OpenZeppelin | security

Royalty Operator Filter Audit

September 13, 2023

Table of Contents

Summary

Type	NFT
Timeline	From 2023-08-04 To 2023-08-18
Languages	Solidity

Total Issues	30 (28 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	2 (2 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	10 (10 resolved)
Notes & Additional Information	17 (15 resolved, 1 partially resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the thesandboxgame/sandbox-smart-contracts repository at the 2c5c66d62505391342295d2a407567b15fbe40b6 commit.

In scope were the following contracts:

```
packages/
└─ dependency-royalty-management/
|  └─ contracts/
|     ├─ MultiRoyaltyDistributer.sol
|     ├─ RoyaltyDistributer.sol
|     ├─ RoyaltyManager.sol
|     ├─ RoyaltySplitter.sol
|     └─ interfaces/
|        ├─ IERC20Approve.sol
|        ├─ IMultiRoyaltyDistributer.sol
|        └─ IRoyaltyManager.sol
|
└─ dependency-operator-filter/
   └─ contracts/
      ├─ OperatorFilterSubscription.sol
      ├─ OperatorFiltererUpgradeable.sol
      └─ interfaces/
         └─ IOperatorFilterRegistry.sol
```

System Overview

This system introduces Catalyst which is an ERC-1155 token. It adopts royalty distribution to users based on the ERC-2981 standard and complies with OpenSea's operator filter registry across all chains to stay updated with prevailing standards. Catalysts have specific tiers that allow the minter role to distribute diverse tokens to users. Administrators have the flexibility to augment these tiers, increasing the range of token-minting possibilities in the future.

Royalty

To implement the ERC-2981 standard for the payment of royalties, this system uses the `MultiRoyaltyDistributor` contract to manage royalties for multiple recipients. The `MultiRoyaltyDistributor` contract calls the `RoyaltyManager` contract to deploy a `RoyaltySplitter` for each creator, which splits the royalty between the creator and The Sandbox platform.

Royalties can be earned in native crypto (such as ETH and MATIC tokens) or in ERC-20 tokens. If the royalties are in native crypto, they are automatically split by the `RoyaltySplitter` contract. For ERC-20 tokens, the split is performed when either the creator or The Sandbox's common recipient address calls the `splitERC20Tokens` function. The split share is defined by the `commonSplit` variable.

Operator Filterer

The `OperatorFilterRegistry` establishes access control for operators. It grants them permission for specific administrative actions. Data for these operators is affected by the OpenSea registry, and the contract's blacklist is synchronized with OpenSea's. Moreover, this contract incorporates modifiers to manage access for verified operators.

Privileged Roles and Trust Assumptions

The audited contracts contain the following privileged roles:

- The `admin` of the `RoyaltyManager` contract can perform the following actions:
  - Change the address of the common recipient
  - Change the value of the common split
  - Change the address of the `trustedForwarder`
- The `CONTRACT_ROYALTY_SETTER_ROLE` role defined in the `RoyaltyManager` contract can set the royalty split for the given `contractAddress`.
- The `owner` role of `RoyaltySplitter` contract can set the `creator` recipient address.

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 the common recipient address which receives The Sandbox's share of royalties is not compromised.

High Severity

H-01 Creator Can Drain `RoyaltySplitter` Contract of All ERC-20 Tokens

The `RoyaltySplitter` contract is designed to split all ETH and ERC-20 tokens sent to the contract as royalties between the creator and the royalty manager. The proportion sent to the creator is determined by the `getCreatorSplit` function in the `RoyaltyManager` contract. When the `RoyaltySplitter` contract holds an ERC-20 token, anyone can call the `splitERC20Tokens` function to send the appropriate proportions of the token to the creator and royalty manager.

The `RoyaltySplitter` contract includes a `proxyCall` function that will make an external call to an arbitrary address with arbitrary calldata. This function will first attempt to call `splitERC20Tokens` on the input `target` address to ensure that tokens are properly split prior to the arbitrary external call. The call to `splitERC20Tokens` is made as an external call from the contract and will always fail due to the check that the caller of `splitERC20Tokens` is either the creator or the royalty manager as the contract itself is not permitted to call this function. This would allow the creator to successfully call this function passing in an ERC-20 token for the `target` argument and calldata that would transfer the full balance of the contract to themselves.

Consider replacing the external call to `splitERC20Tokens` with an `internal` call to the `_splitERC20Tokens` function.

Update: *Resolved in pull request #1102. The Sandbox team stated:*

> *The `proxyCall` function has been removed.*

H-02 Deploying `RoyaltySplitter` Instance for Creator Can Be Front-Run Allowing Royalties to Be Stolen

The `RoyaltyManager` contract is designed to manage the royalty recipients and their respective proportions of royalties. When a `creator` should be eligible to receive royalties, an

instance of the `RoyaltySplitter` contract is deployed for the `creator`. The `deploySplitter` function will deploy the `RoyaltySplitter` contract for the input `creator` and record the contract instance within the `_creatorRoyaltiesSplitter` mapping. If an instance has already been deployed, the function will return the address of the instance associated with the `creator`.

The `deploySplitter` function contains no access control, allowing anyone to front-run its execution. If a `RoyaltySplitter` instance has not been deployed for a `creator`, the deployment can be front-run allowing a malicious user to set the royalty `recipient` to their own wallet. While the `creator` can change their `recipient` address using the `setRoyaltyRecipient` function, the front-running of the transaction may not be detected as the original transaction will not revert. Further, since no events are emitted within the `deploySplitter` function, users may not be aware that the `recipient` was not set to the expected address.

Consider either including access control in the `deploySplitter` function where only allowlisted tokens can call the function, or not setting the `recipient` value and disabling the functionality of the `RoyaltySplitter` until the `creator` has set their desired `recipient` using the `setRoyaltyRecipient` function.

Update: *Resolved in pull request #1103. The Sandbox team stated:*

> *Appropriate role has been added, deploying of splitters is now only available to contracts with that role. Added tests and updated deployment scripts.*

Medium Severity

M-01 Missing Overflow Check From `tryMul` Function

Within the `_splitERC20Tokens` function in the `RoyaltySplitter` contract, the `tryMul` function is used when multiplying the balance of ERC-20 tokens held by the contract with the recipient's royalty portion. The `success` value returned from the call is never checked to validate that an overflow did not occur. As a result, the `amountToSend` value could silently overflow resulting in an incorrect amount of tokens to be sent to the recipients.

Consider validating that `success` is `true` and reverting if it is not.

Update: *Resolved in pull request #1104. The Sandbox team stated:*

> *Added a require statement after the `tryMul` function call. Added test cases.*

Low Severity

L-01 Abstract Contracts Allow Direct Modification of State Variables

Throughout the codebase, there are `abstract` contracts that contain `public` state variables allowing them to be directly modified by child contracts. This may break the expected properties for the state variables and limit off-chain monitoring capabilities due to the lack of event emissions for changes to the variables. The following instances have been identified:

- Values can be set and deleted directly from the `_tokenRoyaltiesSplitter` mapping in the `MultiRoyaltyDistributor` contract. This could cause discrepancies between the `tokenId` values in the `_tokenRoyaltiesSplitter` mapping and the `_tokensWithRoyalties` array which would cause incomplete data to be returned from the `getTokenRoyalties` and `getAllSplits` functions.
- The `royaltyManager` state variable in the `MultiRoyaltyDistributor` contract.
- The `operatorFilterRegistry` state variable in the `OperatorFiltererUpgradeable` contract.
- The `royaltyManager` state variable in the `RoyaltyDistributor` contract.

Consider using `private` visibility for state variables in `abstract` contracts. Additionally, consider creating `internal` functions for updating these variables which emit appropriate events, and verifying if the desirable conditions are met.

Update: *Resolved in pull request #1105. The Sandbox team stated:*

> *Changed state variables visibility to private and added the needed getter and setter functions. Updated test cases.*

L-02 Array May Contain Duplicate Values

Within the `_setTokenRoyalties` function in the `MultiRoyaltyDistributor` contract, there is no check of whether the `_tokenRoyaltiesSplitter` mapping already contains a

value for the input `tokenId`. As a result, the `_tokensWithRoyalties` array can contain duplicate `tokenId` values.

Consider only pushing new `tokenId` values to the `_tokensWithRoyalties` array if the `tokenId` is not recorded in the `_tokenRoyaltiesSplitter` mapping.

Update: *Resolved in pull request #1106. The Sandbox team stated:*

> *The function has been updated and now only pushes to the array when the splitter is deployed for the first time. Tests have been updated.*

L-03 Deploying `RoyaltySplitter` Instance May Emit Incorrect `recipient` in Event

When the `_setTokenRoyalties` function is called in the `MultiRoyaltyDistributor` contract, it will deploy a new instance of the `RoyaltySplitter` contract for the `creator` if there is no existing instance. When an instance exists, the `deploySplitter` call to the `RoyaltyManager` contract will return the address of the existing instance. Since the `deploySplitter` function does not update the `recipient` when an instance has already been deployed, the `recipient` within the instance may differ from the `recipient` passed to the `_setTokenRoyalties` function. The `TokenRoyaltySet` event uses the input `recipient` address rather than the `recipient` stored in the `RoyaltySplitter` instance.

