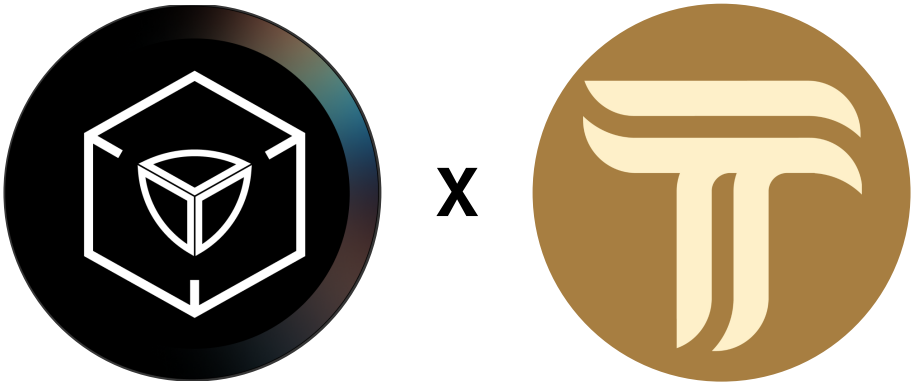
Tharwa Security Review

By Prism



Auditors

Lead Security Researcher: Jacobo Lansac (@Jacopod)

August 5, 2025

Contents

1 Findings Summary

ID	Severity	Issue Title	Team Response
M1	Medium	DoS attack via maturity collision preventing certain types of bonds being purchased	Fixed
L1	Low	Incorrect user bond balances tracking when bonds are traded outside the ‘thBonds‘ contract	Acknowledged

2 Introduction

A time-boxed review of **Tharwa Bonds** smart contracts, developed by Tharwa. The focus of this review is to identify security vulnerabilities, explain their root-cause and provide solutions to mitigate the risk.

Gas optimizations are not the main focus but will also be identified if found.

3 About Prism

Prism delivers specialized security solutions for blockchain and AI companies. We go beyond traditional audits, offering bespoke penetration testing, adversary simulation, and AI security solutions to meet the needs of every client. With tailored services and best-in-class expertise, we safeguard your business against the most sophisticated threats, allowing you to focus on innovation.

Learn more about us at prismsec.xyz

4 About Tharwa

Tharwa Finance aims the first RWA-Collateralized Stablecoin backed by an AI-Driven RWA hedge fund. They aim to do so by tokenizing diversified multi-asset funds to power RWA-backed stablecoin yields.

5 Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time and resource-bound effort to find as many vulnerabilities as possible, but there is no guarantee that all issues will be found. This security review does not guarantee against a hack. Any modifications to the code will require a new security review.

This review does not focus on the correctness of the happy paths. Instead, it aims to identify potential security vulnerabilities and attack vectors derived from an unexpected and harmful usage of the contracts. The devs are ultimately responsible for the correctness of the code and its intended functionality.

A security review is not an endorsement of the underlying business or product and can never be taken as a guarantee that the protocol is bug-free.

6 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

6.1 Impact

- **High**: leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium**: only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low**: can lead to unexpected behavior with some of the protocol's functionalities that are not so critical.

6.2 Likelihood

- **High**: almost certain to happen, easy to perform or highly incentivized.
- **Medium**: only conditionally possible or incentivized, but still relatively likely
- **Low**: requires multiple unlikely conditions, or little-to-no incentive

6.3 Action required for severity levels

- **Critical**: Must fix as soon as possible (if already deployed)
- **High**: Must fix (before deployment if not already deployed)
- **Medium**: Should fix
- **Low**: Could fix

7 Executive Summary

Scope details

Project Name	Tharwa
Type of Project	Bonds, Yield
Repository	tharwa-finance/thBonds/
Review commit	3751b210d42b708c0a19b56df0e374b56bb137b3
Mitigation commit	e93abf5143ef06867c6d77c9c65efb6bb30349e3
Audit Timeline	2025-08-02 - 2025-08-05
Methods	Manual Review, Testing

Files in scope	**nSLOC**
src/TharwaBondVaultV1.sol	143
Total	**143**

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1

8 Architecture review

8.1 Architecture review

8.1.1 Architecture Overview

TharwaBondVaultV1 is an ERC1155 token contract that issues discounted bonds redeemable in THUSD (a stablecoin). Users buy bonds at a discount and can redeem them at full face value after maturity. It essentially creates a time-locked savings product where users lock THUSD for fixed periods to earn yield.

