Audit Report

Glow

v1.0

November 12, 2021

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of this Report

Oak Security has been engaged by Terraform Labs to perform a security audit of the Glow 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/elevenyellow/glow-protocol

Commit hash: `3045d4dcdbb13079b500230c728da9dd03419611`

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

Glow implements a no-loss lottery with certain incentives and sponsorship possibilities. The contracts audited also contain staking, vesting, airdrop and governance functionality.

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 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	Lottery is (almost) never considered started, allowing user interaction during a running lottery which will lead to inconsistent contract states	Critical	Resolved
2	Combinations beginning with 99 will not receive their prices	Major	Resolved
3	Lotto contract's incremental glow emission rate may be bypassed	Major	Resolved
4	Claiming deposits from the lotto contract can run out of gas if a user has many unbonding info entries	Major	Resolved
5	Exchange rate queries in lotto contract may receive outdated values, leading to share fluctuations and unused UST stuck in the contract	Major	Resolved
6	Winning sequence calculation may panic	Minor	Resolved
7	Updates of the lotto contract's split factor config value will leave user funds inaccessible and lead to errors during award calculation	Minor	Resolved
8	Winner prize calculation will leave inaccessible division remainder in lotto contract	Minor	Acknowledged
9	Lack of prize distribution validation in lotto contract could lead to too high prizes distributed	Minor	Resolved
10	Lack of split factor validation in lotto contract could lead to panic during deposits	Minor	Resolved
11	Lack of instant withdrawal fee validation in lotto contract could lead to panic during withdrawals	Minor	Resolved
12	Lotto contract's instant withdrawal fee cannot be updated	Minor	Resolved
13	Lotto contract's unbonding period is assigned to the block time and cannot be updated	Minor	Resolved
14	Vesting contract's lack of start time validation can lead to funds that never vest	Minor	Resolved

15	Vesting contract does not validate that tokens have been received, users could run into insufficient funds errors	**Minor**	**Acknowledged**
16	Staking contract does not validate distribution schedule, allows gaps/overlaps and could cause panics	**Minor**	**Resolved**
17	Missing validation in distributor contract	**Informational**	**Resolved**
18	Lotto contract's logic for leftover deposit tickets assigns the same combination to every depositor/gift recipient in the same block	**Informational**	**Resolved**
19	Deposits in lotto contract allow excess deposits, gifts do not allow that	**Informational**	**Resolved**
20	Lotto contract's total tickets does not count additional tickets for leftover deposits	**Informational**	**Resolved**
21	Unused reserve_factor field in lotto contract's config and associated logic	**Informational**	**Resolved**
22	State query in staking contract may panic for certain block heights	**Informational**	**Resolved**

Code Quality Criteria

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

Detailed Findings

1. Lottery is (almost) never considered started, allowing user interaction during a running lottery which will lead to inconsistent contract states

Severity: Critical

In several places in the lotto contract, the condition `!current_lottery.rand_round == 0` is used to determine whether the current lottery has been started. That condition will never return true though, since `!current_lottery.rand_round` is evaluated first, leading to a binary not operation on the unsigned integer, which is in almost all cases not equal to `0`. The condition is found in `contracts/lotto/src/contract.rs:228`, `366`, `567`, `658`, and `807`.

This issue implies that deposits, gifts, withdrawals, sponsor withdrawals and claims are still possible while a lottery is running, leading to an inconsistent contract state.

Recommendation

We recommend replacing `!current_lottery.rand_round == 0` with `current_lottery.rand_round != 0`.

Status: Resolved

2. Combinations beginning with 99 will not receive their prices

Severity: Major

In the `execute_prize` function of the lotto contract, winners are determined with a storage iteration in `contracts/lotto/src/prize_strategy.rs:181`. The iteration starts at the first two digits of the winning sequence of the lottery and ends at the next integer, excluding that number. For the case of the starting digits 99 though, the next integer would be 100, which is reduced to 99 in line `175`. The iteration uses an exclusive upper bound in line `185`, which means that the iterator would go from 99 to 99, and hence finish directly without any iteration. That implies that any combination starting with 99 will not be determined as a winner, which is incorrect. Conversely, no winners will be stored if the winning sequence starts with 99.

Recommendation

We recommend using an inclusive upper bound for the iteration of the edge case

Status: Resolved

3. Lotto contract's incremental glow emission rate may be bypassed

Severity: Major

The lotto contract's `execute_epoch_ops` function in `contracts/lotto/src/contract.rs:885` is used to update the `glow_emission_rate`. Updates of the `glow_emission_rate` are done in increments/decrements that are determined through the distributor contract's `GlowEmissionRate` query.

