OpenZeppelin | security

The Sandbox Avatar Raffle Audit

June 12, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

Avatar Collection 5

Collection Factory and Collection Proxy 6

Privileged Roles 6

Trust assumptions 7

High Severity 8

H-01 Incorrect Calculation of Remaining Tokens at AvatarCollection Initialization 8

Medium Severity 9

M-01 personalize and burn Functions Can Be Called in Paused State 9

M-02 Users Can Bypass Minting Fee by Withholding Signatures 9

M-03 Mint Might Fail if the Total Supply of Tokens Is Not Increasing 10

Low Severity 11

L-01 Misleading Documentation 11

L-02 Naming Issue 11

L-03 Signature Can Be Replayed if a Mint Fails 11

Notes & Additional Information 12

N-01 Improve _getRandomToken Function's Documentation 12

N-02 Redundant Checks 13

N-03 Typographical Errors 13

N-04 Unused Named Return Variable 13

N-05 WaveSetup Event Should Emit Details About the Minting Phase 14

Conclusions 15

Appendix 16

Monitoring Recommendations 16

Summary

Type	DeFi
Timeline	From 2023-05-16 To 2023-05-24
Languages	Solidity

Total Issues	12 (8 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	3 (2 resolved)
Low Severity Issues	3 (2 resolved)
Notes & Additional Information	5 (3 resolved)

Scope

We audited the thesandboxgame/sandbox-smart-contracts repository at the 4177a31fd6c103d72a6e9744c99b2f9eadabdfb0 commit.

In scope were the following contracts:

```
solc_0.8.15
├── common
│   ├── IERC4906.sol
│   ├── IERC5313.sol
├── avatar
│   ├── AvatarCollection.sol
│   ├── ERC721BurnMemoryEnumerableUpgradeable.sol
│   ├── CollectionAccessControl.sol
├── proxy
│   ├── CollectionFactory.sol
│   ├── CollectionProxy.sol
```

System Overview

The set of contracts in scope can be described in two main groups based on their primary usage:

Avatar Collection

The `AvatarCollection` contract, based on OpenZeppelin's `ERC721EnumerableUpgradeable` contract, implements functionality to allow the minting and customization of Avatars in The Sandbox's metaverse.

This contract has support for Role Based Access Control via the `CollectionAccessControl` contract. Privileged roles along with their capabilities will be discussed in the next section. The contract also relies on `ERC721BurnMemoryEnumerableUpgradeable` contract to perform a "snapshot" of users that burned tokens. This snapshot is saved on-chain, reducing delay and improving user experience and game mechanics.

The contract relies on a signature validation scheme to allow the execution of minting, reveal or token customization functions. To allow users to perform these actions, The Sandbox team has to generate and sign a message with a trusted address ahead of time, namely `signAddress` address; this process is executed off-chain and the signed message is shared with the user. Later on, when the user performs the desired action, the signature is checked on-chain via `_checkSignature` and `_checkPersonalizationSignature` functions.

The contract supports different minting phases or waves: "Marketing", "Allowlist", and "Public", which define minting limits and price.

Minting is only allowed via a specific smart contract which is set as the `allowedToExecuteMint` address. It is expected that this contract will be The Sandbox's SAND contract.

Fees paid by users when minting are transferred to the address set as the `mintTreasury` variable. It is assumed that this address will act in the protocol's best interests.

Last but not least, users should be aware that the contract implements support for OpenSea's Operator Filter Registry, which can be used to forbid certain addresses from interacting with this contract.

Collection Factory and Collection Proxy

The `CollectionFactory` contract along with the `CollectionProxy` contract allows the deployment of new collections or upgrading existing ones by using a modified version of the Beacon proxy pattern. Beacons are based on OpenZeppelin's `UpgradeableBeacon` contract. The `CollectionProxy` contract is based on OpenZeppelin's `BeaconProxy` contract but contains added functionality to support changing the beacon used.

The `CollectionFactory` contract keeps track of all the beacons and proxies deployed or added to it and allows the contract owner to update their configuration.

