Audit Report

Ink Protocol

September 15, 2021

Table of Contents

License

Disclaimer

This audit has been performed by

Philip Stanislaus and **Stefan Beyer**

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by Ink Protocol (https://inkprotocol.finance/) to perform a security audit of the link no loss lottery smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, 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 following GitHub repository:

https://github.com/ink-protocol-finance/contracts

Commit hash: `49ebfa1206c5ed3d4bc64d2338685e7aaa0b610f`

Methodology

The audit has been performed in the following steps:

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
4. Report preparation

Functionality Overview

The smart contracts submitted for review implement a protocol for yield bearing tokens deposited in a no loss lottery.

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**.

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 documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low 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

No	Description	Severity	Status
1	Slashes or other reduction of assets in the Anchor strategy could block drawing a winner while causing a bank run, leaving slow users lose assets	Critical	Resolved
2	Lack of authorization on staking contract's `Add` and `Set` messages allows an attacker to capture all rewards	Critical	Resolved
3	Block author can pick lottery winners and front-run drawing of a winner	Major	Acknowledged
4	Executing the Anchor strategy with CW20 tokens will lead to inaccessible tokens in the strategy contract	Major	Resolved
5	Setting a loot box winner will always fail	Major	Resolved
6	Drawing a winner might fail due to integer overflow	Major	Resolved
7	Blacklist value calculation is unbounded and could block drawing a winner	Major	Acknowledged
8	Excess tax amounts users provide when depositing will be distributed as awards to future winners	Major	Resolved
9	Source of randomness relies on trust in off-chain secret generation and uses low entropy from on-chain data	Major	Acknowledged
10	Updating Anchor aUST and market contract addresses in Anchor strategy could lead to incorrect states	Major	Resolved
11	Storing strategies in a vector in the core's storage is inefficient	Minor	Acknowledged
12	Attacker can prevent setting a loot box winner, resulting in inaccessible loot	Minor	Resolved
13	Lack of validation of strategy state when adding a strategy could lead to failures of award claiming	Minor	Resolved
14	Usage of human instead of canonical addresses in stored entities could eventually break contracts	Minor	Resolved

15	Big shares of blacklisted deposited amounts could cause winner to fail	Minor	Acknowledged
16	Querying blacklisted ids is unbounded and could run out of gas	Minor	Acknowledged
17	Querying player awards is unbounded and could run out of gas	Minor	Resolved
18	Usage of raw queries might break contracts in the future	Minor	Acknowledged
19	Using strategy address as the key for various storage entries is inefficient	Informational	Acknowledged
20	Multiple queries to strategy state store are unnecessary	Informational	Acknowledged
21	Storing own contract address leads to unnecessary storage consumption	Informational	Resolved
22	Exchange rate is queried twice within Anchor strategy's claim function	Informational	Resolved

Code Quality Criteria

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

Detailed Findings

1. Slashes or other reduction of assets in the Anchor strategy could block drawing a winner while causing a bank run, leaving slow users lose assets

Severity: Critical

The current implementation of Ink does not prevent withdrawals after losses of the value of assets (such as slashes) beyond deposits and claimable awards happen within a strategy. It also does not implement proportional withdrawal of the remaining assets after such a loss of value.

Drawing a winner will not be possible in such a situation due to the checks in the `query_award` function in `contracts/ink-core/src/querier.rs:243` and `252`. At the same time, withdrawals will still be possible and the current implementation allows users to withdraw the full amount they deposited, due to the logic in the `claim` function in `contracts/ink-anchor-strategy/src/handler.rs:103`.

Since no winner can be drawn and there is no benefit of remaining in the strategy, and since the sum of all deposits and claimable awards is no longer covered by the remaining assets in the strategy, there is an incentive for users to withdraw their tokens as fast as possible. As in a bank run, the last users to try to withdraw will lose their funds.

Recommendation

We recommend blocking any withdrawals from a strategy if the value of the assets can no longer cover deposits and claimable awards. A recovery from such a state may happen automatically over time, if the remaining assets should still generate some yield. Additionally, we recommend implementing a mechanism to allow proportional withdrawals of the available assets.

