Cryptonics

LoTerra Smart Contracts - Audit Report

April 12, 2021

This audit has been performed by

Philip Stanislaus and **Stefan Beyer**

Cryptonics Consulting S.L.
Ramiro de Maeztu 7
46022 Valencia
SPAIN

https://cryptonics.consulting/

Table of Contents

Table of Contents 2

Disclaimer 3

Introduction 4

Purpose of this Report 4

Codebase Submitted for the Audit 4

Methodology 5

Functionality Overview **5**

How to read this Report **6**

Summary of Findings **7**

Code Quality Criteria 9

Detailed Findings **10**

CosmWasm loterra-contract Smart Contract 10

Play message could be blocked 10

Poll can be blocked by Sybil attack 10

Jackpot message is inefficient and could be blocked 11

Vote weight calculation is unbounded and could be blocked 11

Vote message is inefficient and could be blocked 12

Winners can only claim jackpot until the next winners are drawn 12

Fee for drand worker is applied twice 12

Check for registration phase is inconsistent between register and play message 13

Removing past combinations is inefficient 13

Removing past winners is inefficient 13

Governance messages proposal, vote, present proposal and reject proposal are not respecting safe lock 14

CosmWasm loterra-staking-contract Smart Contract 15

Payout of rewards could be blocked 15

Disclaimer

Introduction

Purpose of this Report

Cryptonics Consulting has been engaged by Terra Capitol to perform a security audit of the Lotera smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the system, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the code submitted in the following GitHub repositories:

https://github.com/LoTerra/loterra-contract

Commit no: `d030dfbd4682dd92fcef524a7f21d7940c3a60de`

https://github.com/LoTerra/loterra-staking-contract

Commit no: `2c65ef7843575d5fee8be9b42a4acb0cc13f0c49`

Methodology

The audit has been performed by a mixed team of smart contract and full-stack auditors.

The following steps were performed:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under- / overflow issues
    c. Key management vulnerabilities
    d. Permissioning issues
    e. Logic errors
4. Report preparation

The results were then discussed between the auditors in a consensus meeting and integrated into this joint report.

Functionality Overview

The submitted code implements a lottery system and related staking smart contract.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged,** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentatio**n, and **test coverage**. We include a table with these criteria for each module, in the corresponding findings section.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

The Anchor smart contracts were found to contain 0 critical issues, 1 major issue, 6 minor issues and 6 informational notes:

No	Description	Severity	Status
CosmWasm loterra-contract Smart Contract			
1	Play message could be blocked	Critical	Pending
2	Poll can be blocked by Sybil attack	Major	Pending
3	Jackpot message is inefficient and could be blocked	Minor	Pending
4	Vote weight calculation is unbounded and could be blocked	Minor	Pending
5	Vote message is inefficient and could be blocked	Minor	Pending
6	Winners can only claim jackpot until the next winners are drawn	Informational	Pending
7	Fee for drand worker is applied twice	Informational	Pending
8	Check for registration phase is inconsistent between register and play message	Informational	Pending
9	Removing past combinations is inefficient	Informational	Pending
10	Removing past winners is inefficient	Informational	Pending
11	Governance messages proposal, vote, present proposal and reject proposal are not respecting safe lock	Informational	Pending

CosmWasm loterra-staking-contract Smart Contract			
11		Minor	Pending

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	Medium-high	-
Level of Documentation	Low-Medium	-
Test Coverage	Medium-high	-

Detailed Findings

CosmWasm loterra-contract Smart Contract

1. Play message could be blocked

Severity: Critical

The `handle_play` function in `src/contract.rs:265` calls `query_all_combination`, which contains an iterative storage lookup that is bounded only to the number of combinations possible. In `src/contract.rs:362`, a loop is performed over all combinations returned from the storage lookup. A further lookup for all combinations is performed in `src/contract.rs:481`, followed by another loop to clear them. Depending on the amount of gas used for these storage lookups and the loops, an attacker could store enough combinations such that the message runs out of gas. In that case, winners could never get their jackpot. Also, the contract would never allow another registration, since `block_time_play` would always be in the past.

Recommendation

We recommend running benchmarks with worst case assumptions to verify whether an out of gas error can happen. If that is indeed the case, we recommend changing the storage for efficient lookup by creating indices on write, such that less combinations need to be queried during the play `handle_play` function.

Status: Pending

2. Poll can be blocked by Sybil attack

Severity: Major

The `handle_present_proposal` function in `src/contract.rs:1192` contains logic that rejects any proposal if the number of `no` votes is bigger than the number of `yes` votes, independent of the respective voters' stake. That allows an attacker to create a higher number of `no` votes than the expected number of `yes` votes to block any vote. Since the stake of the `no` voters is irrelevant in the check, the attack is relatively cheap.

