AutoStake, ThorusMaster & Token

Smart Contract Audit Report Prepared for Thorus

Date Issued:	Jan 6, 2022
Project ID:	AUDIT2021054
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Public

Report Information

Project ID	AUDIT2021054
Version	v1.0
Client	Thorus
Project	AutoStake, ThorusMaster & Token
Auditor(s)	Weerawat Pawanawiwat Patipon Suwanbol
Author	Patipon Suwanbol
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Jan 6, 2022	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 AutoStake, ThorusMaster & Token smart contracts between Dec 23, 2021 and Dec 24, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of AutoStake, ThorusMaster & Token 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 and 1 info-severity issues. With the project team's prompt response, 1 low-severity issue was clarified in the reassessment, while 1 low and 1 info-severity issues were acknowledged by the team. Therefore, Inspex trusts that AutoStake, ThorusMaster & Token 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.

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.

AutoStake, ThorusMaster & Token contracts allow the platform's users to stake their tokens to perform yield farming and gain $THO as a reward. The AutoStake contract allows $THO to be staked in order to compound the farming reward for additional $THO.

Scope Information:

Project Name	AutoStake, ThorusMaster & Token
Website	https://thorus.fi
Smart Contract Type	Ethereum Smart Contract
Chain	Avalanche
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Dec 23, 2021 - Dec 24, 2021
Reassessment Date	Jan 5, 2022

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: (Commit: ea34f1b89a39b89ae12ea6a0a5be0cad81cf464e)

Contract	Location (URL)
ThorusAutoStake	https://github.com/ThorusFi/contracts/blob/ea34f1b89a/ThorusAutoStake.sol
ThorusMaster	https://github.com/ThorusFi/contracts/blob/ea34f1b89a/ThorusMaster.sol
ThosrusToken	https://github.com/ThorusFi/contracts/blob/ea34f1b89a/ThorusToken.sol

Reassessment: (Commit: ea34f1b89a39b89ae12ea6a0a5be0cad81cf464e)

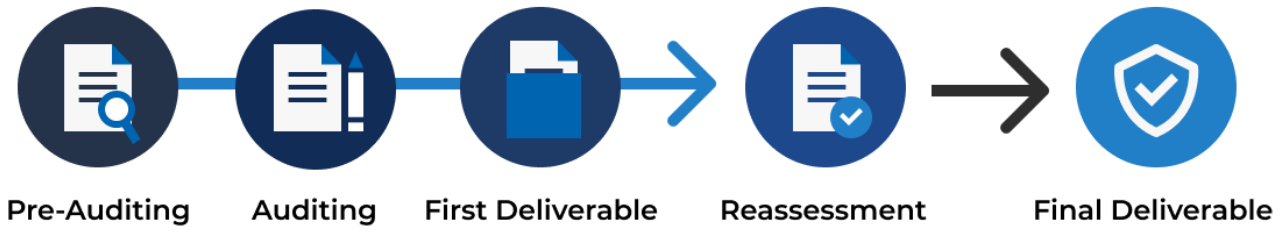
Contract	Location (URL)
ThorusAutoStake	https://github.com/ThorusFi/contracts/blob/ea34f1b89a/ThorusAutoStake.sol
ThorusMaster	https://github.com/ThorusFi/contracts/blob/ea34f1b89a/ThorusMaster.sol
ThosrusToken	https://github.com/ThorusFi/contracts/blob/ea34f1b89a/ThorusToken.sol

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

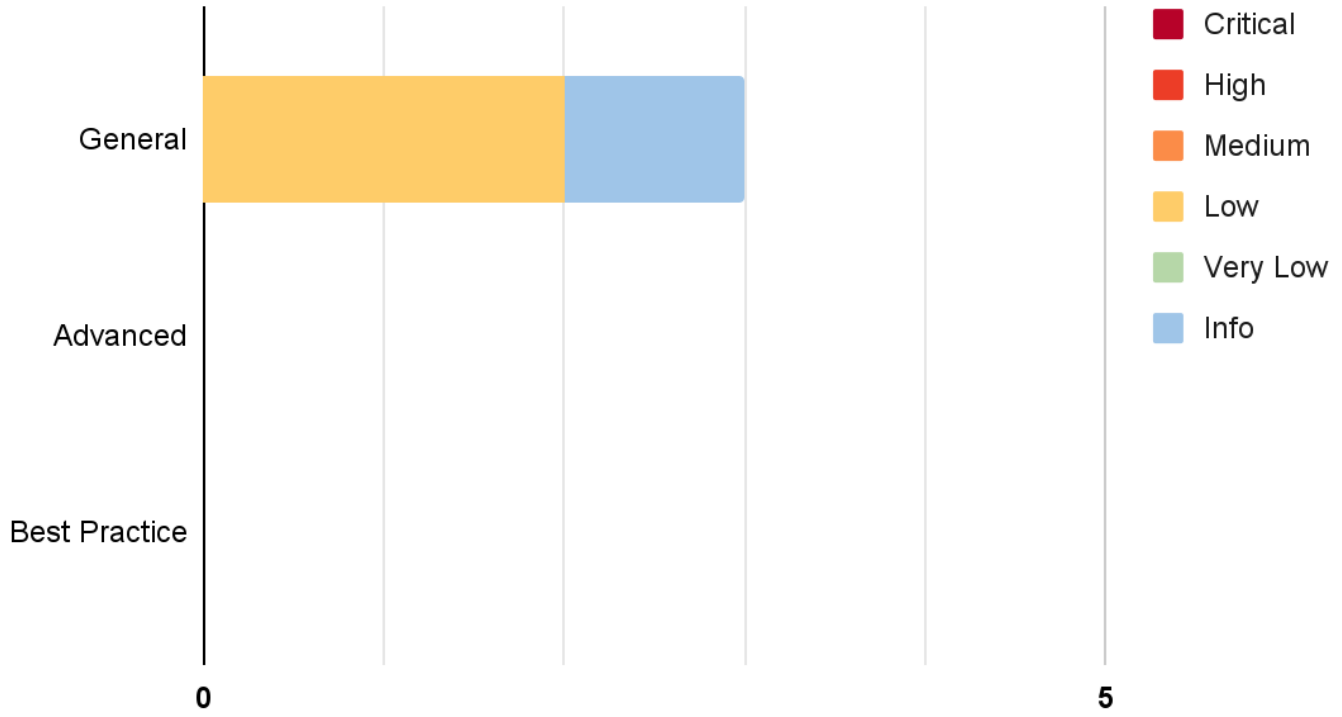
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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**.