Status: Resolved

2. Lack of authorization on staking contract's `Add` and `Set` messages allows an attacker to capture all rewards

Severity: Critical

There is currently no authorization on the handlers for the staking contract's `Add` and `Set` messages in `contracts/ink-staking/src/handler.rs:145` and `186`. Consequently, an attacker can simply change pool parameters to extract value from the stacking contracts.

For example, an attacker could send an `Add` message to create a pool with a very high `alloc_point`, effectively capturing most of the rewards. The attacker could also deploy

their own token contract which would prevent anyone depositing in their high reward pool. The attacker could even obtain all rewards by sending a `Set` message to all other pools to reduce their `alloc_point` to zero.

Recommendation

We recommend adding authorization to the staking contract's `Add` and `Set` messages.

Status: Resolved

3. Block author can pick lottery winners and front-run drawing of a winner

Severity: Major

Randomness in the Ink protocol relies on a combination of an off-chain secret and on-chain entropy. The secret will be revealed as part of the `DrawWinner` message. The on-chain entropy can be influenced by the block author, since it relies on the block time. Since the block creator knows the secret at the time of block creation, the block creator can influence who will win the lottery by picking a suitable block time. Even without influencing the entropy, the block author can add their own Deposit transactions before the `DrawWinner` message to front-run the Ink protocol and have a completely risk-free win.

Recommendation

We recommend using a source of randomness that the block creator cannot influence, e. g. the block time or hash of the previous block. Additionally, we recommend to disallow participation in the lottery after the round end time to prevent the block author from front-running.

Status: Acknowledged

The Ink team plans to move to a decentralized randomness source in a future release. The front-running issue has been resolved.

4. Executing the Anchor strategy with CW20 tokens will lead to inaccessible tokens in the strategy contract

Severity: Major

The `execute` handler for the `Execute` message of the Anchor strategy contract neither forwards CW20 tokens that have been sent to it to Anchor, nor does it revert in the case of CW20 tokens sent in `contracts/ink-anchor-strategy/src/handler.rs:56`. This will lead to CW20 tokens being owned by the Anchor strategy contract where they will be inaccessible.

Since this issue would be caused by a misconfiguration of a strategy, we classify this issue as major instead of critical.

Recommendation

We recommend returning an error in the case of CW20 assets sent to the `execute` handler in `contracts/ink-anchor-strategy/src/handler.rs:56`. We also recommend asserting the denom of the assets sent to be equal to `cfg.asset_info` in the `execute` handler.

Status: Resolved

5. Setting a loot box winner will always fail

Severity: Major

A condition following the `SetRoundWinner` message of the loot box in `contracts/ink-lootbox/src/handler.rs:247` requires a winner to be set for the current or future rounds. At the same time, when a winner is drawn in the core in `contracts/ink-core/src/handler.rs:537`, the round is increased. That implies that the `SetLootboxWinner` message will always fail.

Recommendation

We recommend inverting the condition in `contracts/ink-lootbox/src/handler.rs:247` to require a winner being only set for past periods.

Status: Resolved

6. Drawing a winner might fail due to integer overflow

Severity: Major

In the `draw_winner` function in `contracts/ink-core/src/handler.rs:501`, a hash is calculated on a random draw number which is added to the loop counter `i`. Since overflow checks are enabled for the contract, there is a small chance that this addition will overflow.

We classify this issue as major since the owner of the contract can intervene and restore the correct functioning of the contract by setting a new secret hash.

Recommendation

We recommend using `wrapping_add`.

Status: Resolved

7. Blacklist value calculation is unbounded and could block drawing a winner

Severity: Major

The `get_total_blacklist_value` function currently loops over all blacklist entries in storage in `packages/sortition-sum-tree/src/tree.rs:190` to calculate the total blacklist value. That loop is unbounded and could run out of gas, which would prevent a winner from being drawn.

Recommendation

We recommend storing the black list value, rather than calculating it in a loop.

Status: Acknowledged

Blacklist is used to exclude sponsors from being drawn as winners. The Ink team does not expect the list of sponsor addresses to grow enough to block drawing a winner, but considers a fix for future versions.