The `execute_epoch_ops` function has no access control and no rate limit though, which means that any user can call it repeatedly to bypass incremental adjustments.

Recommendation

We recommend introducing a rate limit to the `execute_epoch_ops` function such that it will only update the `glow_emission_rate` again after several cooldown blocks.

Status: Resolved

4. Claiming deposits from the lotto contract can run out of gas if a user has many unbonding info entries

Severity: Major

During the lotto contract's `claim_deposits` function, an unbounded iteration happens over a depositor's `unbonding_info` vector in `contracts/lotto/src/helpers.rs:44`. If that vector contains too many entries, the call will run out of gas. In the current implementation, that issue cannot be recovered from, which leads to a user being unable to retrieve their funds back.

While this issue can lead to inaccessible user funds, we do not classify it as critical since it is very unlikely for a user to have a large number of `unbonding_info` entries. It is still possible, for example, if another contract builds on top of Glow and withdraws in quick successions.

Recommendation

We recommend changing the storage architecture such that a user has to explicitly supply the id of an `unbonding_info` entry, or limiting the number of entries that can exist in the `unbonding_info` vector.

Status: Resolved

5. Exchange rate queries in lotto contract may receive outdated values, leading to share fluctuations and unused UST stuck in the contract

Severity: Major

In multiple places in the codebase, Anchor's aUST/UST exchange rate is queried through an `EpochState` query and then used calculate the aUST/UST value of deposited, gifted, sponsored or withdrawn UST/aUST and during lottery execution (in `contracts/lotto/src/contract.rs:260`, `399`, `507`, `580`, `674`, and `74`). The query does not include the current block number though. Without a block number, Anchor returns the exchange rate based on the last interest computation, not the latest one. That implies that the actual exchange rate may be lower than expected, with the following consequences:

- During deposit, gift, and sponsor, the stored share will be higher than the actual aUST received. That will cause the shares to vary slightly between depositors.
- During withdrawal, sponsor withdrawal, and lottery execution, too much aUST is burned, leading to unused UST stuck in the contract.

Additionally, in the `withdraw`, `sponsor_withdraw` and `execute_lottery` functions the UST amount sent back from Anchor will have taxes deducted, while the amount used in the lotto contract's calculations is not considering taxes. That causes too many funds to be distributed, which implies that the last users will not be able to claim their deposits back due to the condition in `contracts/lotto/src/contract.rs:863`.

Recommendation

We recommend refactoring the code such that the actually received tokens are used rather than tokens calculated from the queried exchange rate. This can be accomplished by first querying and storing the balance of aUST/UST, then depositing/redeeming from Anchor through a sub-message, and then querying and comparing the updated balance of aUST/UST in a reply to determine the received amount. Alternatively, the `EpochState` query can be performed with the current block number to use an up-to-date interest computation. In that case, taxes should be deducted twice in the `withdraw`, `sponsor_withdraw` and `execute_lottery` functions.

Status: Resolved

6. Winning sequence calculation may panic

Severity: Minor

The winning sequence of a lottery is determined in the `sequence_from_hash` function of the lotto contract in `contracts/lotto/src/oracle.rs:14`. Within that function, a hexadecimal string is filtered by decimal characters, and then the resulting string is sliced into the winning sequence. That slicing is done by index access of the string in line `16`, which will panic if the filtered string is shorter than 6 characters. Since a hexadecimal string can contain no single decimal character (an example is `0xaaaa..aa`), such a panic will eventually occur.

We consider this issue not to be critical since it is very unlikely and recoverable by triggering the `execute_prize` function again, which should result in a different hash and resolve the issue.

Recommendation

We recommend changing the `sequence_from_hash` function to convert hexadecimal characters into decimal characters instead of filtering them out.

Status: Resolved

7. Updates of the lotto contract's split factor config value will leave user funds inaccessible and lead to errors during award calculation

Severity: Minor

In the `update_config` function of the lotto contract in `contracts/lotto/src/contract.rs:1010`, the `split_factor` can be updated. That `split_factor` is used during deposits/gifts and withdrawals to update the pool's `lottery_deposits`, `lottery_shares`, and `deposit_shares` values. If it is changed while the lotto contract holds any user deposits, any subsequent withdrawal will apply a different `split_factor` and hence lead to an inconsistent state.

