Quantstamp Security Assessment Certificate

January 18th 2023 — Quantstamp Verified

OpenSea Royalties (Sandbox)

This audit report was prepared by Quantstamp, the leader in blockchain security.

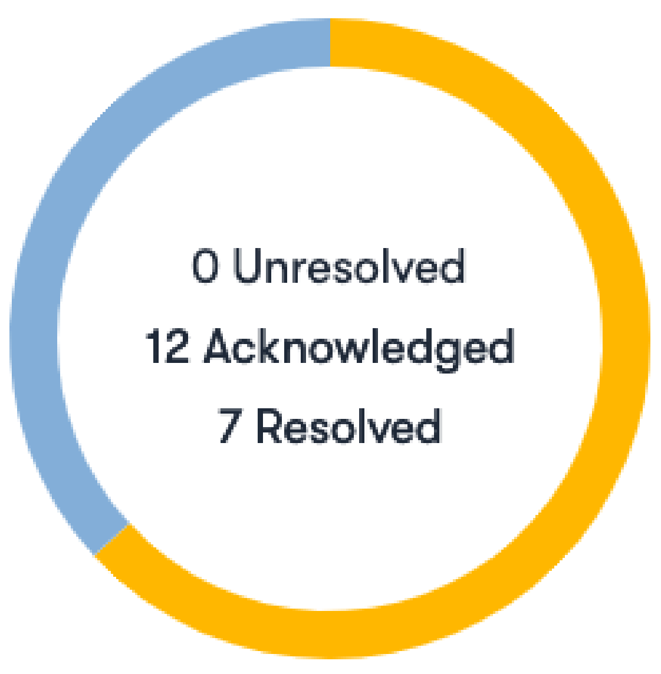
Executive Summary

Type	NFT
Auditors	Jennifer Wu, Auditing Engineer Rabib Islam, Auditing Engineer Cameron Biniamow, Auditing Engineer Mustafa Hasan, Auditing Engineer
Timeline	2023-01-11 through 2023-01-18
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Sandbox's OpenSea Operator Filter Registry Implementation Raffle Smart Contract Operator Filter Registry
Documentation Quality	Medium
Test Quality	Low

Source Code

Repository	Commit
thesandboxgame/sandbox-smart-contracts	0dcaefd initial audit

Total Issues	**19**	(7 Resolved)
High Risk Issues	0	(0 Resolved)
Medium Risk Issues	0	(0 Resolved)
Low Risk Issues	**8**	(2 Resolved)
Informational Risk Issues	**10**	(4 Resolved)
Undetermined Risk Issues	**1**	(1 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has performed an audit of Sandbox's `GenericRaffle` extended with `UpdatableOperatorFiltererUpgradeable`. The `GenericRaffle` contract has been updated to remain compliant with OpenSea creator fee enforcement requirement. All Sandbox NFT collections are implementations of the `GenericRaffle` contract. This `GenericRaffle` update ensures that all marketplaces blocked by OpenSea will be also blocked by the NFT contracts created by Sandbox. Quantstamp had previously performed an audit of Sandbox's `GenericRaffle` contract in the `Sandbox Avatar Collection` report. This report is an extension of the `Sandbox Avatar Collection` report and should be reviewed in conjunction.

Furthermore `UpdatableOperatorFiltererUpgradeable` and `GenericRaffle` contracts are highly dependent on Opensea's `OperatorFilterRegistry` contract, which is out of scope. Issues with OpenSea's `OperatorFilterRegistry` could cause `GenericRaffle` to behave incorrectly or break. This is largely mitigated since the Operator Filter Registry can be updated; however, this is dependent on the owner of `GenericRaffle` to make the update.

Update: The team addressed the issues related to `GenericRaffle` and acknowledged the issues regarding `UpdatableOperatorFiltererUpgradeable`. The client considers the `UpdatableOperatorFiltererUpgradeable` contract to be temporary until OpenSea accepts the pending pull request to include the `UpdatableOperatorFiltererUpgradeable` contract. The team plans to reference this contract from the project once the PR is merged.

ID	Description	Severity	Status
QSP-1	Wave Setup Is Possible During Contract Pause	Low	Fixed
QSP-2	Registry Enforcement May Not Match Opensea	Low	Acknowledged
QSP-3	Registry May Be Undefined	Low	Acknowledged
QSP-4	Signatures Do Not Expire and Cannot Be Revoked	Low	Acknowledged
QSP-5	Potential Collusion Between Signer and Minter Can Cause Unintended Purchases	Low	Acknowledged
QSP-6	Operator Filter Registry with Incorrect Implementation Blocks All Transfers	Low	Acknowledged
QSP-7	Generic Raffle Tokens Transferred to a Blocked Operator	Low	Acknowledged
QSP-8	Solidity Version `0.8.13` is Bugged	Low	Fixed
QSP-9	Unable to Access Operator Registry After Update if Ownership Is Undefined	Informational	Acknowledged
QSP-10	Pseudorandom Values Are Known Ahead of Time	Informational	Acknowledged
QSP-11	Storage Collision when Updating Existing `GenericRaffle` Implementation Contract	Informational	Acknowledged
QSP-12	Certain State Changes Lack Corresponding Events	Informational	Mitigated
QSP-13	ERC20 Token Is Caller of `GenericRaffle.mint()`	Informational	Acknowledged
QSP-14	Privileged Roles	Informational	Mitigated
QSP-15	Upgradeable Contract Storage Gaps	Informational	Acknowledged
QSP-16	Disable Initializer in Implementation Contract	Informational	Fixed
QSP-17	Clone-and-Own	Informational	Acknowledged
QSP-18	Unlocked Pragma	Informational	Fixed
QSP-19	Lack of Code Documentation	? Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Wave Setup Is Possible During Contract Pause

Severity: Low Risk

Status: Fixed

File(s) affected: `GenericRaffle.sol`

Description: The `setupWave()` function does not revert when the contract is paused, which allows waves to be set up. The Raffle Smart Contract documentation states that `setupWave()` should fail when paused. The function `setupWave()` does not include any logic that checks the state of `paused`.

Recommendation: Confirm that `paused` is disabled before allowing `setUpWave()`.

Update: The client added a `require` statement in `setupWave()` that reverts the call when the contract is paused. The fix is implemented in: `5c058a1c88d65d736cf4aaff733dc64d0c1a1981`.

QSP-2 Registry Enforcement May Not Match Opensea

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GenericRaffle.sol`, `UpdatableOperatorFiltererUpgradeable.sol`

Description: According to the Sandbox's OpenSea Operator Filter Registry Implementation specification, NFT contracts created by Sandbox must block any marketplaces blocked by OpenSea. Based on `GenericRaffle` and `UpdateableOperatorFiltererUpgradeable` implementations, no enforcements are applied when creating or updating the registry.

Recommendation: Update the specification and confirm whether this behavior is intentional. If this behavior is not expected, implement input validation when initializing `GenericRaffle`.

Update: The client acknowledged the issue and provided the following explanation:

> Our implementation is generic in this way because we want to be able to fully control if and to what address to block, this is intended behavior. We do agree that our documentation specified that a full block is mandatory, as such we modified the documentation to also mention that we are reserving the right to change the block addresses/registry used.

QSP-3 Registry May Be Undefined

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GenericRaffle.sol`

Description: If an inheriting token contract is deployed to a network without the registry deployed, the modifier will not revert, but the contract needs to be registered with the registry once it is deployed in order for the modifier to filter addresses. According to the specification, NFT contracts created by Sandbox must block any marketplaces blocked by OpenSea. However, during `GenericRaffle` initialization, `_registry` may be undefined. The intention of leaving `_registry` undefined allows the registry contract to be deployed after. However, if the `registry` is not updated in `GenericRaffle`, the modifier to filter addresses is bypassed.
In `GenericRaffle`, the modifiers `onlyAllowedOperator` and `onlyAllowedOperatorApproval` are applied to the following functions:

1. `GenericRaffle.setApprovalForAll()`
2. `GenericRaffle.approve()`
3. `GenericRaffle.transferFrom()`
4. `GenericRaffle.safeTransferFrom()`

Recommendation: Consider confirming that `_registry` is a contract to ensure address filter enforcement.

Update: The client acknowledged the issue and provided the following explanation:

> Our implementation is generic in this way because we want to be able to fully control if and to what address to block, this is intended behavior. We do agree that our documentation specified that a full block is mandatory, as such we modified the documentation to also mention that we are reserving the right to change the block addresses/registry used.

QSP-4 Signatures Do Not Expire and Cannot Be Revoked

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GenericRaffle.sol`

Description: Within the functions `mint()` and `personalize()`, signatures from `signAddress` are needed to execute calls. Since the signatures from `signAddress` do not expire, nor can they be revoked, the caller can choose to use the signature to mint or personalize tokens at any time.

Recommendation: Include an expiration time in the `mint()` and `personalize()`signatures and create a function that allows the owner or signer address to revoke signatures.

Update: The client acknowledged the issue and provided the following explanation:

> We acknowledge that this may be an issue in case we will ever make signatures for 3rd party, non-Sandbox staff. We do not intend to, and, currently, consider this acceptable.

QSP-5 Potential Collusion Between Signer and Minter Can Cause Unintended Purchases

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GenericRaffle.sol`

Description: At `GenericRaffle.sol#170`, in the execution of `mint()`, ERC20 tokens are transferred from `_wallet` to `sandOwner`. However, the address executing this transfer is `allowedToExecuteMint`, which may be different from `_wallet`. Moreover, `signAddress` enables the calling of `mint()`. Therefore, if `_wallet` gives an infinite approval, tokens may be taken from `_wallet` without the user's consent through the `mint()` function.

Recommendation: Require the signature of `_wallet` on each execution of the `mint()` function if token transfers are required.

Update: The client acknowledged the issue and provided the following explanation:

> The issue appears if `signAddress` and `allowedToExecuteMint` addresses would ever be set to a malicious party. As it is, we only assign it to authorized trusted members of our team and set it as required by our policy. We do not consider this an issue at the moment.

QSP-6 Operator Filter Registry with Incorrect Implementation Blocks All Transfers

Severity: Low Risk

Status: Acknowledged

File(s) affected: `UpdatableOperatorFiltererUpgradeable.sol`

Description: The owner of `GenericRaffle` can update the Operator Filter Registry by calling `updateOperatorFilterRegistry()` and passing the new address. When calling the approve or transfer functions of `GenericRaffle`, the modifiers `onlyAllowedOperatorApproval` and `onlyAllowedOperator` call `isOperatorAllowed()` from the Operator Filter Registry when the registry is a deployed contract. If the owner updates the Operator Filter Registry to a contract that does not include `isOperatorAllowed()`, calls to the following functions of `GenericRaffle` will fail:

1. `GenericRaffle.setApprovalForAll(address operator, bool approved)`
2. `GenericRaffle.approve(address operator, uint256 tokenId)`
3. `GenericRaffle.transferFrom(address from, address to, uint256 tokenId)`
4. `GenericRaffle.safeTransferFrom(address from, address to, uint256 tokenId)`
5. `GenericRaffle.safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data)`

Recommendation: When initializing and updating the Operator Filter Registry, ensure that the registry contract implements all necessary functions.

Update: The client acknowledged the issue and provided the following explanation:

> The used code that works with the Operator Filter Registry is considered by us as temporary until OpenSea will accept our PR, the code itself was derived from their already existing functions. We do not intend to add any extension to this part of the code for the time being. Regarding this issue, currently, we are ok with mitigating it by changing the registry to a valid one, if ever an invalid one was set.

QSP-7 Generic Raffle Tokens Transferred to a Blocked Operator

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `GenericRaffle.sol`

Description: Although `GenericRaffle` restricts approving a blocked operator and transferring tokens from a blocked operator, the owner of a token can still transfer the token to a blocked operator. Once a blocked operator becomes the owner of a `GenericRaffle` token, all future transfers will fail.

Recommendation: Consider adding reverting logic for transfers of `GenericRaffle` tokens to blocked operators.

