOwnership(uint256 id_) external override {
    if (msg.sender != newOwners[id_]) revert Auctioneer_NotAuthorized();
    markets[id_].owner = newOwners[id_];
}
```

Since the new owner of the market does not have access to the callback contract that is connected to the market, they will be able to use all its essential functionalities, such as withdrawing and depositing funds. Moreover, should the previous market owner turn malicious, they could potentially inflict monetary losses to the callback contract since they still have ownership over it.

Because one callback contract can be linked to several markets, transferring ownership of such a contract to a new owner will not solve the problem but only create new issues.

So for these types of markets, we also recommend changing the address of the callback contract altogether. In this case, however, you also need to make sure that the new owner is added to the `callbackAuthorized` list.

The Bond Labs team describes the intended usage and behavior of the market ownership change, below:

> We do not expect that one market owner will want to transfer the ownership of a market to a different individual/entity. The purpose of the transfer ownership is to be able to deploy markets with one address and potentially manage them with another. Because that is something that would be useful to certain users, forcing a callback change on an ownership change could complicate that. Furthermore, Bond Labs doesn't believe it other security implications.

5 Audit Results

At the time of our audit, the code was not deployed to mainnet EVM.

During our audit, we discovered two findings. Of these, one was of low risk and the remaining was a suggestion (informational). Bond Labs acknowledged all findings and implemented fixes.

5.1 Disclaimers

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any additional code added to the assessed project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.