Bond

Smart Contract Audit Report Prepared for Thorus

Date Issued:	Dec 18, 2021
Project ID:	AUDIT2021053
Version:	v1.0
Confidentiality Level:	Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021053
Version	v1.0
Client	Thorus
Project	Bond
Auditor(s)	Weerawat Pawanawiwat Patipon Suwanbol
Author	Patipon Suwanbol
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Dec 18, 2021	Full report	Patipon Suwanbol

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Thorus, Inspex team conducted an audit to verify the security posture of the Bond smart contracts between Dec 15, 2021 and Dec 16, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Bond smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 2 low-severity issues. With the project team's prompt response, 1 low-severity issue was resolved in the reassessment, while another 1 low-severity issue was acknowledged by the team. Therefore, Inspex trusts that Bond smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

This smart contract passes
Inspex's security verification
standard, and is trustworthy.
Approved by Inspex on Dec 18, 2021
inspex
CYBERSECURITY
PROFESSIONAL
SERVICE
PASS

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Thorus is an all in one cross-chain DeFi 2.0 Platform with an adaptable treasury system, and a token holder first approach. All protocol functions are designed to reinforce this mentality. Each feature is part of an ecosystem that continually drives value back to the THO token, benefiting holders and stakers above all.

Bond is a smart contract that allows the platform's users to buy the $THO as bonds. $THO will be paid out to the buyers over time linearly.

Scope Information:

Project Name	Bond
Website	https://thorus.fi
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Dec 15, 2021 - Dec 16, 2021
Reassessment Date	Dec 17, 2021

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit:

Contract	Location (URL)	SHA256 Checksum
ThorusBond	https://public.inspex.co/audit/Thorus_Bond/ThorusBond.sol	6ac2ebb3748784af9c45fa47e546f5ce4e56161fdfaf35a2eecc04cf44c36d96

Please note that the initial audit was done on a contract in Thorus's private repository. The file has been uploaded to Inspex's storage for public access.

Reassessment: (Commit: e959bb8a18a6b7f4a44749d2e33b98e862f61b3e)

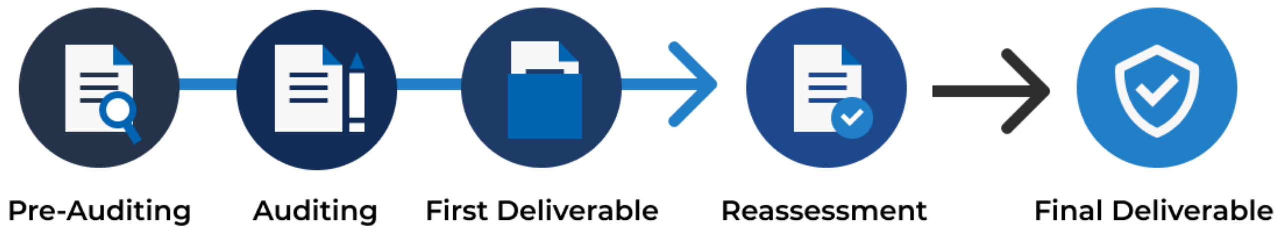
Contract	Location (URL)
ThorusBond	https://github.com/ThorusFi/contracts/blob/e959bb8a18/ThorusBond.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