Consider using the `_recipient` value directly from the instance of the `RoyaltySplitter` deployed at the `creatorSplitterAddress` in the `TokenRoyaltySet` event to ensure the `recipient` value in the event matches the deployed contract.

Update: *Resolved in pull request #1107. The Sandbox team stated:*

> The event mentioned has been removed and the code emit a new event in another place.

L-04 ERC-20 Transfer Return Value Lacks Validation

The `_splitERC20Tokens` function in the `RoyaltySplitter` contract contains the logic to transfer the entire balance of an ERC-20 token held by the contract to the appropriate recipients. In the event that the ERC-20 token transfer calls on lines 166 or 175 fail and return `false` rather than reverting, the function will execute successfully. Further, the

`ERC20Transferred` event will be emitted falsely indicating that tokens have been transferred.

Consider using the `SafeERC20` transfer and returning `false` from the function when the transfer fails.

Update: *Resolved in pull request #1108. The Sandbox team stated:*

> *Changed the code to use safeTransfer as suggested.*

L-05 Incomplete Data in Struct With Royalty Information

The `getTokenRoyalties` function in the `MultiRoyaltyDistributor` contract returns an array of `TokenRoyaltyConfig` structs containing the token split for all `tokenId` values that have royalties set. The `TokenRoyaltyConfig` struct contains the corresponding `tokenId`, the `royaltiesBPS` corresponding to the number of basis points from the sale price that should be paid as royalties, and a list of the royalty `recipients`. The `getTokenRoyalties` function never sets the `royaltyBPS` value in the struct and as a result, all structs will contain the default value of `0` for `royaltyBPS`.

Consider setting the appropriate `royaltyBPS` within the struct to ensure the returned data contains the expected information.

Update: *Resolved in pull request #1109. The Sandbox team stated:*

> *This function has been removed.*

L-06 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 `getContractRoyalty` function in the `RoyaltyManager` contract states that the function "returns the commonRecipient and EIP2981 royalty split". However, it only returns the "royalty split" for the user-provided `_contractAddress`.
- The docstring above the `setSplit` function in the `RoyaltyManager` contract states that the function "sets the common recipient and common split". However, it only sets the "common split".

- The docstring above the `setRecipient` function in the `RoyaltyManager` contract states that the function "sets the common recipient and common split". However, it only sets the "common recipient".
- The docstring for the `getCreatorSplit` function in the `RoyaltyManager` contract states that it is "to be called by the splitters to get the common recipient and split", but it returns the amount of basis points allocated to the creator.

Update: *Resolved in pull request #1111. The Sandbox team stated:*

> *Documentation has been updated*

L-07 Initializer Functions Can Be Called After Initialization

Throughout the codebase, there are `internal` initializer functions that initialize the state of the contracts which lack restrictions that prevent inheriting contracts from calling these functions multiple times. In particular:

- The `__RoyaltyDistributor_init` function in the `RoyaltyDistributor` contract.
- The `__MultiRoyaltyDistributor_init` function in the `MultiRoyaltyDistributor` contract.

Consider restricting calls to these functions to only be called once by the `initialize` function in a parent contract such as by including the `onlyInitializing` modifier.

Update: *Resolved in pull request #1112. The Sandbox team stated:*

> *Add onlyInitializing modifier*

L-08 Lack of Event Emission After Sensitive Action

The following functions do not emit relevant events after executing sensitive actions:

- When the `_trustedForwarder` is set in the `initialize` function in the `RoyaltyManager` contract.
- When the `_trustedForwarder` is set in the `setTrustedForwarder` function in the `RoyaltyManager` contract.

- When a new instance of the `RoyaltySplitter` contract is deployed for a `creator` in the `RoyaltyManager` contract.
- When the `royaltyManager` is set in the `__RoyaltyDistributor_init` function in the `RoyaltyDistributor` contract.
- When the `royaltyManager` is set in the `__MultiRoyaltyDistributor_init` function in the `MultiRoyaltyDistributor` contract.
- When the `_recipient` is set in the `setRecipients` function in the `RoyaltySplitter` contract.

