Smart Contract Audit Report

The Colony – Collectible Drops

Coffee
COFFEEXDEV April 25, 2024

Overview

The following is a review of the Colony Collectibles smart contract system, a series of smart contracts for managing tokenized physical collectibles and collections of collectible drops, to be deployed on the Blast network.

Contracts in scope for this review include:

- `CollectiblesRegistry.sol`
- `DropsRegistry.sol`
- `WrappedDrops.sol`
- `RandomnessProviderGelato.sol`
- `GelatoVRFConsumerBase.sol`
- `TransferMiddleware.sol`

This review was initially based on commit SHA `fd1e6f2474f5c4c4cd3efdb91138b45a5364646e`. It aims to identify security vulnerabilities and general best practices with regards to the contracts in scope. This review should not be considered an endorsement of the project, nor is it a guarantee of security.

Additional changes for remediation were reviewed in commit `200f9786663b093ce162816499d81f454ba9f946` which provided resolution to issues as mentioned in the Status text.

Findings

Minters/Proposers can create drops with 0 tokens.

Severity: Low

`DropsRegistry.commitDrop` does not check that the number of tokens in `drop.TokenIds` is greater than zero. This potentially allows a proposer to create a drop with zero tokens which could have unintended effects or allow a proposer to grief the contract by exhausting the available drop IDs with empty drops.

Recommendation: ensure that the number of tokens in a drop is greater than zero.
Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`

Minters/Proposers can add tokens to other drops

Severity: Low

`DropsRegistry.pushTokensToDrop` does not check that the user pushing collectibles to a drop is the original creator of the drop. Because of this, any authorized proposer can grief drop creation by pushing many low value collectibles to someone else's drop or accidentally lose value by adding cards to the wrong drop.

Recommendation: ensure that the user pushing tokens to a drop is the original creator of the drop.
Status: Acknowledged

Duplicated code in closeUnsuccessfulDrop

Severity: Informational
`DropsRegistry.closeUnsuccesfulDrop` checks the same condition twice for a revert.
Recommendation: Remove duplicated code

```
if (block.timestamp < drop.saleEndTime) revert("sale still ongoing");
//TODO: clean
if (block.timestamp < drop.saleEndTime) {
    _revert(RefundNotAvailable.selector);
}
```

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`

Users can burn and refund any wrapped drop tokens after any unsuccessful drop.

Severity: Critical
`DropsRegistry.claimRefund` does not check the range of tokens being refunded belong to the drop. This allows anyone to trigger the burning and refunding of any tokens from any drop as soon as any unsuccessful drop has occurred. This can result in inconsistent state, loss of funds, and loss of assets for users.

Example:
Drop 1 is unsuccessful.
Drop 2 is ongoing.

- Malicious actor calls claimRefund and passes drop id 1 but wrappedTokenIds belonging to drop 2. Tokens from drop 2 are burned and refunded at the price of tokens from drop 1. This is especially problematic if drop 1 was more expensive than drop 2 as users could immediately claim a refund for more than they paid.

Recommendation: Ensure that each of the wrappedTokenIds passed to claimRefund are within the scope of the drop being refunded.

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`

Usage of abi.encodePacked with multiple dynamic sized elements.

Severity: Low

`abi.encodePacked` should not be used with multiple dynamic sized input elements. If you use `keccak256(abi.encodePacked(a, b))` and both `a` and `b` are dynamic types, it is easy to craft collisions in the hash value by moving parts of `a` into `b` and vice-versa. More specifically, `abi.encodePacked("a", "bc") == abi.encodePacked("ab", "c")`. If you use `abi.encodePacked` for signatures, authentication or data integrity, make sure to always use the same types and check that at most one of them is dynamic. Unless there is a compelling reason, `abi.encode` should be preferred.
Usages:
`CollectiblesRegistry.getTokenizeDigest`
`DropsRegistry.swap`

Recommendation: While these do not appear to be exploitable, it is still recommended to prefer `abi.encode` over `abi.encodePacked` as a best practice.

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`. Code was updated to check for equal array lengths so collision is not possible.

Inconsistent variable naming and redundant code

Severity: Informational
RandomnessProviderGelato has a public mapping of requestIdtoDropId which maps the VRF request id to the collectible drop id. This mapping is accessed with variables having inconsistent names (requestId and dropId). Upon further investigation, the base contract is using the drop id as the request id so the access is correct with either name but the mapping is redundant.

Recommendation: remove the requestIdToDropId mapping and refactor variable naming to be consistent.
Status: Redundant code Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`. Variable naming Acknowledged

WrappedDrops tokens are never marked as committed

Severity: Informational/Low
Code is in place in the WrappedDrops contract to prevent transfer/trade of the wrapped drops tokens while they are marked as a status of “committed” but this status is never updated on the token. WrappedDrops are always in the initial status of “none”
If the intent is to prevent trading of the tokens while a drop is ongoing, this code branch will never be hit.

Recommendation: If needed, ensure that the `extraData.status` field is updated to

properly set the tokens to committed. If unneeded, remove the `extraData.status` field and unnecessary code in `WrappedDrops._update`.

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`.
Code in transfer was deemed unnecessary and was removed.

State mutability warnings

Severity: Informational
Adjust state mutability per compiler warnings.

- `CollectiblesRegistry._startTokenId()` can be marked pure
- `WrappedDrops._onlyDropsRegistry()` can be marked view
- `DropsRegistry._onlyProposer()` can be marked view
- `DropsRegistry._checkExchangeAndReturnValues()` can be marked view

Recommendation: Adjust state mutability to always use the most restrictive mutability option.

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`

Inconsistent naming of internal functions

Severity: Informational
The following functions are marked internal and do not follow the underscore prefix for internal function names followed throughout the rest of the codebase:

- `DropsRegistry.uint256ArrayToUint48Array`
- `TransferMiddleware.splitSignature`
- `WrappedDrops.assignDropToToken`

Recommendation: Prefix all internal function names with an underscore (_)

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`

Remove unused variables/parameters

Severity: Informational
The following variables or parameters are unused and can be removed:

- `WrappedDrops.unrevealedTokensOfDrop`: `_dropsRegistry` local variable
- `DropsRegistry._executeSwapAndGetLeaves`: `breakdownRoot` parameter
- `RandomnessProviderGelato._fulfillRandomness`: `dropId` local variable

Recommendation: ensure disuse of variables/parameters is correct and remove if unnecessary.

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`

Unused struct parameter

Severity: Informational

`WrappedDrops.UserDropStats.numPurchased` is never used (this is tracked on DropsRegistry).

Recommendation: Remove `UserDropStats` struct from `WrappedDrops` and only track balance in drop if needed, or properly track number purchased if needed on `WrappedDrops`.

Status: Resolved in commit `200f9786663b093ce162816499d81f454ba9f946`