Lottery

Smart Contract Audit Report Prepared for Thorus

Date Issued:	Feb 21, 2022
Project ID:	AUDIT2022011
Version:	v1.0
Confidentiality Level:	Public

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022011
Version	v1.0
Client	Thorus
Project	Lottery
Auditor(s)	Peeraphut Punsuwan Wachirawit Kanpanluk
Author(s)	Wachirawit Kanpanluk
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Feb 21, 2022	Full report	Wachirawit Kanpanluk

Contact Information

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

1. Executive Summary **1**

1.1. Audit Result 1

1.2. Disclaimer 1

2. Project Overview **2**

2.1. Project Introduction 2

2.2. Scope 3

3. Methodology **4**

3.1. Test Categories 4

3.2. Audit Items 5

3.3. Risk Rating 6

4. Summary of Findings **7**

5. Detailed Findings Information **9**

5.1. Arbitrary Update of rewardOffered 9

5.2. Centralized Control of State Variable 11

5.3. Unwithdrawable Excessive Reward 13

6. Appendix **15**

6.1. About Inspex 15

6.2. References 16

1. Executive Summary

As requested by Thorus, Inspex team conducted an audit to verify the security posture of the Lottery smart contracts on Feb 15, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Lottery 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 1 medium, 1 low, and 1 info-severity issues. With the project team's prompt response, 1 medium and 1 info-severity issues were resolved in the reassessment, while 1 low-severity issue was acknowledged by the team. Therefore, Inspex trusts that Lottery 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 Feb 21, 2022

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.

Lottery is a new feature of Thorus platform, an automated lottery that allows the platform's users to use $THO to buy tickets and gain $DAI as a reward with fairness and transparency through blockchain technology.

Scope Information:

Project Name	Lottery
Website	https://thorus.fi/
Smart Contract Type	Ethereum Virtual Machine
Chain	Avalanche
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Feb 15, 2022
Reassessment Date	Feb 18, 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: 9713802c5cfbb8b6cf02f958f53a58df10ca1a76)

Contract	Location (URL)
ThorusLottery	https://github.com/ThorusFi/contracts/blob/9713802c5c/ThorusLottery.sol

Reassessment: (Commit: 0e8b423dfef520d9a0460beffd262262d36eeb03)

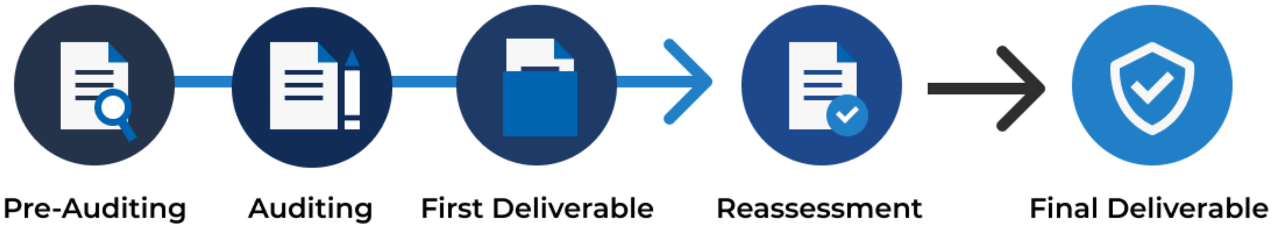
Contract	Location (URL)
ThorusLottery	https://github.com/ThorusFi/contracts/blob/0e8b423dfe/ThorusLottery.sol

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
Centralized Control of State Variable
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