Update: The client acknowledged the issue and provided the following explanation:

> The used code that works with the Operator Filter Registry is considered by us as temporary until OpenSea will accept our PR, the code itself was derived from their already existing functions. We do not intend to add any extension to this part of the code for the time being. Regarding the mentioned issue; OpenSea registry logic dictates that 3rd party contracts from blocked marketplaces should not be allowed to transfer on behalf of a user. There is no restriction in sending NFTs to them. Also, there is no sense in transferring to them directly (nor there is any reason for a user to do so, as he would effectively give up his NFTs) also, doing this check on every transfer will cause unnecessary gas usage for a particular case.

QSP-8 Solidity Version `0.8.13` is Bugged

Severity: *Low Risk*

Status: Fixed

File(s) affected: `GenericRaffle.sol`

Description: Solidity versions `0.8.13` and `0.8.14` contain important bugs, that if exploited, could result in serious issues. These bugs were fixed in `0.8.15`.

Recommendation: We recommend upgrading the compiler version to `0.8.15` or above.

Update: The client fixed the issue in `41282f661da63e61c1803231b2fb8bcf9ec5897a`.

QSP-9 Unable to Access Operator Registry After Update if Ownership Is Undefined

Severity: *Informational*

Status: Acknowledged

File(s) affected: `UpdatableOperatorFiltererUpgradeable.sol`

Description: Contracts inheriting `UpdatableOperatorFiltererUpgradeable` can update `operatorFilterRegistry` address through `updateOperatorFilterRegistryAddress()`. Since ownership implementation is up to inheriting contract, the ownership may be undefined. If ownership is undefined, after updating to a new registry, `operatorFilterRegistry` functions with the modifier `onlyAddressOrOwner` become inaccessible.
The UpdatableOperatorFilterer is not limited to GenericRaffle usage only. The lack of ownership is a concern if another contract were to inherit `UpdatableOperatorFiltererUpgradeable`.

Recommendation: Consider confirming ownership before proceeding with registry updates.

Update: The client acknowledged the issue and provided the following explanation:

> The used code that works with the Operator Filter Registry is considered by us as temporary until OpenSea will accept our PR, the code itself was derived from their already existing functions. We do not intend to add any extension to this part of the code for the time being. Regarding the mentioned issue; although there is a possibility for an undefined Owner to exist in implementation, but in that case, there is no way to call updateOperatorFilterRegistryAddress which would result in an update of the registry. Thus there can not be an update if the contract is having undefined ownership.

QSP-10 Pseudorandom Values Are Known Ahead of Time

Severity: *Informational*

Status: Acknowledged

File(s) affected: `GenericRaffle.sol`

Description: Within `getRandomToken()`, a "random" token ID is generated using the hash of the wallet address, block difficulty, block timestamp, and the number of remaining tokens. Block proposers can manipulate the values of block difficulty, block timestamp, and remaining tokens to alter the token ID. Given that a particular token ID is known to be valuable, block proposers can manipulate the aforementioned variables to obtain a desired token ID.
Note that `block.difficulty` invokes the `PREVRANDAO` opcode, which produces a known value generated by the consensus layer in the preceding epoch. Therefore, the output integer `rand` is predictable.

Recommendation: Consider using `Chainlink's Verifiably Random Function` (VRF) as a source of randomness instead of generating a random value in `GenericRaffle`. If this is not an option, consider using `PREVRANDAO` with a look-ahead period (four epochs are sufficient in most cases).

Update: The client acknowledged the issue and provided the following explanation:

> We acknowledge that this may be an issue but consider that our use for randomness is not that critical. Although a malicious minor/actor X can target a specific token to have at mint, they do not know the rarity of such a mint. The cost for such an attack would, regardless, outweigh any benefit gained from mining a specific token ID.

QSP-11 Storage Collision when Updating Existing `GenericRaffle` Implementation Contract

Severity: Informational

Status: Acknowledged

File(s) affected: `GenericRaffle.sol`

Description: Since `GenericRaffle` is an upgradeable implementation contract, storage collision can occur when updating existing `GenericRaffle` contracts. The previous implementation of `RaffleContract` did not inherit the `UpdatableOperatorFiltererUpgradeable` contract which has a state variable `operatorFilterRegistry`. Therefore replacing the existing `GenericRaffle` implementation contract can potentially cause storage collision between `UpdatableOperatorFiltererUpgradeable.operatorFilterRegistry` and `GenericRaffle.maxSupply`.

Recommendation: Confirm with the client plans to upgrade existing `GenericRaffle` implementation contracts. If upgrades are planned, the contract inheritance should be revised to the following:

```
contract GenericRaffleV2 is
    UpdatableOperatorFiltererUpgradeable,
    GenericRaffle
```

Update: The client clarified that the new `GenericRaffle` contract applies only to new contracts. The contracts already deployed are eligible for royalties since the contracts have been deployed before the deadline given by OpenSea. This issue has been downgraded to informational since the client does not intend to upgrade existing `GenericRaffle` contracts.
The client has acknowledged the issue and provided the following explanation:

> We have already stated that we do not intend to upgrade old contracts. No further action is to be taken.

QSP-12 Certain State Changes Lack Corresponding Events

Severity: Informational

Status: Mitigated

File(s) affected: `UpdatableOperatorFiltererUpgradeable.sol`, `GenericRaffle.sol`

Description: The emission of events is useful for monitoring state changes and incidents.

Recommendation: Consider adding an event for `UpdatableOperatorFiltererUpgradeable.updateOperatorFilterRegistryAddress()`. Also, consider adding information to `GenericRaffle.ContractInitialized()`.

Update: The client fixed `GenericRaffle` in `7e5ee1f0dc53dc3325e790c9d39f79d00e62c077`. The client expanded the event `ContractInitialized` to include the registry address, operator filterer subscription address, and the subscription state.

The client provided the following explanation:

> The mentioned alterations in the `GenericRaffle` contract were made. The used code that works with the Operator Filter Registry is considered by us as temporary until OpenSea will accept our PR, the code itself was derived from their already existing functions. We do not intend to add any extension to this part of the code for the time being.

QSP-13 ERC20 Token Is Caller of `GenericRaffle.mint()`

Severity: Informational

Status: Acknowledged

Description: When minting from `GenericRaffle`, the caller needs to be the payment token as the `safeTransferFrom()` call uses `_msgSender()` as the ERC20 address to transfer. Using this implementation restricts which ERC20 tokens can be used as payment for minting. The ERC20 payment token implementation must include logic that calls `GenericRaffle.mint()`, otherwise the mint will fail.

Recommendation: Ensure that `allowedToExecuteMint` is set to an ERC20 address with the necessary implementation to call `GenericRaffle.mint()`.

Update: The client has acknowledged the issue and provided the following explanation:

> This is expected behavior and we do not consider this an issue.

QSP-14 Privileged Roles

Severity: Informational

Status: Mitigated

File(s) affected: `GenericRaffle.sol`, `UpdatableOperatorFiltererUpgradeable.sol`

Description: There are a number of functions that can only be called by designated addresses.

- `GenericRaffle.setupWave()` -- callable by `owner`
- `GenericRaffle.mint()` -- callable by `allowedToExecuteMint`
- `UpdatableOperatorFiltererUpgradeable.updateOperatorFilterRegistryAddress()` -- callable by `owner`

Recommendation: Write end-user-facing documentation that describes these permissions.

Update: The client added inline comments in `GenericRaffle` that state the privileged roles of the contract in `329bebe3e3dccac010f57dc41b229ab34a6d2b64`.

QSP-15 Upgradeable Contract Storage Gaps

Severity: Informational

Status: Acknowledged

File(s) affected: `UpdatableOperatorFiltererUpgradeable.sol`

Description: For future update considerations, add storage gaps in upgradeable abstract contracts to avoid storage collisions when introducing new variables. This pattern is used in OpenZeppelin's upgradeable contracts.

Recommendation: Add a storage gap at the end of the upgradeable abstract contract.

Update: The client has acknowledged the issue and provided the following explanation:

> The used code that works with the Operator Filter Registry is considered by us as temporary until OpenSea will accept our PR, the code itself was derived from their already existing functions. We do not intend to add any extension to this part of the code for the time being.

QSP-16 Disable Initializer in Implementation Contract

Severity: Informational

Status: Fixed

File(s) affected: `GenericRaffle.sol`

Description: Implementation contracts are not guaranteed to have their initializer disabled. This makes it possible for a malicious actor to call the `initialize()` function on the implementation contracts, which may impact the proxy.
Using `_disableInitializers()` in the constructor is recommended as a best practice to prevent initialization of the implementation contract itself. For the placement of this function, refer to OpenZeppelin documentation.

Recommendation: Add `_disableInitializers()` to the constructor of upgradeable `GenericRaffle` .

Update: The client added `disableInitializer()` to the constructor of `GenericRaffle`in `2b9bdd8c8b9aea2c8e5dd71afe431201b75b911a`.

QSP-17 Clone-and-Own

Severity: Informational

Status: Acknowledged

File(s) affected: `IOperatorFilterRegistry.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve traceability of the file.

Update: The client has acknowledged the issue and provided the following explanation:

> We agree and are already aware of this, this is why we are pushing the code to the owing project https://github.com/ProjectOpenSea/operator-filter-registry/pull/80 No further alterations will be done to the local version of the registry code.

QSP-18 Unlocked Pragma

Severity: Informational

Status: Fixed

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret to lock the file onto a specific Solidity version.

Update: The client fixed the issue in `41282f661da63e61c1803231b2fb8bcf9ec5897a`.

QSP-19 Lack of Code Documentation

Severity: Undetermined

Status: Fixed

File(s) affected: `GenericRaffle.sol`

Description: There is a general dearth of documentation with regard to the `GenericRaffle` contract. While this may not be the case in the present audit, a lack of documentation may substantially impair developers in implementing functionality as desired and may hamper code reviewers from properly determining the correctness of the code.

Recommendation: Write documentation specifying the functional requirements of `GenericRaffle`.

Update: The client added documentation throughout GenericRaffle.sol such that every function and event have corresponding documentation. The client has addressed the issue in `ec70b5251923a22928885441b3cfdc4b3f475098`, `ec70b5251923a22928885441b3cfdc4b3f475098`, and `fb21338273c29f527978ddeeec02179c93c9bccf`.

Automated Analyses

Slither

We have executed slither and filtered the issues that were reported and incorporated the valid ones in the report.

Notes:

1) We only ran Slither on the `src/solc_0.8.13/*` folder.

2) We only reviewed issues reported for the files under the scope of this audit.

Adherence to Specification

1. The Raffle Smart Contract documentation states that `waveType` can be assigned the value 0 or 1; however, in `setupWave()` the wave type can be assigned a value of 0, 1, or 2.
2. The Raffle Smart Contract documentation states that `setupWave()` should fail when paused. `setupWave()` does not include any logic that checks the state of `paused`.
3. `withdraw()` public payable is not implemented in `GenericRaffle` despite being defined in the Raffle Smart Contract documentation.
4. Add comments describing `UpdatableOperatorFiltererUpgradeable.sol`.
5. Add NatSpec comments to all public and external functions in `GenericRaffle.sol`.

Adherence to Best Practices

1. To reduce code duplication in `UpdatableOperatorFiltererUpgradeable.sol`, consider adding an internal function which checks whether the code size of the registry is greater than zero and then checks the registry for whether the operator is allowed (`msg.sender` in the case of modifier `onlyAllowedOperator`, `operator` in the case of modifier `onlyAllowedOperatorApproval`).
2. `GenericRaffle.sol#89` local variable `_trustedForwarder` in `__GenericRaffle_init()` function shadows `ERC2771HandlerUpgradeable._trustedForwarder`; consider renaming the variable to avoid shadow conflicts.
3. Only a small range of values can be assigned to `GenericRaffle.waveType`. Create an enum for these values to improve readability.
4. Set `_tokenId` as `indexed` for the event `Personalized` in `GenericRaffle` to allow for filtering by token ID.
5. Consider changing `GenericRaffle.contractAddress` to a more descriptive name for improved readability.
6. Change the type of `GenericRaffle.paused` from `uint256` to `bool` since `paused` can only be `0` or `1`.
7. Change the type of `GenericRaffle.signatureIds` from `mapping(uint256 => uint256)` to `mapping(uint256 => bool)` since `signatureIds` can only be `0` or `1`.

