OpenZeppelin | security

OpenSea Operator Filterer Audit

OpenSea

January 18, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	NFT
Timeline	From 2022-12-12 To 2022-12-23
Languages	Solidity

Total Issues	26 (15 resolved, 3 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	2 (0 resolved, 2 partially resolved)
Medium Severity Issues	1 (0 resolved, 1 partially resolved)
Low Severity Issues	7 (7 resolved)
Notes & Additional Information	16 (8 resolved)

Scope

We audited the "OpenSeaProtocol/operator-filter-registry-audit" repository at the `adbb12ef37519e674cfc3482c9a43d7db7a1b31f` commit.

In scope were the following contracts:

```
├── src/
│   ├── DefaultOperatorFilterer.sol
│   ├── IOperatorFilterRegistry.sol
│   ├── OperatorFilterer.sol
│   ├── OperatorFilterRegistry.sol
│   ├── OperatorFilterRegistryErrorsAndEvents.sol
│   ├── OwnedRegistrant.sol
│   ├── RevokableDefaultOperatorFilterer.sol
│   ├── RevokableOperatorFilterer.sol
│   ├── UpdatableOperatorFilterer.sol
│   ├── example/
│   │   ├── ExampleERC1155.sol
│   │   ├── ExampleERC721.sol
│   │   ├── RevokableExampleERC1155.sol
│   │   ├── RevokableExampleERC721.sol
│   │   ├── upgradeable/
│   │   │   ├── ExampleERC1155Upgradeable.sol
│   │   │   ├── ExampleERC721Upgradeable.sol
│   │   │   ├── RevokableExampleERC1155Upgradeable.sol
│   │   │   ├── RevokableExampleERC721Upgradeable.sol
│   ├── upgradeable/
│   │   ├── DefaultOperatorFiltererUpgradeable.sol
│   │   ├── OperatorFiltererUpgradeable.sol
│   │   ├── RevokableDefaultOperatorFiltererUpgradeable.sol
│   │   ├── RevokableOperatorFiltererUpgradeable.sol
```

System Overview

The Operator Filter Registry is a system where any contract or EOA may register a list of addresses or smart contract code hashes to "filter". When an address or codehash is "filtered", it typically means it is marked for exclusion or denial of the ability to call `transferFrom` or `approve` in the NFT contract. It is important to note that the NFT contract controls this (not the registry).

Denial of actions is defined by the NFT contracts, which interface with the registry. Any NFT can query if some address is filtered, and using this may limit what actions that address can take. OpenSea provides examples in which `approval` and `transferFrom` are limited, though these restrictions may be extended by individual NFT projects. OpenSea discourages limiting `transfer` in any way, to avoid "stuck" NFTs even when the owner is a filtered operator.

The registry allows addresses to "register" themselves, which enables them to begin creating their filters. Registrants may also "subscribe" to other registrants, which causes the subscriber's filters to be overridden in favor of the registrant's filters. This means that if the registered address changes filters, later on, those changes will be reflected for themselves as well as all of their subscribers. Notably, subscription chains are not followed, and only the immediate first degree filters are used.

It is worth noting that NFT projects may opt out of using this registry, and that there are known workarounds for many aspects of the registry for a determined entity.

OpenSea, as a part of the Creator Ownership Research Institute (CORI), will manage their own list of filters in the registry and allow NFT owners to subscribe to it. The goal of this list is to filter any contracts or addresses which do not respect royalties for NFT sales. OpenSea has stated that NFTs launched after January 2nd, 2023, which do not make a good faith attempt to filter out marketplaces which do not respect royalty fees, will be marked by OpenSea as ineligible to receive royalties on the OpenSea platform.

Privileged Roles

While the system is designed to be accessible by all, and allows any entity to register their own filter lists or change their subscriptions from one filter list to another, there are still some notable relationships that carry important power dynamics.

The registrant, typically an NFT contract, and the owner of a registrant contract (specifically, the address returned by `owner()`) may at any time modify the registration details. These details contain addresses or code hashes that are "filtered" in the registry for their registration, and a possible subscription to another registrant. The specific actions taken based on the filtered state are determined by the NFT contract. These actions are not strictly limited to `approve` or `transferFrom`. Additionally, the deployer of the NFT contract has the same powers as the owner, by virtue of choosing the capabilities of the NFT contract itself.

If an NFT contract that has integrated with the Operator Filterer codebase subscribes to another registrant within the Operator Filterer Registry, then the subscribed-to registrant has the same power as the owner in the above described case. Fortunately, since many subscribers may be subscribed to a single registrant (and these actions affect all subscribed NFTs equally by subscribing to a list with many subscribers), an NFT can avoid being specifically targeted by subscribing to a sufficiently large list.

Finally, one privileged group is the Creator Ownership Research Institute (CORI), which is partially controlled by OpenSea, as well as other NFT marketplaces and NFT industry players. This group will have total control over the "default" filter list and as such, must be trusted to manage the list responsibly, and not to abuse this power for financial gain.

Integration Concerns

This project provides examples for NFT project developers to use as a reference for their integrations. It is intended for use by various third parties in their integration endeavors.

For NFT projects to safely integrate with the registry, they should be aware of how the registry works and ensure that they will be able to call any necessary functions in the registry if they intend to change their filters or subscriptions. They should note that only the `owner` of the NFT contract, or the NFT contract itself, may call any functions within the registry marked with the `onlyAddressOrOwner` modifier.

For upgradeable contracts, they must manage any potential storage conflicts that may arise during the process of upgrading. They should consider leveraging OpenZeppelin's Upgrades Plugins or another storage layout management system to ensure proper and safe smart contract upgrades.

Finally, they should be familiar with the OpenSea developer documentation and all inline documentation within the codebase.

High Severity

H-01 Add time delay to reduce abuse of power

When an operator or a codehash is added to the `OperatorFilterRegistry`, the effect is immediate. This may cause excessive unintended damage to a mistakenly or controversially filtered operator, as well as the communities that rely on this operator. For example, if a token project subscribes to a registrant, this gives the registrant power over it, as they can update filtered operators or codehashes that take effect immediately. In addition, the effect of filtering addresses or codehashes may be used to disrupt time-sensitive NFT project actions, such as front-running trades with registry updates to prevent them from happening.

In some cases, the impacted communities can react to this change. For example, in the case of tokens subscribing to a malicious or misaligned registrant, the token project can `unsubscribe`. In the case of a marketplace or contract that was incorrectly filtered (e.g., lending contract, or a generic proxy codehash), an off-chain "appeal" to the registrant can be coordinated. However, these reactions will take a long time to coordinate, and in the meantime, the damage from this filtering is ensuing. Thus, a filter enforcement delay may balance the powers between a registrant and its subscribers.

Consider implementing a minimum delay of multiple weeks, starting at the time an address or codehash is updated from "unfiltered" to "filtered", while maintaining the immediate removal of a filtered address or codehash. This can be implemented by storing timestamps when updates are made, and using them to calculate whether or not an address or codehash is filtered.

Update: *Partially resolved in pull request #2 at commit `d7a3113`. The CORI subscription's ownership has been transferred to a Timelock Controller. The README has been updated to reflect this change. Note that the `TimelockController` has not been audited in this engagement. Other subscriptions are still instantly changeable, unless managed by a similar timelock.*

H-02 `isOperatorAllowed` poorly named and misused

The implementation of `isOperatorAllowed` does not match the expected behavior as described in the integration examples for the following interfaces:

- `OperatorFilterer`
- `UpdatableOperatorFilterer`
- `OperatorFiltererUpgradeable`
- `OperatorFiltererUpgradeable`
- `RevokableOperatorFiltererUpgradeable`
- `RevokableOperatorFiltererUpgradeable`

According to these examples, the function should return `false` if the transfer is to be blocked. However, the current implementation does not return `false` in this case and instead reverts the transaction without reaching the `OperatorNotAllowed` error. Additionally, the naming pattern "isSomeValue" does not suggest that the function will revert on certain conditions (as opposed to "revertIf").