8. Excess tax amounts users provide when depositing will be distributed as awards to future winners

Severity: Major

In the `deposit` function in `contracts/ink-core/src/handler.rs:294`, `deposit_asset` is calculated by subtracting the user provided `tax` amount from the `sent_asset` amount the user sent to the contract. That `deposit_asset` amount is then used to update the withdrawal lock time and is stored as the user's deposit. The actual amount sent to the strategy is the `sent_asset` amount minus the actual tax on that amount, as reported by TerraSwap. When a user later wants to withdraw their tokens, they will only be able to claim the `deposit_asset` amount. That implies the difference between the user provided `tax` amount and the actual taxes needed will be gifted to the strategy, which will eventually increase awards.

Recommendation

We recommend removing the need for users to provide the tax amount, using actual taxes to calculate `deposit_asset`. We also recommend allowing users to supply an optional max tax cap.

Status: Resolved

9. Source of randomness relies on trust in off-chain secret generation and uses low entropy from on-chain data

Severity: Major

Randomness in the Ink protocol relies on a secret that is provided by a party that knows a previous secret, plus on-chain entropy. Both sources of randomness have issues:

1. Provision of the secret follows a commit-reveal scheme, where the Keccak 256 hash of the secret is set by the contract owner. After the end time of a round, any party knowing the secret can send the `DrawWinner` message with that secret and the hash for the next secret. That puts trust in the external party to use enough entropy to generate the off-chain secret. For example, there is currently no logic in the code base that ensures that the provided secret is different in each round.
2. The on-chain entropy is generated by querying Terra for the swap result between LUNA and TerraUSD for the amount of block height plus the block time in seconds. Since the swap amount will increase monotonically by a relatively fixed amount, and the exchange rate between LUNA and TerraUSD is expected to fluctuate by a few percentage points only per block, the resulting value will have little entropy.

Recommendation

We recommend relying on a more robust source of randomness.

Status: Acknowledged

The Ink team states in their documentation that it plans to move to an Oracle's VRF as the source of randomness in a future version.

10. Updating Anchor aUST and market contract addresses in Anchor strategy could lead to incorrect states

Severity: Major

The Anchor strategy allows updating the aUST and market contract addresses in `contracts/ink-anchor-strategy/src/handler.rs:176` and `190`. A migration of those contracts without a migration of the underlying state could lead to incorrect states, with consequences such as failure of withdrawing or drawing a winner.

Recommendation

We recommend removing the possibility to update aUST and market contract addresses.

Status: Resolved

11. Storing strategies in a vector in the core's storage is inefficient

Severity: Minor

The addresses of strategies are currently stored in a vector in `contracts/ink-core/src/state.rs:25`. That vector is fully loaded in many places throughout the codebase, and its elements are accessed by their `sid`, which is a strategy's index in the vector. With more and more strategies being added to Ink over time, the vector will grow. That is inefficient and leads to increasing gas costs to use Ink as the protocol matures.

Recommendation

We recommend changing the storage layout from a vector in a singleton to a bucket.

Status: Acknowledged

The Ink team plans to consider this recommendation as more strategies are added in future versions.

12. Attacker can prevent setting a loot box winner, resulting in inaccessible loot

Severity: Minor

During the `SetRoundWinner` message of the loot box an unbounded loop is used to read loot assets in `contracts/ink-lootbox/src/helper.rs:14`. Since anyone can sponsor loot, an attacker could create many CW20 assets and send tiny amounts of those assets to the loot box. That could cause the unbounded loop to run out of gas, which would prevent a loot box winner from being set and render the loot for that round inaccessible.

We classify this issue as minor, since deploying enough CW20 asset contracts and sending those assets to the loot box will probably be prohibitively expensive to not execute the outlined attack.

Recommendation

We recommend restricting the assets that can be sent to the loot box, e. g. by specifying a predefined set of assets denoms/contracts or by tracking and limiting the number of assets per round.

Status: Resolved

13. Lack of validation of strategy state when adding a strategy could lead to failures of award claiming

Severity: Minor

The fields of the optional `strategy_state` argument passed to the `add_strategy` function in `contracts/ink-core/src/handler.rs:416` are currently not validated. This can lead to inconsistent states, e. g. `claimable_award` asset could diverge from the actually available asset, which might lead to failure of drawing a winner due to the check in `contracts/ink-core/src/querier.rs:252`.