Privileged Roles

The `AvatarCollection` contract contains the following privileged roles:

- The `owner` of the contract can perform the following actions:
  - Pause or resume the contract's operations.
  - Change the treasury address.
  - Change the signing address.
  - Change the contract allowed to call `mint` function.
  - Change the address used as registry by the Operator Filterer contract.
  - Possess the `ADMIN_ROLE`.
  - Configure and revoke `CONFIGURATOR_ROLE` and `TRANSFORMER_ROLE` roles.

In addition to the capabilities mentioned above, the owner can also perform the actions of the `CONFIGURATOR_ROLE` and `TRANSFORMER_ROLE` roles. Ownership can be transferred using a two-step transfer mechanism. This role cannot be renounced.

- The `CONFIGURATOR_ROLE` has the following capabilities:
  - Configure minting waves via the `setupWave` function.
  - Set up minting phases via the `setMarketingMint`, `setAllowlistMint` and `setPublicMint` functions.

    - Configure the contract's base URI by calling `setBaseURI` function.
- The `TRANSFORMER_ROLE` has the following capabilities:
    - Personalize tokens via the `operatorPersonalize` function.

The `CollectionFactory` contract has the following privileged roles:

- The `owner` of the contract can perform the following actions:
    - Deploy new beacons or add existing ones.
    - Change the implementation pointed to by a beacon.
    - Transfer ownership of an owned beacon.
    - Deploy new proxies (collections) or add existing ones.
    - Change the beacon used by a collection.
    - Transfer ownership of an owned collection.

Ownership can be transferred using a two-step transfer mechanism. This role cannot be renounced.

The `CollectionProxy` contract has the following privileged roles:

- The `admin` of the contract can perform the following actions:
    - Change the beacon used by the proxy.
    - Set a new admin.

Trust assumptions

We assume that the Sandbox's backend functionalities (such as signing transactions, updating the avatar during personalization, and others) work as intended and that the transaction-signing addresses are trusted entities.

High Severity

H-01 Incorrect Calculation of Remaining Tokens at `AvatarCollection` Initialization

During the contract's initialization, the `__AvatarCollection_init` function of the `AvatarCollection` contract incorrectly calculates the number of `remaining` tokens by subtracting the total supply of tokens from the state variable `maxSupply`. Given that the contract has not been initialized and no tokens are minted, the value of `totalSupply()` is zero. Moreover, the `maxSupply` variable is also not initialized until this point in the codebase, hence its value is also zero. This makes the value of `remaining` zero.

The contract further checks that the maximum number of tokens minted per wallet for the `Public` and `Allowlist` minting phases and the number of tokens minted during the `Marketing` phase should be less than or equal to the `remaining` amount.

Given that the value of `remaining` is zero, the only way these checks can pass is if the value of `_mintingDefaults.maxPublicTokensPerWallet` is zero, the value of `_mintingDefaults.maxAllowlistTokensPerWallet` is zero and the value of `_mintingDefaults.maxMarketingTokens` is zero.

If these values of the `mintingDefaults` parameters are zero, then the `setMarketingMint`, `setAllowlistMint` and `setPublicMint` functions would set the value of the `waveMaxTokensPerWallet` variable to zero and no minting could take place in either of these waves. The Sandbox team expects to use these setters to cover 99% of the minting requirements.

Alternatively, if `_mintingDefaults` parameters are non-zero values, the contract cannot be initialized.

Consider replacing the `maxSupply` state variable with the `_maxSupply` input variable, which should have a non-zero value.

Additionally, consider removing any unnecessary lines of code from the codebase. Given that at initialization the value of `remaining` would always be equal to `_maxSupply`, consider using `_maxSupply` within the `__AvatarCollection_init` function instead of the `remaining` variable.

Update: *Resolved in pull request #979 at commit acd1cd1.*

Medium Severity

M-01 `personalize` and `burn` Functions Can Be Called in Paused State

The `personalize` function defined in the `AvatarCollection` contract allows the owner of a token to change the token's traits. The `burn` function that the `AvatarCollection` contract inherits from the `ERC721BurnMemoryEnumerableUpgradeable` contract allows users to burn their tokens.