This is confusing and error-prone, since the calling code may expect to handle a `false` return value, and not be prepared to handle a reverting `view` function. Depending on the caller's implementation this may lead to high-severity issues. For example, if the caller is expecting to handle or block multiple transfers depending on the registry's return value, a single revert will cause all transfers to fail (even those that do not need to be blocked).

Consider renaming the `isOperatorAllowed` function to `revertIfTransferBlocked`, removing the boolean return value from the interface, and updating the usage in examples to not expect a return value, but to expect a revert. Removing the boolean return in this case will simplify the code and make its intention easier to understand for future developers. Alternatively, instead of renaming, consider changing the implementation to not revert, and instead return `false` as the current name, interface, and usage in examples imply. In this case, the example code provided does not need to be modified.

Update: *Partially resolved in pull request #19 at commit `c09db68`. The functions relying on `isOperatorAllowed` (now renamed to `validateOperator`) still do not expect a revert, and the `OperatorNotAllowed` error is still unreachable. OpenSea has acknowledged this is by design.*

Medium Severity

M-01 Incomplete test coverage

Some of the code suggested by the community for integration lacks sufficient testing. This can lead the community to assume that the functionality has been thoroughly tested, and they may integrate untested code into their token contracts. Full test coverage could have also prevented some of the other findings, such as the interface mismatch with `isOperatorAllowed` that causes it to revert instead of returning false as expected. Ensuring coverage of `OperatorNotAllowed` reverts would have revealed this issue.

Some of the examples of missing test cases are:

- Reverting with `OperatorNotAllowed` error in e.g., `OperatorFilterer` (and similarly in `UpdatableOperatorFilterer`, `OperatorFiltererUpgradeable`, `RevokableOperatorFiltererUpgradeable`)
- All cases for `updateOperatorFilterRegistryAddress` in `UpdatableOperatorFilterer`
- `isOperatorFilterRegistryRevoked` in `RevokableOperatorFiltererUpgradeable`
- The `if` branches of `__OperatorFilterer_init` in `OperatorFiltererUpgradeable` where only 1 branch is tested out of at least 5
- Testing that `approve` calls filter functionality for the example tokens (e.g., in `ExampleERC721`, and similarly for `RevokableExampleERC721`, `ExampleERC721Upgradeable`, and `RevokableExampleERC721Upgradeable`)
- Reverts of `revokeOperatorFilterRegistry` in `RevokableOperatorFilterer`
- Reverts of `revokeOperatorFilterRegistry` in `RevokableOperatorFiltererUpgradeable`

Consider ensuring a minimum standard of complete test coverage for solidity code to follow best practices, and reduce the likelihood of introducing faulty code into other projects. For example, an HTML coverage report for the audited repository can be generated by running the following shell command: `forge coverage --report=lcov && genhtml lcov.info -o report --branch-coverage`. Consider automating the measurement and reporting of test coverage in Pull Requests (i.e., using GitHub Actions) to prevent coverage reduction.

Update: *Partially resolved in pull request #4 at commit `5fc0ef7`. Note that test cases for `approve` of `Example...` tokens are still missing.*

Low Severity

L-01 Examples lack EIP-2981 implementation

EIP-2981, the NFT royalty standard, is a standardized way to retrieve royalty payment information to allow for universal support for royalty payments across NFT marketplaces. The operator filterer's policy will require that contracts deployed after January 2nd implement EIP-2981 as their objective source of truth for creator fee preferences. However, the examples lack implementation of EIP-2981 and thus the creators may use a different standard or mechanism for royalties that is not recognized by the operator filterer. For instance, note that none of the examples implement a `royaltyInfo` function as specified by the EIP.

Consider inheriting from the OpenZeppelin `ERC721Royalty` or the OpenZeppelin `EIP2981` contracts in the provided examples to make it easier for NFT creators to follow EIP-2981.

Update: *Resolved in pull request #5 at commit* `f5f6bdd`.

L-02 Lacking length views for `EnumerableSets`

The `view` functions `subscriberAt`, `filteredCodeHashAt`, and `filteredOperatorAt` all expect an index into an enumerable set. However an external caller may not know the range of possible indices without a corresponding length getter. This may limit the ability to inspect the contents of the lists of filtered entries or subscribers for some registrants.

A possible limited workaround would be to call, for example, `subscribers()` and check the length of the full array, but that may be prohibitively gas-expensive or may fail completely for lengths above a certain limit.

Consider adding view functions for the lengths of these sets to allow off-chain and on-chain access to the sets' contents.

Update: *Resolved in pull request #6 at commit* `12a24a7`.

L-03 Missing docstrings

Docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned, and the events emitted. If the codebase does not have proper docstrings, it hinders reviewers' understanding of the code's intention, which is fundamental to correctly assessing security and correctness.

Throughout the codebase, there are several files that do not have docstrings. For instance:

- The `IOperatorFilterRegistry` interface does not have any docstrings.
- Events and errors in `OperatorFilterRegistryErrorsAndEvents` do not have NatSpec comments.
- The contracts within the example folder should be fully documented with detailed docstrings, as they are intended to serve as a reference for interested users. However, it has been observed that the current docstrings are insufficient in this regard.
- Both the `RevokableDefaultOperatorFiltererUpgradeable` and `OperatorFiltererUpgradeable` contracts lack docstrings that describe their purpose.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' 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 #7 at commit* `1535f8e`.

L-04 Unchecked blocks can be reduced

There are a few instances in `OperatorFilterRegistry` where the unchecked keyword is used to cover entire loops and their contents in `updateOperators`, `updateCodehashes`, and `_copyEntries`. This appears to be done in order to save gas when the loop counter increments in each case.

Since the loop counter starts at 0 and increments by 1 each time, it is unlikely to overflow, making the use of `unchecked` math safe. However, to make the intent of the `unchecked` keyword clearer and reduce the risk of errors, consider limiting the scope of the `unchecked` block to only include the increment operation. This can be done by inlining the unchecked block within the `for` loop definition.

Note that using the `unchecked` keyword can be dangerous if applied to more operations than necessary.

Update: *Resolved in pull request #8 at commit `adf45d4`.*

L-05 Regular storage used in upgradable contract

A boolean value is stored in regular linear storage in the `RevokableOperatorFiltererUpgradeable` contract.

This may complicate storage slot considerations for integrating upgradable token contracts and can potentially cause storage slot collisions if not properly managed.

Consider using a specific storage location (similarly to EIP-1967) and the OpenZeppelin StorageSlot helper library to reduce storage layout risks for upgradable contracts integrating this mixin.

Update: *Resolved in pull request #9 at commit `90da83b`.*

L-06 Unowned token contracts cannot execute privileged functions

The `OperatorFilterRegistry` contract contains several privileged functions, including `register`, `unregister`, `registerAndSubscribe`, `registerAndCopyEntries`, `updateOperator`, `updateCodeHash`, `updateOperators`, `updateCodeHashes`, `subscribe`, `unsubscribe`, and `copyEntriesOf`, that can only be called by the owner of the token contract or the token contract itself. If a token contract inherits from `OperatorFilterer` or `DefaultOperatorFilterer`, but does not implement either an `owner` method or a method that can call one of these privileged functions, the token contract will maintain the same filtering as it had right after construction. The token contract will be unable to add or remove operators or codehashes, and if it is already subscribed, it will be unable to unsubscribe.

Consider ensuring that an `owner` that returns a non-zero address exists during construction in `OperatorFilterer` and `DefaultOperatorFilterer`. Additionally, consider making it very clear in the docstrings and external documentation that if the token contract inherits from `OperatorFilterer`, and does not have an `owner` nor methods to call these privileged functions, then it will be locked into its subscription during construction.

Update: *Resolved in pull request #10 at commit `ac59e67`. Note that only documentation changes were made.*

L-07 Zero codehash unhandled

According to the EIP-1052, the `codehash` of an account with no contract code, no balance and a zero nonce is `0`. This value differs from the `EOA_CODEHASH`, which is correct for accounts that have a nonzero nonce or a nonzero balance.

In many places in the codebase, the condition `code.length > 0` is used to filter out both empty and non-empty EOAs. However, in the functions `updateCodeHash` and `updateCodeHashes`, the only check performed to filter out EOAs ensures that the codehash is not the `EOA_CODEHASH`. As a result, these functions do not filter out empty EOAs.

Consequently, the `_filteredCodeHashes` enumerable set for the registrant can have the codehash `0` added. This will be reflected when calling the following functions:

- `isCodeHashFiltered`
- `isCodeHashOfFiltered`
- `filteredCodeHashes`
- `filteredCodeHashAt`

While none of these functions are directly called within the codebase, note that they are externally callable `view` functions. Projects building on top of this codebase may erroneously assume some empty EOA is filtered out by relying on these calls. Importantly, this is an error since the `isOperatorAllowed` function only filters contracts with `code.length > 0`.

Consider changing the `updateCodeHash` and `updateCodeHashes` functions to disallow filtering the codehash `0`.

Update: *Resolved in pull request #11 at commit `2bd964b`.*

Notes & Additional Information

N-01 Constants not using upper case format

The constants declared in `OwnedRegistrant` and `OperatorFiltererUpgradeable` are not in all upper case with underscores separating words. This conflicts with the Solidity style guide. For better readability, consider following this convention.

Update: *Resolved in pull request #13 at commit `d14f148`.*

N-02 Confusing semantics of `subscriptionOf`

The view function `subscriptionOf` reverts on subscription to `address(0)`, but then returns `address(0)` if the subscription is made to itself. This is confusing and the intent of this behavior is unclear.

Furthermore, the internal name of `_registrations` maps to the external concept of subscriptions. This is confusing to read and reason about.

Consider renaming `_registrations` to `subscriptions` and making it a public mapping to take advantage of the automatic getter and reduce the complexity of the interface semantics. Furthermore, consider either removing `subscriptionOf`, or detailing the reason its return values differ from those of the current `_registrations` mapping in the docstrings. Note that otherwise, external developers may not be aware of this small but important difference.

Update: *Resolved in pull request #12 at commit `77fe082`.*

N-03 Gas inefficiencies

There are instances throughout the codebase where changes can be made to improve gas consumption. For example:

- The gas for the SLOAD instruction when loading `_filteredCodeHashes` in `isOperatorAllowed` can be saved if the execution does not reach its usage later in the code (e.g., due to failing the address check, or due to code length check). Consider moving the SLOAD next to the place where it will be used. Doing so could potentially save 2100 gas per call, especially if the data being loaded is not frequently accessed (i.e., "cold"). This would be a significant improvement, especially since this view function is expected to be heavily used by tokens.
- The current implementation incurs additional gas costs by checking the `operator.code.length`. Instead, we can check if `operator.codehash` is equal to `EOA_CODEHASH` or `bytes32(0)`, which represents the code hash of a non-existent account. To optimize gas usage, consider loading the codehash value into memory before the `if` statement, and then comparing it to `EOA_CODEHASH` or `bytes32(0)` in the `if` condition. This would save 100 gas for cases where the `if` block is executed.

When performing these changes, aim to reach an optimal tradeoff between gas optimization and readability. Having a codebase that is easy to understand reduces the likelihood of errors in the future and improves transparency for the community.

Update: *Resolved in pull request #14 at commit `f377262`. The Opensea team stated:*

> *Noting that removing `_filteredCodeHashes` only saves a couple hundred gas, as it is merely computing a storage reference, and not doing an actual `SLOAD`. Regardless, it is inefficient to calculate it before it is needed.*

N-04 Duplicate check of registry address

The `operatorFilterRegistry` address is loaded and checked against the zero address in both `RevokableOperatorFilterer` and its `super` method in `UpdatableOperatorFilterer`. Consider completely removing the overriding method in `RevokableOperatorFilterer`.

Update: *Resolved in pull request #15 at commit `d0acc3d`.*

N-05 Duplicated code

Code is duplicated between `updateOperator` and `updateOperators`, and similarly between `updateCodeHash` and `updateCodeHashes`. This is error prone and the methods can reuse some of the code without a significant difference in execution gas.

Consider refactoring to use an `internal` "multiple" version, and calling it from both the "single" and "multiple" `external` methods. Doing so will lower the likelihood of errors caused by differences between each function.

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

> *May still get to this, but for a note-severity issue, the risk of the scope of code changes required likely outweighs the potential upside.*

N-06 Hard-coded address should be a constant instead

To improve readability and follow best practices, consider replacing the hard-coded registry address with a named constant. This change can help with error checking and understanding the purpose of the code. Descriptive names can also enhance understandability.

Update: *Resolved in pull request #16 at commit* `c29cc7f`.

N-07 OpenZeppelin imports in incorrect format

In files that use OpenZeppelin contract imports, the current format is `import {SomeContract} from "openzeppelin-contracts/..."`. With the current import statement, NFT creators may run into compilation errors when trying to integrate their NFT contract with the operator filterer registry. For ease of consumption for NFT creators using this operator filterer registry, consider changing the imports to be written as `import {SomeContract} from "@openzeppelin/contracts/..."`.

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

> *The repository is configured as a Foundry project, and thus favors (what are currently) Foundry-style imports. As part of releasing on NPM, the workflow re-maps Foundry-style imports to NPM-compatible imports. As long as developers are using some form of*

> *dependency management (`.gitmodules` for Foundry or `npm` for Truffle/Hardhat-style) they should not run into issues with paths.*

N-08 Integrating token may incorrectly prevent transfers

The intended use of the filtering is for "pull" transfers (`.transferFrom`), but a token integrator may assume filtering is expected for `.transfer` and block those as well. Although the audited reference implementation allows `from==msg.sender` to pass unfiltered, it is still possible that the integrating token will not use this implementation, omitting the `msg.sender` pass through check, and possibly implement filtering on `.transfer`. Additionally, relayed transactions or meta-transactions where `msg.sender` is replaced by a `_msgSender()` method may pose similar issues.

As a result, some assets may become stuck.

Consider clearly documenting via inline docstrings and external documentation that under no circumstances should the `.transfer` method be filtered. Also consider renaming `isOperatorAllowed` to `nonOwnerTransferUnfiltered` or some other name that better implies the intent of the modifier. Finally, consider adding checks in the suggested compliance validation test ensuring that a filtered operator is able to successfully call `.transfer`.

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

> *These tools are meant for non-fungible and semi-fungible tokens that are either upgradeable or will be deployed in the future, since immutable contracts cannot have their code updated. The `transfer` method was included in early drafts of the ERC-721 specification, but was ultimately dropped in the final specification.*

N-09 Misnamed error

The `CannotFilterEOAs` error is thrown when a codehash to be filtered is equivalent to the "uninitialized" codehash.

The error implies that an EOA is being filtered, but does not include the case where a contract is under construction (during which the codehash returns the uninitialized value). For greater clarity, consider renaming this error to `CannotFilterUninitializedCodeHash` or something similar to illustrate the possibility of a non-EOA account.

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

> *The `EOA_CODEHASH` constant is the hash of any account with balance/nonce but no code, including both EOAs and smart contracts within the context of their constructor. The hash of accounts with no state is `bytes32(0)`, as noted in L-07. Since it is not possible to distinguish between an EOA and a smart contract within the context of its constructor, and because the concept of "initialized" vs "uninitialized" accounts with differing codehashes may be unfamiliar or abstract, the name `CannotFilterEOAs` is more descriptive of the intent of the error.*

N-10 Missing events

Upon executing sensitive actions that modify the state, the following functions do not emit any events:

- The function `updateOperatorFilterRegistryAddress` in the contracts `RevokableOperatorFilterer.sol` and `UpdatableOperatorFilterer.sol` updates the registry address, but does not emit an event.
- The function `revokeOperatorFilterRegistry` in the contracts `RevokableOperatorFilterer.sol` and `RevokableOperatorFiltererUpgradeable.sol` revokes the `OperatorFilterRegistry` address, but does not emit an event.

It is recommended to emit events for all updates to storage in order to facilitate efficient tracking of contract state by off-chain tools.

Update: *Resolved in pull request #17 at commit `f13cae1`.*

N-11 Modifier implementation is inconsistent and expensive

Within `OperatorFilterer.sol`, the `onlyAllowedOperator` and `onlyAllowedOperatorApproval` modifiers rely on an internal function `_checkFilterOperator` to consolidate their logic, which is nearly identical.

However, within `RevokableOperatorFiltererUpgradeable` and `OperatorFiltererUpgradeable`, the `onlyAllowedOperator` and `onlyAllowedOperatorApproval` modifiers instead implement nearly identically the same process without using `_checkFilterOperator`.

Consider defining `_checkFilterOperator` for both `RevokableOperatorFiltererUpgradeable` and `OperatorFiltererUpgradeable`, and consolidating the logic from `onlyAllowedOperatorApproval` and `onlyAllowedOperator` into this internal function, similarly to within `OperatorFilterer`. The surface for error will be reduced, as implementing nearly identical code multiple times can result in unseen differences between each implementation. Additionally, these changes will reduce the bytecode size.

Update: *Resolved in pull request #21 at commit `d8109c4`.*

N-12 Naming suggestions

The intended implementation is specific and unambiguous, but some contract, method, and variable names are generic and ambiguous. Those names can confuse readers, such as community members who may not understand that a token is integrating with a third-party contract to block asset transfers. Additionally, integrators may incorrectly interpret boolean values that are not properly named in interfaces.

Consider renaming the following:

- `OperatorFilterRegistry` to `TransferBlocklist` ("Operator" suggests a privileged management role, but it is unclear what the context of the operator is. "Filtering" is ambiguous and could refer to either allowing to pass through or blocking. This also shortens the name from 9 syllables to 4.)
- `OperatorFilterer` to `BlocklistRegistrant`
- `isOperatorAllowed` to `isNotBlocked`
- `bool filtered` in multiple places (such as the definition of `updateOperator`) to `bool blocked`
- `updateOperator` to `updateBlockedAddress`
- `updateCodeHash` to `updateBlockedCodeHash`

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

> *OpenSea's language regarding the registry is ossified at this point.*

N-13 Consider paginating copies

The `copyEntries` function uses loops to copy and store potentially large length `EnumerableSet`s. Eventually, some registrants' `_filteredOperators` or

`_filteredCodeHashes` lists may become so long that these loops are unexecutable, causing execution to exhaust all gas within a block.

This will have the effect of making certain registrants' lists uncopyable within `copyEntriesOf`, `registerAndCopyEntries`, and `unsubscribe` (if `copyExistingEntries` is `true`).

Consider implementing a paginated version of the `copyEntriesOf` function, which would allow users to specify an index to start and stop copying at, thereby limiting the gas cost of each operation. Although there are alternative approaches that can achieve the same result, pagination may be a more efficient and user-friendly solution. In the documentation, consider making users aware of the limitations of these functions and providing instructions on how to work around out-of-gas errors using the currently available functions.

