PUBLIC

Code Assessment

of the Staking and Tokens Smart Contracts

July 14, 2021

Produced for

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Sir or Madam,

First and foremost we would like to thank Rarible Inc. for giving us the opportunity to assess the current state of their Staking and Tokens system. This document outlines the findings, limitations, and methodology of our assessment.

This assessment has revealed two medium and numerous low severity findings. While most of them were fixed for the final version of this report, one low severity issue was only partially addressed, because the contracts are already deployed. Potentially this issue can cause problems with hardware wallets integrations.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2
• Code Corrected	1
• Specification Changed	1
Low-Severity Findings	14
• Code Corrected	11
• Specification Changed	2
• Code Partially Corrected	1

2 Assessment Overview

In this section we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Staking and Tokens repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	21 June 2021	cd18f3164d6c733eedbfd8a9e53a372b804a4a52	Staking Version 1
2	21 June 2021	2cf9b0c03668e9b2ec78b69fa77f528894b3d097	Tokens Version 1
3	9 July 2021	e9fe7a0c61e05960de2b20296cf76e06f99bfe8f	Version 2
4	14 July 2021	1c397bca6fcb236016c79dff0dcc44580cf3d6a0	Version 3

For the solidity smart contracts, the compiler version `0.7.6` was chosen.

For Tokens, solidity contracts in following folders at corresponding commit are in scope:

- `tokens/contracts/erc-721`
- `tokens/contracts/erc-1155`

For Staking contract, solidity contracts in following folders at corresponding commit are in scope:

- `staking/contracts`

2.1.1 *Excluded from scope*

Any imported libraries, contracts and interfaces that were not in scope folders. Any factory contracts and proxy contracts that may interact with reviewed contracts were not in scope of this assessment.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section we have added a version icon to each of the findings to increase the readability of the report.

The 3 separate smart contracts were part of the audit: the staking contract, an ERC721 token contract, and an ERC1155 token contract.

2.2.1 *ERC721 and ERC1155 Tokens*

The ERC721 and ERC1155 contracts are build using OpenZeppelin's upgradeable token templates. The following additional functionality was implemented:

- Multiple creators. A minted token is associated with one or multiple creators, namely the addresses that signed the minting data. Each creator of a token owns a share of it. The shares of a token need to add up to 10000, which represents 100% with 2 decimal places of precision.

- Mint and transfer. The function `mintAndTransfer` receives the minting data and signatures of creators, and then mints and transfers a token within the same contract call. The first creator of the token is the **minter** of the token. The `mintAndTransfer` function needs to be called by the minter or by a party approved by the minter.
- Default operators. The owner of the contract can assign the role of **default operator**. A default operator can transfer tokens of any user of the contract without prior approval from the user. The approval of a default operator cannot be revoked, and they can `mintAndTransfer` tokens on behalf of any minter.

There also a "User" version of the ERC721 and ERC1155 token contracts that does not have the "default operator" functionality. According to Rarible Inc., these contracts are meant to be deployed using a proxy factory. They enable users to have their own contracts with the `mintAndTransfer` functionality. The minter of tokens on these contracts can only be the owner of the contract.

2.2.2 Staking contract

Staking contract allows users to lock ERC20 tokens for a period of time. The smallest unit of time that the contract tracks is one week. Each individual stake is identified by an ID and consists of 2 lines. The first line is the **lock line** and it is defined by the following 3 parameters:

- Bias - the amount of tokens that the user locks inside the contract.
- Slope - the mount of tokens that are unlocked every week.
- Cliff - the period (in weeks), during which there is no unlocking.

The second line is the **stake line**, which equals the lock line scaled (multiplied) by a coefficient. That coefficient is computed based on cliff and slope period durations of the lock line. The longer any of the two periods is, the greater the scaling coefficient will be. Each unique stake is associated with an owner and a delegate addresses. The delegate address is the address whose stake balance is increased by the stake line of the given stake. The **stake balance** of the delegate address is the sum of all stake lines of all stakes that are delegated to it. The owner of a line can do following actions with the lock lines:

- Change the delegate of a stake line using the `delagateTo` function.
- Delete an existing lock line (identified by its ID), as well as update a lock line with a new bias and slope, provided that the bias is above the current locked amount and the new line terminates no earlier than the original line. That can be done by using `restake` function.
- Withdraw the already unlocked funds from all owned stakes. The amount still locked is determined by the sum of all lock lines owned by address.