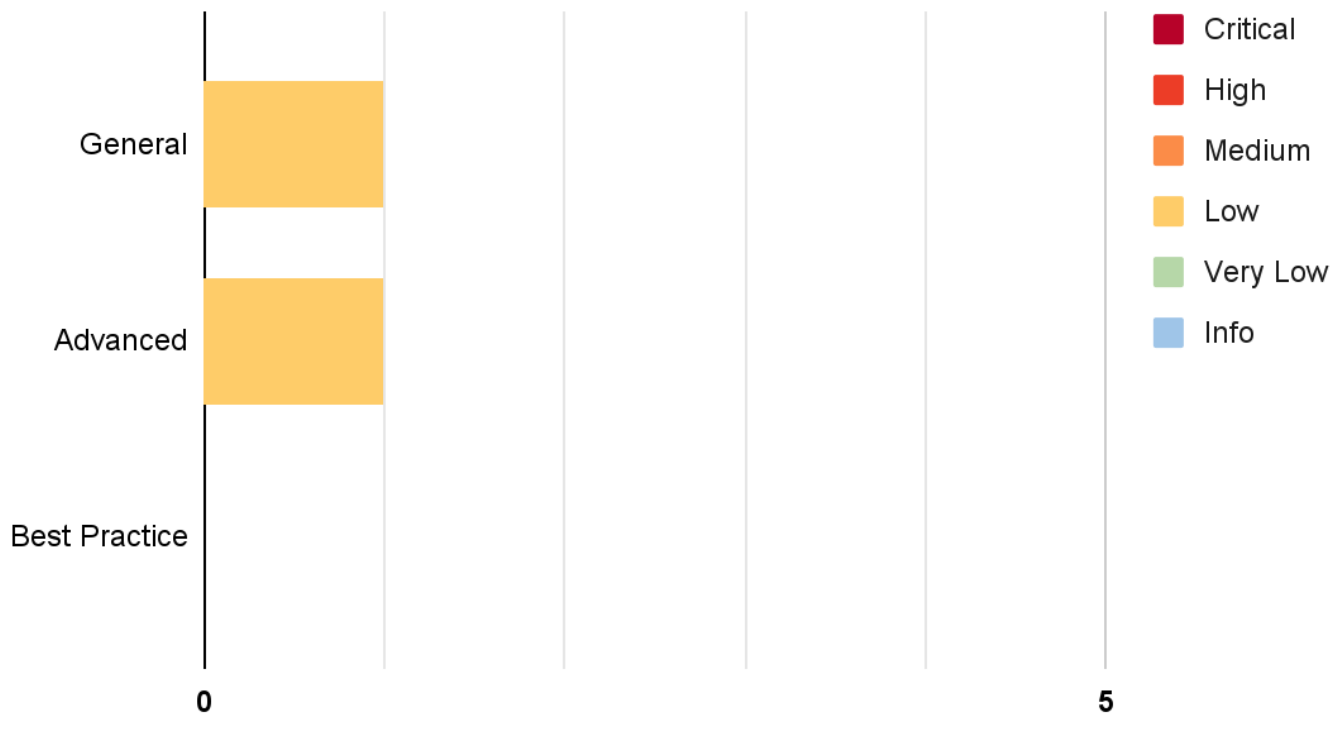
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Likelihood / Impact	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

4. Summary of Findings

From the assessments, Inspex has found 2 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Centralized Control of State Variables	General	**Low**	**Acknowledged**
IDX-002	Improper Reset of Claimable Amount	Advanced	**Low**	**Resolved**

* The mitigations or clarifications by Thorus can be found in Chapter 5.

5. Detailed Findings Information

5.1. Centralized Control of State Variables

ID	IDX-001
Target	ThorusBond
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Low **Impact:** Medium The controlling authority can change the critical state variables to gain additional profit. Thus, it is unfair to the users. **Likelihood:** Low There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner. Nevertheless, the benefit in doing this compared to the risk of reputation loss for the platform results in low motivation for the malicious action.
Status	**Acknowledged** Thorus team has acknowledged this issue as they want to have a maximum efficient bonding mechanism. The price rates of the newly purchased bonds will be adjusted by an external oracle and manually set metrics, depending on the staking profits. Having a 24 hours timelock is not suitable for the current business design, as it is impossible for the oracle to predict the prices of $THO for the next day. Furthermore, the states modified do not have any effect on the bonds already purchased.

5.1.1. Description

Critical state variables can be updated any time by the controlling authority. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The contract owner can set the `thorusPerPrincipal` and `_ratioPrecision` state variables through the `setRatio()` function, which is the factors to calculate the amount of $THO distributed to the users.

ThorusBond.sol

```
function setRatio(uint256 _thorusPerPrincipal, uint256 _ratioPrecision) external onlyOwner {
    emit RatioChanged(thorusPerPrincipal, _thorusPerPrincipal, ratioPrecision, _ratioPrecision);
    require(_thorusPerPrincipal != 0, 'ratio cant be zero');
    thorusPerPrincipal = _thorusPerPrincipal;
    require(_ratioPrecision != 0, 'precision cant be zero');
    ratioPrecision = _ratioPrecision;
}
```

Therefore, it is possible for the contract owner to set the price manually to benefit from this ability to control the price.

5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. Therefore, the `setRatio()` function should be removed.

However, if modifications are needed, Inspex suggests mitigating this issue by limiting the use of this function using a timelock mechanism to delay the changes for a reasonable amount of time, e.g., 24 hours.

5.2. Improper Reset of Claimable Amount

ID	IDX-002
Target	ThorusBond
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Medium The amount of the user's currently claimable $THO will be reset to zero, regardless of the duration passed since the previous deposit. This causes monetary loss for the user and reputation damage to the platform. **Likelihood:** Low The users can claim their reward before depositing the principal to be unaffected by this issue.
Status	Resolved Thorus team has resolved this issue in commit `b9b76ec59e611aaea68584ae5f9506809cce45e9` by calling the `claim()` function in the `deposit()` function if the user has a pending claimable payout.