Recommendation

We recommend requiring the `claimable_award` in the `strategy_state` argument to be `None`.

Status: Resolved

14. Usage of human instead of canonical addresses in stored entities could eventually break contracts

Severity: Minor

In several places in the codebase, human addresses are stored or used for comparison. As outlined in the CosmWasm documentation, that human representation could potentially be changed in the future, which may lead to the contracts no longer working as expected. Examples are:

- The elements of the `strategies` vector of the Ink core's `State` in `contracts/ink-core/src/state.rs:25`
- The `id` of entries in the sortition sum tree, e. g. in `contracts/ink-core/src/handler.rs:136` and `171` as well as in `contracts/ink-core/src/querier.rs:168`
- The `id` of entries in the sortition sum tree's blacklist, e. g. in `contracts/ink-core/src/handler.rs:668` as well as in `contracts/ink-core/src/querier.rs:377`
- The stored address of the loot box in `lootbox_address_store` and `lootbox_address_store_read` in `contracts/ink-core/src/state.rs:36` and `40`
- The `address` field of the `Winner` struct in `packages/ink/src/core.rs:227`
- The `address` field of the `Player` struct in `packages/ink/src/core.rs:207`
- The key to access `winner_by_round_store` and `winner_by_round_store_read` in `contracts/ink-lootbox/src/handler.rs:249`, `260`, `285` and `314`
- The `sponsor_address` field of the `SponsoredLoot` struct in `packages/ink/src/lootbox.rs:98`

- The `address` field of the `LootWinner` struct in `packages/ink/src/lootbox.rs:103`
- The comparison in `contracts/ink-core/src/querier.rs:337`
- The comparison in `contracts/ink-core/src/querier.rs:576`
- The comparison in `contracts/ink-staking/src/handler.rs:44`

Recommendation

We recommend converting all addresses to their canonical form, which is stable across human representations. In cases where strings are required, we recommend encoding canonical addresses to a string type, rather than using human addresses.

Status: Resolved

15. Big shares of blacklisted deposited amounts could cause drawing a winner to fail

Severity: Minor

When blacklisted addresses are drawn as winners from the sortition sum tree in `packages/sortition-sum-tree/src/tree.rs:161`, they will be skipped and another draw will be executed. The higher the ratio of the token amount by blacklisted addresses to the total token amount deposited to a strategy, the more inefficient drawing could become. This could even cause the drawing of a winner to run out of gas.

Recommendation

We recommend removing blacklisted addresses from the sortition sum tree and storing the amount deposited by them in a separate storage to keep drawing from the tree efficient.

Status: Acknowledged

The Ink team considers this scenario to be unlikely, since bigger sponsored shares also attract more regular users.

16. Querying blacklisted ids is unbounded and could run out of gas

Severity: Minor

The `get_blacklist_ids` function, which is called by the core's `BlacklistAddresses` query message, contains an unbounded iteration over storage entries in `packages/sortition-sum-tree/src/tree.rs:207`. While not security relevant to the current implementation of Ink, this could have adverse effects in the future or on third party contracts integrating with Ink.

Recommendation

We recommend adding offset and limit arguments to the query that calls the `get_blacklist_ids` function.

Status: Acknowledged

The Ink team plans to monitor the impact of the amount of blacklisted ids. Since blacklisting is performed by the contract owner, the Ink team can control the impact accordingly.

17. Querying player awards is unbounded and could run out of gas

Severity: Minor

The `query_player_awards` function, which is called by the core's `PlayerAwards` query message, contains an unbounded iteration over all rounds of a strategy. Within every iteration, winners are queried in `contracts/ink-core/src/querier.rs:332`. While not security relevant to the current implementation of Ink, this could have adverse effects in the future or on third party contracts integrating with Ink.

Recommendation

We recommend adding offset and limit arguments for the rounds to the query that calls the `query_player_awards` function.

Status: Resolved

18. Usage of raw queries might break contracts in the future

Severity: Minor

In `packages/moneymarket/src/querier.rs:50` and `68` raw queries are used. Using raw queries is problematic, since they tie the caller to a specific implementation of the target contract's storage layout, which might change in a future migration.