Both of these functions are missing the `whenNotPaused` modifier, which means that they can be called when the system is on pause.

To avoid making unnecessary changes to the state of the tokens when the system is paused, consider adding the `whenNotPaused` modifier to the `personalize` and `burn` functions.

Update: *Resolved in pull request #980 at commit 4106d14. Along with the suggested fix, the Sandbox team has decided to remove the `whenNotPaused` modifier from the following privileged functions: `setupWave`, `setMarketingMint`, `setAllowlistMint` and `setPublicMint`.*

M-02 Users Can Bypass Minting Fee by Withholding Signatures

For the Avatar Collection minting, the `signAddress` address provides a signature for each mint request in the backend. This `signature` is validated and recorded in the `_signatureIds` mapping to ensure that the signature is not reused.

Consider a scenario where a user of The Sandbox game is allowed to mint tokens during the `Allowlist` and/or `Public` minting waves via the normal signature scheme.

The user could decide to delay sending the transaction containing the message signed, having an unused valid signature. Then call the `approveAndCall` function from the `PolygonSand` contract, which in turn calls the `mint` function in the `AvatarCollection` contract during the `Marketing` minting phase (where the minting fee is zero). Since all the input parameters from the previously saved `mint` transactions are still valid, the contract mints the token for the user's address at no fee.

Consider adding minting phase information as an input to the `mint` function.

Update: *Acknowledged, not resolved. The Sandbox team stated:*

> *Acknowledged, we decided against fixing it at this stage since it is not a scenario that we envision happening in the future (i.e., going from the marketing mint phase (price 0) to allowlist/public minting (100 SAND mint), then back to marketing phase, and somebody premeditating this). We accept the risk.*

M-03 Mint Might Fail if the Total Supply of Tokens Is Not Increasing

The `_getRandomToken` function in the `AvatarCollection` contract is used to generate a unique token ID for an Avatar token. It takes the total number of tokens minted as an input parameter, and uses the number of tokens left to be minted to calculate the random token ID. This logic correctly assumes the total number of minted tokens would always be incrementing.

This function is called when a token is minted and the total supply of tokens is passed as the input parameter. Since users are allowed to burn tokens, the total supply of tokens can reduce if more tokens are burned than minted; or stay the same, if the tokens are minted and burned in the same proportion.

If the total supply decreases or stays the same when the `_getRandomToken` function is called, the algorithm might generate a previously used token ID. This may cause the function to regenerate a previously minted token. If this previously minted token has been burned, the same token is regenerated. However, if this previously generated token exists, the mint would fail.

Consider changing the logic of the random token ID generation for non-incrementing token supplies.

Update: *Resolved in pull request #972 at commit 7f7a83c. The Sandbox team has decided to disable the default burn functionality to prevent the total supply from decreasing. Burning of tokens is allowed only when the owner enables the burn functionality.*

Low Severity

L-01 Misleading Documentation

In the `CollectionFactory` contract, the docstring above the `transferBeacon` function states that the function "Transfers a beacon from the factory and all linked collections". However, this function only transfers the ownership of the beacons without performing any changes to the collections.

To improve clarity, consider editing the documentation to better reflect the function's purpose.

Update: *Resolved in pull request #981 at commit 429f5f3.*

L-02 Naming Issue

In the `AvatarCollection` contract, the `_getRandomToken` function takes the `_totalMinted` variable as an input, which represents the `totalSupply` of the tokens. This could be confusing since the total supply of the tokens can increase and decrease by burning and minting. However, the total tokens minted should be a non-decreasing value.

To favor explicitness and readability, consider renaming the `_totalMinted` variable.

Update: *Resolved in pull request #982 at commit ad8d1b4.*

L-03 Signature Can Be Replayed if a Mint Fails

The AvatarCollection contract keeps track of signatures that are generated by the backend system and have been used to mint avatars on-chain, thus preventing the signature replay attack.