5.2.1. Description

In the ThorusBond contract, the user can deposit the principal token to get $THO linearly over the vesting period.

The amount of currently claimable $THO is determined using the duration of time passed since the last interaction, calculated in line 539, compared to the remaining vesting duration. If the vesting duration has been exceeded, the whole payout can be claimed as seen in line 543, if not, the amount is calculated in line 546.

ThorusBond.sol

```
537 function claim(bool autoStake) external returns (uint256) {
538     UserInfo memory info = userInfo[msg.sender];
539     uint256 secondsSinceLastInteraction = block.timestamp -
info.lastInteractionSecond;
540     uint256 payout;
541 
542     if(secondsSinceLastInteraction >= info.remainingVestingSeconds) {
543         payout = info.remainingPayout;
544         delete userInfo[msg.sender];
545     } else {
```

```
546         payout = info.remainingPayout * secondsSinceLastInteraction / info.remainingVestingSeconds;
547         userInfo[msg.sender] = UserInfo({
548             remainingPayout: info.remainingPayout - payout,
549             remainingVestingSeconds: info.remainingVestingSeconds - secondsSinceLastInteraction,
550             lastInteractionSecond: block.timestamp
551         });
552     }
553 
554     if(autoStake) {
555         IERC20(thorus).approve(staking, payout);
556         IAutoStake(staking).deposit(msg.sender, payout);
557     } else {
558         IERC20(thorus).safeTransfer(msg.sender, payout);
559     }
560 
561     emit Claim(msg.sender, payout, autoStake);
562     return payout;
563 }
```

However, if the user has previously deposited without claiming, the current claimable amount will be reset to zero when the user deposits again, since the `lastInteractionSecond` is set to the current timestamp in line 521.

ThorusBond.sol

```
508 function deposit(uint256 amount) external returns (uint256) {
509     uint256 payout;
510     payout = amount * thorusPerPrincipal / ratioPrecision;
511 
512     require(payout > 0, "too small");
513     require(thorusAvailableToPay >= payout, "sell out");
514 
515     IERC20(principal).safeTransferFrom(msg.sender, treasury, amount);
516     totalPrincipalReceived += amount;
517     thorusAvailableToPay -= payout;
518     userInfo[msg.sender] = UserInfo({
519         remainingPayout: userInfo[msg.sender].remainingPayout + payout,
520         remainingVestingSeconds: vestingSeconds,
521         lastInteractionSecond: block.timestamp
522     });
523 
524     emit Deposit(msg.sender, amount, payout);
525     return payout;
526 }
```

Therefore, the users will not be able to claim the amount they are eligible for regardless of the duration passed since the previous deposit.

5.2.2. Remediation

Inspex suggests adding the logic to claim $THO for the user in the **`deposit()`** function to clear the pending claimable amount before updating the state with the new amount. For example, adding the **`autoStake`** parameter to the **`deposit()`** function and adding the claiming code in line 515 - 517:

ThorusBond.sol

```
508  function deposit(uint256 amount, bool autoStake) external returns (uint256) {
509      uint256 payout;
510      payout = amount * thorusPerPrincipal / ratioPrecision;
511  
512      require(payout > 0, "too small");
513      require(thorusAvailableToPay >= payout, "sell out");
514  
515      if (userInfo[msg.sender].remainingPayout > 0 && block.timestamp >
     userInfo[msg.sender].lastInteractionSecond) {
516          claim(autoStake);
517      }
518  
519      IERC20(principal).safeTransferFrom(msg.sender, treasury, amount);
520      totalPrincipalReceived += amount;
521      thorusAvailableToPay -= payout;
522      userInfo[msg.sender] = UserInfo({
523          remainingPayout: userInfo[msg.sender].remainingPayout + payout,
524          remainingVestingSeconds: vestingSeconds,
525          lastInteractionSecond: block.timestamp
526      });
527  
528      emit Deposit(msg.sender, amount, payout);
529      return payout;
530  }
```

Please note that the visibility of the **`claim()`** function should be changed to **`public`** to make it callable from the **`deposit()`** function.

6. Appendix

6.1. About Inspex

CYBERSECURITY
PROFESSIONAL
SERVICE

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] "OWASP Risk Rating Methodology." [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

inspex

CYBERSECURITY PROFESSIONAL SERVICE