Impact \ Likelihood	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 3 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 Variable	General	**Low**	**Acknowledged**
IDX-002	Design Flaw in massUpdatePools() Function	General	**Low**	**Resolved ***
IDX-003	Outdated Compiler Version	General	**Info**	**No Security Impact**

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

5. Detailed Findings Information

5.1 Centralized Control of State Variable

ID	IDX-001
Target	ThorusMaster ThorusAutoStake
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Low **Impact:** Medium The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood:** Low There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner and there are boundaries set for the privileged functions. Hence, by doing malicious actions the platform's owner will likely get few rewards which are not worth it compared to the loss of the platform's reputation.
Status	**Acknowledged** ThorusFinance has acknowledged this issue since using timelock mechanism for the privileged function affects to the business model of the platform. However, as the privileged functions are applied with the boundaries, there will be a tiny room for the profit for doing malicious actions which are not worth it compared to the loss of a platform's reputation.

5.1.1 Description

Critical state variables can be updated any time by the controlling authorities. 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 controllable privileged state update functions are as follows:

Target	Contract	Function	Modifier
ThorusMaster.sol (L: 573)	ThorusMaster	add()	onlyOwner
ThorusMaster.sol (L: 591)	ThorusMaster	set()	onlyOwner

ThorusMaster.sol (L: 716)	ThorusMaster	setThorusPerSecond()	onlyOwner
ThorusAutoStake.sol (L: 1102)	ThorusAutoStake	setPerformanceFee()	onlyOwner
ThorusAutoStake.sol (L: 1108)	ThorusAutoStake	setCallFee()	onlyOwner
ThorusAutoStake.sol (L: 1114)	ThorusAutoStake	setWithdrawFee()	onlyOwner
ThorusAutoStake.sol (L: 1120)	ThorusAutoStake	setWithdrawFeePeriod()	onlyOwner

5.1.2 Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a mechanism to delay the changes for a reasonable amount of time

Please note that the `emergencyWithdraw()`, `pause()` and `unpause()` functions are considered as emergency functions to prevent immediate malicious actions which could damage the platform and the platform users. If the mitigation is done by using a timelock mechanism, Inspex suggests applying a new privilege modifier to exclude them from `onlyOwner`'s timelock effect, for example, adding the `onlyOperator` privilege role for those functions.

5.2 Design Flaw in massUpdatePools() Function

ID	IDX-002
Target	ThorusMaster
Category	General Smart Contract Vulnerability
CWE	CWE-400: Uncontrolled Resource Consumption
Risk	**Severity:** Low **Impact:** Medium The `massUpdatePools()` function will eventually be unusable due to excessive gas usage, causing disruption to the service of the platform. **Likelihood:** Low It is very unlikely that the `poolInfo` size will be raised until the `massUpdatePool()` is eventually unusable.
Status	Resolved * ThorusFinance has confirmed that the platform will not add that huge amount of pools, which means there will be no scenario to make the `massUpdatePools()` function unusable.

5.2.1 Description

The `massUpdatePools()` function executes the `updatePool()` function, which is a state modifying function for all added farms as shown below:

ThorusMaster.sol

```
610 function massUpdatePools() public {
611     uint256 length = poolInfo.length;
612     for (uint256 pid = 0; pid < length; ++pid) {
613         updatePool(pid);
614     }
615 }
```

With the current design, the added pools cannot be removed. They can only be disabled by setting the `pool.allocPoint` to 0. Even if a pool is disabled, the `updatePool()` function for this pool is still called. Therefore, if new pools continue to be added to this contract, the `poolInfo.length` will continue to grow and this function will eventually be unusable due to excessive gas usage.

5.2.2 Remediation

Inspex suggests making the contract capable of removing unnecessary or ended pools to reduce the loop round in the `massUpdatePools()` function.

5.3 Outdated Compiler Version

ID	IDX-003
Target	ThorusToken ThorusMaster ThorusAutoStake
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact ThorusFinance has acknowledged this issue since the contracts are already deployed and there are known bugs affected to the code.

5.3.1 Description

The Solidity compiler versions specified in the smart contracts were outdated. There are improvements and bug fixes applied in the newer version, so it is more suitable to be used.

ThorusMaster.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.10;
```

The outdated Solidity compilers for the contracts are as follows:

Contract	Solidity Compiler Version
ThorusToken	0.8.10
ThorusMaster	0.8.10
ThorusAutoStake	0.8.10

5.3.2 Remediation

Inspex suggests upgrading the Solidity compiler to the latest stable version[2].

During the audit activity, the latest stable versions of Solidity compiler in major 0.8 (0.8.x) is 0.8.11.

6. Appendix

6.1. About Inspex

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]

[2] "Releases — Ethereum Solidity Releases" [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 28-December-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE