Audit of Decentraland Marketplace V2

v1.0.0

September 28th, 2018

Introduction

The Decentraland team asked us to review their Marketplace V2 and related contracts. The commit used for the initial version of this audit is 26cf5d3.

We reviewed the code and laid out the results below. We have worked with the Decentraland team before, and once again, their code is well-written and easy to follow. They follow known best practices that make the code easier to read and audit. There were no critical issues found in the code. We recommended a few changes all of which are outlined below.

Some particularly strong patterns in the code include: (a) use of the SafeMath library; (b) simple, concise code; and (c) thorough use of sanity checks to protect against user error.

Note: *Additional programmatic tests are being written for the purposes of this audit. These tests may surface additional issues. When testing is complete, this report will be updated and the Decentraland team will be notified.*

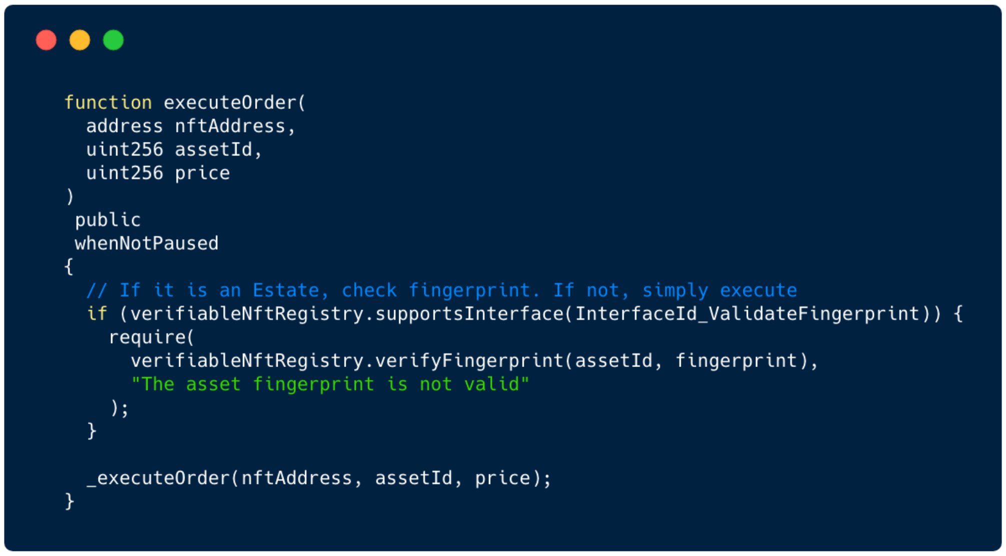
Issues

1. Require that each executeOrder transaction is checked for Estate or Land rather than assuming one or the other based on the called function

High

Marketplace.sol#L253 - A user with an Estate can put the Estate up for sale and subsequently remove all of the land in this estate prior to the execution of the sale. If a buyer then calls `executeOrder()` on this `assetId` (as opposed to `safeExecuteOrder()`), they will be able to purchase this empty Estate.

Consider having only one function that executes `_executeOrder()` so that users must check for a fingerprint if one exists. The code would look as follows:



```
function executeOrder(
  address nftAddress,
  uint256 assetId,
  uint256 price
)
 public
 whenNotPaused
{
  // If it is an Estate, check fingerprint. If not, simply execute
  if (verifiableNftRegistry.supportsInterface(InterfaceId_ValidateFingerprint)) {
    require(
      verifiableNftRegistry.verifyFingerprint(assetId, fingerprint),
      "The asset fingerprint is not valid"
    );
  }

  _executeOrder(nftAddress, assetId, price);
}
```

2. Collisions are possible with EstateRegistry's getFingerprint()

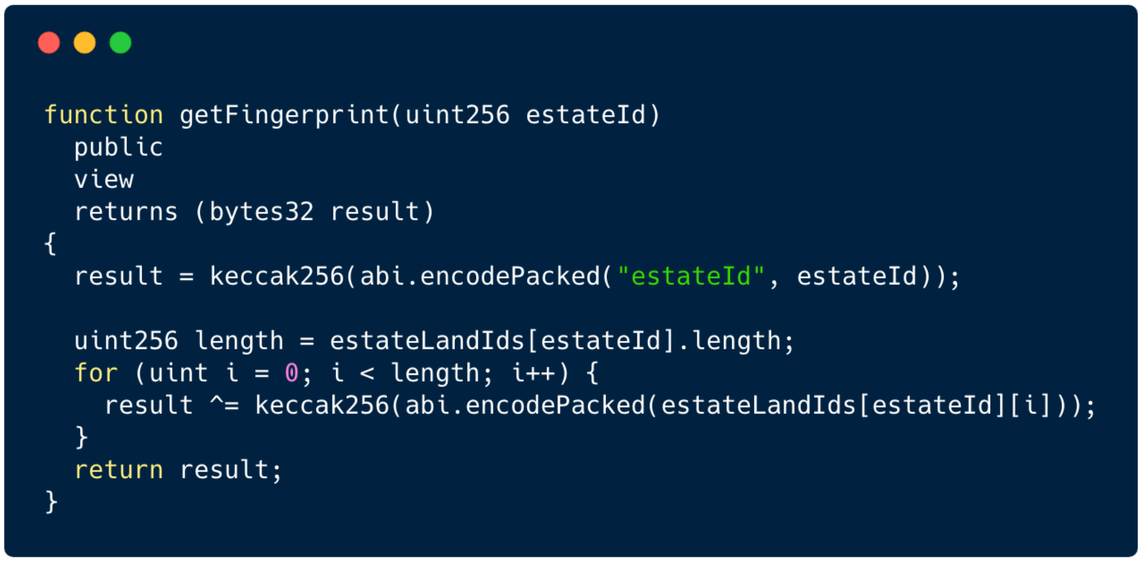
High

EstateRegistry.sol#L198 - It's possible to create colliding fingerprints with different estates and land compositions. This is due to the fact that `estateId`'s and `landId`'s can be identical. For example, the 5th estate would have an `estateId` of `5` and the land parcel at (0,5) also has a `landId` of `5`. If the 5th estate contained the parcel (0,5) the fingerprint would be `0`. This is possible for all combinations where the `estateId` is the same as the `landId` of its only parcel of land. Collisions are also possible by adding an extra parcel to the situation above (e.g.

the 5th estate containing (0,5) and (3,4) will have the same fingerprint as the 4th estate containing (0,4) and (3,4)). Additionally, collisions are possible in cases such as the 5th estate containing (0,4) and the 4th estate containing (0,5).

These collisions do not present any issues for the `Marketplace` contract as it is but may present an attack vector in future iterations of the `Marketplace` contract or in other contracts that assume estate fingerprints will be unique.

Consider adding the string “estateId” as a parameter to the the `abi.encodePacked()` function on this line. No collisions will be possible with the new `getFingerprint()` function. (We have not formally proven that no collisions are possible but have had the modification reviewed by an independent third party.)



```
function getFingerprint(uint256 estateId)
  public
  view
  returns (bytes32 result)
{
  result = keccak256(abi.encodePacked("estateId", estateId));

  uint256 length = estateLandIds[estateId].length;
  for (uint i = 0; i < length; i++) {
    result ^= keccak256(abi.encodePacked(estateLandIds[estateId][i]));
  }
  return result;
}
```

3. Checks-Effects-Interactions

Medium

Marketplace.sol#L304-L319 - In `_executeOrder()`, it doesn’t follow the checks-effects-interactions guideline. The “effect” here is deleting the order, so that should come before the “interaction” of transfering the token and NFT.

4. Require that the NFT being sold or bought is an approved token

Medium

Marketplace.sol#L253 - Consider creating a whitelist of approved NFTs that can be transacted in the marketplace.

Users can list any NFT for sale and users may be fooled into interacting with malicious NFT implementations. Adding a whitelist of approved NFTs will protect users from this set of attacks.

5. openzeppelin-zos doesn't follow semantic versioning (yet)

Medium

We recommend you pin `openzeppelin-zos` without a caret (^) in `package.json`. Until version 2.0.0 for `openzeppelin-zos`, Zeppelin explicitly notes in the README that semver isn't followed.

6. Require that nftAddress follows the ERC721 interface

Low

Marketplace.sol - In `createOrder()`, `cancelOrder()`, `safeExecuteOrder()`, and `executeOrder()`, consider requiring that `nftAddress` implements the correct ERC721 interface by ensuring that `ERC721Interface(nftAddress).supportsInterface(0x80ac58cd)` returns `true`.

Warning: *Before this is implemented the `LANDRegistry` contract should be updated to return `true` for the current ERC721 interface (`0x80ac58cd`).*

7. Some ERC20s do not revert when a transfer fails

Low

Marketplace.sol - Consider requiring that all ERC20 transfers return `true` in the following locations: #L175, #L297, #L305. This does not present an issue when Decentraland's MANA is the accepted token.

8. Test a realistic failing fingerprint

Low

The test "`should fail on execute a created order :: (wrong fingerprint)`" should use a more realistic fake fingerprint than `-1`. While a different fake fingerprint wouldn't change the results with the current implementation of fingerprinting, this would be a more reliable test to confirm that it's failing for the reason that the fingerprint didn't match perfectly and not some more trivial reason - this makes this test more robust for any future fingerprint implementations.

