RNDR Token Transfer Audit

OPENZEPPELIN SECURITY | NOVEMBER 29, 2018

Security Audits

The OTOY team asked us to review and audit their RNDR Token contracts. We looked at the code and now publish our results.

The audited code is located in the Token-Audit and the Token-Airdrop repositories. The versions used for this report are commits `e946177747e57312690775204834b8fca1bbb0d5` and `d96202acf6fb5d0305368bac36aa960d455cbffe` respectively.

Disclaimer: While the RNDR Token incorporates a migration strategy from a legacy token, we were not provided with the code of the token to be migrated, so this audit does not cover potential security issues associated with the intended migration. A new security audit is advised when the live migration is to be performed, for which it is suggested to use the coming version 2.0 of ZeppelinOS.

Here is our assessment and recommendations, in order of importance.

Update: *The OTOY team made some fixes based on our recommendations on commits* `bad2e4d0c283a1fa75840806a7026869dab057ad` *and* `0d71bb89d20a192453614c80be815880b2ef3eac` *of the Token-Audit and the Token-Airdrop repositories respectively. The updates in the issues below refer to these new commits.*

Critical Severity

None.

In the `Escrow` contract, the address of the associated token contract is tracked by the `renderTokenAddress` variable. The owner of the `Escrow` is allowed to change this variable at any time by calling `changeRenderTokenAddress`. If the owner called `changeRenderTokenAddress` passing the address of an account controlled by them as a parameter, they would therefore be allowed to call `fundJob` from that account and arbitrarily increase any job balance, at any point in time, without spending any tokens.

Consider analyzing the removal of the `changeRenderTokenAddress` function, or at least properly documenting the rationale behind its inclusion in the contract, so users are aware of such a dangerous scenario. While an attempt to do so was found in `Escrow.sol` , where a comment states that the function "*[…] is included as a failsafe*", the issues that this mechanism would prevent from occurring are never explained.

Update: *an event is now emitted when the `changeRenderTokenAddress` function is called, and documentation was added explaining the rationale for having this function.*

Unsafe arithmetic operations in Airdrop contract

The `Airdrop` contract contains a series of arithmetic operations which are not being addressed with caution (in lines 30 , 39 , 56 , 58 , 64 and 65) , leading to attempts to store numbers outside the range of the data types of their target variables. There are in particular two situations which could potentially cause integer overflows/underflows.

The first case is related to an assignment to a storage variable inside a loop in the `addManyUsers` function. This function iterates through a `_recipients` array of `addresses` and a paired `_amounts` array of `uint256`'s. In each iteration, the function `addUser` is called, which adds the respective user amount to the storage variable `totalBonus`, in charge of accumulating the sum. For an extremely large list of users and amounts, the variable `totalBonus` may reach its maximum possible value and finally overflow (*i.e.* start again from 0). In this scenario, an inconsistency between the total bonus sum and each user's bonus amount would be reached.

variable (or the function was externally called without parameters) , the unsigned variable `idTo` would be assigned the result of the operation 0 + 0 – 1, resulting in an underflow and a stored value of 2^{256} – – While in this case the immediately followin `if` `clause` would prevent something unexpected to happen, this approach is error-prone and not advised.

Consider using OpenZeppelin's `SafeMath` library to avoid underflows/overflows when doing mathematical operations.

Update: *the `SafeMath` library is now being used throughout.*

Contract owners can change business logic unnoticeably

In several contracts, the owner can arbitrarily change the business logic by setting new contract addresses, without properly warning users of those changes. Examples are:

- `RenderToken#setEscrowContractAddress` :the address that will hold tokens in escrow and keep a ledger of funds available for jobs.
- `Escrow#changeRenderTokenAddress` : the address of the token contract.
- `Escrow#changeDisbursalAddress` : the address authorized to distribute tokens for completed jobs.

Consider emitting events to notify users about any modifications of such importance in the contracts' business logic.

Update: *events are now emitted to provide users with a mechanism to track these changes.*

Inconsistent and experimental use of Solidity versions

Across the audited contracts, several versions of the Solidity compiler are used: `^0.4.18` in `Escrow`, `^0.4.14` in `RenderToken`, `^0.4.21` in the migration example contracts `LegacyToken` and `MigratableERC20`, and `^0.4.0` in `Airdrop`, which also uses the pragma `#experimental "v0.5.0"`.

While the individual contracts can be compiled using different versions of the Solidity compiler, profuse versioning among the same codebase is confusing and error-prone. As indicated by its

throughout the code.

Update: *Solidity version* `v0.4.24` *is now used throughout the project.*

Medium Severity

Two different minting functions coexist in the RenderToken contract

As per the disclaimer in the header, we haven't been able to assess the migration strategy for not having been provided with the code corresponding to the legacy contract to be migrated. Instead, the `LegacyToken.sol` and `MigratableERC20.sol` files were extracted from an example provided as a guide in an early version of ZeppelinOS. This guide explains how to migrate old token balances by burning old and minting new tokens, for which it introduced a `_mint` function in the new token, which extended `StandardToken` from `openzeppelin-zos` (now renamed to `openzeppelin-eth`).