However, there is no process to track these signatures when a mint transaction fails. With every failed mint transaction, an unused signature is publicly available to be replayed. Since, the signature validates the recipient address to which the token should have been minted, the replaying of the signature cannot be used to steal the token. However, it can be exploited to mint tokens to the recipient in undesirable circumstances.

For instance, consider a scenario where Alice has given infinite SAND allowance to the Avatar contract. She sends a mint transaction with a valid signature on-chain and the mint fails. An

attacker withholds Alice's valid signature. The attacker can then misuse the signature in the following ways:

- Keep signatures to mint tokens for Alice in a wave that is much more expensive than the intended one.
- Make Alice mint more tokens than she might have wanted. Imagine Alice only wanted to mint one token at the start - her first transaction fails and she sends another transaction to mint her token. The unused signature from the failed transaction can be replayed by an attacker, forcing Alice to mint two tokens.
- An attacker can keep the signature to make batch transactions that happen later fail by hitting the mint limit. Imagine Alice has a limit of 10 mint tokens, and she is sending a transaction that mints 10 new tokens. The attacker can potentially make this transaction fail if they already stored a valid signature from Alice. This can cause 10 new signatures to become public for a later attack.

To prevent the signature replay attack, consider adding the nonce and wave number to the signature generation scheme.

Update: *Acknowledged, not resolved. The Sandbox team stated:*

> *We acknowledge it but will not implement a fix since it's an edge case for which the extra required backend overhead is not worth it, for now.*

Notes & Additional Information

N-01 Improve `_getRandomToken` Function's Documentation

The `_getRandomToken` function in the `AvatarCollection` contract produces a pseudo-random number that can be predicted by dedicated users. Even though in the context of this system this does not pose a threat (as The Sandbox team stated that there are off-chain processes that perform shuffling of the IDs), the documentation should clarify that this function should not be trusted as a source of randomness.

Consider clarifying in the function's documentation, explaining that the function does not provide true randomness.

Update: *Resolved in pull request #983 at commit 401182f.*

N-02 Redundant Checks

Through the codebase, the following redundant checks were identified:

- In the `transferBeacon` function of `CollectionFactory` contract, there is a `require` statement validating that the `newBeaconOwner` address is not zero. However, this check is also performed in the `transferOwnership` function defined in the `Ownable` contract.
- In the `transferCollections` function of the `CollectionFactory` contract, there is a `require` statement validating that the `newCollectionOwner` address is not zero. However, this check is also performed in the `_setAdmin` function defined in the `ERC1967Upgrade` contract.

To improve clarity and save gas, consider removing redundant checks.

Update: *Resolved in pull request #984 at commit 34d0e74.*

N-03 Typographical Errors

Consider addressing the following typographical errors.

- On lines 83 and 173 of `AvatarCollection.sol` "*tot*" should be "*to*".
- On line 31 of `CollectionProxy.sol`, "*wile*" should be "*while*".
- On lines 45 and 61 of `CollectionProxy.sol`, "reroute *to* here" should be "*reroute here*".

Update: *Resolved in pull request #985 at commit dd34ad4.*

N-04 Unused Named Return Variable

Named return variables are a way to declare variables that are meant to be used within a function's body for the purpose of being returned as the function's output. They are an alternative to explicit in-line return statements.

In the `AvatarCollection` contract, the `sender` return variable for the `_msgSender` function is unused.

Consider removing this unused named return variable.

Update: *Acknowledged, not resolved. The Sandbox team stated:*

> *The return named variable is used. If by "unused" the auditors are refering to being used only as a return mechanism, then we acknowledge it and will leave it as is.*

N-05 `WaveSetup` Event Should Emit Details About the Minting Phase

In the `AvatarCollection` contract, the `WaveSetup` event is emitted when a minting wave is set up (i.e., from the `setupWave`, `setMarketingMint`, `setAllowlistMint`, and `setPublicMint` functions). While this event emits information about the number and price changes of the tokens per minting wave, there is no way to filter these emitted events by minting phase.

Including the minting phase in the event can be useful to find the token's price for the public minting phase.