Recommendation

We recommend using well defined query interfaces instead of raw queries.

Status: Acknowledged

Those raw queries are part of the external anchor-market package and are not currently used in the Ink protocol.

19. Using strategy address as the key for various storage entries is inefficient

Severity: Informational

Currently, a strategy's address is used as the key to query several strategy specific storage entries within the core contract. Since messages are sent with the strategy index `sid` rather than the address, a storage read first needs to read the strategy address before executing the actual storage read of interest. That leads to computation overhead.

An example is the `query_round` function in `contracts/ink-core/src/querier.rs:204`.

Recommendation

We recommend using the strategy index `sid` as a key to all storage reads instead of the strategy address.

Status: Acknowledged

The Ink team plans to consider this recommendation as more strategies are added in future versions.

20. Multiple queries to strategy state store are unnecessary

Severity: Informational

The handlers for several messages of the core contract contain multiple reads of the stored strategy state. That is unnecessary and can be prevented. Specific examples:

- During the `deposit_token` function in `contracts/ink-core/src/handler.rs:229`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in the `assert_strategy_start_time` function in `contracts/ink-core/src/handler.rs:118`.
- During the `deposit` function in `contracts/ink-core/src/handler.rs:294`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53`, in the `assert_deposit_tax` function in `contracts/ink-core/src/handler.rs:63` and `70` and in the `assert_strategy_start_time` function in `contracts/ink-core/src/handler.rs:118`.
- During the `withdraw` function in `contracts/ink-core/src/handler.rs:356`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/handler.rs:373`.

- During the `draw_winner` function in `contracts/ink-core/src/handler.rs:472`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53`, in `contracts/ink-core/src/handler.rs:482` and in the `query_award` function in `contracts/ink-core/src/querier.rs:224` and in `229`.
- During the `claim_award` function in `contracts/ink-core/src/handler.rs:565`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/handler.rs:590`.
- During the `update_period` function in `contracts/ink-core/src/handler.rs:628`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/handler.rs:641`.
- During the `set_secret_hash` function in `contracts/ink-core/src/handler.rs:675`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/handler.rs:687`.
- During the `set_withdraw_lock_fee_bps` function in `contracts/ink-core/src/handler.rs:695`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/handler.rs:707`.
- During the `set_withdraw_lock_period` function in `contracts/ink-core/src/handler.rs:715`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/handler.rs:727`.
- During the `query_round` function in `contracts/ink-core/src/querier.rs:204`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/querier.rs:212`.
- During the `query_award` function in `contracts/ink-core/src/querier.rs:224`, the stored strategy state is read within the `query_strategy` function in `contracts/ink-core/src/querier.rs:53` and in `contracts/ink-core/src/querier.rs:229`.
- During the `query_player_awards` function in `contracts/ink-core/src/querier.rs:224`, the stored strategy state is read within the `query_strategy` function in

`contracts/ink-core/src/querier.rs:53` and in the `query_winners` function in `contracts/ink-core/src/querier.rs:182`.

Recommendation

We recommend reading the stored strategy state only once per message, and pass the query response to helper functions instead.

Status: Acknowledged

The Ink team plans to optimize the number of storage reads in a future version.

21. Storing own contract address leads to unnecessary storage consumption

Severity: Informational

The Anchor strategy, loot box and staking contracts store their own address as `self_address_raw`, `self_address` and `contract_address_raw` in their state in `contracts/ink-anchor-strategy/src/contract.rs:38`, `contracts/ink-lootbox/src/contract.rs:33` and `contracts/ink-staking/src/contract.rs:32`. That leads to unnecessary storage entries, since a contract's address can always be retrieved through `env.contract.address`.

Recommendation

We recommend using `env.contract.address` instead of storing the own address.

Status: Resolved

22. Exchange rate is queried twice within Anchor strategy's claim function

Severity: Informational

Within the Anchor strategy's `claim` function in `contracts/ink-anchor-strategy/src/handler.rs:68`, the `get_a_ust_exchange_rate` function is called twice, once in the call in line `88` and once in the call in line `103`.

Recommendation

We recommend querying the exchange rate once.

Status: Resolved