Apart from the two copied files that function as placeholder, the `RenderToken` `contract` implements the `_mint` function that is declared in `MigratableERC20`, from which it extends. However, `RenderToken` also extends from `openzeppelin-zos`'s `MintableToken`, which already has a minting function named `mint`. Duplicating the minting functionality is confusing and potentially dangerous. Whichever form the migration ends up taking, consider using the already existing `mint` function for it.

Update: *The* `RenderToken` *contract now extends from* `openzeppelin-eth's` `StandardToken` *instead of* `MintableToken`*, and there is now a single minting function called* `_mintMigratedTokens`*.*

Input arrays with mismatched length will make addManyUsers throw

The `addManyUsers` function in the `AirDrop` contract, in charge of registering the `_recipients` of the airdrop and their respective bonus `_amounts`, simultaneously iterates over both arrays based on the length of just one of them (`_recipients`). If the number of elements in `_amounts` is less than that in `_recipients`, the whole transaction will be reverted for attempting to access an out-of-bounds index.

Update: *a* `require` *statement now checks for matching array length.*

Omission of the transfer in disburseJob leads to inconsistent balance state

The `RenderToken` contract interacts with the `Escrow` contract by funding jobs, transferring tokens to increase the different jobs' balances, which are tracked by the `Escrow`. The `disburseJob` function later takes care of redistributing these funds among the `recipients`.

This function, however, does not actually transfer the tokens to the recipients, but simply sets an allowance, which the recipients can then use to transfer the tokens themselves. While there is merit in using a pattern where the beneficiaries are in charge of withdrawing their funds, the `disburseJob` function will leave the `Escrow` in an inconsistent state where the sum of the total `jobBalances` is different from its total token balance, since the former are depleted by the function but the latter isn't.

Even if the `jobBalances` mapping cannot be traversed to get the total balance without an external listing of jobs, consider implementing the `Escrow` balance tracking in a way that doesn't lead to inconsistencies, or using the `PullPayment/Escrow` solution provided in the OpenZeppelin suite.

Update: *the* `disburseJob` *function now performs the token transfers.*

payUser fails silently if the bonus was already paid

In the `payUser` function of the `AirDrop` contract, an `if` clause is used to check whether a bonus has already been paid. In the case the condition `amount > 0` is not satisfied, the payment will not be performed, giving the caller no notice that it didn't go through apart from the lack of an associated event.

Consider complementing the `if` with an `else` clause that handles the logic when the condition fails.

Update: *an event is now emitted in case the* `if` *condition fails.*

Unchecked ERC20 transfer operation

would be emitted despite the transfers not getting their intended

Consider using OpenZeppelin's `SafeERC20` library and its `safeTransfer` `function`, or surrounding the transfer operation with a require statement.

Update: *the `SafeERC20` library is now used.*

Missing checks for null addresses in RenderToken and Airdrop contracts

In `RenderToken`, the `setEscrowContractAddress` function allows the token's owner to change the escrow's address (*i.e.* the contract variable `escrowContractAddress`). However, the function does not implement a check to prevent the null address from being set.

Similarly in `Airdrop`, the contract's constructor receives as a parameter an address that is assigned to the contract variable `renderTokenAddress` with no checks preventing the null address from being used.

Consider implementing no-null address validations before setting these variables to avoid potential problems downstream.

Update: *null checks are now [in(https://github.com/jeualvarez/Token-Audit/blob/master/contracts/RenderToken.sol#L88) place for address changes.*

Storage modification on event emission

In the `addUser` function of the `Airdrop` contract, an item is pushed into the `bonusAddresses` array. The result of the operation, which is the length for that array after the addition, is used as a parameter in the `AddedUser` event emission. This, while valid Solidity, is confusing and error-prone.

Consider performing the storage modification and keeping the resulting value in a temporary variable before emitting the event for code clarity.

Update: *the temporary value is now assigned to a variable before emitting the event.*

Low Severity

Event parameters are not indexed

balance histories, user additions or payments.

In case these are to be tracked, consider adding the `indexed` keyword to at least the `userAddress` variables in the `AddedUser` and `PaidUser` events, and the `_jobId` variable in the `JobBalanceUpdate` event.

Update: *the `AddUser` and `PaidUser` events now index their parameters, but the `JobBalanceUpdate` one still doesn't. String indexing had an associated [web3(https://github.com/ethereum/web3.js/issues/434#issuecomment-321526814) issue , which is purportedly solved in version 1.0. Alternatively, two workarounds for this issue are discussed here.*

Deceptive inline comment in Escrow contract

Considering the issue "*The Escrow's owner can arbitrarily increase the balance of any job without spending RNDR tokens*" above, the comment in `Escrow.sol` that states: "*Jobs can only be created through the RNDR contract*" is false and may be misleading for users reading the contract's code. Consider rephrasing it clearly to state that job balances can be arbitrarily incremented by whatever account the owner of the contract sets as the `renderTokenAddress`.