Bond Types:

- 90-day bonds: ~5% APY (price: 0.9878 THUSD)
- 180-day bonds: ~7% APY (price: 0.9666 THUSD)
- 360-day bonds: ~12% APY (price: 0.8941 THUSD)

User Operations:

- purchaseBond(): Buy bonds at discounted prices
- redeemBond(): Redeem matured bonds at full face value
- earlyExit(): Exit before maturity with penalty (0.2% per day, max 20%)

Admin Functions:

- Price/cap adjustments per bond duration, for all issued bonds
- Pause/unpause functionality affecting deposit and redeem
- Token rescue capability
- URI management for metadata

8.1.2 Centralization review

The admin has limited control over the contract, but still has some control:

- Modifying conditions of future bonds (config changes do not affect ongoing bonds)
- Pausing/unpausing deposits and redeems. This means that if paused, users won't be able to redeem their bonds even at maturity
- There is no guarantee enforced in the smart contract that the protocol will be solvent and the bonds will be paid. A large degree of trust in the protocol is needed

9 Findings

9.1 Medium Risk

9.1.1 [M1] - DoS attack via maturity collision preventing certain types of bonds being purchased

Location:

Contract: `TharwaBondVaultV1`, function `purchaseBond()`

Description:

For a given maturity date, there can only be one Duration-bond. Furthermore, there is no minimum `faceAmount` when purchasing a bond, so it is possible to create a `1 wei` bond.

These two things together allow an attacker to perform a DoS attack by purchasing a minimal amount of bonds (1 wei) daily at midnight to prevent other users from buying bonds of different durations that would have the same maturity timestamp. This happens because the `_midnightUTC()` function truncates all timestamps to midnight (intentional), so one bond-duration can deny the existence of the others for that same day.

Example "attack":

- The contract is deployed.
- The attacker starts purchasing every day 1 wei of the 180-day duration bond.
- For the first 90 days, the contract works normally.
- After 90 days, users can only purchase bonds of 180 & 360 days, denying the 90-day bond.
- After 270 days and onwards, users can only purchase bonds of 180, denying both 90 and 360-day bonds.

Impact:

An attacker can prevent legitimate users from purchasing certain types of bonds at a cheap cost.

Mitigation:

- Implement a minimum face amount for bond purchases
- Remove the `_midnightUTC()` truncation and use actual timestamps for maturity (although this has other implications)
- Allow multiple bond durations to have the same maturity date by removing the collision check

Code Context:

```
function purchaseBond(BondDuration duration, uint256 faceAmount) external whenNotPaused nonReentrant {
    if (faceAmount == 0) revert ZeroAmount();
    // ...

    uint256 maturity = _midnightUTC(block.timestamp + series.duration);

    if (maturitySeries[maturity] != duration && totalSupply(maturity) > 0) {
        revert MaturityCollision();
    }

    // ...
}
```

Response: Fixed

The team added a minimum `faceAmount` of $10 which makes the attack impractical.

9.2 Low Risk

9.2.1 [L1] - Incorrect user bond balances tracking when bonds are traded outside the `thBonds` contract

Location:

Contract: `TharwaBondVaultV1`, function `purchaseBond()`

Description:

The `_userOwnedBonds` mapping is only updated within the `TharwaBondVaultV1` contract operations, (during bond purchases, early exits, and redemptions). However, since bonds are ERC1155 tokens, they can be traded on NFT marketplaces using standard transfers. When bonds are traded externally, the `_userOwnedBonds` mapping becomes outdated and inaccurate.

