

Audit "Trustless Rewards"

Overview


Introduction

I was asked to audit the smart contract "Trustless Rewards" in Clarity programming language. The contract is used on the site <https://stacksdegens.com/game-lobby>.

The purpose of the smart contract is to manage a list of game lobbies with leaderboards. An administrator should be able to set scores per lobby and distribute rewards to lobby guest according to the published scores.

This audit was executed and written by Friedger (OpenIntents). The final report was published on October 17th, 2022.

Code

The contract was already deployed on Stacks mainnet blockchain at the address `'SP1SCEXE6PMGPAC6B4N5P2MDKX8V4GF9QDE1FNNGJ trustless-rewards'`. The source code is also available on github copied from the blockchain for better review: Trustless Rewards  ,

A git repo was provided that contains tests for the smart contract. The tests for the contract are hosted at <https://github.com/BowTiedDeployer/trustless-rewards>. For the deployed contract, the git commit `eda81d` contains the relevant source code. The contract in this repo only differs in some comments and the used traits for SIP9 and SIP10. The deployed contract uses correctly the trait definitions mentioned in the SIP documents.

Scope and Approach

The audit focussed on the following topics:

- correctness
- misuse of funds
- error handling
- centralization
- runtime errors

For the audit, I used

- manual code inspection
- manual tests with clarity console
- reproducible tests

Findings

Description

The contract code is well structured and good to read. No issues of high risks were found. Tests cover 100% of the code.

The distribution of rewards is under the full control of the contract administrator. Users can verify how the administrator distributed the rewards because all information is on-chain. However, they have to trust the administrator to execute the distribution as intended.

I have implemented some of the recommendations <https://github.com/friedger/trustless-rewards/commit/134de04c4ccdd8a40a963fcee6aa413aa1754588>.

More changes have been added to the repository to address the issues mentioned in a first draft shared with the team. The state of the issues is reflected in the summary.

Summary

High Risk

None

Medium Risk

- Security: Use of traits with as-contract (won't fix)
- Correctness: Mismatch between assets and state (won't fix)

Low Risk

- Error Handling: No error on wrong id (fixed)
- Error Handling: Use of unwrap-panic (fixed)
- Error Handling: Use of unwrap-panic instead of asserts! (fixed)
- Performance: Unnecessary unwrap of map-get? result (fixed)
- Performance: Unnecessary unwrap of map-get? and wrap as result (fixed)

- Performance: Unnecessary unwrap of function call and wrap as result (fixed)
- Performance: Use of tuple with single value (fixed)
- Code Readability: Mix of state data with storage data (won't fix)
- Code Readability: Duplication of code (fixed)

Details

Medium Risk

Security: Use of traits with `as-contract`

In functions `nft-transfer` and `ft-transfer` traits are called in the contract context (`as-contract`) potentially giving the traits access to all assets of the contract. For more details, see <https://app.sigle.io/friedger.id/HuOT9tNQC8fTXOsK28D7e>.

The security risk is limited because these functions can be called only by the administrator (`contract-owner`).

Recommendations: Call these functions only with whitelisted contracts and always use appropriate postconditions.

Correctness: Mismatch between assets and state

The function `stx-transfer` allows the administrator (`contract-owner`) to change the assets owned by the contract while there is no on-chain information about the available funds. The function is probably intended to transfer commission fees. However, this functions can be abused by the administrator.

The function `finish-result-many` allows the administrator (`contract-owner`) to transfer any amount of Stacks tokens out of the contract.

The risk is limited because the function can be called only by the administrator (`contract-owner`).

Recommendations: Add a `total-current-balance` variable holding the current balance of all lobbies and guard the `stx-transfer` function (and even `nft-transfer` and `ft-transfer` for the reasons mentioned above) with a check whether there is a mismatch between the `stx` balance of the contract and the current total balance. Similarly, add a check in the `finish` function so that no more than the current balance of the lobby can be transferred out. The current balance will always be below or equal to the `balance` (total funds raised).

Low Risk

Error Handling: No error on wrong id

In function `disable-lobby`, the result is the same whether the lobby exists or not.

Recommendations: Return with an error if the lobby does not exist in the map.

Error Handling: Use of `unwrap-panic`

In function `add-balance`, the call to `stx-transfer` is wrapped in `unwrap-panic!`.
Thereby, errors in `stx-transfer` are not visible for the caller.

Recommendation: Use `try!` instead and return value of type `(Response bool uint)`.
Then wrap calls to `add-balance` in a `try!`.

Error Handling: Use of `unwrap-panic` instead of `asserts!`

In function `publish`, to check whether a user has joined the relevant lobby `unwrap-panic` is used.

Recommendations: Similar to the checks for active status of the lobby and administration privileges of the transaction caller, the check for lobby membership should use `asserts!` together with `is-some`.

Performance: Unnecessary unwrap of `map-get?` result

In function `add-balance`, the result of `map-get?` is unwrapped in a `match` call and in a `defaults-to` call.

In function `join`, the map result is unwrapped three times.

Recommendation: Use `let` to define variable `lobby`.

Performance: Unnecessary unwrap of `map-get?` and wrap as result

In function `get-lobby` and `get-score`, the result of `map-get?` is unwrapped and then wrapped into an `ok` response.

Recommendations: Return just the result of `map-get?` for the read-only function.

Performance: Unnecessary unwrap of function call and wrap as result

In private functions `publish-only` and `finish-only`, the result of `publish` and

`finish` is unwrapped and then wrapped into an `ok` response.

Recommendations: Use function result directly as result and remove `publish-only` and `finish-only`

Performance: Use of tuple with single value

The map `lobbies` uses a tuple with a single `uint` value as the key type.

Recommendations: Use `(define-map lobbies uint ...)` instead. For code readability, add a comment that the `uint` represents the id of the lobby.

Code Readability: Mix of state data with storage data

In function `create-lobby`, the map `lobbies` stores both data relevant for the state of the contract and data for storage in one tuple.

Recommendations: In map `lobbies` use tuple value containing one tuple `meta-data` for storage data and one tuple `values` for state relevant data. The tuple `values` should contain value `active` and `balance`, `price`.

Code Readability: Duplication of code

Functions `publish` and `finish` only differ in a function call `stx-transfer?`.

Recommendations: In function `finish` call the `publish` function.