# Security Audit
# Decentraland
# Collections v2 contracts

Tuesday, 13-Oct-2020

Agustín Aguilar

# Introduction

The Decentraland team requested a security audit of their system of contracts that compose the Collections v2 system. The audited repository follows:

https://github.com/decentraland/wearables-contracts

The contracts that constitute the project follow.

- **ERC721BaseCollectionV2**: ERC721 contract that allows defining internal items, with their ID and supply restrictions.
- **MinimalProxyFactory**: Enables the creation of child proxies connected to a single implementation contract.
- **ERC721CollectionFactoryV2**: Inherits MinimalProxyFactory, transfers the owner of the child contracts to the ownership of the collection factory upon creation.
- **CollectionStore**: Mints collection tokens in exchange for a defined ERC20 token.
- **Strings**: Utility contract is used to convert Solidity variables into their string representation.

## Fuzz testing

The following invariants have been tested using echidna[1], with at least 50.000.000 iterations.

- Single item supply cannot go beyond 100.000
- Single item supply cannot go beyond specified by rarity
- Collection cannot be approved and not completed
- The Balance of a single address cannot exceed the total supply
- ERC721 total supply cannot exceed the total item supply
- The price must be set, or the item must have no beneficiary

No broken invariants have been found.

# Issues

## Low severity

### L1 - Proxy factory can change the address of deployed contracts

The `MinimalProxyFactory` contract implements the method `getAddress`, used to retrieve the address of a deployed or undeployed contract given the salt and the `msg.sender` used on the `createProxy` call.

This method counterfactually computes the contract's address using those values and the current `codeHash`, which is a derived value from the current `implementation`.

[1] Echidna fuzz testing framework (https://github.com/crytic/echidna)

The `implementation` of the factory is an upgradeable property that will cause the factory to return a different address on `getAddress`, even if the child contract has already been deployed.

Proposed solutions:

a) Make `implementation` an immutable value in the factory's context and deploy additional factory contracts if multiple implementations are required.

b) Store the address of already deployed contracts on the factory storage, use the counterfactual address only if a contract with that salt + sender has not been deployed.

c) Make `implementation` a parameter of both `createProxy` and `getAddress`.

# Notes

Optimizations, unclear cases and minor findings, presented on no specific order.

## N1 - Compute proxy code without using storage

The `MinimalProxyFactory` creates child proxy contracts by inserting the implementation contract's address in the middle of a generic proxy creation code, forming the creation code for a proxy contract pointed to the implementation.

This creation code is computed at `_setImplementation` and stored on the contract storage to later be retrieved during `getAddress` and `createProxy`.

The cost of loading a variable-length bytes variable from storage can vary between 600 and 1200 gas, and it is bound to increase to 5000 gas on the upcoming Ethereum forks[2], while the cost of computing `code` from the implementation can go as low as 100 gas[3].

Proposed solution:

a) Remove `code` and `codeHash` from contract storage, compute those values using `implementation` when necessary.

b) Remove `implementation` from the contract storage, make it a parameter of both `createProxy` and `getAddress`.

[2] EIP-2929 Draft (https://github.com/ethereum/EIPs/pull/2929)
[3] Ethereum yellow paper (https://ethereum.github.io/yellowpaper/paper.pdf)

## N2 - canMint can be optimized for best-scenario

The `canMint` modifier on the `ERC721BaseCollectionV2` contracts validates if the `msg.sender` is either the creator of the collection or an authorized minting address for the given collection.

It performs this validation in the following order:

1) Check if `msg.sender` is the creator
2) Check if `msg.sender` is a global minter
3) Check if `msg.sender` is a `itemMinter`

Given that `issueToken` is most likely going to be called by a `CollectionStore` instance, and such instance will be defined as a minter, consider moving `msg.sender == creator` to the last checked condition.

## N3 - Keep unmodified values in memory during loops

The `ERC721BaseCollectionV2` contract defines `items.length` value, stored on contract storage, this value is accessed within loops.

The functions that contain this pattern are:

- `setItemsMinters`
- `setItemsManagers`
- `editItemsSalesData`
- `editItemsMetadata`
- `rescueItems`
- `issueItems`

The `CollectionStore` contract defines `fee`, this value is assessed inside a loop on the `buy()` function, but the value cannot be modified from within the loop itself.

Consider keeping these values in memory to avoid repeated storage accesses inside loops.

## N4 - Some require conditions are unreachable

Some `required` conditions are unreachable and thus are left untested; consider replacing them with `assert` to improve readability and static analysis[4].

- On the `ERC721BaseCollectionV2` contract, the `rarity > 0 && rarity <= MAX_ISSUED_ID` condition can never be triggered, given that `rarity` is statically defined by `getRarityValue`.

- On the `ERC721BaseCollectionV2` contract, the `newItemId < MAX_ITEM_ID` condition is virtually unreachable given that it would require the creation of `1099511627775` items.

[4] assert and require, Solidity 7.3 docs (https://solidity.readthedocs.io/en/v0.7.3/control-structures.html#id4)

### N5 - Mark fixed variables as immutable

The `CollectionStore` contract defines `acceptedToken`; this variable's value is defined during contract construction but cannot be updated after the deployment.

Consider marking the const as `immutable` to avoid storage accesses.

# Final thoughts

The contracts are well written, self-documented and have high test coverage, no high or medium severity issues have been found in the contract. The project is ready to be deployed in a production environment.

- October 2020 - Agustín Aguilar