Test Results

Test Suite Results

The tests for `UpdatableOperatorFilterer.sol` and `IOperatorFilterRegistry.sol` are obtained from the ProjectOpenSea PR project.
The tests used to evaluate `GenericRaffle.sol` are from `PlayboyPartyPeopleV2.test.ts`.

```
Running 6 tests for test/example/upgradeable/UpdatableExampleERC721Upgradeable.t.sol:UpdatableExampleERC721UpgradeableForUpgradableTest
[PASS] testExcludeApprovals() (gas: 127775)
[PASS] testExclusionExceptionDoesNotApplyToOperators() (gas: 152074)
[PASS] testFilter() (gas: 55709)
[PASS] testOwnersNotExcluded() (gas: 125861)
[PASS] testOwnersNotExcludedSafeTransfer() (gas: 164280)
[PASS] testUpgradeable() (gas: 2545451)
Test result: ok. 6 passed; 0 failed; finished in 2.01ms

Running 8 tests for test/UpdatableOperatorFilterer.t.sol:UpdatableOperatorFiltererTest
[PASS] testConstructor_copy() (gas: 361762)
[PASS] testConstructor_subscribe() (gas: 281131)
[PASS] testConstructory_noSubscribeOrCopy() (gas: 405697)
[PASS] testFilter() (gas: 43767)
[PASS] testRegistryNotDeployedDoesNotRevert() (gas: 377615)
[PASS] testUpdateRegistry() (gas: 17623)
[PASS] testUpdateRegistry_onlyOwner() (gas: 12749)
[PASS] testZeroAddressBypass() (gas: 27363)
Test result: ok. 8 passed; 0 failed; finished in 1.41ms

RafflePlayboyPartyPeopleV2
     ✓ should be able to mint with valid signature (3714ms)
     ✓ should be able to mint 1_969 different tokens (36538ms)
     ✓ should be able to mint 1_969 different tokens in 3 waves (35890ms)
    - should be able to mint 1_969 different tokens in 3 waves in 3 txs
     ✓ should be able to personalize with valid signature (64ms)
     ✓ should not be able to personalize with invalid signature (73ms)
     ✓ should be able to differentiate a personalized asset (77ms)
     ✓ should not be able to personalize twice with the same signature (58ms)

  7 passing (1m)
  1 pending
```

Code Coverage

The coverage for `UpdatableOperatorFilterer.sol` was generated using tests from ProjectOpenSea.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
solc_0.8.15/common/BaseWithStorage/ERC2771/	50	25	60	44.44	
ERC2771HandlerUpgradeable.sol	50	25	60	44.44	23,30,39,40,42
solc_0.8.15/common/OperatorFilterer/	13.33	5.56	25	9.52	
IOperatorFilterRegistry.sol	100	100	100	100	
UpdatableOperatorFiltererUpgradeable.sol	13.33	5.56	25	9.52	… 60,68,69,71
solc_0.8.15/raffle/	84.38	55	63.64	84.54	
DanceFight.sol	100	100	100	100	
GenericRaffle.sol	83.7	55	58.62	83.87	… 465,565,574
MadBalls.sol	100	100	100	100	
PlayboyPartyPeopleV2.sol	100	100	100	100	
Rabbids.sol	100	100	100	100	

File	% Lines	% Statements	% Branches	% Funcs
src/upgradeable/ OperatorFiltererUpgradeable.sol	57.14% (4/7)	50.00% (3/6)	12.50% (1/8)	100.00% (1/1)

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
cddcf30859e33e251d6284a04325d3e7e959a92f86776c87299cbe93c379d4d4 ./solc_0.8.13/raffle/GenericRaffle.sol

cc2f7bd9cddd8709a3e5a04f0d0e13f038fd7ab11c6a10150d2828d5531af3cc ./solc_0.8.13/common/OperatorFilterer/IOperatorFilterRegistry.sol

3663562951adb573a89b3dacac62c89b7dccaf1423034441d50d5f3e93084d1f
./solc_0.8.13/common/OperatorFilterer/UpdatableOperatorFiltererUpgradeable.sol
```

Changelog

- 2023-01-12 - Initial report
- 2023-02-02 - Fix review

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT