OpenZeppelin | security

Asset Audit

THE SANDBOX

November 27, 2023

Table of Contents

Summary

Type	NFT
Timelines	From 2023-08-22 To 2023-09-01 From 2023-11-20 To 2023-11-22
Languages	Solidity

Total Issues	13 (11 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	3 (3 resolved)
Notes & Additional Information	9 (7 resolved)

Scope

We audited the `thesandboxgame/sandbox-smart-contracts` repository at the 1e04f35 commit.

In scope were the following contracts:

```
packages
└─ asset/
   └─ contracts/
      ├─ Asset.sol
      ├─ AssetCreate.sol
      ├─ AuthSuperValidator.sol
      ├─ interfaces/
      │  ├─ IAsset.sol
      │  ├─ IAssetCreate.sol
      │  ├─ IAssetReveal.sol
      │  └─ ITokenUtils.sol
      └─ libraries/
         └─ TokenIdUtils.sol
```

System Overview

The `Asset` contract implements functionality to allow the minting and burning of Assets (ERC-1155 compliant tokens) in The Sandbox's metaverse.

The users can interact with the `AssetCreate` contract to create new assets if they have a valid signature from The Sandbox and sufficient Catalyst tokens of the specific tier. This contract is granted the `MINTER_ROLE` role for the `Asset` contract and `BURNER_ROLE` role for the `Catalyst` contract.

The creation of assets is done by validating the signature, generating a unique token ID, burning the tier-specific catalysts and then minting the asset. Assets can be minted individually or as a batch. This functionality is pausable.

The `AssetCreate` contract also allows the `SPECIAL_MINTER_ROLE` privileged role to mint special assets, represented by tier-0.

Within the `Asset` contract, the minting of assets can only be done by the `MINTER_ROLE` role (which is the `AssetCreate` contract). A `RoyaltySplitter` contract is deployed for each of the minted tokens to ensure that the `creator` is paid the royalty for their creation. This contract also inherits from the `OperatorFiltererUpgradeable` to filter out the operators who do not pay royalties to the creators.

Additionally, the `Asset` contract allows the burning of asset tokens either by the users or by the `BURNER_ROLE` privileged role.

The `AuthSuperValidator` contract is used for the verification of signatures and the `TokenIdUtils` library is used for the generation of asset token IDs.

Privileged Roles and Trust Assumptions

The audited contracts contain the following privileged roles:

- The `admin` of the `Asset` contract can perform the following actions:
    - Alter the base URI of the contract.
    - Change the address of the `trustedForwarder`.
    - Change the royalty address of a token.
    - Change the `Operator Filterer Registry` address.
    - Add the `Asset` contract to the `Operator Filterer Registry`.
- The `MINTER_ROLE` defined in the `Asset` contract can mint individual or batches of ERC-1155 asset tokens.
- The `BURNER_ROLE` defined in the `Asset` contract can burn individual or batches of ERC-1155 asset tokens from a specific address.
- The `MODERATOR_ROLE` defined in the `Asset` contract can assign a new URI for a valid `tokenId`.
- The `admin` of the `AssetCreate` contract can change the address of the `trustedForwarder`.
- The `PAUSER_ROLE` defined in the `AssetCreate` contract can halt the minting of new assets.
- The `SPECIAL_MINTER_ROLE` defined in the `AssetCreate` contract has permission to create special assets like TSB-exclusive tokens.
- The `admin` of the `AuthSuperValidator` contract can set signers for the contracts.

During the audit, it is assumed that the privileged addresses are trusted entities and would work in the best interest of the platform and its users.

Additionally, it is assumed that only the `AssetCreate` contract will be assigned the `MINTER_ROLE` in the `Asset` contract.

It is also assumed that the `BURNER_ROLE` in the Catalyst contract would not prevent the asset-minting transactions by front-running the `createAsset` and `createMultipleAssets` functions and burning the catalyst tokens for the creator.

Medium Severity

M-01 `DEFAULT_ADMIN_ROLE` Can Renounce Role

The `DEFAULT_ADMIN_ROLE` role for the `AuthSuperValidator` contract can be renounced by calling the `renounceRole` function inherited from the OpenZeppelin contract library's `AccessControl` contract.

The `AuthSuperValidator` contract is not upgradable. Once the `DEFAULT_ADMIN_ROLE` role is renounced, it cannot be recovered, and would result in the inability to set or change signers for the `AssetCreate` and other contracts that implement similar signature verification.

Consider overriding the `renounceRole` function to ensure that the `DEFAULT_ADMIN_ROLE` role is not lost.

Update: *Resolved in pull request #1135. The Sandbox team stated:*

> *Added condition to not renounce the `DEFAULT_ADMIN_ROLE`.*

Low Severity

L-01 Incorrect Documentation

Several docstrings and inline comments throughout the codebase were found to be erroneous and should be fixed. In particular:

- The docstring above the `generateTokenId` function in the `TokenIdUtils` contract incorrectly states that `chain index` is one of the fields used for generating the token ID.
- The docstring above the `setOperatorRegistry` function in the `Asset` contract incorrectly states that the function "sets filter registry address deployed in test". The function will also be used on the mainnets.

Update: *Resolved in pull request #1136. The Sandbox team stated:*

> *Documentation has been updated.*

L-02 Lack of `gap` Variables

Throughout the codebase, there are multiple upgradeable contracts that do not have a gap variable. For instance:

- The `Asset` contract
- The `AssetCreate` contract

Consider adding a gap variable to avoid future storage clashes in upgradeable contracts.

Update: *Resolved in pull request #1137. The Sandbox team stated:*

> *Added `gap` variables.*

L-03 Missing Docstrings

Within the codebase there are several parts that do not have docstrings.

For instance:

- Line 33 in `Asset.sol`
- Line 65 in `Asset.sol`
- Line 171 in `Asset.sol`
- Line 4 in `IAsset.sol`
- Line 26 in `IAsset.sol`
- Line 33 in `IAsset.sol`
- Line 39 in `IAsset.sol`
- Line 45 in `IAsset.sol`
- Line 47 in `IAsset.sol`
- Line 4 in `IAssetCreate.sol`
- Line 4 in `IAssetReveal.sol`
- Line 6 in `ITokenUtils.sol`
- Line 7 in `ITokenUtils.sol`
- Line 9 in `ITokenUtils.sol`
- Line 11 in `ITokenUtils.sol`
- Line 13 in `ITokenUtils.sol`
- Line 15 in `ITokenUtils.sol`
- Line 6 in `TokenIdUtils.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #1138. The Sandbox team stated:*

> *Function documentation has been updated.*

Notes & Additional Information

N-01 Incomplete Docstrings

To improve the readability of the code, consider updating the following incomplete docstrings:

- The docstring above the `getData` function in the `TokenIdUtils` library does not document the return value.
- The docstring above the `mint` function in the `Asset` contract does not document all the input parameters.
- The docstring above the `mintBatch` function in the `Asset` contract does not document all the input parameters.
- The docstring above the `initialize` function in the `AssetCreate` contract does not document all the input parameters.
- The docstring above the `createAsset` function in the `AssetCreate` contract does not document all the input parameters.
- The docstring above the `createMultipleAssets` function in the `AssetCreate` contract does not document all the input parameters.
- The docstring above the `_hashMint` function in the `AssetCreate` contract does not document all the input parameters.
- The docstring above the `_hashBatchMint` function in the `AssetCreate` contract does not document all the input parameters.

Update: *Resolved in pull request #1139. The Sandbox team stated:*

> *Documentation has been updated.*

N-02 Inconsistent Error Message

Within the codebase, there are instances where the error messages are inconsistent. For example, in the `Asset` contract, the error message states the contract name followed by the error statement, whereas there is no mention of the contract's name in the error messages of the `AssetCreate` contract.

Consider adding contract names to all the error messages to know where the error occurred.

Update: *Resolved in pull request #1140. The Sandbox team stated:*

> *Added consistent error messages.*

N-03 Lack of Indexed Event Parameter

Within `IAssetReveal.sol`, the `AssetRevealBurn`, `AssetRevealBatchBurn`, `AssetRevealMint` and `AssetRevealBatchMint` events do not have indexed parameters.

Consider indexing event parameters to improve the ability of off-chain services to search and filter for specific events.

Update: *Resolved in pull request #1141. The Sandbox team stated:*

> *Added indexed parameters on suggested events .*

N-04 Typographical Errors

To improve code readability, consider removing the following typographical errors from the codebase:

- On line 242 and line 270 of `Asset.sol`, "aditional" should be "additional".
- On line 288 of `Asset.sol`, "creactor" should be "creator".
- On line 339 of `Asset.sol`, "Opensea.can" should be "Opensea. Can".

