PUBLIC

Code Assessment

of the Portfolio Debt Token Smart Contracts

February 22, 2023

Produced for

by

Contents

1 Executive Summary

Dear Archblock and TrueFi Teams,

Thank you for trusting us to help you with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Portfolio Debt Token according to Scope to support you in forming an opinion on their security risks.

Archblock on behalf of TruFi implemented a Portfolio Debt Token - a Solidity smart contract that is intended to be used for distributing assets, recovered from defaulted loans.

The most critical subjects covered in our audit are asset solvency, access control and functional correctness. Security regarding all the aforementioned subjects is high.

The general subjects covered are upgradeability and gas efficiency. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security. This assessment did not uncover any issues that need immediate fixing. However, you might consider addressing the informational findings.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the `contracts/PortfolioDebtToken.sol` source code files inside the `trusttoken/portfolio-debt-token` GitHub repository based on the provided pdf documentation file. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	20 February 2023	02866d10fcd3f1a3e0a6b400a93c163a0c484fe3	Initial Version

For the solidity smart contracts, the compiler version `0.8.17` was chosen.

2.1.1 Excluded from scope

Third-party dependencies, files inside `mocks` folder and any other file not listed above are outside the scope of this review. Imported libraries like OpenZeppelin are considered to be safe and act according to their high-level specification.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Archblock on behalf of TrueFi developed a Portfolio Debt Token, an ERC4626 that is intended as a means of distributing funds recovered from defaulted loans, in a fair way. It is meant to be used with in the TrueFi lending protocol.

In case of a default, lenders who have invested in a portfolio receive `PortfolioDebtTokens` as a representation of their title to the debt. If the funds are recovered, for example as a result of legal action, lenders can get their proportional (pro rata) share.

2.2.1 Portfolio Debt Token

The contract `PortfolioDebtToken` is a tokenized vault contract (`ERC4626`) that is also `Ownable`. The contract is deployed through a `UUPS` upgradeable proxy and the underlying asset is the borrowed asset of a defaulted loan. A new proxy is deployed every time the amount from a defaulted loan can be recovered to reimburse the lenders.

When initialized, the caller must specify the underlying `ERC20` asset, a `mintDeadline` and a `redeemDeadline`, such that `block.timestamp < mintDeadline < redeemDeadline`.

The contract implements a finite state machine with three states: `Mint`, `Redeem`, and `Recover`. The contract starts in the `Mint` state up until the `mintDeadline`, then it enters the `Redeem` state until

`redeemDeadline`, and the contract will be in the `Recover` state after `redeemDeadline`. During each state, a restricted set of actions is allowed:

- `Mint`: the contract's owner can call the function `mintShares` with the addresses and associated shares amount.
- `Redeem`: the minting is blocked, the lenders can call `withdraw()` or `redeem()` to receive their share of the recovered funds.
- `Recover`: withdrawal and redemption by the lenders are blocked, the contract's owner can call the `recover` function. This function can sweep any remaining `ERC20` tokens from the contract, underlying asset or unintentionally sent `ERC20` tokens.

`ERC4626` functionalities:

- the `_transfer` function is disabled, hence disabling the standard `transfer` and `transferFrom` functions
- the functions `maxDeposit` and `maxMint` always return `0`
- the functions `maxWithdraw` and `maxRedeem` return the result from `ERC4626Upgradeable.maxWithdraw/maxRedeem` when the contract' state is `Redeem`, `0` otherwise.

2.2.2 Trust Model

- The owner of the `PortfolioDebtToken` is assumed to be a trusted party since the owner can fully upgrade the implementation of the contract.
- The users are assumed to be untrusted.

2.2.3 Explicit assumptions

The following statements are assumed to be true. Violation of that statement can cause certain problems that Archblock & TrueFi is aware of and described in the documentation.

- The recovered debt is repaid via a single standard ERC20 transfer. Multiple debt repayment installments can lead to a violation of pro rata distribution.
- The assets sent to the `PortfolioDebtToken` cannot be disabled. Thus, disproportionate distribution of non-debt assets is considered OK.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 Debt Recovery

Informational **Version 1**

Debt recovery via single installment is not enforced. The `Redeem` stage of the `PortfolioDebtToken` can start with no assets inside the contract. In addition, the single redemption event is not guaranteed as well. These properties are hard to enforce, but users need to be aware of the potential issues.

6.2 Gas Optimizations

Informational **Version 1**

The function `PortfolioDebtToken.status` does two storage loads (`SLOAD`) upon the assertion `assert(mintDeadline < redeemDeadline);` followed by one or two more `SLOAD` in the `if-else` block. The same result can be achieved, by burning less gas, and by loading only one of the two variables in the assertion.

6.3 Mint and Redeem Stage Durations

Informational **Version 1**

While the checks enforce the proper sequence of `PortfolioDebtToken` statuses (Mint->Redeem->Recover), the actual durations can be too short for the actions to be performed. In the extreme case, the difference between `mintDeadline` and `redeemDeadline` can be less than the difference between timestamps of two consecutive blocks.

6.4 Zero Value Events

Informational **Version 1**

Even though the mint and deposit are disabled via the 0 max value setting, the 0 value deposits can still be performed by the users at any point. In the same way, the 0 value withdrawals and redeems can be performed not in redeem state of the PortfolioDebtToken. While such actions do not change the state of the contract, Deposit and Withdraw events will be still emitted.