Depending on the business requirement, consider adding the minting phase to the `WaveSetup` event.

Update: *Acknowledged, not resolved. The Sandbox team stated:*

> *Acknowledged, but we will not implement this since we can determine the phases off-chain if needed and adding this information would require changes in our backend which we will not do solely for this.*

Conclusions

One high-severity, two medium-severity, and a few low-severity issues and notes were reported, mainly addressing improvement opportunities to the overall quality of the codebase.

Appendix

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, The Sandbox team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues affecting production environments. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring.

Governance

Critical: The `CollectionFactory` contract plays a central role in the architecture of the system. The contract's owner can deploy new beacons and proxies or modify existing ones, affecting how deployed collections behave. Consider monitoring any changes in the contract's ownership via `OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner)` and `OwnershipTransferred(oldOwner, newOwner)` events.

Critical: `CollectionAccessControl` contract manages what activities can be performed by privileged addresses over avatar collections. This contract implements ownership and role-based access controls. Consider monitoring the following events: `OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner)`, `RoleAdminChanged(bytes32 role, bytes32 previousAdminRole, bytes32 newAdminRole)`, `RoleGranted(bytes32 role, address account, address sender)`, `RoleRevoked(bytes32 role, address account, address sender)`

Technical

Critical: Beacons interact with proxies by providing them with the address of the corresponding implementation contract. All changes affecting the list of available beacons or their configuration should be reviewed. Consider monitoring the following events: `BeaconAdded(bytes32 indexed beaconAlias, address indexed`

`beaconAddress)`, `BeaconUpdated(address indexed oldImplementation, address indexed newImplementation, bytes32 indexed beaconAlias, address beaconAddress)`, `BeaconRemoved(bytes32 indexed beaconAlias, address indexed beaconAddress, address indexed newBeaconOwner)`, `BeaconOwnershipChanged(address indexed oldOwner, address indexed newOwner)`, `Upgraded(address indexed implementation)`.

Critical: Proxies act as the entry point for any deployed collection. In the same way as beacons, they are managed by the `CollectionFactory` contract. Any change to proxies should be monitored. Consider monitoring the following events: `CollectionAdded(address indexed beaconAddress, address indexed collectionProxy)`, `CollectionUpdated(address indexed proxyAddress, bytes32 indexed beaconAlias, address indexed beaconAddress)`, `CollectionRemoved(address indexed beaconAddress, address indexed collectionProxy)`, `CollectionProxyAdminChanged(address indexed oldAdmin, address indexed newAdmin)` and `AdminChanged(address previousAdmin, address newAdmin)`.

Critical: The `AvatarCollection` contract uses `signAddress` address to check the validity of signed messages. Any changes to this address should be monitored. Consider monitoring the `SignAddressSet(address indexed _signAddress)` event.

Critical: The `AvatarCollection` contract uses `mintTreasury` address to determine the address where user payments are transferred during minting. Consider monitoring the `TreasurySet(address indexed _owner)` event. Unexpected changes to this address could signal attack attempts.

Medium: The `AvatarCollection` contract implements an emergency pause mechanism. Consider monitoring for the `Paused(address)` and `Unpaused(address)` events.

Suspicious activity

High: Actions such as minting new tokens or token customization via the `personalize` function are previously approved by The Sandbox team by crafting a specific signed message. Under normal circumstances, these transactions are likely going to be successfully executed. Consider monitoring unusually large numbers of transactions executing `mint` or `personalize` functions that revert with the message `"AvatarCollection: signature failed"` as this could signal attempts to perform signature replay attacks.

Medium: The `AvatarCollection` contract whitelists the address allowed to call the `mint` function via `allowedToExecuteMint` parameter. Consider monitoring changes to this address via `AllowedExecuteMintSet(address indexed _address)` event.

Medium: As token customization via `personalize` function has to be previously whitelisted by The Sandbox team in their back end, it is possible to roughly estimate when and how this function will be used. Consider monitoring unexpected volumes of `Personalized(uint256 indexed _tokenId, uint256 indexed _personalizationMask)` events.