That implies that the subtractions in `contracts/lotto/src/contract.rs:737-738` will panic for the last users trying to withdraw their tokens, leaving them unable to access their funds. Additionally, the award calculation of the `execute_lottery` function in `contracts/lotto/src/prize_strategy.rs:69-102` would lead to unexpected errors.

We classify this issue as minor since it can only be caused by the contract owner.

Recommendation

We recommend either adding state migration to an update of the `split_factor` to keep the state consistent, or removing the possibility to update the `split_factor` whenever deposits exist in the lotto contract.

Status: Resolved

8. Winner prize calculation will leave inaccessible division remainder in lotto contract

Severity: Minor

During the lotto contract's winner prize calculation, integer division is used to determine the prize for the winner in `contracts/lotto/src/helpers.rs:78`. That integer division will leave a remainder in the contract, which is inaccessible by anyone.

Recommendation

We recommend tracking remainders and adding them back to the award.

Status: Acknowledged

9. Lack of prize distribution validation in lotto contract could lead to too high prizes distributed

Severity: Minor

The `prize_distribution` passed to the lotto contract's `instantiate` function and used in `contracts/lotto/src/contract.rs:71` is not validated. If the sum of its items is greater than one, higher prizes will be assigned to winners than available, and the last winners to claim their prizes will run into an `InsufficientClaimableFunds` error.

We still classify this issue as minor since it can only be caused by the owner during instantiation.

Recommendation

We recommend validating the `prize_distribution` in the `instantiate` function as already done in the `update_lottery_config` function in `contracts/lotto/src/contract.rs:1061`.

Status: Resolved

10. Lack of split factor validation in lotto contract could lead to panic during deposits

Severity: Minor

The `split_factor` passed to the lotto contract's `instantiate` function and used in `contracts/lotto/src/contract.rs:74` is not validated. If it is greater than one, the subtraction in the `deposit` function in `contracts/lotto/src/contract.rs:301` will panic, which will imply that users cannot deposit.

We still classify this issue as minor since it can only be caused by the owner during instantiation.

Recommendation

We recommend validating the `split_factor` in the `instantiate` function as already done in the `update_config` function in `contracts/lotto/src/contract.rs:1007`.

Status: Resolved

11. Lack of instant withdrawal fee validation in lotto contract could lead to panic during withdrawals

Severity: Minor

The `instant_withdrawal_fee` passed to the lotto contract's `instantiate` function and used in `contracts/lotto/src/contract.rs:75` is not validated. If it is greater than one, the subtraction in the `withdraw` function in `contracts/lotto/src/contract.rs:760` will panic, which will imply that users cannot withdraw their deposits.

We still classify this issue as minor since it can only be caused by the owner during instantiation.

Recommendation

We recommend validating the `instant_withdrawal_fee` in the `instantiate` function as already done in the `update_config` function in `contracts/lotto/src/contract.rs:1014`.

Status: Resolved

12.Lotto contract's instant withdrawal fee cannot be updated

Severity: Minor

In the lotto contract, the `update_config` function accepts the `instant_withdrawal_fee` accepted as an argument in `contracts/lotto/src/contract.rs:980`. It is validated in line `1014`, but never updated in the config.

Recommendation

We recommend assigning the new `instant_withdrawal_fee` value in the config after validating it.

Status: Resolved

13. Lotto contract's unbonding period is assigned to the block time and cannot be updated

Severity: Minor

In the lotto contract, a new `unbonding_period` value is accidentally assigned to the config's `block_time` in `contracts/lotto/src/contract.rs:1020`, which may confuse the owner and implies that the `unbonding_period` cannot be updated.

Recommendation

We recommend assigning the `unbonding_period` to the config field with the same name.

Status: Resolved

14. Vesting contract's lack of start time validation can lead to funds that never vest

Severity: Minor

In the vesting contract, vesting schedules are validated to have their end time after the start time, but there is currently no validation that the start time is at or after the `genesis_time`. If the start time is before the `genesis_time`, a user can never receive the full vested amount, since a user's initial `last_claim_time` is set to the `genesis_time` in `contracts/vesting/src/contract.rs:116`.

The owner can help users recover from that situation (or render further vesting funds inaccessible) by updating the `genesis_time`.

Recommendation

We recommend adding validation that the start time is at or after the `genesis_time`.