Update: *Resolved in pull request #1142. The Sandbox team stated:*

> *Fixed the suggested typographical errors.*

N-05 Unused Named Return Variables

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.

Throughout the codebase, there are multiple instances of unused named return variables. For instance:

- The `sender` return variable in the `_msgSender` function in `Asset.sol`.
- The `creator` return variable in the `getCreatorAddress` function in `Asset.sol`.
- The `tier` return variable in the `getTier` function in `Asset.sol`.
- The `sender` return variable in the `_msgSender` function in `AssetCreate.sol`.

Consider either using or removing any unused named return variables.

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

> *We have added named exports but decided to keep the explicit returns. We find that named exports enhance the clarity of what the function returns, so we have chosen to integrate them alongside explicit returns.*

N-06 Gas Optimization

In the `AssetCreate` contract, the `createMultipleSpecialAssets` function accepts the `amounts` and `metadataHashes` parameters and stores them in `memory`. Given that these arrays are user-defined inputs, their length can be large. `calldata` is useful for passing large amounts of data to a function without having to copy said data into memory, which can otherwise prove to be expensive in terms of gas usage.

Consider using `calldata` instead of `memory`. This will avoid the overhead of copying data into memory, and help reduce the amount of gas needed to execute the function.

Update: *Resolved in pull request #1324.*

N-07 Inconsistent Variable Naming

In the `Asset` contract, the names of the return variables for `name` and `symbol` functions start with `_`. This is inconsistent with how other return variables are named in the contract.

Consider renaming the `_name` and `_symbol` variables, adopting a consistent naming convention. This will help improve the consistency and readability of the code.

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

> *Acknowledged but we have decided not to make any changes in this regard.*

N-08 Initialize Variables in the Same Order in Which They Are Received as Arguments

In the `initialize` function of the `Asset` contract, the `baseUri` variable is initialized before the `forwarder` and `assetAdmin` variables.

Consider initializing the variables in the same order in which they are received as arguments. This will help improve code readability.

Update: *Resolved in pull request #1327. The Sandbox team stated:*

> *The order of initialized functions was changed.*

N-09 Replace Zero-Address Checks With Code Existence Checks

Several instances were identified where it would be better to replace the zero-address checks with code existence checks:

- At line 206 of `Asset.sol`
- At line 362 of `Asset.sol`
- At line 369 of `Asset.sol`
- At line 331 of `AssetCreate.sol`
- At line 362 of `Asset.sol`

Consider verifying whether a smart contract exists at a given address instead of performing a zero-address check. The `isContract` function from the OpenZeppelin contracts library's `Address` contract can be used for this purpose.

Update: *Resolved in pull request #1328.*

Recommendations

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 `AssetCreate` and `Asset` contracts manage what activities can be performed by privileged addresses over asset tokens as well as the burning of catalyst tokens. This contract implements ownership and role-based access controls. Consider monitoring the following events: `RoleAdminChanged(bytes32 role, bytes32 previousAdminRole, bytes32 newAdminRole)`, `RoleGranted(bytes32 role, address account, address sender)`, and `RoleRevoked(bytes32 role, address account, address sender)`.

Critical: The admin for the `AuthSuperValidator` contract manages the list of addresses that can be signers for a contract. Consider monitoring the following events: `RoleAdminChanged(bytes32 role, bytes32 previousAdminRole, bytes32 newAdminRole)`, `RoleGranted(bytes32 role, address account, address sender)`, and `RoleRevoked(bytes32 role, address account, address sender)`.

Technical

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

Suspicious activity

High: Actions such as minting new assets 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 an unusually large number of transactions executing the `createAsset` or `createMultipleAssets` functions which revert with the message `"Invalid signature"`, as this could signal attempts to perform signature replay attacks.

Conclusion

The audit was conducted over the course of two weeks. The codebase of the Asset contracts is well-written, making it easy to understand its functionalities and potential vulnerabilities. The Sandbox team was very responsive during the audit, ensuring doubts were addressed in a timely manner. One medium and several low-severity issues were reported, alongside some notes addressing improvement opportunities to the overall quality of the codebase.