The staking is assumed to be done with the Rari ERC20 token .

The staking contract can be stopped by the contract's owner. After stopping, no stakes can be created or changed. Users can withdraw all the tokens that they had in the contract.

The contract's owner can put the staking contract into a migration mode. In this mode stakes can be created and changed, but the user can also `migrateTo` specific ID stakes into a new contract.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Tokens EIP-721 Typehash Code Partially Corrected

5.1 Tokens EIP-721 Typehash

Correctness Low Version 1 Code Partially Corrected

The EIP-712 defines set of rules, how the solidity structs should be hashed.

Current LibERC721LazyMint and LibERC1155LazyMint contracts have few violations of this standart:

- Mint1155Data struct has `uri` field, but `tokenURI` is used in MINT_AND_TRANSFER_TYPEHASH
- In Mint1155Data struct, the `supply` field follows the `uri` field. In MINT_AND_TRANSFER_TYPEHASH, the `supply` field follows the `tokenId` field.
- Mint721Data struct has `uri` field, but `tokenURI` is used in MINT_AND_TRANSFER_TYPEHASH

According to EIP-712, such mismatches are not compatible with the standard.

Code partially corrected:

Field `uri` was renamed to `tokenURI` in both contracts. Currently, these contracts are already deployed and changing the `supply` and `tokenId` order cannot be fixed.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2

- ERC721 Use of Unsafe Methods Code Corrected
- Staking Formula Differs From Specificaiton Specification Changed

Low-Severity Findings	13

- Mismatch in Mint Data Specification Specification Changed
- Code Is Not Compilable Specification Changed
- Initialization of Staking Contracts Not According to OZ Guidelines Code Corrected
- Initializer Not Using Unchained Initializer of Ancestors Code Corrected
- Missing updateAccount Function in ERC1155Lazy Code Corrected
- Solidity Compiler Versions Code Corrected
- Staking Coefficient Can Make Stake Line Longer Code Corrected
- Staking Contracts Are Missing __gap Field Code Corrected
- Staking Events Data Code Corrected
- Staking Exposing Getters Code Corrected
- Staking Is Possible When Migrating Code Corrected
- Staking Restake Can Cut the Corner Code Corrected
- Staking Slope Period Definition Code Corrected

6.1 ERC721 Use of Unsafe Methods

Design Medium Version 1 Code Corrected

The ERC721 standard focuses on ensuring that token transfers do not lock / loose tokens. That is why the use of "safe" functions such as `safeTransferFrom` was introduced. This applies not only to transfers, but to minting as well. However, the implementation of `mintAndTransfer` in the contract `ERC721Lazy` does not use the "safe" `_safeMint` but the `_mint` function, whose use is discouraged.

Code corrected:

Function `mintAndTransfer` now uses `_safeMint` function.

6.2 Staking Formula Differs From Specificaiton

Design | Medium | Version 1 | Specification Changed

The specs define the formula as:

```
stake = k * tokens.
K = (0.07 + 0.93 * (cliffPeriod / 104) ^ 2 + 0.5 * (0.07 + 0.93 * (slopePeriod / 104) ^ 2)).
```

This formula differs from solidity implementation, mostly due to use of following constants:

```
uint256 constant ST_FORMULA_MULTIPLIER = 1081000;        //stFormula multiplier = TWO_YEAR_WEEKS^2 * 100
uint256 constant ST_FORMULA_COMPENSATE = 1135050;        //stFormula compensate = (0.7+0.35) * ST_FORMULA_MULTIPLIER
uint256 constant ST_FORMULA_SLOPE_MULTIPLIER = 465;      //stFormula slope multiplier = 0.93 * 0.5 * 100
uint256 constant ST_FORMULA_CLIFF_MULTIPLIER = 930;      //stFormula cliff multiplier = 0.93 * 100
```

- `ST_FORMULA_MULTIPLIER` should be 1086000 to comply with specs.
- `ST_FORMULA_COMPENSATE` should be 1135680 to comply with specs.
- `ST_FORMULA_SLOPE_MULTIPLIER` should be 46.5 to comply with specs.
- `ST_FORMULA_CLIFF_MULTIPLIER` should be 93 to comply with specs.

Specification corrected:

Specificaion of formula was changed in commit `6167554ff40b7b7f9f6d1ce808fd7d62d04ab3f6` to:

```
stake = K * tokens / 1000;
K = ( 11356800 + 9300 * (cliffPeriod)^2 + 4650 * (slopePeriod)^2) / 10816;
```

Code was adjusted in commit `888761566077d51937cf7676417ebe0839f2bdfe` implemented according to this specificaion.

6.3 Mismatch in Mint Data Specification

Correctness | Low | Version 2 | Specification Changed

The `Mint721Data` and `Mint1155Data` structs in LibERC721LazyMint.sol and LibERC1155LazyMint.sol were updated. Yet, the documentation in `tokens/readme.md` was not updated. The documentation specifies that the mint data structs have element `uri`, while in code `uri` was changed to `tokenURI`.

Moreover, both structs have an element `royalties`, that are named as `fees` in the `tokens/readme.md`. That is a mismatch in the specification.

Specification corrected:

The names were fixed in version 3. Names in specs match the ones in code now.

6.4 Code Is Not Compilable

Design | Low | Version 1 | Specification Changed

Staking contract and token contracts are not compilable due to imports of `@rarible` libraries. Node pulls outdated libraries that are not compatible with the new code. If imports by path are used, the code would have been compilable. Not using direct imports by path can lead to compilations with outdated libraries. Taking into account the monorepo structure of rarible github, it is safer to use direct path imports.

Specification corrected:

The compilation instructions were added to the readme.

6.5 Initialization of Staking Contracts Not According to OZ Guidelines

Design | Low | Version 1 | Code Corrected

The initialization style of the contract is not according to guidelines from OpenZeppelin libary. In contract `StakingBase` a `__StakingBase_init` function is defined as public function. Usually such functions are defined as private and only most derived contract defines

public function `initialize` where the private chained/unchained init function are called.

Moreover, the initializer should call all unchained initializers of all ancestors of the contract. Currently the `__StakingBase_init` calls `__Ownable_init_unchained` but not `__Context_init_unchained`. The latter has no effect in current version of the OZ library, but that can change in future.

Code corrected:

`__StakingBase_init` was renamed to `__StakingBase_init_unchained`. It does not call any initializer of ancestors anymore and was made internal. The initialization was moved to the Staking contract where `__Staking_init` calls all unchained initializers of parent contracts. The guidelines are now followed.

6.6 Initializer Not Using Unchained Initializer of Ancestors

Design | Low | Version 1 | Code Corrected

The contract `Mint721Validator` implements `__Mint721Validator_init_unchained` function, that calls `__EIP712_init` from OpenZeppelin's `EIP712Upgradeable` contract. The OpenZeppelin docs suggests using unchained initializers of parent contracts to prevent initializing a contract twice.

Code corrected:

`__Mint721Validator_init_unchained` calls `__EIP712_init_unchained` now.

6.7 Missing `updateAccount` Function in `ERC1155Lazy`

Design | Low | Version 1 | Code Corrected

The contracts should support Rarible on-chain royalties. As in `ERC721Lazy` there should be a method calling `_updateAccount` from `AbstractRoyalties`. That enables a registered royalty account to change the address for the royalties for some token ID. Such function is missing in the `ERC1155Lazy`.

Code corrected:

The `updateAccount` function was added. It allows to perform the previously described actions.

6.8 Solidity Compiler Versions

Design | Low | Version 1 | Code Corrected

The defined solidity versions are too different across staking contracts:

```
pragma solidity >=0.6.2 <0.8.0;
pragma solidity ^0.7.0;
```

Too wide version range `>=0.6.2 <0.8.0` can cause compilation with not the latest compiler version, where bugs can present.

Code corrected:

The Solidity compiler version was set to `0.7.6` for all contracts in scope.

6.9 Staking Coefficient Can Make Stake Line Longer

Design | Low | Version 1 | Code Corrected

When the staking bias and slope are computed, the computation is done with a certain precision. If bias has a significant difference compared to slope, the resulting params of stake line can make it longer compared to locked line. For example, bias of 5200, with slope of 100 and cliff of 52 weeks, the stake line will have bias of 23592 and slope of 453. Right after 104 weeks, the lock line will be depleted, and all funds will be withdrawable. The stake line after the same period will have remaining 36 (23592 mod 453) stake that will unlock only on 53rd week. This behavior is not documented in specs and violates the statement that stake behaves like locked tokens and is just multiplied by value.