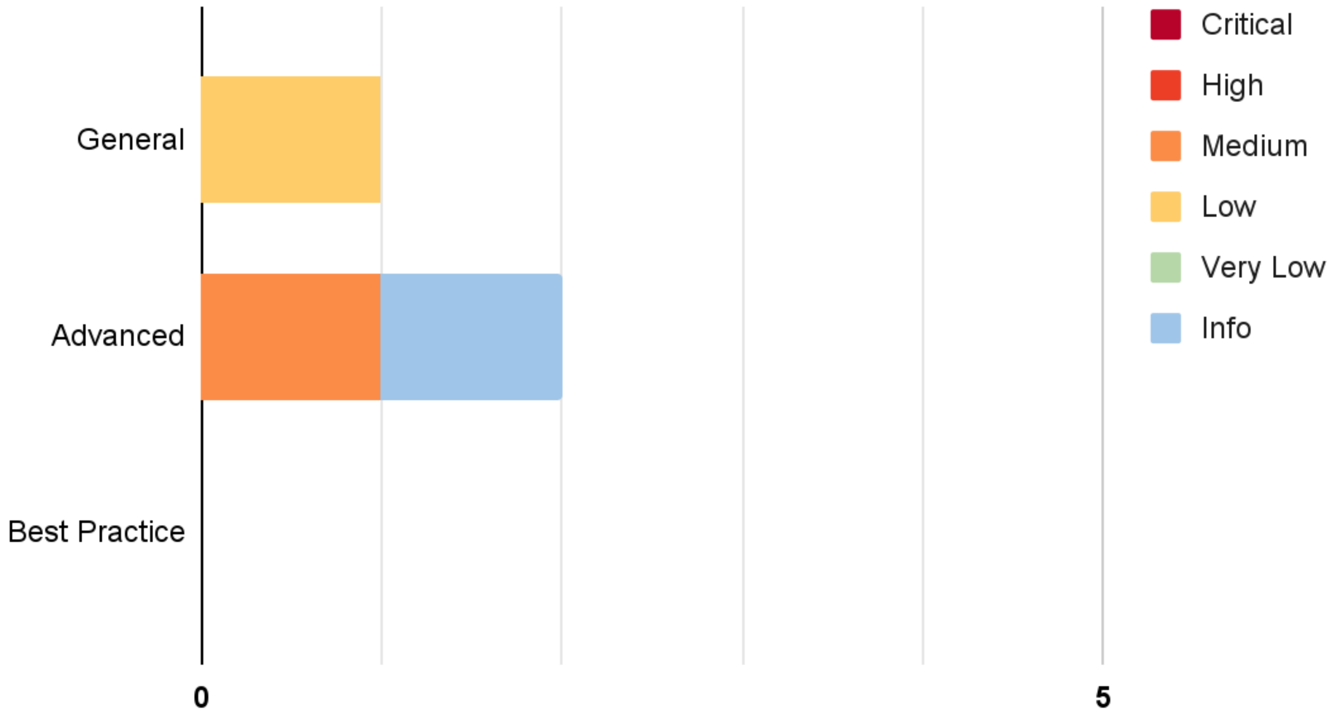
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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 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	Arbitrary Update of rewardOffered	Advanced	**Medium**	Resolved
IDX-002	Centralized Control of State Variable	General	**Low**	Acknowledged
IDX-003	Unwithdrawable Excessive Reward	Advanced	**Info**	Resolved

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

5. Detailed Findings Information

5.1. Arbitrary Update of rewardOffered

ID	IDX-001
Target	ThorusLottery
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** High The `rewardOffered` state can be updated by the owner only, resulting in the rewards of the winning ticket can be lower than the user accepts while buying the ticket. **Likelihood:** Low The rewards that the owner has transferred to the `ThorusLottery` contract will be unwithdrawable, As a result, it is a low motivation and no profit for the platform owner to execute this scenario.
Status	Resolved The Thorus team has resolved this issue by adding the condition which checks the new `rewardOffered` is not lower than the current `rewardOffered` value in commit `0e8b423dfef520d9a0460beffd262262d36eeb03`.

5.1.1. Description

When the users intend to buy the lottery tickets, they expect the reward that the platform offers (`rewardOffered`). However, the `rewardOffered` can be updated anytime by the platform owner, resulting in the rewards of the users can be less than the reward offered at the time that users have been buying.

ThorusLottery.sol

```
function setRewardOffered(uint256 _rewardOffered) external onlyOwner {
    require(!buyingAllowed, "buying still allowed");
    require(!claimingAllowed, "claiming already allowed");
    require(dai.balanceOf(address(this)) >= _rewardOffered, "transfer needed funds first!");

    rewardOffered = _rewardOffered;
    emit RewardSet();
}
```

However, there is no motivation for the platform owner to change the `rewardOffered` state lower than the current `rewardOffered` state because there is no implementation to withdraw that leftover in the contract. This means the rewards that the platform owner transferred will be stuck on the contract forever.

5.1.2. Remediation

For the `setRewardOffered()` function, the platform owner can increase the rewards by setting the reward for the users (`rewardOffered`). This reward value is used to motivate the users to participate in the lottery.Hence, to be fair, the new rewards offered should not be lower than the current rewards.

Inspex suggests verifying whether the new `rewardOffered` state should not be lower than the current `rewardOffered` value.

ThorusLottery.sol

```
function setRewardOffered(uint256 _rewardOffered) external onlyOwner {
    require(!buyingAllowed, "buying still allowed");
    require(!claimingAllowed, "claiming already allowed");
    require(dai.balanceOf(address(this)) >= _rewardOffered, "transfer needed funds first!");
    require(_rewardOffered > rewardOffered, "The new rewards should be more than the current rewards");

    rewardOffered = _rewardOffered;
    emit RewardSet();
}
```

5.2. Centralized Control of State Variable

ID	IDX-002
Target	ThorusLottery
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 by the owner. However, only some owner roles can call these functions to change the states.
Status	**Acknowledged** The state variables can be updated at any time by the controlling authorities. However, there is no profit to motivate the platform owner to update these state variables' value.

5.2.1. Description

The state variables can be updated at 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, as the contract is not yet deployed, there is potentially no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier
ThorusLottery.sol (L:617)	ThorusLottery	allowBuying()	onlyOwner
ThorusLottery.sol (L:625)	ThorusLottery	disallowBuying()	onlyOwner
ThorusLottery.sol (L:632)	ThorusLottery	setRewardOffered()	onlyOwner
ThorusLottery.sol (L:641)	ThorusLottery	allowClaiming()	onlyOwner
ThorusLottery.sol (L:686)	ThorusLottery	settleRandomResult()	onlyOwner
ThorusLottery.sol (L:802)	ThorusLottery	withdrawThorus()	onlyOwner

5.2.2. 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 `Timelock` contract to delay the changes for a sufficient amount of time

The `withdrawThorus()` function has no direct impact to the users, so if the platform decides to mitigate this issue with the `Timelock` contract, it is suggested changing the privilege role from `onlyOwner` to other privilege role such as `onlyOperator` role.

5.3. Unwithdrawable Excessive Reward

ID	IDX-003
Target	ThorusLottery
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Thorus team has resolved this issue by checking the value in the `allowClaiming()` function when the value of `rewardOffered` is greater than the `dai.balanceOf(address(this))`, it then transfers the excess reward token in the contract to the treasury in commit `0e8b423dfef520d9a0460beffd262262d36eeb03`.

5.3.1. Description

In the `ThorusLottery` contract, the $DAI, which is used as the reward, is directly transferred from the platform owner. Then, the platform owner will execute the `setRewardOffered()` to initialize the prize pool as shown below:

ThorusLottery.sol

```
function setRewardOffered(uint256 _rewardOffered) external onlyOwner {
    require(!buyingAllowed, "buying still allowed");
    require(!claimingAllowed, "claiming already allowed");
    require(dai.balanceOf(address(this)) >= _rewardOffered, "transfer needed funds first!");

    rewardOffered = _rewardOffered;
    emit RewardSet();
}
```

While initializing the prize pool, the `setRewardOffered()` verifies whether the current balance of $DAI is sufficient to pay off to the users or not. Therefore, if the balance of $DAI is greater than the `rewardOffered` value (prize pool), the $DAI will be left in the contract and the owner can not withdraw this left-over $DAI since the `ThorusLottery` contract does not have the withdraw function for $DAI.

5.3.2. Remediation

Inspex suggests withdrawing the exceed $DAI in the `ThorusLottery` contract before allowing users to claim the rewards, for example:

ThorusLottery.sol

```
function allowClaiming() external onlyOwner {
    require(!claimingAllowed, "claiming already allowed");
    require(ticketsWithdrawn, "tickets not yet withdrawn");
    require(rewardOffered > 0, "reward not yet set");
    uint256 excessAmount = dai.balanceOf(address(this)) - rewardOffered;
    if(excessAmount > 0)
        dai.safeTransfer(treasury, excessAmount);

    claimingAllowed = true;
    emit ClaimingStarted();
}
```

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]

inspex

CYBERSECURITY PROFESSIONAL SERVICE