Zellic

Porter Finance

Smart Contract Security Assessment

April 25, 2022

Prepared for:

Jordan Meyer

Porter Finance

Prepared by:

Chad McDonald and Vlad Toie

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 Introduction

1.1 About Porter Finance

Porter is a protocol that enables creditworthy DAOs to obtain fixed-rate financing using their project tokens as collateral.

We were approached to audit the contracts that handle the issuance and management of these bonds.

1.2 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, etc. 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, etc.

Complex integration risks. Several high-profile exploits have been the result of not any bug within the contract itself, but rather an unintended consequence of its 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, etc.

Code maturity. We review for possible improvements in the codebase in general. We

look for violations of industry best practices and guidelines, or code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, etc.

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 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, project timelines, etc. We aim to provide useful and actionable advice to our partners that consider their long-term goals, rather than simply a list of security issues at present.

1.3 Scope

The engagement involved a review of the following targets:

Porter Contracts

Repository	https://github.com/porter-finance/v1-core
Versions	`1605befb12f2437c806966dc07a95b230271b704`
Contracts	• Bond.sol • BondFactory.sol
Type	Solidity
Platform	Ethereum

1.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants, for a total of 1 person-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:

Chad McDonald, Engineer
chad@zellic.io

Vlad Toie, Engineer
vlad@zellic.io

1.5 Project Timeline

The key dates of the engagement are detailed below.

April 15, 2022	Kick-off call
April 18, 2022	Start of primary review period
April 22, 2022	End of primary review period
TBD	Closing call

1.6 Disclaimer

This assessment does not provide any warranties on finding all possible issues within its scope; i.e., the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees on any additional code added to the assessed project after our assessment has concluded. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program. Finally, this assessment report should not be considered financial or investment advice.

2 Executive Summary

Zellic conducted an audit for Porter Finance from April 18th to April 22nd, 2022 on the scoped contracts and discovered 2 findings. Fortunately, no critical issues were found. We applaud Porter Finance for their attention to detail and diligence in maintaining incredibly high code quality standards in the development of Porter Finance.

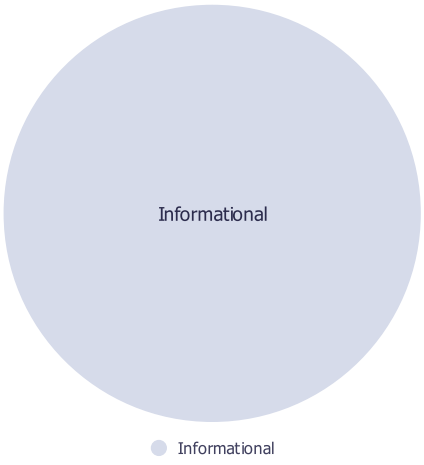
Of the 2 findings, both were informational in nature. Additionally, Zellic recorded its notes and observations from the audit for Porter Finance's benefit at the end of the document.

Zellic thoroughly reviewed the Porter Finance codebase to find protocol-breaking bugs as defined by the documentation, or any technical issues outlined in the Methodology section of this document. Specifically, taking into account Porter's threat model, we focused heavily on issues that would render the bonds unusable for the borrower, or that would not allow lenders to withdraw their redeemable bonds.

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.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	2



3 Detailed Findings

3.1 Borrower protocol fees are not implemented

- **Target**: Bond
- **Category**: Business Logic
- **Likelihood**: N/A
- **Severity**: N/A
- **Impact**: Informational

Description

It is mentioned within the official documentation of the project that there is to be a protocol fee paid to Porter Finance for use of their protocol. However, at the time of writing, this fee is not implemented in the smart contracts.

> Establishment fees are paid by borrowers. This fee is paid to the Porter Treasury.

Impact

Implementing the protocol fee is essential for funding the protocol. Lack of a protocol fee allows anyone to repurpose the code with no drawbacks whatsoever. Although this does not pose a direct security risk to the smart contracts themselves, this may prove to be problematic in the long term for the overall protocol.

Lack of a clear funding mechanism or renumeration also leaves unclear the incentives of a project's developers and team. This may, in certain circumstances, reduce the subjective trustworthiness of a project.

Recommendations

Implement the fees described in the documentation, and/or expand documentation on how the protocol will be funded.

Remediation

The issue has been acknowledged by the Porter team and will be addressed in future development.

3.2 Discrepancies between interfaces and contract code

- **Target**: IBond, IBondFactory
- **Category**: Code Maturity
- **Likelihood**: N/A
- **Severity**: N/A
- **Impact**: Informational

Description

The natspec documentation does not match the current state of the code. Some previously-used variables' or parameters' names were not updated as the code was changed.