Status: Resolved

15. Vesting contract does not validate that tokens have been received, users could run into insufficient funds errors

Severity: Minor

In the vesting contract's `register_vesting_accounts` function, vesting accounts can be added, but there is no validation that the vesting contract actually controls enough funds to distribute the vested amounts. That is problematic since the last users to claim their vested tokens might run into insufficient funds errors.

Recommendation

We recommend wrapping the `RegisterVestingAccounts` message in a `Cw20ReceiveMsg` and asserting that the total amount to be vested is equal to the total amount of Glow tokens received.

Status: Acknowledged

16. Staking contract does not validate distribution schedule, allows gaps/overlaps and could cause panics

Severity: Minor

In the staking contract's `instantiate` function, there is currently no validation that the start block is less than or equal to the end block of a `distribution_schedule` entry. That can cause panics in `contracts/staking/src/contract.rs:223`. Also, the lack of validation allows gaps/overlaps of the schedule entries.

Recommendation

We recommend adding validation of distribution schedules.

Status: Resolved

17. Missing validation in distributor contract

Severity: Informational

The `increment_multiplier` and `decrement_multiplier` config values of the distributor contract are currently not validated, neither in the `instantiate`, nor in the `update_config` functions. Consequently, an `increment_multiplier` could be set such that it decrements the rate or leaves it steady, and the `decrement_multiplier` could be set such that it increments the rate or leaves it steady. Also, validation that the `emission_cap` is greater than or equal to the `emission_floor` is not checked in the `instantiate` function.

These issues have no impact on security but may confuse users.

Recommendation

We recommend asserting in the `instantiate` and `update_config` functions that:

- The `increment_multiplier` is greater than or equal to one,
- The `decrement_multiplier` is less than or equal to one, and
- The `emission_cap` is greater than or equal to the `emission_floor`.

Status: Resolved

18. Lotto contract's logic for leftover deposit tickets assigns the same combination to every depositor/gift recipient in the same block

Severity: Informational

The additional tickets that depositors/gift recipients get for leftover deposits in `contracts/lotto/src/contract.rs:242` and `381` use a function of the current block time as the extra combination. That implies that every depositor/gift recipient that gets additional tickets in the same block ends up with the same combination.

Recommendation

We recommend making the extra combination a function of not just the current block time, but also the depositor/gift recipient address and their current number of tickets. That allows for a more even distribution of tickets.

Status: Resolved

19. Deposits in lotto contract allow excess deposits, gifts do not allow that

Severity: Informational

Deposits in the lotto contract allow a deposit of more funds than the tickets for the lottery cost. Those leftover funds can later lead to additional tickets for the depositor through the logic in `contracts/lotto/src/contract.rs:242`. Gifting tickets uses almost the identical code as deposits but does not support higher amounts than the total price of the tickets. Accordingly, additional tickets for leftover amounts are not provided during the `gift_tickets` function.

Recommendation

We recommend merging the code of `gift_tickets` and `deposit` to simplify the codebase and remove differences between the functions.

Status: Resolved

20. Lotto contract's total tickets does not count additional tickets for leftover deposits

Severity: Informational

During the lotto contract's `deposit` and `gift_tickets` functions, an additional ticket is given to users that have enough leftover deposit in `contracts/lotto/src/contract.rs:242` and `381`. That additional ticket is not considered when the state's `total_tickets` field is updated.

We consider this issue as informational since the total ticket amount is only used to determine whether a lottery can be executed. An incorrect value could lead to UI inconsistencies though.

Recommendation

We recommend using `new_combinations` instead of `amount_tickets` in line `297` and `436` to fix this issue.

Status: Resolved

21. Unused `reserve_factor` field in lotto contract's config and associated logic

Severity: Informational

The lotto contract's config contains a field `reserve_factor` in `contracts/lotto/src/state.rs:41`, which is currently unused. Consequently, the logic used to transfer reserves defined in `contracts/lotto/src/contract.rs:906-915` will never execute. Unused code bloats the contract size and leads to lower maintainability.

Recommendation

We recommend adding functionality for a reserve or removing unused code.

Status: Resolved

22. State query in staking contract may panic for certain block heights

Severity: Informational

In the staking contract's `query_state` function, an optional `block_height` is accepted. If that block height is less than `state.last_distributed`, the subtraction in `contracts/staking/src/contract.rs:221` may panic.

Recommendation

We recommend adding an assertion to ensure that the `block_height` is greater than `state.last_distributed`, similar to the condition in `contracts/lotto/src/contract.rs:1158`.

Status: Resolved