Consider emitting events after sensitive changes occur to facilitate tracking and notify off-chain clients following the contracts' activity.

Update: *Resolved in pull request #1113. The Sandbox team stated:*

> *Added all suggested events*

L-09 Lack of `gap` Variables

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

- The `OperatorFiltererUpgradeable` contract
- The `RoyaltyManager` contract
- The `RoyaltyDistributor` contract
- The `MultiRoyaltyDistributor` contract

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

Update: *Resolved in pull request #1114. The Sandbox team stated:*

> *Added gap variables.*

L-10 Missing Docstrings

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

For instance:

- Line 4 in `IOperatorFilterRegistry.sol`
- Line 11 in `RoyaltyDistributor.sol`
- Line 77 in `RoyaltyManager.sol`

- Line 53 in `RoyaltySplitter.sol`
- Line 5 in `IERC20Approve.sol`
- Line 6 in `IERC20Approve.sol`
- Line 8 in `IERC20Approve.sol`
- Line 6 in `IRoyaltyManager.sol`
- Line 13 in `IRoyaltyManager.sol`
- Line 15 in `IRoyaltyManager.sol`
- Line 17 in `IRoyaltyManager.sol`
- Line 19 in `IRoyaltyManager.sol`
- Line 21 in `IRoyaltyManager.sol`
- Line 23 in `IRoyaltyManager.sol`
- Line 25 in `IRoyaltyManager.sol`
- Line 27 in `IRoyaltyManager.sol`
- Line 29 in `IRoyaltyManager.sol`
- Line 31 in `IRoyaltyManager.sol`
- Line 4 in `IRoyaltyUGC.sol`
- Line 5 in `IRoyaltyUGC.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 #1115. The Sandbox team stated:*

> *Function documentation has been updated*

Notes & Additional Information

N-01 Coding Style Deviates From Solidity Style Guide

The codebase deviates from the recommended Solidity Style Guide. Most notably, there are `public` state variables that are prefixed with an underscore. In particular:

- The `_recipient` variable in the `RoyaltySplitter` contract.
- The `_royaltyManager` variable in the `RoyaltySplitter` contract.
- The `_tokenRoyaltiesSplitter` variable in the `MultiRoyaltyDistributor` contract.
- The `_creatorRoyaltiesSplitter` variable in the `RoyaltyManager` contract.

According to the Solidity Style Guide, only non-external functions and state variables should be prefixed with an underscore.

To favor readability, consider always following a consistent style throughout the codebase.

Update: *Resolved in pull request #1118. The Sandbox team stated:*

> *Updated the names according to the guide.*

N-02 Constant Not Using `UPPER_CASE` Format

In `OperatorFilterSubscription.sol`, the `operatorFilterRegistry` constant is not declared using `UPPER_CASE` format.

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

Update: *Resolved in pull request #1119. The Sandbox team stated:*

> *Changed to uppercase.*

N-03 Contract Should Be Abstract

The `RoyaltyDistributor` contract is intended to be inherited by a token that implements EIP-2981 where royalties are paid to a single common recipient.

Since this contract should never be deployed as a standalone contract, consider marking it as `abstract` to ensure it is always inherited.

Update: *Resolved in pull request #1120. The Sandbox team stated:*

> *Changed contract to abstract, updated tests.*

N-04 `_disableInitializers()` Not Called in Multiple Initializable Contract Constructors

An implementation contract in a proxy pattern would allow anyone to call the `initialize` function on the implementation contract. While not a direct security concern, it is a good practice to prevent the implementation contract from being initialized as this could allow an attacker to take over the contract. This would not affect the functionality of the proxy contract as only the storage of the implementation contract would be affected.

There are multiple Initializable contracts without `_disableInitializers()` called in the constructor in codebase that should be manually checked. For instance:

- The `RoyaltySplitter` contract
- The `RoyaltyManager` contract

Consider calling `_disableInitializers()` in Initializable contracts' constructors to prevent front-running attacks.

Update: *Resolved in pull request #1121.*

N-05 Incomplete Docstrings

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

- The docstring for the `setTokenRoyalties` function in the `IMultiRoyaltyDistributor` interface does not document the function's parameters.

- The docstring for the `getTokenRoyalties` function in the `IMultiRoyaltyDistributor` interface does not document the return value.
- The docstring for the `getAllSplits` function in the `IMultiRoyaltyDistributor` interface does not document the return value.
- The docstring for the `supportsInterface` function in the `MultiRoyaltyDistributor` contract does not document the return value.
- The docstring for the `_setTokenRoyalties` function in the `MultiRoyaltyDistributor` contract does not document the `recipient` parameter.
- The docstring for the `getTrustedForwarder` function in the `RoyaltyManager` contract does not document the return value.
- The docstring for the `setContractRoyalty` function in the `RoyaltyManager` does not document the `contractAddress` parameter.