Recommendation

We recommend changing the logic of the governance system to require a locked stake at the time of voting, to create a cost for casting no votes. Additionally, we recommend to not reject votes based on the number of votes, but rather by the stake of the voters.

Status: Pending

3. Jackpot message is inefficient and could be blocked

Severity: Minor

The `handle_jackpot` function in `src/contract.rs:640` calls `query_all_winner` and then iterates over all winners for all ranks to determine whether the message sender is a winner. This iteration is not needed if the storage layout is changed. The iteration is also theoretically unbounded since there could be an unlimited amount of winners. In that case, the message would run out of gas and no winner could claim their share of the jackpot. It is very unlikely for that to ever happen, since a high number of winners is extremely unlikely. For an attacker, it would be extremely hard to exploit this issue: The attacker would have to enter an extremely high amount of combinations and also win with those. The cost of that attack is most likely too high in any circumstance.

Recommendation

For efficiency reasons, we would still recommend adjusting the storage layout such that winners are stored by address. That removes the need for the loop in `src/contract.rs:670`.

Status: Pending

4. Vote weight calculation is unbounded and could be blocked

Severity: Minor

The `total_weight` function in `src/contract.rs:1165` iterates over all voters and queries their token balances to determine the weight of voters in a poll. This iteration is unbounded. An attacker could create a high number of addresses and have them all vote on a poll to make the `total_weight` function run out of gas, effectively blocking enactment of a poll.

Recommendation

We recommend changing the logic to set the voting weight at the time of the vote and store the weight of `yes` and `no` voters at vote time. That would eliminate the need to iterate over voters.

Status: Pending

5. Vote message is inefficient and could be blocked

Severity: Minor

The `handle_vote` function in `src/contract.rs:1054` ensures that a voter does not vote multiple times by checking whether the `Vecs` of `yes` and `no` voters already contain the address of the voter. Those checks are unbounded. An attacker could create a high number of addresses and have them all vote on a poll to make the `handle_vote` function run out of gas, effectively blocking any other voters.

Recommendation

We recommend changing the storage of `yes` and `no` voters to a storage by address as the key with the vote as the value. That would eliminate the `Vecs`, removing any unbounded checks.

Status: Pending

6. Winners can only claim jackpot until the next winners are drawn

Severity: Informational

The `handle_play` function in `src/contract.rs:294` removes all past winners, implying that a past winner will no longer be able to claim their share of a won jackpot as soon as the next winners are stored.

Recommendation

We recommend to permanently store won and collected amounts, removing any time constraint for jackpot collection.

Status: Pending

7. Fee for drand worker is applied twice

Severity: Informational

In the `handle_play` function in `src/contract.rs:342` , the `fee_for_drand_worker_in_percentage` is applied twice to the jackpot.

Recommendation

We recommend applying the fee only once..

Status: Pending

8. Check for registration phase is inconsistent between register and play message

Severity: Informational

The `handle_play` function in `src/contract.rs:304` considers the registration phase to include the block stored in `state.block_time_play`, while the `handle_register` function in `src/contract.rs:215` excludes that block.

Recommendation

We recommend either including or excluding the block stored in `state.block_time_play` in both functions.

Status: Pending

9. Removing past combinations is inefficient

Severity: Informational

The `handle_play` function in `src/contract.rs:481` looks up all past combinations and then removes them individually. That results in many storage reads and writes, which is inefficient.

Recommendation

We recommend removing the storage by prefix instead.

Status: Pending

10. Removing past winners is inefficient

Severity: Informational

The `handle_play` function in `src/contract.rs:288` looks up all past winners and then removes them individually. That results in many storage reads and writes, which is inefficient.

Recommendation

We recommend removing the storage by prefix instead.

Status: Pending

11. Governance messages proposal, vote, present proposal and reject proposal are not respecting safe lock

Severity: Informational

The governance messages proposal, vote, present proposal and reject proposal are not respecting the safe lock.

Status: Pending

CosmWasm loterra-staking-contract Smart Contract

12. Payout of rewards could be blocked

Severity: Minor

The `handle_payout_reward` function in `src/contract.rs:383` queries all stakers from storage, calculates the total staked tokens, and then updates the stake of every staker. An attacker could create many accounts, stake small amounts and hence make the `handle_payout_reward` function so gas intensive that it will fail.

Recommendation

We recommend changing the reward calculation logic to be index based. Such a system would store a global reward index, that is increased whenever new tokens are received as rewards. Each staker would also have a staker reward index. Whenever a staker's stake would change, the staker would receive their reward proportional to the difference between the staker reward index and the global reward index. Then the staker reward index would be reset to the global reward index. A reference implementation of this logic can be found in https://github.com/Anchor-Protocol/anchor-bAsset-contracts.

Status: Pending