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

> *`copyEntries` is already gas-inefficient compared to the bulk `updateOperators`/`updateCodeHashes` methods; it is useful for simple configuration of small entries in the registry, but adding pagination would add more complexity to the registry logic and encourage more gas-intensive operations.*

N-14 Redundant control flow

The `if` in `isOperatorFiltered`, and similar code in `isCodeHashFiltered`, `isCodeHashOfFiltered`, `filteredOperators`, `filteredCodeHashes`, `filteredOperatorAt`, and `filteredCodeHashAt` are redundant and can be simplified to call the `if` body unconditionally. This is because the implicit `else` branch will result in the same return value.

To simplify the code, consider using only the inline version of the `if` statement's body in the relevant cases. This may help streamline the code and improve readability.

Update: *Resolved in pull request #18 at commit `684a74a`.*

N-15 Registration step is redundant

The interface includes steps for registering and unregistering, which may add unnecessary complexity and make it more difficult for integrators to understand and use. These `registration`-related concepts and steps may increase the risk of errors and make the interface unnecessarily complex.

Since `address(0)` is unowned, it cannot have any filtered operators or codehashes. Thus, subscribing to it via `subscribe` is functionally equivalent to unregistering, and similarly subscribing to an existing address is equivalent to registering.

Consider removing the `register` semantic from the external interface, and instead only keeping it internally for the gas optimizations (to avoid checking `address(0)` entries). This will make the interface less error prone and more clear for integrators.

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

> *The concept of registration is semantically useful when compared to the concept of "subscribing" to oneself before one has made an entry in the registry. It should be necessary to be registered before managing one's entries in the registry, or else one may update entries without theirs being reflected; similarly, "unregistering" is a clearer step of "opting-out" of the checks, which is not necessarily implied by subscribing to `address(0)` (though they are technically the same).*

N-16 State variable visibility not explicitly declared

Throughout the codebase there are state variables that lack an explicitly declared visibility:

- The state variable `DEFAULT_SUBSCRIPTION` in `DefaultOperatorFilterer.sol`
- The state variable `EOA_CODEHASH` in `OperatorFilterRegistry.sol`
- The state variable `registry` in `OwnedRegistrant.sol`
- The state variable `DEFAULT_SUBSCRIPTION` in `RevokableDefaultOperatorFilterer.sol`
- The state variable `DEFAULT_SUBSCRIPTION` in `DefaultOperatorFiltererUpgradeable.sol`
- The state variable `DEFAULT_SUBSCRIPTION` in `RevokableDefaultOperatorFiltererUpgradeable.sol`

For clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

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

> *May still get to this, but the scope will touch most files for relatively little benefit.*

Conclusions

Two high severity issues have been found among several lower-severity issues. The OpenSea team has been responsive to auditors' questions and suggestions during the audit. Some recommendations were proposed to follow best practices and reduce the potential attack surface.

Appendix

Monitoring Recommendations

To keep the protocol secure and operating as intended, we recommend monitoring the following:

- Monitor changes to the default list to detect unplanned changes. This will help monitor the safety of the keys controlling the default list. Additionally, consider publishing and maintaining a list of all changes to the default list in a human-readable format.
- For NFT projects, monitor the account your NFT contract is subscribed to in order to detect changes to its filter list or subscription status.
- For NFT projects, monitor all `Transfer` event emissions and ensure that a creator fee was paid for each transaction. This can be used to detect marketplaces that are not respecting creator fees.
- Monitor each registered NFT project's ability to change its filter list or subscription status by checking the NFT contract's `owner`, checking the ability of the NFT contract to call functions in the registry, and by checking whether the NFT contract can be upgraded. This can be open-sourced to show whether an NFT can change its filtering properties.
- Monitor upgrades to NFT contracts that interface with the registry.
- Monitor newly registered or upgraded NFT contracts, and perform static analysis to ensure that `onlyAllowedOperator` and `onlyAllowedOperatorApproval` are not applied to the `transfer` function.