Update: *Resolved in pull request #1122. The Sandbox team stated:*

> *Documentation has been updated.*

N-06 Inconsistent Coding Style

Throughout the codebase, the `_msgSender` function is used to determine the address that is calling a function. This is generally used for compatibility with ERC-2771 allowing for the use of meta-transactions. The `OperatorFiltererUpgradeable` contract uses `msg.sender` in the `onlyAllowedOperator` modifier which would prevent this modifier from properly validating the message sender if it were to be included in a contract that supports ERC-2771.

Consider using `_msgSender` within all contracts that are intended to be inherited elsewhere in the codebase to ensure meta-transactions are universally supported.

Update: *Resolved in commit 2cf4f66. The Sandbox team stated:*

> *This has been fixed with the initial fixes delivered after giving commit hash to OZ. Provided link to code of one of the other PRs made for previous note.*

N-07 Inconsistent Use of Named Return Values

The following contracts contain some functions where named return values are used and some functions where named return values are not used. For example, within the `RoyaltyDistributor` contract, the `royaltyInfo` function returns named variables,

`receiver` and `royaltyAmount`, as return values, whereas the returned boolean value from the `supportsInterface` function does not have a variable name.

Other instances can be found within the following contracts and interfaces:

- The `RoyaltySplitter` contract
- The `RoyaltyManager` contract
- The `MultiRoyaltyDistributor` contract
- The `IRoyaltyManager` interface
- The `IOperatorFilterRegistry` interface

Consider using named return values consistently throughout each contract and library to provide a clear understanding of the code's behavior.

Update: *Resolved in pull request #1123. The Sandbox team stated:*

> *We have added named exports but decided to keep the explicit returns, We believe named exports add clarity to what the function returns but decided to combine it with explicit returns.*

N-08 Lack of Indexed Event Parameters

Throughout the codebase, several events do not have their parameters indexed. For instance:

- On line 16 of `IMultiRoyaltyDistributor.sol`, the `tokenId` and the `recipient` should be indexed.
- On line 19 of `IMultiRoyaltyDistributor.sol`, the `recipient` should be indexed.
- On line 21 of `IMultiRoyaltyDistributor.sol`, the `recipient` and `splitter` should be indexed.
- On line 7 of `IRoyaltyManager.sol`, the `commonRecipient` should be indexed
- On line 11 of `IRoyaltyManager.sol`, the `contractAddress` should be indexed

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

Update: *Resolved in pull request #1124. The Sandbox team stated:*

> *Added indexed on suggested parameters.*

N-09 Missing Calls to Inherited Contract Initializers

Throughout the codebase, there are contracts that contain initializer functions that do not call their parent initializers. In particular:

- The `initialize` function in the `RoyaltySplitter` contract does not call the `__ERC165_init` function from the `ERC165Upgradeable` contract.
- The `__RoyaltyDistributor_init` function in the `RoyaltyDistributor` contract does not call the inherited `__ERC165_init_unchained` function from the `ERC165Upgradeable` contract.
- The `__MultiRoyaltyDistributor_init` function in the `MultiRoyaltyDistributor` contract does not call the inherited `__ERC165_init_unchained` function from the `ERC165Upgradeable` contract.

Additionally, functions named following the `__{ContractName}_init` pattern should linearize calls to all parent initializers while functions named following the `__{ContractName}_init_unchained` pattern do not make calls to parent initializers.

Consider either ensuring all parent initializers are called, or renaming the functions using the `__{ContractName}_init_unchained` pattern to indicate that the functions do not call parent initializers.

Update: *Resolved in pull request #1125. The Sandbox team stated:*

> *Added missing init function calls.*

N-10 `public` Functions That Should Have `external` Visibility

The following `public` functions should be `external`:

- The `initialize` function in the `RoyaltySplitter` contract.
- The `splitETH` function in the `RoyaltySplitter` contract.
- The `splitERC20Tokens` function in the `RoyaltySplitter` contract.
- The `getTrustedForwarder` function in the `RoyaltyManager` contract.

Consider changing the visibility of these functions to `external` in order to clarify that these functions will only be called by external contracts.

Update: *Partially resolved in pull request #1126.*

The `getTrustedForwarder` function's visibility in the `RoyaltyManager` contract has not been changed.

N-11 Redundant Code

Consider making the following changes to eliminate redundant code:

- In the `OperatorFilterSubscription` contract, the `DEFAULT_SUBSCRIPTION` address does not need to be cast to an address.
- In the `_setRecipients` function in the `RoyaltySplitter` contract, deleting the current `_recipient` value is not required as a new value is set immediately after.
- The `target` argument to the `proxyCall` function in the `RoyaltySplitter` contract does not need to be `payable` as ETH is never sent to the address within the function.

Update: *Resolved in pull request #1127. The Sandbox team stated:*

> *Removed redundant code.*

N-12 Returned Recipients Array Can Contain Zero Address

The `getAllSplits` function in the `MultiRoyaltyDistributor` contract aggregates all addresses that can receive royalties over all `tokenId` values. Since the `_tokenRoyaltiesSplitter` has `public` visibility, it is possible for values to be deleted from the mapping without removing the corresponding `tokenId` from the `_tokensWithRoyalties` array by an inheriting contract. As a result, the array returned from the `getAllSplits` function may contain the zero address.

Consider only returning non-zero `recipient` addresses from this function by validating that addresses are non-zero prior to including them in the `splits` array.

Update: *Resolved in pull request #1106.*

N-13 Typographical Errors

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

- On line 9 of `OperatorFilterSubscription.sol`, "*openSea*" should be "*OpenSea*"

- On line 17 of `OperatorFilterSubscription.sol`, "*open sea*" should be "*OpenSea*"
- On line 9 of `OperatorFiltererUpgradeable.sol`, "*subscibe*" should be "*subscribe*"
- On line 9 of `OperatorFiltererUpgradeable.sol`, "*addess*" should be "*address*"
- On line 9 of `OperatorFiltererUpgradeable.sol`, "*inherting*" should be "*inheriting*"
- On line 15 and line 17 of `MultiRoyaltyDistributor.sol`, "*MultiRoyaltyDistributer*" should be "*MultiRoyaltyDistributor*"
- On line 18 of `MultiRoyaltyDistributor.sol`, "*slip*" should be "*split*"
- On line 161 of `RoyaltyManager.sol`, "*contarct*" should be "*contract*"

Update: *Resolved in pull request #1128. The Sandbox team stated:*

> *Fixed suggested and other found typographical errors.*

N-14 Unbounded Array Length

Within the `MultiRoyaltyDistributor` contract, the `_tokensWithRoyalties` array may grow to an arbitrary length as new values are pushed to the array each time the `_setTokenRoyalties` function is called. This could result in an out-of-gas situation when calling the `getTokenRoyalties` and `getAllSplits` functions as these functions iterate through the entire array.

Although none of these loops impose an immediate risk, consider monitoring these array lengths as the protocol evolves to prevent an out-of-gas situation.

Update: *Resolved. The function was removed in PR 1109. The Sandbox team stated:*

> *getTokenRoyalties function has been removed in one of the previous issues, we no longer iterate over _tokensWithRoyalties.*

N-15 Unused Imports

Throughout the codebase, there are imports that are unused and could be removed. For instance:

- Import `EnumerableSet` of `MultiRoyaltyDistributor.sol`
- Import `Clones` of `MultiRoyaltyDistributor.sol`
- Import `Initializable` of `RoyaltyManager.sol`
- Import `IRoyaltySplitter` of `RoyaltyManager.sol`

- Import `IRoyaltySplitter` of `IMultiRoyaltyDistributor.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved. The issue was resolved here. The Sandbox team stated:*

> *Removed as part of the code delivered shortly after audit start. Simi was notified and accepted.*

N-16 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.

Consider either using or removing the following instance of unused named return variables.

- The `royaltyBps` return variable in the `getContractRoyalty` function in `RoyaltyManager.sol`.

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

> *We have decided to keep the named return with explicit return statement. We believe the code is more readable that way.*

N-17 Unused Variables

Within the `RoyaltySplitter` contract, there are state variables that remain unused. In particular:

- The `IERC20_APPROVE_SELECTOR` variable
- The `SELECTOR_MASK` variable

Consider removing these unused state variables.

Update: *Resolved in pull request #1130. The Sandbox team stated:*

> *Removed unused variables*

Conclusion

Two high-severity and one medium-severity issues were identified in this audit. Several low-severity issues and notes were reported, mainly addressing improvement opportunities to the overall quality of the codebase.