In `IBond.sol`, the `accessControl` for the `initialize()` function was replaced with `OwnerUpgradable`, but notes remain about access control in `IBond` for the `initialize()` function. The name of the `owner` parameter was also changed to `bondOwner`.

In `IBondFactory`, the `BondCreated` variable names are also inconsistent with the natspec documentation.

Impact

Outdated documentation can result in developer confusion, potentially leading to future bugs. This is especially true when there are numerous variables and names to consider. In general, outdated documentation impedes auditors and external developers from reading, understanding, and extending the code. The problem would also be carried over if the code is ever forked or re-used.

Recommendations

Consider updating the documentation so that it is consistent with the rest of the code. Also, future code changes should be reflected in documentation, and the two should be kept in sync.

Remediation

The issue was fixed by Porter Finance by implementing our recommendations.

4 Discussion

In this section, we discuss miscellaneous interesting observations discovered during the audit that are noteworthy and merit some consideration.

Credit Risk

The Porter team requested that we also consider attack scenarios that exist outside of the smart contracts.

The most plausible scenarios are that (1) a borrower will purposefully default on their loan, or (2) that the collateral token will drop in value, removing the financial incentive for the borrower to pay off the loan. While defaulting on a loan is not unexpected behavior, it is bad for the credibility of the system. Naturally, it is important that lenders carefully consider a borrower's credibility before purchasing their bonds. However, a high value default or a string of defaults will diminish a lender's trust in Porter.

In the Porter system, similar to conventional bond markets, lenders bear credit risk. If a borrower defaults or if the collateral token backing a loan drops in value, the lender suffers losses. Therefore, it is essential that a lender conduct proper due diligence on a borrower before purchasing bonds. Depending on the skill level of a typical lender, this can be challenging. Conducting proper due diligence may require a review of the borrower's contracts, their financial statements, their security practices, etc.

Porter may want to emphasize the importance of credit risk to its users in its documentation. Porter has provided documentation surrounding the risks associated with purchasing bonds. At launch, Porter will maintain an allowlist of manually vetted borrowers who must meet the following criteria to issue a bond:

- Collateral tokens must have a historical volatility less than 150% over the last 90 days
- Bonds cannot be issued with tenors over 12 months
- Bonds must be at least 250% collateralized

To ensure a smooth launch, it may be advisable to restrict borrowers to those conforming to the above criteria. However, it should be noted that these mitigations are currently not implemented in Porter's contracts. Hence, it is up to the discretion of the Porter team to implement and enforce the criteria. We also considered an additional criteria to be appropiate: namely, ensuring that all issued bonds are sold at time of auction, such that a potentially malicious owner would not be allowed to manipulate the total supply of bonds once the selling phase has concluded.

Re-entrancy

The risk of re-entrancy is carefully mitigated throughout the code base. In the `Bond` contract, all relevant external functions are marked `nonReentrant`. In the `BondFactory` contract, all relevant external functions are permissioned behind access control mechanisms.

Ownership best practices

As the `Bond` contract inherits from the standard OpenZeppelin `OwnableUpgradeable` contract, by default the `renounceOwnership` and `transferOwnership` methods are public. It may be worthwhile to consider overriding and disabling these functions if they are not desired. Additionally, the default ownership transfer procedure does not enforce a two-step process to avoid accidentally transferring ownership to the wrong address. Such a mistake would lead to a lockout of the owner-only functionality. It may be worthwhile to consider overriding and adding this functionality if desired, as ownership transfers are a delicate procedure and should be handled carefully.

Bond tokens may be burned by users directly

The `Bond` contract is a burnable ERC20. By default, the `ERC20BurnableUpgradeable` contract exposes the `burn` function as `public`. In the past, there have been bugs where a token can be transferred or burned directly, bypassing any dedicated functions in a subclassed contract. Nevertheless, we do not believe this poses a security risk for the `Bond` contract, as the contract uses the ERC20 `totalSupply` rather than any internal bookkeeping. Additionally, burning bond tokens is also not economical. Still, it is worthwhile to remain cognizant of this and vigilant for any future bugs.

Code maturity

Overall, the quality of the code is very commendable, and the test coverage reaches 95%. The test suite relies on the state of the Bond contract to test functionality. Each external function is tested for appropriate behavior in the active, paid early, defaulted and paid states.

In addition to unit tests, the project also makes use of the smart contract fuzzing tool Echidna. All the important Bond functionalities have been tested here, namely, creating, converting, redeeming, and paying bonds. We hope that the Porter team continues to demonstrate the diligence and best practices they have adopted in developing the protocol.

The code is also entirely documented, each function having their subsequent documentation within the interface of the contract. This will help future developers adopt and extend the Porter protocol.