Impact:

The `getBondsForUser()` function will return incorrect data when bonds are traded on NFT marketplaces. While this is not a critical risk, it can cause confusion and ultimately bad UX. When a seller sells an ERC1155 bond in an NFT marketplace:

- the seller will still see the bond inside the return from `getBondsForUser()`.
- the buyer won't see his new bond in that same function.

Mitigation:

- Remove reliance on `_userOwnedBonds` mapping and use `ERC1155.balanceOf()` for all possible bond tokens instead
- Implement an off-chain indexer that tracks ERC1155 transfer events to display which bonds are owned by the user

Code Context:

```
function purchaseBond(BondDuration duration, uint256 faceAmount) external whenNotPaused nonReentrant {
    // ...

    _mint(msg.sender, maturity, faceAmount, "");
    // @audit this mapping will be inaccurate if the bonds are acquired outside this contract
    _userOwnedBonds[msg.sender].add(maturity);

    // ...
}
```

Response: Acknowledged

The team doesn't use that view function to track user's bonds.

9.3 Informationals

10 Informational Issues

10.1 1. Harmless early exit with zero balance

Location:

Contract: `TharwaBondVaultV1`, function `earlyExit()`

Description:

It's possible to call `earlyExit()` with a tokenId for which the user has no balance. The function won't revert and will transfer 0 THUSD while applying 0 penalty. While this doesn't pose a security risk, it can potentially pollute indexers with meaningless transactions.

Impact:

Minimal impact - only creates noise in transaction logs and indexers.

Code Context:

```
function earlyExit(uint256 tokenId, uint256 faceAmount) external whenNotPaused returns (uint256 payout)
↪  {
    if (block.timestamp >= tokenId) revert BondMatured();

    _burn(msg.sender, tokenId, faceAmount);
    // ...
}
```

10.2 2. Harmless redeem with zero balance

Location:

Contract: `TharwaBondVaultV1`, function `redeemBond()`

Description:

It's possible to call `redeemBond()` with a non-existing tokenId (where user has 0 balance). The function won't revert and will transfer 0 THUSD. Similar to the early exit case, this can pollute indexers but poses no security risk.

Impact:

Minimal impact - only creates noise in transaction logs and indexers.

Code Context:

```
function redeemBond(uint256 tokenId) external whenNotPaused {
    if (block.timestamp < tokenId) revert BondNotMatured();
    uint256 bal = balanceOf(msg.sender, tokenId);

    _burn(msg.sender, tokenId, bal);
    // ...
}
```

10.3 3. Redundant condition check in redeemBond

Location:

Contract: `TharwaBondVaultV1`, function `redeemBond()`

Description:

The condition `if (balanceOf(msg.sender, tokenId) == 0)` will always be true after burning the full balance, making the condition check redundant. The tokenId will always be removed from `_userOwnedBonds`.

Impact:

No functional impact - the code works as intended but contains unnecessary logic.

Code Context:

```
function redeemBond(uint256 tokenId) external whenNotPaused {
    // ...
    _burn(msg.sender, tokenId, bal);
+   // as the full balance is burned, the msg.sender can be safely removed
+   _userOwnedBonds[msg.sender].remove(tokenId);
-   if (balanceOf(msg.sender, tokenId) == 0) {
-       _userOwnedBonds[msg.sender].remove(tokenId);
-   }
    // ...
}
```

10.4 4. APY calculation mismatch between comments and actual implementation

Location:

Contract: `TharwaBondVaultV1`, Constructor function

Description:

The comment states "For 12% APY at 360 days" but the actual price value `894169800200000000` (0.8941e18) results in an APY of approximately 11.83%. Similar discrepancies exist for other bond durations where comments don't match the actual APY calculations.

Code Context:

```
constructor(address thusd_, uint256 initialCap)
↪   ERC1155("https://bondmeta.tharwa.finance/v1/bond/{id}.json") {
    // ...

    // For 12% APY at 360 days
    bondSeries[BondDuration.ThreeSixtyDays] = BondSeries({
        price: 894169800200000000, // 0.8941e18
        duration: 360 days,
        cap: initialCap,
        outstanding: 0
    });
}
```