Update: *the comment was fixed to read: "Jobs can only be created by the address stored in the renderTokenAddress variable ".*

Missing error messages in require statements

There are several require statements (such as `Escrow` :L70 , `Escrow` :L89 , `Escrow` :L109 , `RenderToken` :L40 , `RenderToken` :L45) that provide no error messages. Consider including specific and informative error messages in all require statements.

Update: *all `require` statements now provide appropriate error messages.*

Missing docstrings in contract and functions in Airdrop contract

The `AirDrop` contract's source code, which handles token distribution, has no inline documentation whatsoever. Consider documenting with *docstrings* everything that is part of the

Untested functions in RenderToken

The `RenderToken` contract implements functions (*e.g.* holdInEscrow) that are not being tested in the test suite. Consider testing all functions implemented in contracts to ensure they behave as expected.

Update: *some testing of* `holdInEscrow` *was done in the* `Escrow.js` *file, but an additional test for this function was added to the* `RenderToken.js` *file.*

Broken testing instructions in README files

Instructions in `Token-Audit/README.md` file do not work if followed literally. An error "*Cannot find ./config module*" is thrown while running `npm test`. Instructions in `Token-Airdrop/README.md` file also do not work, with the error "*Could not find artifacts for Airdrop from any sources*" thrown while running `truffle test`. These errors might arise from differences in casing: the name of the contract in the `Airdrop.sol` file is `AirDrop`, but the artifact being required is `Airdrop`. This in turn might be due to the fact that in OS X, strings are case-insensitive. While this works on Mac's filesystems, it can lead to setup errors when working across different operating systems.

Consider updating the instructions and including a working cross-platform configuration so developers and auditors can successfully run the test suite. Furthermore, given that the test suite for the `Airdrop` contract can only be run using the `truffle v5.0.0-beta` release, consider including this version of truffle as a dev dependency in the `package.json` file of the project.

Update: *the* `README` *files were updated with new testing instructions.*

Erroneous documentation in initialize functions

Initialize functions (in `Escrow` and `RenderToken`) are incorrectly documented, since these functions are not contract constructors. Consider updating the inline documentation to fix these errors.

Update: *documentation now refers to the functions as initializers instead of constructors.*

explicit imports (*e.g.* missing imports for `Migratable` and `Ownable` in `Escrow.sol`, and for `MintableToken` in `RenderToken.sol`). Consider explicitly importing all necessary contracts in each contract throughout the codebase to improve code consistency and legibility.

Update: *all imports use now a consistent style.*

Inconsistent coding style among different files

There is a significant coding-style difference between the contracts in the `Token-Audit` repository and those in `Token-Airdrop` repository. The contracts in the first one use docstrings, libraries like OpenZeppelin's `SafeMath` , and 2-space indentation. The latter, on the other hand, has no comments in the source code nor takes security considerations into account by using already audited libraries, and uses 4-space indentation. Consider following best practices and applying the same style guidelines across all files.

Update: *both repositories use now a consistent coding style.*

Notes & Additional Information

- The addresses of job funders are presumably meant to be tracked off-chain, but consider adding an event-based tracking layer as a failsafe (*i.e., _emitting an event identifying the contributor in* `RenderToken`*'s* `holdInEscrow` *function).*
  _Update: *an event is now emitted.*
- In the `Airdrop` contract, consider prefixing all internal functions with an underscore to clearly denote their visibility.
  Update: *internal functions are now prefixed with an underscore.*
- Several public functions can be restricted to external. In particular: functions `fundJob`, `changeDisbursalAddress`, `changeRenderTokenAddress`, `disburseJob`, and `jobBalance` in the `Escrow` contract, and functions `addManyUsers`, `payManyUsers`, `finalizeList`, `returnTokens` and `getUserCount` in the `AirDrop` contract.
  Update: *these functions were restricted to external.*
- Consider including brackets in all control flow statements (*e.g.* in `Airdrop.sol`) , to prevent issues with future versions of the language.
  Update: *brackets were added.*

`Airdrop.sol` :L47).

Update: *all* `uint` *types are now explicitly* `uint256`.

- Variables `listFinalized` and `nextUserToBePaid` are explicitly initialized in `AirDrop`, but `totalBonus` is not. Consider initializing this last variable explicitly as well for code consistency.

  Update: `totalBonus` *is now initialized.*
- Consider explicitly marking all contract variables as private and defining getters/setters where appropriate, or at least explicit setting the visibility of all contract variables (*e.g.* `jobBalances` in the `Escrow` contract is not declared as public).

  _Update: `jobBalances` is now declared as `private`.
- In order to load the `Airdrop` contract to execute the payments, it is necessary that someone with minting privileges or enough tokens adds balance from a `RenderToken` contract to the address of the recently deployed `Airdrop` contract. For clarity purposes, consider expanding step number five on the instructions in `Token-Airdrop/README.md` file to clearly state this precondition.

  Update: *the* `README` *file has updated wording on this point.*

Conclusion

No critical and four high severity issues were found. Some changes were proposed to follow best practices and reduce the potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the OTOY RNDR Token contracts. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits