Decentralization Ambassadors Smart Contract Code Review Report

Introduction

Overview

This code review was performed by @clemlak for the Hydro team, as discussed here.

The goal of this code review was to evaluate the current master state of the Solidity code in the DA Program smart contract, in order to identify any areas of potential improvement.

The code review focused on several points, such as: security, syntax, extendability, functionality, gas usage, ...

Scope

The code review was performed on the following file: * DA.sol from the hydrogen-dev/smart-contracts/decentralization-ambassadors repository (commit hash `35db7183f0cfa286f9c06fb9528a6873097f3de4`).

The `DA.sol` file included the following declarations: * `Ownable` contract: This contract was taken from the Open Zeppelin framework, hence only a static analysis was performed on it, and no dynamic analysis. * `SafeMath` library: This library was also taken from the Open Zeppelin framework, hence only a static analysis was performed on it. * `addressSet` library: This library was created by the Hydro team and has been included in the code review, both in the static and dynamic analysis. * `DecentralizationAmbassadors` contract: This contract was the main target of this code review.

Nota: The following files has been added to the code review directory, but were NOT reviewed: `Voting/Voting.sol`, `Hydro/HydroToken.sol`, `Hydro/Ownable.sol`, `Hydro/Raindrop.sol`, `Hydro/SafeMath.sol`.

Methodology

This code review was performed through several steps:

1) A basic "human" analysis of the code, in order to understand how the contract should work and try to find any obvious errors or unexpected behaviors.

2) A static analysis of the code, performed using Solhint, Solium, Mythril and Smart Check.

3) A dynamic analysis, performed using Ganache and Truffle, Mocha, several tests and attacks.

Nota: The `setHydroAddress()` *function has been added to the* `DA.sol` *contract in order to use a custom instance of the Hydro contract and manipulate "test" Hydro tokens.*

Terminology

Impact	Description
Low	An issue that does not have an impact on the smart contract behavior or execution and is likely to be subjective.
Medium	An issue that could impact on the smart contract behavior or execution or introduce weakness that may be exploited.
High	An issue that represents a significant security vulnerability or failure of the smart contract.

Disclaimer

This code review was made from my current understanding of the best practises for Solidity and Smart Contracts. Development in Solidity and for Blockchain is an emerging area of software engineering which still has a lot of room to grow, hence my current understanding of best practise may not find all of the issues in this code and design.

Findings

The DA program smart contract CANNOT NOT be deployed at its current state.

Even though the expected behavior can be reached, a critical issue has been found regarding the payout of the tokens from the DA smart contract to the ambassadors. **Fixing this issue is mandatory before any deployment.**

Also, no limit regarding the maximum number of ambassadors has been implemented in the smart contract. I strongly recommend to set a fixed limit within the DA smart contract, and to check this limit before any ambassador nomination or addition.

Additionally, the current smart contract lets new ambassadors ask for an instant payout as soon as they are added to the program. I recommend to initialize each new ambassador `lastPayoutBlock` to the current `block.number` to make them wait at least 30 days before any payout.

Here are all the relevant findings that (in my opinion) need to be fixed and my recommendations to fix them:

High

- **Line 212 - Security** - The ambassadors can successively call the `recieveHydro()` function and withdraw an unlimited amount of tokens (as shown in the `DAAbusiveTest.test.js` test file). This issue is related to line 212, where the following operation is made: `lastPayout[msg.sender] = daLastPayoutBlock + payoutBlockNumber;`. To fix this issue, the `lastPayout[msg.sender]` variable should be set to the current `block.number`, after each payout.
- **Line 210 - Security** - The use of the `block.number` as a timestamp is risky, as the average block time may change. A good way to fix this issue is to use the timestamp provided by `now` or `block.timestamp`. Since miners can only manipulate time up to a couple of minutes, the use of a true timestamp is safe here.

Medium

- **Line 97, 106, 108, 109, 210, 212 - Security** - These lines contain unchecked math operations. Since underflows and overflows exist in Solidity, these operations may return unexpected results or be exploited. I recommend to use the SafeMath library to safely do these operations and to fix this issue.
- **Line 1 - Security** - The version of Solidity is not fixed. Thus, compiler may use a higher version of Solidity with major changes and leave the smart contract vulnerable. I recommend to fix it to `0.4.24`
- **Line 204 - Security** - The `receiveHydro()` function is misspelled (original name: `recieveHydro()`). The name of the function should be fixed in order to avoid any future confusion.
- **Line 207 - Security** - `this` is used to get the address of the contract, before checking its Hydro balance. This will be depreciated and may lead to unexpected result. `address(this)` should be used to fix this issue.
- **Line 148, 157, 167, 175, 182, 191, 199, 204 - Gas Usage** - All these functions are marked as `public`, however, they are never called internally. Marking these functions as `external` will reduce the gas used to execute them.
- **Line 141, 208 - Gas Usage** - The `lastPayout` and `daLastPayoutBlock` variables are currently `uint256`. Since these variables refer to a block number and since the contract should only be used during a 48 months timeframe, the maximum expected block number will be around 10 490 240. Hence, the variable type of `lastPayout` and `daLastPayoutBlock` can be changed to `uint24` (max value of 16 777 216) to save some gas. *Nota: Explicit conversion of these variables will be needed in the `recieveHydro()` function.*
- **Line 142 - Gas Usage** - The `payoutBlockNumber` is currently a `uint256`. The variable type can be changed to `uint24` to save some gas. *Nota: Explicit conversion of this variable will be needed at line 210 and 212.*
- **Line 143 - Gas Usage** - The `payoutHydroAmount` is currently a `uint256`. The variable type can be changed to `uint80` to save some gas. *Nota: Explicit conversion of this variable will be needed at line 207, 210 and 213. Also the `Payout` event parameters may be updated.*

Low

- **Line 142, 143, 145, 146 - Functionality** - The `payoutBlockNumber`, `payoutHydroAmount`, `hydroAddress`, `votingAddress` variables should have a fixed visibility level. Setting the visibility level to `public` for these variables can fix this issue.
- **Line 142, 143, 145, 146 - Extendability** - The `InitiateNomination` and `InitiateRemoval` events have almost the same name as functions within the `Voting.sol` contract. Events and functions should have explicit different names, in order to avoid confusion. For example, `Log` can be added at the beginning of the name of the events.
- **Line 27, 36, 153, 158, 168, 176, 183, 192, 207 - Extendability** - The `require` used at these lines do not return any error message. Adding error messages to the `require` will fix this issue.
- **Syntax** - Several syntax errors have been found:

Line(s)	Category	Description	Recommendation
8	Syntax	Missing 1 blank line before the Ownable contract definition.	Add a new blank line at line 2.
11	Syntax	Useless blank line at line 11.	Remove blank line at line 11.
19	Syntax	Constructor function of Ownable contract has 1 additionnal blank line before its definition.	Remove blank line at line 14.
40	Syntax	Useless blank line at line 40.	Remove blank line at line 40.
47	Syntax	Missing 1 blank line before the SafeMath library definition.	Add a new blank line at line 42.
48	Syntax	Useless blank line at line 48.	Remove blank line at line 48.
49 - 86	Syntax	Found indentation of 2 spaces instead of 4 spaces in the SafeMath library.	Change indentation from 2 spaces to 4 spaces.
89	Syntax	Contract name "addressSet" should be in CamelCase.	Change "addressSet" library name to "AddressSet".
89	Syntax	Missing 1 blank line before "addressSet" library definition.	Add a new blank line at line 88.
124	Syntax	Missing 1 blank line before "Voting" interface definition.	Add a new blank line at line 124.
129	Syntax	Missing 1 blank line before "HydroToken" interface definition.	Add a new blank line at line 129.
134	Syntax	Missing 1 blank line before "DecentralizationAmbassadors" contract definition.	Add a new blank line at line 134.
168 - 169	Syntax	Missing a blank line between lines 168 and 169.	Add a new blank line between lines 168 - 169.
170	Syntax	Indentation of 10 spaces instead of 12.	Change indentation from 10 spaces to 12 spaces at line 170.
171	Syntax	Indentation of 10 spaces instead of 12.	Change indentation from 10 spaces to 12 spaces at line 171.
194	Syntax	Indentation of 10 spaces instead of 12.	Change indentation from 10 spaces to 12 spaces at line 194.
195	Syntax	Indentation of 10 spaces instead of 12.	Change indentation from 10 spaces to 12 spaces at line 195.
207	Syntax	Useless additional space before ">=".	Remove the space before the ">=".
210	Syntax	Missing space before the "{".	Add a space before the "{".
223	Syntax	Useless blank line at line 223.	Remove blank line at line 223.

Conclusion

During this code review, important issues regarding security (marked as *High* in the **Findings** section) have been found in the DA Program smart contract. These issues lead to vulnerabilities in the contract and must be fixed as soon as possible. Furthermore, fixing the *Medium* and *Low* issues too is **strongly** recommended.

Moreover, the DA Program smart contract is meant to be used with the Voting smart contract. Since this contract was not finished during this code review, a specific attention should be paid in order to avoid any unexpected behaviors in the DA smart contract due to external calls from the Voting contract.

Appendix

The following reports can be found in the `/reports` directory.

Tool reports

- Solhint analysis report
- Solium analysis report
- Mythril analysis report
- SmartCheck analysis report

Test reports

- DA Basic test report
- DA Abusive Test Report
- AddressSet Test Report

Nota: The tests were performed using Ganache-cli, through a forked blockchain of the Main net from Infura. The script can be found in `ganache`*.*

References

This code review was made using the following tools, guides, techniques and frameworks: * Solium * Solhint * Mythril * Smart Check * Truffle * Ganache * Mocha * miguelmota/solidity-audit-checklist * ConsenSys/smart-contract-best-practices