Code corrected:

Calculations were changed for the staking bias and slope values. The `slopePeriod` is now defined as `ceil(amount / slopePeriod)`. The staking coefficient affects only staking bias. The staking slope value is computed as `ceil(staking amount / slopePeriod)`. As a result of this changes, the "remained" will be counted as a last period of `slopePeriod`. The stake line and balance line will deplete at the same time.

6.10 Staking Contracts Are Missing __gap Field

Design | Low | Version 1 | Code Corrected

All staking contracts, `Staking`, `StakingRestake` and `StakingBase`, do not have the recommended `__gap` field for upgradeable contracts. Without this field adding new state variables in the future can be problematic.

Code corrected:

`__gap` was added to the above mentioned contracts.

6.11 Staking Events Data

Design | Low | Version 1 | Code Corrected

Some events emitted in staking contract do not provide enough information to easily reconstruct the state of contract.

- In function `restake` the emitted event Restake does not emit new value of `counter` field. Also the `account` that performs restake is not emitted. The StakeCreate emits both those fields.
- Event Delegate does not emit `account` that performs redelegation.
- No events emitted on `stop` function call. That is important function that greatly impacts the contract functionality.
- No events emitted on `startMigration` function call. That is important function that greatly impacts the contract functionality.

Code corrected:

The events are now emitted for all actions, described above.

6.12 Staking Exposing Getters

Design | Low | Version 1 | Code Corrected

In current implementation of the staking contract some fields and data is hard to query due to missing getters. The list of information that can improve user experience and minimize human errors:

- Output of `getStake` function. Currently it is internal pure function.
- Remaining locked amount
- Amount available for withdrawal
- For a given Line id, the owner and delegate addresses.
- Field `stopped` of the staking contract
- Getting "current week" of the contract. E.g. `roundTimestamp` function output.

Code corrected:

Following changes were done, to fix the issues from the above list:

- Function `getStake` is declared as public.
- Function `locked` was added.
- Function `getAvailableForWithdraw` was added.

- Function `getAccountAndDelegate` was added.
- Field `stopped` was made public.
- Function `getWeek` was added.

6.13 Staking Is Possible When Migrating

Design | Low | Version 1 | Code Corrected

The migration is irreversible process that is needed to move stakes to a new contract. But such actions like `stake`, `delegateTo` and `restake` are still allowed during this process. Users can by mistake performed such actions and will need to submit extra transaction to migrate their actions to a new contract.

Code corrected:

Function mentioned above now use `notMigrating` operator that restricts their usage when the contract is migrating.

6.14 Staking Restake Can Cut the Corner

Correctness | Low | Version 1 | Code Corrected

The `restake` function checks that the new amount of locked tokens is not lower than the old amount. Also there is a check that the end time of the new line should not be earlier that the old end time. These constraints allow the user to restake with a long slopePeriod, but without a cliffPeriod, which leads to stake being unlocked earlier than in the original line. Geometrically, this amounts to "cutting the corner" of the brokenline.

Code corrected:

The `restake` function now performs checks, that the new line is greater (not strictly) then the old line. Geometrically, this can be interpreted as new line to be always "above or equal" the old stake line. This is done by a call to `verification` internal function.

6.15 Staking Slope Period Definition

Correctness | Low | Version 1 | Code Corrected

The current slope period is computed as `amount // slope`, that is using an integer division that rounds down. The effect of this is that, in the week after the slope period ends, the `amount % slope` tokens will still be staked. This remainder does not contribute to the staking coefficient K. The remainder cannot be withdrawn, but can be delegated to and restaked. Also this extra period does not counted towards the 2 year limit for the staking period. The current version of the specification does not describe this behavior.

Code corrected:

The new slope period is computed as `ceil(amount // slope)`. Now the "remainder" week is accounted for staking coefficient and 2 year limit.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Missing Documentation for Public Functions

Note Version 1

Some functions and state variables in StakingBase.sol are now public. However, functions and getters are not documented. Following functions and variables are lacking documentation in the readme:

- `getStake`
- `counter`
- `stopped`
- `migrateTo`
- `totalSupplyLine`