9. Test coverage

High

An exhaustive test suite is crucial for verifying the intended functionality of the marketplace v2. Here are suggested tests that will improve the coverage of important code paths, sorted by the attention level each test warrants:

Critical tests

a) Integration tests
   - i) When an order is created, it should transfer the accepted token to owner (in the correct amount based on publication fee, if that fee is greater than 0) and create an order
   - ii) When an order is executed correctly
     1) Land: When the owner cut is 0, it should transfer the accepted token to seller, the land to the buyer, and delete the order
     2) Estate: When the owner cut is 0, it should transfer the accepted token to seller, the estate to the buyer, and delete the order
   - iii) When an NFT is changed after the order is started
     1) An order for land should revert if the land has been transferred
     2) An order for an estate should revert if the estate has been transferred
     3) it should revert if an estate's number of land has been changed
     4) it should revert if a parcel of land in an estate has been switched with another parcel

b) Unit test: `createOrder`
   - i) it should create a new order with the correct parameters with nftAddress `isApprovedForAll` as the path for meeting the require
   - ii) it should transfer `publicationFeeInWei` to owner
   - iii) it should revert if the `msg.sende`r does not own the assetId and the asset is land

c) Unit test: `executeOrder`
   - i) it should call `_executeOrder`

d) Unit test: `_executeOrder`
   - i) it should revert if the seller is no longer the owner

Medium tests

e) Integration tests
   - i) When an NFT is changed after the order is started

                1) it should complete the order for land if the land's metadata has been changed
                2) it should complete the order for the estate if the estate’s metadata has been changed
        ii) it should revert if an order that has already been executed is cancelled
f) Unit test: `initialize`
    i) Should fail if initialized twice
    ii) Should initialize `msg.sender` as the owner
g) Unit test: `createOrder`
    i) it should revert if the `msg.sender` puts land for sale and owns the land but only via an estate, so they don't technically own the assetId
    ii) it should revert if contract is paused
    iii) it should revert if `msg.sender` does not have enough tokens to pay the `publicationFeeInWei`
h) Unit test: `cancelOrder`
    i) it should revert if contract is paused
i) Unit test: `safeExecuteOrder`
    i) It should revert if the contract is paused
j) Unit test: `executeOrder`
    i) It should revert if the contract is paused
k) Unit test: `_executeOrder`
    i) It should revert if no such asset exists with that assetId

Low tests

l) Integration tests
    i) Executing an order should revert if an order has never been created
m) Unit test: `initialize`
    i) It does set `acceptedToken`
    ii) It reverts if `_acceptedToken` is not a contract
n) Unit test: `setPublicationFee`
    i) It should emit a `ChangedPublicationFee` event
o) Unit test: `setOwnerCut`
    i) It should emit a `ChangedOwnerCut` event
p) Unit test: `createOrder`
    i) it should emit an `OrderCreated` event with the correct parameters
    ii) it should revert if the `priceInWei` is 0
    iii) it should revert if `expiresAt` is less than 1 minute in the future
    iv) it should revert if the NFT is not approved for transfer by the marketplace contract
    v) it should revert if publication fee transfer is not approved
q) Unit test: `cancelOrder`
    i) it should emit an `OrderCancelled` event with the correct parameters
    ii) it should revert if the order does not exist

r) Unit test: _executeOrder
    i) It should emit an `OrderSuccessful` event with the correct parameters

Notes

- `priceInWei` is used to describe the price in the smallest unit of the `acceptedToken`. Wei implies the smallest unit of ETH but there is also no general word to describe the smallest unit of an arbitrary token that we know of. Consider using another variable name like `priceInBaseUnit` to avoid confusion.
- Consider using an npm script to run the local truffle version, to minimize the chance of a developer contributing to this repo having upgraded to a newer major truffle version globally and running into issues.
- Consider renaming the variable `ownerCut` to `_ownerCutPercentage` in order to be fully clear that this number is the desired percentage. The same holds anywhere else `ownerCut` is used without "Percentage" as a suffix.
- It is possible for a seller to front-run `_executeOrder` to cancel an order or for anyone to front-run `_executeOrder` to get the order before the original buyer. Neither of these are profitable or successful griefing strategies. Details
- The dependency erc821 is not being used and can be removed.
- Marketplace.setOwnerCut uses uint8 but this does not save gas. The Solidity docs state "It is only beneficial to use reduced-size arguments if you are dealing with storage values because the compiler will pack multiple elements into one storage slot, and thus, combine multiple reads or writes into a single operation. When dealing with function arguments or memory values, there is no inherent benefit because the compiler does not pack these values." Consider using uint256 instead.
- Consider importing Zeppelin's ERC20 contract instead of using ERC20Interface.
- The time period "1 minutes" is a "magic constant". Consider defining a constant such as `MINIMUM_EXPIRATION_TIME`.
- **Style: Consistent naming for fake/test contracts used for testing**
    - In the `land` repo, `.sol` files used only for testing were suffixed with Test, whereas here they're prefixed with Fake. Also, in this repo they're in a subdirectory of the `contracts` directory, whereas in the `land` repo they're in the `test` directory. A convention one way or the other would be an improvement.
- **Typos and grammar**
    - Grammar: "between 0 to 100" should change to "between 0 and 100"
    - Typo: "NTF" -> "NFT". It would be best to do a find/replace in the whole repo for this.
    - **createOrder docstring should be about creating an order, not cancelling it**
      The docstring for `createOrder()` does not describe the right function. Consider updating it to reflect the actions of the function.

Conclusion

No critical issues were found. In general, the code is very well written and easy to follow. The Decentraland team did a great job of keeping the code simple and clear while performing necessary functionality and checks.