SFXDX

Libertas: Token, Tipping and Staking Pool

Smart Contract audit report

By: SFXDX Team
For: Libertas Project

www.sfxdx.com

Libertas: Token, Tipping and Staking Pool:
Smart Contracts audit report

The following Audit was prepared by the Sfxdx team and the purpose of this audit is to analyze the codebase of three smart contracts with an overview and a description of the bugs and vulnerabilities found and fixed.

This document describes the compliance of Libertas platform`s Smart Contracts with the logical, architectural and security requirements.

During the audit, we classified the shortcomings into four categories of severity:

- Informational - shortcomings that are described in the document for informational purposes only. They are not required for correction, do not violate the intended architecture and business logic.

- Low - shortcomings that do not violate the requirements, but can be eliminated to improve the quality of the code. Their elimination can be considered as a recommendation.

- Medium - shortcomings that are not serious, but should be better eliminated. Their elimination can be considered as a recommendation.

- High - shortcomings that lead to the fact that the smart contract does not meet the requirements. For example, business logic violation, inconsistency with the planned architecture etc. This type of shortcomings must be eliminated from the smart contract code.

All security and other critical issues were resolved.

A more detailed description of the found issues is provided in the document below.

Introduction

Git repository with contracts:

https://github.com/TheLibertasProject/TippingContract

Audited contracts:

LibertasToken.sol
Tipping.sol
StakingPool.sol

Below are summary conclusions regarding the contract codebase considering the business logic analysis, correctness of logic implementation, security analysis, gas efficiency analysis and style guide principles.

LibertasToken

26.10.2021

https://github.com/TheLibertasProject/TippingContract/commit/a434a555e1409e98299811f6ae5b34deff62f702

The “LibertasToken” contract audit results includes the defined shortcomings of the following categories:

Severity		Count
Informational	-	1
Low	-	2
Medium	-	1
High	-	1

The shortcomings are detailed below.

Hardcoded name, decimals and symbol of the token

Informational

The token contract is inherited from Openzeppelin Contract Library realization of the ERC20 standard [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol], which implies to pass both name and symbol for the token in constructor. Also the methods “decimals()” return a hardcoded amount of 2. This approach may be efficient as gas optimisation, but should either name or symbol, or decimals require edit, the contract won’t allow it.

Taken measures: create a custom getter and setter for the token.

Duplication of versioning

Low

The token contract has a field called “version”. Since the token use Transparent Upgradeability Proxy pattern to allow upgrade functionality, this function is redundant due to versioning in TransparentUpgradeableProxy [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/transparent/TransparentUpgradeableProxy.sol] parent of the token contract.

Initial cap value is not captured

Low

The token contract mints an initial cap at the deploy stage. The cap is stored then in the “totalSupply()” function value, which alters over the time according to the burning scheme of the tokenomics. So it would improve readability of the code and readability of previous states of the contract if the cap was stored in a constant.

Taken measures: Store initial cap at state constant.

The contract instance is not implicitly convertible to the address type

High

The token contracts method “approveAndCall(...)” is misusing the instance type of the state variable “this”. It’s not implicitly convertible to the “address” type, so it needs to be converted explicitly.

Taken measures: Change “this” to “address(this)”.

Duplicating standard realization of ERC20

Medium

The token contract inherits from a complete copy of standard realization but with possible security issues because of handicrafted solution. The Openzeppelin Contracts Library keeps track of modern issues of the ERC20 standard and implements all possible security checks at the moment, so not using the prewritten ERC20 realization could lead to unknown security issues.

Taken measures: Remove StandardToken and its children and replace it with ERC20 from Openzeppelin Contracts Library [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol].

Tipping

26.10.2021

https://github.com/TheLibertasProject/TippingContract/commit/a434a555e1409e98299811f6ae5b34deff62f702

The “Tipping” contract audit results includes the defined shortcomings of the following categories:

Severity		Count
Informational	-	2
Low	-	5
Medium	-	0
High	-	0

The shortcomings are detailed below.

Naming of non-constant fields are not by style guides

Informational

The Tipping contract has several constant fields such as:

```
address public _STAKING_VAULT;
address public _LIBERTAS;
address public _FUND_VAULT;
```

The naming of those fields violates style guide recommendation [https://docs.soliditylang.org/en/v0.8.7/style-guide.html#local-and-state-variable-names] about using naming of state variables.

Taken measures: rename field according to style guides.

Too low granularity for rates

Low

The Tipping contract uses rate coefficients that are in range [0..1000]. This range restricts the variance of the rates therefore restricts flexibility of the usage.

The basis points prove both flexibility of usage and ease of use. If someone were to review or re-audit the smart-contracts or the tokenomics, it would be much easier to understand if the system were using the basis points as a metric of rates.

Taken measures: Expand rate ranges from [0..1000] to [0..10000].

Using pulling of accumulated funds to vaults by the admin not by the user

Informational

It is considered more capital efficient for the product owner, to make users pay fees for their transfer transactions. But in some cases, when the product targets the broader audience, it would be more convenient to implement the payment of transfer of some admin fees by admin.

Taken measures: the contract could sent the base of the transfer (already with all fees taken) excluding the following transfers:

```
_libertas.safeTransfer(address(0), burnAmt);
_libertas.safeTransfer(_FUND_VAULT, fundAmt);
_libertas.safeTransfer(_STAKING_VAULT, rewardAmt);
```

Then, to implement intended functionality, add the “burntAmt”, “fundAmt”, “rewardAmt” to the fields that aggregates sums of the parameters, then move transfers above to different admin only function, but instead of “burntAmt”, “fundAmt”, “rewardAmt” parameters pass the aggregated sums of the parameters. So the fees transfer will be on the admins expense, not the users.

Duplicity of IERC20 interface use

Low

The Tipping contract uses a custom ILibertasToken interface which fully duplicates the IERC20 standard Openzeppelin Contract Library interface.

Taken measures: Delete ILibertasToken.sol and replace it with IERC20 usage [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol].

Redundant “_transfer” internal method

Low

In the main method “transfer” of the Tipping contract used the internal “_transfer” method to move tokens and its sub-groups (fees). This method is just wrapping up DELEGATECALL to the LIBERTAS token and does nothing more, so the extraction of it into another internal function is useless and it increases the contract size.

Taken measures: Remove “_transfer” method and replace it with “safeTransfer” method of the SafeERC20 library from Openzeppelin Contracts Library [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol].

Redundant “_transferFrom” internal method

Low

The Tipping contract uses the internal “_transferFrom” method to move tokens and its sub-groups (fees). This method is just wrapping up DELEGATECALL to the LIBERTAS token and does nothing more, so the extraction of it into another internal function is useless and it increases the contract size.

Taken measures: Remove “_transferFrom” method and replace it with “safeTransferFrom” method of the SafeERC20 library from Openzeppelin Contracts Library [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol].

Redundant “_validateTransfer” internal method

Low

The Tipping contract uses the internal “_validateTransfer” method to check if transfer could work out well. But the logic of it already realized in the ERC20 contract (standard realization from Openzeppelin Contract Library [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol]). So this method is redundant

Taken measures: Remove “_validateTransfer” method.

StakingPool

26.10.2021

https://github.com/TheLibertasProject/TippingContract/commit/a434a555e1409e98299811f6ae5b34deff62f702

The “StakingPool” contract audit results includes the defined shortcomings of the following categories:

Severity		Count
Informational	-	0
Low	-	7
Medium	-	0
High	-	1

The shortcomings are detailed below.

Duplicity of the reward calculation in LIBERTAS tokens

Low

The StakingPool smart-contract has multiple lines with duplicate logic of the reward calculation in method “claim” at line 70 and in method “availableReward” at line 82.

```
function claim() external {
        ...
        uint256 pending = user.amount
            .mul(accLibertasPerShare).div(1e12).sub(user.rewardDebt);
       ...
}
function availableReward() external view returns (uint256) {
      ...
      uint256 pending = user.amount
            .mul(accLibertasPerShare).div(1e12).sub(user.rewardDebt);
     ...
}
```

Taken measures: Extract the reward calculation in the internal function of the smart-contract.

Opportunity to merge if statement

Low

At the line of 36 and 42 of the contract StakingPool, there are two “if” statements, which could be merged to one.

Taken measures: Merge “if” statements.

A scalar value extractable to constant variable

Low

In all the methods of the StakingPool there are several encounters of the 1e12 scalar value. Its purpose is to normalize the value of the product of user staked amount and token reward per share.

Taken measures: To bring up readability of the contract extract the scalar to the constant state variable.

Unauthorized access to the supply reward function

High

The method “supplyReward” could execute anyone so some malicious party could manipulate the reward rate.

Taken measures: Inherit the contract from AccessControl of Openzeppelin Contracts Library [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol] restrict the usage of the “supplyReward” to the role “DEFAULT_ADMIN_ROLE”.

Misuse of underscore naming

Low

The style guides of the solidity language says about underscore as opportunity to avoid naming collisions - [https://docs.soliditylang.org/en/v0.7.0/style-guide.html#avoiding-naming-collisions]. In some cases underscore could be used to mark either local variables or private state variables, not both at the same time.

Taken measures: Remove underscore naming.

Misuse of “public” modifier

Low

The “public” modifier could be used if the function in the StakingPool is used both in external calls and internal calls. But almost all of the functions are used only externally.

Taken measures: Change modifier “public” to “external”, it would save some gas.

Redundant revert messages

Low

Every bit of chars if the source file of the StakingPool contract costs gas. So for the sake of gas economy it would be nice to use either error codes or Camel Case standard in error messages. The second approach would be in this kind of project more efficient, because there is not so much contract, and it would be both quicker and easier to read the errors in the tx and save some gas. Why Camel Case? It’s readable and does not generate whitespace characters or underscores or any other characters that not used in words.

Taken measures: Rewrite error messages to Camel Case standard.

Ambiguous return of “claim” function

Low

The return of the “claim” function is considered ambiguous because of the lack of return information about success or failure. And to fetch this information from storage is harder for the average user.

Taken measures: Rewrite “if” statement to “require” function usage in the “claim” function.