EARN’M DropBox Audit Report

Prepared by Cyfrin
Version 2.0

Lead Auditor
Dacian

August 15, 2024

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

EARN'M's `DropBox` protocol is a new product similar to the previous `MysteryBox` product which was also audited by Cyfrin.

`DropBox` is a rewards program where users receive reward codes which are used to claim `ERC721` drop boxes of various tiers. Each tier allows users to redeem their drop boxes for a specific amount of `EARNM ERC20` tokens associated with that tier; the longer a user waits to redeem the more tokens they get for their drop box.

To ensure fairness and transparency in the distribution of box tiers the protocol integrates with Chainlink's Verifiable Random Function (VRF) API v2.5.

Outside the protocol users are generally free to trade `ERC721` drop boxes amongst eachother or on NFT marketplaces, however the `NFT` is burned in the same transaction at which the `EARNM` token redemption occurs. This prevents malicious sellers from defrauding a buyer of their drop box by redeeming the `EARNM` tokens associated with a box before transferring ownership of that box to the buyer.

The protocol also integrates with Limit Break's Creator Token Standards (CTS) library to enable protocol admins to configure royalty payments on NFT marketplaces and fine-grained access control for NFT transfers.

Centralization Risks

While the `DropBox` contract is immutable, the protocol does contain admin functionality which allows protocol admins to:

- associate codes with an address and a specific amount of boxes to receive
- remove unused codes from an address
- disable the ability for users to use codes to claim their drop box `NFT`
- disable the ability for users to use their drop box `NFT` to redeem `EARNM ERC20` tokens
- update the Chainlink VRF handler address
- update fees to be paid by users when claiming their `NFT` and using it to redeem `EARNM` tokens

- update default and per-`NFT` royalty fees
- via Limit Break's CTS APIs, implement fine-grained access control for transfers of drop box `NFTs`

Threat Model

User interaction with the protocol is limited to:

- using associated codes to claim drop box `NFTs`
- redeeming the claimed `NFTs` to receive `EARNM ERC20` tokens

Both claiming and redeeming are protected from re-entrancy using the `nonReentrant` modifier. While users are free to transfer their `NFTs`, they are always burned when claiming `EARNM` tokens which prevents malicious users from selling worthless `NFTs`. Additionally the protocol contains many safety checks which prevent attack vectors such as codes being used multiple times or using non-existent `NFTs` to redeem `EARNM` tokens.

The one major risk to the protocol is if there would exist an error that would prevent users from claiming their drop boxes and/or using their claimed drop boxes to redeem `EARNM` tokens; the examination of this risk factor led to finding `C-1`.

5 Audit Scope

The following contracts together with associated interfaces and dependencies were included in the scope for this audit:

```
src/DropBox.sol
src/DropBoxFractalProtocol.sol
```

To mitigate finding `M-2` a new contract was added which was also audited:

```
src/VRFHandler.sol
```

The audit was conducted using a "collaborative agile" methodology with the client fixing issues soon after they were found and the auditor daily updating to the latest code version.

6 Executive Summary

Over the course of 13 days, the Cyfrin team conducted an audit on the EARN'M DropBox smart contracts provided by EARN'M. In this period, a total of 37 issues were found.

The findings consist of 1 Critical, 2 Medium and 2 Low severity issues with the remainder being informational and gas optimizations.

The one major Critical finding was due to a couple of excessive safety checks, some drop boxes would be impossible to claim and hence users could never redeem the `EARNM ERC20` tokens associated with them.

Additionally Cyfrin provided significant advice on how to refactor the immutable `DropBox` contract to insulate it from future breaking Chainlink `VRF` changes, reduce code complexity, reduce the storage slot requirements from 46 to 33 and eliminate many storage read/write operations; these findings resulted in significant refactoring of the `DropBox` contract and the introduction of a new `VRFHandler` contract.

As part of the audit Cyfrin also created a comprehensive invariant fuzz testing suite using both Medusa & Foundry fuzzers which was provided as an additional deliverable.

Summary

Project Name	EARN'M DropBox
Repository	dropbox-smart-contracts
Commit	f1d6baf1646b...
Audit Timeline	July 29th - Aug 14th 2024
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	1
High Risk	0
Medium Risk	2
Low Risk	2
Informational	7
Gas Optimizations	25
Total Issues	37

Summary of Findings

[C-1] Impossible to reveal any drop boxes linked to codes once enough codes have been associated such that `remainingBoxesAmount == 0`	Resolved
[M-1] `DropBox::tokenURI` returns data for non-existent `tokenId` violating `EIP721`	Resolved
[M-2] Use upgradeable or replaceable `VRFHandler` contracts when interacting with Chainlink VRF	Resolved
[L-1] Protocol can collect less or more fees than expected due to constant `claimFee` per transaction regardless of how many boxes are claimed	Acknowledged
[L-2] Use inheritance order from most "base-like" to "most-derived" so `super` will call correct parent function in `DropBox::supportsInterface`	Resolved
[I-1] Resolve discrepancy between expected `randomNumber` in `DropBox::_assignTierAndMint` and `DropBoxFractalProtocol::_determineTier`	Resolved
[I-2] Avoid floating pragma unless creating libraries	Resolved
[I-3] Refactor duplicate fee sending code into one private function	Resolved
[I-4] Use constant `TIER_IDS_LENGTH` in constructor inputs	Resolved
[I-5] `VRFHandler` shouldn't inherit from `ConfirmedOwner` since it inherits from `VRFConsumerBaseV2Plus` which already inherits from `ConfirmedOwner`	Resolved
[I-6] Use named imports instead of importing the entire namespace	Resolved
[I-7] Use named `mapping` parameters	Resolved
[G-01] Don't initialize to default values	Resolved

[G-02] Cache storage variables when same values read multiple times	Resolved
[G-03] Remove `amountToClaim > 0` check in `DropBox::claimDropBoxes` as previously reverted if `amountToClaim == 0`	Resolved
[G-04] Remove `< 0` checks for unsigned variables since they can't be negative	Resolved
[G-05] Initialize `DropBox::boxIdCounter` to 1 avoiding default initialization to 0	Resolved
[G-06] Enforce ascending order for boxIds to implement more efficient duplicate prevention in `DropBox::claimDropBoxes`	Resolved
[G-07] Prefer `calldata` to `memory` for external function parameters	Resolved
[G-08] Prefer assignment to named return variables and remove explicit `return` statements	Resolved
[G-09] Change `Box` struct to use `uint128` saves 1 storage slot per `Box`	Resolved
[G-10] Use `else if` for sequential `if` statements to prevent subsequent checks when a previous check is `true`	Resolved
[G-11] Remove call to `_requireCallerIsContractOwner` from `DropBox::setDefaultRoyalty` and `setTokenRoyalty` since they already have the `onlyOwner` modifier	Resolved
[G-12] Bypass first `for` loop in `DropBoxFractalProtocol::_determineTier` when `randomNumber >= TIER_1_MAX_BOXES`	Resolved
[G-13] Remove redundant `tierId` validity check from `DropBox::claimDropBoxes`	Resolved
[G-14] Remove redundant `balance > 0` check from `DropBox::claimDropBoxes`	Resolved
[G-15] Simplify checks to fail faster and remove negation operator on final boolean result	Resolved
[G-16] Use a simplified and more efficient implementation for `DropBoxFractalProtocol::_determineTier`	Resolved
[G-17] Only read and write once for storage locations `boxIdCounter` and `mintedTierAmount` in `DropBox::_assignTierAndMint`	Resolved
[G-18] Cache result of `_getOneTimeCodeHash` to prevent repeated identical calculation in `DropBox::associateOneTimeCodeToAddress`	Resolved
[G-19] Remove second `for` loop in `DropBox::claimDropBoxes` and reduce 1 storage read per deleted box	Resolved
[G-20] Only read and write storage location `unclaimedTierAmount` once per changed tier in `DropBox::claimDropBoxes`	Resolved
[G-21] Only read and write storage location `unclaimedTierAmount` once per changed tier in `DropBox::_assignTierAndMint`	Resolved
[G-22] Use `uint32` for more efficient storage packing when tracking minted and unclaimed tier box amounts	Resolved
[G-23] Use `uint128` for `revealFee` and `claimFee` for more efficient storage packing	Resolved
[G-24] Remove redundant check from `VRFHandler::constructor`	Resolved

[G-25] Delete fulfilled vrf request from `vrfRequestIdToRequester` in `VRFHandler::fulfillRandomWords`	Resolved

7 Findings

7.1 Critical Risk

7.1.1 Impossible to reveal any drop boxes linked to codes once enough codes have been associated such that `remainingBoxesAmount == 0`

Description: In `DropBox::associateOneTimeCodeToAddress`, the `boxAmount` linked to allocated codes is deducted from `remainingBoxesAmount` storage:

```
// Decrease the amount of boxes remaining to mint, since they are now associated to be minted
remainingBoxesAmount -= boxAmount;
```

Then in `DropBox::revealDropBoxes` when the user attempts to reveal the boxes previously linked with a code, if all the codes have been previously associated such that `remainingBoxesAmount == 0` the following safety checks cause the transaction to erroneously revert with `NoMoreBoxesToMint` error:

```
// @audit read current `remainingBoxesAmount` from storage
// the box amount linked with this code has already been
// previously deducted in `associateOneTimeCodeToAddress`
uint256 remainingBoxesAmountCache = remainingBoxesAmount;

// @audit since the box amount linked with this code has
// already been deducted from `remainingBoxesAmount`, if
// all codes have been associated, even though not all codes have been
// revealed attempting to reveal any codes will erroneously revert since
// remainingBoxesAmountCache == 0
if (remainingBoxesAmountCache == 0) revert NoMoreBoxesToMint();

// @audit even if the previous check was removed, the transaction would
// still revert here since remainingBoxesAmountCache == 0 but
// oneTimeCodeData.boxAmount > 0
if (remainingBoxesAmountCache < oneTimeCodeData.boxAmount) revert NoMoreBoxesToMint();
```

Impact: Once all codes have been associated it is impossible to use any codes to reveal boxes. This makes it impossible to mint the remaining drop boxes linked to the associated but unrevealed codes and hence makes it impossible to claim the `EARNM` tokens associated with them.

Proof of Concept: Firstly in `test/DropBox/DropBox.t.sol` change the number of boxes such that there is only 1 box:

```
    dropBox = new DropBoxMock(
      address(users.operatorOwner),
      "https://api.example.com/",
      "https://api.example.com/contract",
      address(earnmERC20Token),
      address(users.apiAddress),
      [10_000_000, 1_000_000, 100_000, 10_000, 2500, 750],
-     [1, 10, 100, 1000, 2000, 6889],
+     [uint16(1), 0, 0, 0, 0, 0],
      ["Mythical Box", "Legendary Box", "Epic Box", "Rare Box", "Uncommon Box", "Common Box"],
      address(vrfHandler)
    );
```

Then add the PoC function to `test/DropBox/behaviors/revealDropBoxes.t.sol`:

```
  function test_revealDropBoxes_ImpossibleToMintLastBox() public {
    string memory code = "validCode";
    uint32 boxAmount = 1;
    _setupAndAllowReveal(code, users.stranger, boxAmount);
    _mockVrfFulfillment(code, users.stranger, boxAmount);
    _validateMinting(dropBox.mock_getMintedTierAmounts(), boxAmount, code);
  }
```

Run the PoC with: `forge test --match-test test_revealDropBoxes_ImpossibleToMintLastBox -vvv`.

The PoC stack trace shows it fails to reveal the last and only box due to `[Revert] NoMoreBoxesToMint()` in `DropBoxMock::revealDropBoxes`.

Commenting out the `remainingBoxesAmountCache` checks in `DropBox::revealDropBoxes` then re-running the PoC shows that the last box can now be revealed.

Recommended Mitigation: Remove the `remainingBoxesAmountCache` checks from `DropBox::revealDropBoxes` since the boxes associated with codes are already deducted inside `DropBox::associateOneTimeCodeToAddress`.

Mode: Fixed in commit 974fd2c.

Cyfrin: Verified.

7.2 Medium Risk

7.2.1 `DropBox::tokenURI` returns data for non-existent `tokenId` violating `EIP721`

Description: `DropBox::tokenURI` overrides OpenZeppelin's default implementation with the following code:

```
function tokenURI(uint256 tokenId) public view override returns (string memory) {
  return string(abi.encodePacked(bytes(_baseTokenURI), Strings.toString(tokenId), ".json"));
}
```

This implementation does not check that `tokenId` exists and hence will return data for non-existent `tokenId`. This is in contrast to OpenZeppelin's implementation and contrary to EIP721 which states:

```
/// @notice A distinct Uniform Resource Identifier (URI) for a given asset.
/// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined in RFC
///  3986. The URI may point to a JSON file that conforms to the "ERC721
///  Metadata JSON Schema".
function tokenURI(uint256 _tokenId) external view returns (string);
```

Impact: Apart from simply returning incorrect data, the most likely negative effect is integration problems with NFT marketplaces.

Recommended Mitigation: Revert if `tokenId` does not exist:

```
function tokenURI(uint256 tokenId) public view override returns (string memory) {
+ _requireOwned(tokenId);
  return string(abi.encodePacked(bytes(_baseTokenURI), Strings.toString(tokenId), ".json"));
}
```

Mode: Fixed in commit 2d8c3c0.

Cyfrin: Verified.

7.2.2 Use upgradeable or replaceable `VRFHandler` contracts when interacting with Chainlink VRF

Description: Cyfrin has been made aware by our private audit clients that Chainlink intends on bricking VRF 2.0 at the end of November 2024 such that *"VRF 2.0 will stop working"* even for existing subscriptions.

Impact: All immutable contracts dependent on VRF 2.0 will be bricked when Chainlink bricks VRF 2.0. The same will apply in the future to immutable contracts depending on VRF 2.5 when/if Chainlink does the same to it.

Recommended Mitigation: Immutable contracts should be insulated from directly interacting with Chainlink VRF. One way to achieve this is to create a separate `VRFHandler` contract that acts as a bridge between immutable contracts and Chainlink VRF; this contract should:

- be either a replaceable immutable contract using VRFConsumerBaseV2Plus such that a new `VRFHandler` can be deployed, or an upgradeable contract using VRFConsumerBaseV2Upgradeable
- allow the contract owner to set the address of the immutable contract (and vice versa in the immutable contract to set the address of the `VRF Handler`)
- provide an API to the immutable contract that will not need to change
- handle all the Chainlink VRF API calls and callbacks, passing randomness back to the immutable contract as required

Mode: Fixed in commit dc3412f by implementing a replaceable immutable `VRFHandler` contract to act as a bridge between `DropBox` and Chainlink VRF.

Cyfrin: Verified.

7.3 Low Risk

7.3.1 Protocol can collect less or more fees than expected due to constant `claimFee` per transaction regardless of how many boxes are claimed

Description: `DropBox::claimDropBoxes` allows users to pass an array of `boxIds` to claim per transaction while charging a constant `claimFee` per transaction:

```
// Require that the user sends a fee of "claimFee" amount
if (msg.value != claimFee) revert InvalidClaimFee();
```

This means that (not including gas fees):

- claiming 1 box costs the same as claiming 100 boxes, if done in the 1 transaction
- claiming 100 boxes 1-by-1 costs 100x more than claiming 100 boxes in 1 transaction

Impact: The protocol will collect:

- less fees if users claim batches of boxes
- more fees if users claim boxes one-by-one

Recommended Mitigation: Charge `claimFee` per transaction taking into account the number of boxes being claimed, eg:

```
// Require that the user sends a fee of "claimFee" amount per box
if (msg.value != claimFee * _boxIds.length) revert InvalidClaimFee();
```

Mode: Acknowledged; this was required by the product specification.

7.3.2 Use inheritance order from most "base-like" to "most-derived" so `super` will call correct parent function in `DropBox::supportsInterface`

Description: Use inheritance order from most "base-like" to "most-derived"; this is considered good practice and can be important since Solidity searches for parent functions from right to left.

`DropBox` inherits and overrides the following function from two different parent contracts and calls `super`:

```
  function supportsInterface(bytes4 interfaceId) public view virtual override(ERC721C, ERC2981) returns
  ↪   (bool) {
    return super.supportsInterface(interfaceId);
  }
```

The current ordering is:

```
contract DropBox is OwnableBasic, ERC721C, BasicRoyalties, ReentrancyGuard, DropBoxFractalProtocol,
↪   IVRFHandlerReceiver {
```

As Solidity searches from right to left, this will bypass `ERC721C::supportsInterface` and instead execute `ERC2981::supportsInterface` since `BasicRoyalties` appears after `ERC721C` in the inheritance order.

`ERC2981::supportsInterface` ends up executing `ERC165::supportsInterface` but `ERC721C::supportsInterface` explicitly intends to override `ERC165::supportsInterface`:

```
    /**
     * @notice Indicates whether the contract implements the specified interface.
     * @dev Overrides supportsInterface in ERC165.
     * @param interfaceId The interface id
     * @return true if the contract implements the specified interface, false otherwise
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return
        interfaceId == type(ICreatorToken).interfaceId ||
        interfaceId == type(ICreatorTokenLegacy).interfaceId ||
        super.supportsInterface(interfaceId);
    }
```

Hence due to the current inheritance order `DropBox::supportsInterface` will bypass `ERC721C::supportsInterface` to execute `ERC2981::supportsInterface` and `ERC165::supportsInterface` which appears to be incorrect.

Recommended Mitigation: File: `DropBox.sol`:

```diff
- contract DropBox is OwnableBasic, ERC721C, BasicRoyalties, ReentrancyGuard, DropBoxFractalProtocol,
↪  IVRFHandlerReceiver {
+ contract DropBox is IVRFHandlerReceiver, DropBoxFractalProtocol, ReentrancyGuard, OwnableBasic,
↪  BasicRoyalties, ERC721C {

  function supportsInterface(bytes4 interfaceId) public view virtual override(ERC721C, ERC2981) returns
   ↪  (bool) {
-    return super.supportsInterface(interfaceId);
// @audit make it explicit which parent function to call
+   return ERC721C.supportsInterface(interfaceId);
  }
```

File: `VRFHandler.sol`:

```diff
- contract VRFHandler is VRFConsumerBaseV2Plus, IVRFHandler {
+ contract VRFHandler is IVRFHandler, VRFConsumerBaseV2Plus {
```

Mode: Fixed in commit c66de99.

Cyfrin: Verified.

7.4 Informational

7.4.1 Resolve discrepancy between expected `randomNumber` in `DropBox::_assignTierAndMint` and `DropBoxFractalProtocol::_determineTier`

Description: `DropBox::_assignTierAndMint` generates `randomNumber` between 0 and 9999 and then calls `DropBoxFractalProtocol::_determineTier` passing this as input:

```
// Generate a random number between 0 and 9999
uint256 randomNumber = boxSeed % 10_000;

// Determine the tier of the box based on the random number
uint128 tierId = _determineTier(randomNumber, mintedTierAmountCache);
```

But `DropBoxFractalProtocol::_determineTier` has a comment expecting `randomNumber` to be between 0 and 999_999:

```
/// @notice Function to determine the tier of the box based on the random number.
/// @param randomNumber The random number to use to determine the tier. 0 to 999_999
/// @param mintedTierAmountCache Total cached amount minted for each tier.
/// @return determinedTierId The tier id of the box.
function _determineTier(uint256 randomNumber, ...)
```

Resolve this discrepancy; it appears that the comment in the latter is incorrect.

Mode: Fixed in commit 4e48b15 & 5a3cbf9.

Cyfrin: Verified.

7.4.2 Avoid floating pragma unless creating libraries

Description: Per SWC-103 compiler versions in pragmas should be fixed unless creating libraries. Choose a specific compiler version to use for development, testing and deployment, eg:

```
- pragma solidity ^0.8.19;
+ pragma solidity 0.8.19;
```

Mode: Fixed in commit 668011e.

Cyfrin: Verified.

7.4.3 Refactor duplicate fee sending code into one private function

Description: `DropBox::claimDropBoxes` and `revealDropBoxes` are two `payable` external functions which both implement duplicate fee sending code; refactor this into 1 private function:

```
  function _sendFee(uint256 amount) private {
    bool sent;
    address feesReceiver = feesReceiverAddress;
    // Use low level call to send fee to the fee receiver address
    assembly {
      sent := call(gas(), feesReceiver, amount, 0, 0, 0, 0)
    }
    if (!sent) revert Unauthorized();
  }
```

Then simply call it from both external functions:

```
_sendFee(msg.value);
```

Mode: Fixed in commit 327c12b.

Cyfrin: Verified.

7.4.4 Use constant `TIER_IDS_LENGTH` in constructor inputs

Description: Use constant `TIER_IDS_LENGTH` in constructor inputs:

```
// DropBox:
  constructor(
    uint256 _subscriptionId,
    address _vrfCoordinator,
    address _feesReceiverAddress,
    string memory __baseTokenURI,
    string memory __contractURI,
    address _earmmERC20Token,
    bytes32 _keyHash,
    address _apiAddress,
    uint24[TIER_IDS_LENGTH] memory _tierTokens,
    uint16[TIER_IDS_LENGTH] memory _tierMaxBoxes,
    string[TIER_IDS_LENGTH] memory _tierNames
  )

// DropBoxFractalProtocol:
  constructor(uint24[TIER_IDS_LENGTH] memory tierTokens,
              uint16[TIER_IDS_LENGTH] memory tierMaxBoxes,
              string[TIER_IDS_LENGTH] memory tierNames) {
```

Mode: Fixed in commit d8af391.

Cyfrin: Verified.

7.4.5 `VRFHandler` shouldn't inherit from `ConfirmedOwner` since it inherits from `VRFConsumerBaseV2Plus` which already inherits from `ConfirmedOwner`

Description: `VRFHandler` shouldn't inherit from `ConfirmedOwner` since it inherits from `VRFConsumerBaseV2Plus` which already inherits from `ConfirmedOwner`:

```
- contract VRFHandler is ConfirmedOwner, VRFConsumerBaseV2Plus, IVRFHandler {
+ contract VRFHandler is VRFConsumerBaseV2Plus, IVRFHandler {
```

Mode: Fixed in commit 7fe63a3.

Cyfrin: Verified.

7.4.6 Use named imports instead of importing the entire namespace

Description: Use named imports as they offer a number of advantages compared to importing the entire namespace.

File: `DropBox.sol`

```
// DropBoxFractalProtocol
import {DropBoxFractalProtocol} from "./DropBoxFractalProtocol.sol";

// VRF Handler Interface
```

```
import {IVRFHandler} from "./IVRFHandler.sol";
import {IVRFHandlerReceiver} from "./IVRFHandlerReceiver.sol";

// LimitBreak implementations of ERC721C and Programmable Royalties
import {OwnableBasic, OwnablePermissions} from
↪  "@limitbreak/creator-token-standards/src/access/OwnableBasic.sol";
import {ERC721C, ERC721OpenZeppelin} from "@limitbreak/creator-token-standards/src/erc721c/ERC721C.sol";
import {BasicRoyalties, ERC2981} from
↪  "@limitbreak/creator-token-standards/src/programmable-royalties/BasicRoyalties.sol";

// OpenZeppelin
import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import {Strings} from "@openzeppelin/contracts/utils/Strings.sol";
```

File: `VRFHandler.sol`

```
// VRF Handler Interface
import {IVRFHandler} from "./IVRFHandler.sol";
import {IVRFHandlerReceiver} from "./IVRFHandlerReceiver.sol";

// Chainlink
import {VRFConsumerBaseV2Plus} from "@chainlink/contracts/src/v0.8/vrf/dev/VRFConsumerBaseV2Plus.sol";
import {IVRFCoordinatorV2Plus} from
↪  "@chainlink/contracts/src/v0.8/vrf/dev/interfaces/IVRFCoordinatorV2Plus.sol";
import {VRFV2PlusClient} from "@chainlink/contracts/src/v0.8/vrf/dev/libraries/VRFV2PlusClient.sol";
```

Mode: Fixed in commit 9afe010.

Cyfrin: Verified.

7.4.7 Use named `mapping` parameters

Description: Solidity 0.8.18 introduced named `mapping` parameters; use this feature in `DropBox` for clearer mappings like this:

```
  /// @notice Mappings
  mapping(uint256 boxId => Box boxData) internal boxIdToBox;
  mapping(bytes32 code => bool usedInd) internal oneTimeCodeUsed; // ensure uniqueness of one-time codes
  mapping(bytes32 codeAddressHash => OneTimeCode codeData) internal oneTimeCode; // ensure integrity of
  ↪  one-time code data
  mapping(bytes32 codeAddressHash => uint256[] randomWords) internal oneTimeCodeRandomWords;
  mapping(uint256 vrfRequestId => bytes32 codeAddressHash) internal activeVrfRequests;
  mapping(uint256 vrfRequestId => bool fulfilledInd) internal fulfilledVrfRequests;
```

Mode: Fixed in commit 8f6a168.

Cyfrin: Verified.

7.5 Gas Optimization

7.5.1 Don't initialize to default values

Description: Don't initialize to default values:

File: `src/DropBox.sol`

```
142:    isEarnmClaimAllowed = false;
143:    isRevealAllowed = false;
265:      for (uint32 i = 0; i < boxAmount; i++) {
424:    uint256 amountToClaim = 0;
437:    for (uint256 i = 0; i < _boxIds.length; i++) {
478:    for (uint256 i = 0; i < _boxIds.length; i++) {
537:    for (uint256 i = 0; i < _randomWords.length; i++) {
620:    for (uint32 i = 0; i < boxAmount; i++) {
676:    uint256 totalLiability = 0;
784:    for (uint256 i = 0; i < boxIds.length; i++) {
```

Mode: Fixed in commit 16ecc3b.

Cyfrin: Verified.

7.5.2 Cache storage variables when same values read multiple times

Description: Cache storage variables when same values read multiple times:

File: `src/DropBox.sol`

```
232:    if (remainingBoxesAmount == 0) revert NoMoreBoxesToMint();
235:    if (remainingBoxesAmount < boxAmount) revert NoMoreBoxesToMint();

329:    if (remainingBoxesAmount == 0) revert NoMoreBoxesToMint();
354:    if (remainingBoxesAmount < oneTimeCodeData.boxAmount) revert NoMoreBoxesToMint();

357:    if (oneTimeCodeRandomWords[otpHash].length == 0) revert InvalidVrfState();
360:    uint256[] memory randomWords = oneTimeCodeRandomWords[otpHash];
```

Mode: Fixed in commit 3cfec7f.

Cyfrin: Verified.

7.5.3 Remove `amountToClaim > 0` check in `DropBox::claimDropBoxes` as previously reverted if `amountToClaim == 0`

Description: Remove `amountToClaim > 0` check in `DropBox::claimDropBoxes` as previously reverted if `amountToClaim == 0`:

```
// [Safety check] In case the amount to claim is 0, revert
if (amountToClaim == 0) revert AmountOfEarnmToClaimIsZero();
// -----------------------------------------------------------------------------------------

//
//
//
//  Contract Earnm Balance and Safety Checks
// -----------------------------------------------------------------------------------------
// Get the balance of Earnm ERC20 tokens of the contract
uint256 balance = EARNM.balanceOf(address(this));

// [Safety check] Validate there is more than 0 balance of Earnm ERC20 tokens
// [Safety check] Validate there is enough balance of Earnm ERC20 tokens to claim
// @audit remove `amountToClaim > 0` due to prev check which enforced that
// amountToClaim != 0 and amountToClaim is unsigned so can't be negative
if (!(amountToClaim > 0 && balance > 0 && balance >= amountToClaim)) revert InsufficientEarnmBalance();
```

Mode: Fixed in commit e6231a7.

Cyfrin: Verified.

7.5.4 Remove `< 0` checks for unsigned variables since they can't be negative

Description: Remove `< 0` checks for unsigned variables since they can't be negative:

File: `src/DropBox.sol`

```
538:      if (_randomWords[i] <= 0) revert InvalidRandomWord();
```

Mode: Fixed in commit ccc358a.

Cyfrin: Verified.

7.5.5 Initialize `DropBox::boxIdCounter` to 1 avoiding default initialization to 0

Description: Initialize `DropBox::boxIdCounter` to 1 avoiding default initialization to 0:

```
- uint256 internal boxIdCounter; // Counter for the box ids, starting from 1 (see _assignTierAndMint
↪  function)
+ uint256 internal boxIdCounter = 1; // Counter for the box ids
```

And change this line in `_assignTierAndMint`:

```
- uint256 newBoxId = ++boxIdCounter;
+ uint256 newBoxId = boxIdCounter++;
```

Mode: Fixed in commit 4e2d90b.

Cyfrin: Verified.

7.5.6 Enforce ascending order for boxIds to implement more efficient duplicate prevention in `DropBox::claimDropBoxes`

Description: Enforce ascending order for boxIds to implements more efficient duplicate prevention in `DropBox::claimDropBoxes`:

```
-       for (uint256 j = i + 1; j < _boxIds.length; j++) {
-         if (_boxIds[i] == _boxIds[j]) revert BadRequest();
-       }
+      if(i > 0 && _boxIds[i-1] >= _boxIds[i]) revert BadRequest();
```

Mode: Fixed in commit 756fc82.

Cyfrin: Verified.

7.5.7 Prefer `calldata` to `memory` for external function parameters

Description: Prefer `calldata` to `memory` for external function parameters:

File: `src/DropBox.sol`:

```
158:  function removeOneTimeCodeToAddress(string memory code, address allowedAddress) external
↪  only(apiAddress) {
203:    string memory code,
308:  function revealDropBoxes(string memory code) external payable nonReentrant {
406:  function claimDropBoxes(uint256[] memory _boxIds) external payable nonReentrant {
781:  function getBoxTierAndBlockTsOfIds(uint256[] memory boxIds) external view returns (uint256[3][]
↪  memory) {
798:  function getOneTimeCodeData(address sender, string memory code) external view returns
↪  (OneTimeCode memory) {
805:  function getOneTimeCodeToAddress(address sender, string memory code) external view returns
↪  (address) {
929:  function setContractURI(string memory __contractURI) external onlyOwner {
939:  function setBaseTokenURI(string memory __baseTokenURI) external onlyOwner {
```

Mode: Fixed in commit 9776bc3 & b0cc24a.

Cyfrin: Verified.

7.5.8 Prefer assignment to named return variables and remove explicit `return` statements

Description: Prefer assignment to named return variables and remove explicit return statements.

This applies to most of the codebase however we provide one example of how `DropBoxFractalProtocol::_calculateAmountToClaim` could be refactored to remove the `tokens` variable and `return` statement by using a named return variable:

```
  /// @return tokens The amount of Earnm tokens to claim.
  function _calculateAmountToClaim(uint256 tier, uint256 daysPassed) internal view returns (uint256
  ↪  tokens) {
      if (tier == 6) tokens = TIER_6_TOKENS;
      if (tier == 5) tokens = TIER_5_TOKENS;
      if (tier == 4) tokens = TIER_4_TOKENS;
      if (tier == 3) tokens = TIER_3_TOKENS;
      if (tier == 2) tokens = TIER_2_TOKENS;
      if (tier == 1) tokens = TIER_1_TOKENS;

      // The maximum amount of tokens to claim is the total amount of tokens
      // Which is obtained after 4 years
      if (daysPassed > FOUR_YEARS_IN_DAYS) {
          tokens *= (10 ** EARNM_DECIMALS);
      }
      else {
          tokens = (tokens * (10 ** EARNM_DECIMALS)) * daysPassed / FOUR_YEARS_IN_DAYS;
      }
  }
```

Mode: Fixed in commit 88c5f55.

Cyfrin: Verified.

7.5.9 Change `Box` struct to use `uint128` saves 1 storage slot per `Box`

Description: The current `Box` struct looks like this:

```
struct Box {
  uint256 blockTs;
  uint256 tier;
}
```

This will require 2 storage slots for each `Box`. However neither `blockTs` nor `tier` require `uint256`; both can be changed to `uint128` which will pack each `Box` into 1 storage slot:

```
struct Box {
  uint128 blockTs;
  uint128 tier;
}
```

This will halve the number of storage reads/writes when reading/writing boxes from/to storage.

Mode: Fixed in commit 88c5f55.

Cyfrin: Verified.

7.5.10 Use `else if` for sequential `if` statements to prevent subsequent checks when a previous check is `true`

Description: Use `else if` for sequential `if` statements to prevent subsequent checks when a previous check is `true`. For example `DropBoxFractalProtocol::_determineTier` has this code:

```
if (tierId == 6) maxBoxesPerTier = TIER_6_MAX_BOXES;
if (tierId == 5) maxBoxesPerTier = TIER_5_MAX_BOXES;
if (tierId == 4) maxBoxesPerTier = TIER_4_MAX_BOXES;
if (tierId == 3) maxBoxesPerTier = TIER_3_MAX_BOXES;
if (tierId == 2) maxBoxesPerTier = TIER_2_MAX_BOXES;
if (tierId == 1) maxBoxesPerTier = TIER_1_MAX_BOXES;
```

Here even if the first condition is true, all the other `if` statements will still be executed even though they can't be true. Refactor to:

```
if (tierId == 6) maxBoxesPerTier = TIER_6_MAX_BOXES;
else if (tierId == 5) maxBoxesPerTier = TIER_5_MAX_BOXES;
else if (tierId == 4) maxBoxesPerTier = TIER_4_MAX_BOXES;
else if (tierId == 3) maxBoxesPerTier = TIER_3_MAX_BOXES;
else if (tierId == 2) maxBoxesPerTier = TIER_2_MAX_BOXES;
else if (tierId == 1) maxBoxesPerTier = TIER_1_MAX_BOXES;
```

The same issue also applies in a second loop in the same function though the ordering of checks is different and to `DropBoxFractalProtocol::_calculateAmountToClaim`.

Mode: Fixed in commit fc8466e.

Cyfrin: Verified.

7.5.11 Remove call to `_requireCallerIsContractOwner` from `DropBox::setDefaultRoyalty` and `setTokenRoyalty` since they already have the `onlyOwner` modifier

Description: Remove call to `_requireCallerIsContractOwner` from `DropBox::setDefaultRoyalty` and `setTokenRoyalty` since they already have the `onlyOwner` modifier:

```
function setDefaultRoyalty(address receiver, uint96 feeNumerator) external onlyOwner {
-  _requireCallerIsContractOwner();
```

```
  _setDefaultRoyalty(receiver, feeNumerator);
}

function setTokenRoyalty(uint256 tokenId, address receiver, uint96 feeNumerator) external onlyOwner {
-  _requireCallerIsContractOwner();
  _setTokenRoyalty(tokenId, receiver, feeNumerator);
}
```

Mode: Fixed in commit 2939602.

Cyfrin: Verified.

7.5.12 Bypass first `for` loop in `DropBoxFractalProtocol::_determineTier` when `randomNumber >= TIER_1_MAX_BOXES`

Description: Tier 1 is the lowest tier with the most boxes and this condition inside the first `for` loop ensures the first `for` loop will always fails to allocate a `tierId` when `randomNumber >= TIER_1_MAX_BOXES`:

```
// Check if the randomNumber falls within the current tier's range and hasn't exceeded the mint limit.
if ((randomNumber < maxBoxesPerTier) && hasCapacity) return tierId;
```

Hence there is no point entering the first `for` loop when `randomNumber >= TIER_1_MAX_BOXES`; skip the first `for` loop and go straight to the second one:

```
  function _determineTier(
    uint256 randomNumber,
    uint256[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache
  )
    internal
    view
    returns (uint128 /* tierId, function returns before the end of the execution */ )
  {
-     // Iterate backwards from the highest tier to the lowest.
-     for (uint128 tierId = TIER_IDS_LENGTH; tierId > 0; tierId--) {

+     // impossible for first loop to allocate tierId when randomNumber is
+     // >= lowest tier max boxes, so in that case skip to second loop
+     if(randomNumber < TIER_1_MAX_BOXES) {
+       // Iterate backwards from the highest tier to the lowest.
+       for (uint128 tierId = TIER_IDS_LENGTH; tierId > 0; tierId--) {
```

Mode: Fixed in commit a3d7641.

Cyfrin: Verified.

7.5.13 Remove redundant `tierId` validity check from `DropBox::claimDropBoxes`

Description: `DropBox::claimDropBoxes` performs this check before calling `DropBoxFractalProtocol::_calculateVestingPeriodPerBox`:

```
// [Safety check] Validate that the box tier is valid
if (!(box.tier > 0 && box.tier <= TIER_IDS_LENGTH)) revert InvalidTierId();
```

However `DropBoxFractalProtocol::_calculateVestingPeriodPerBox` performs the same check, hence this check is redundant and should be removed from `DropBox::claimDropBoxes`.

Mode: Fixed in commit 7ebd568.

Cyfrin: Verified.

7.5.14 Remove redundant `balance > 0` check from `DropBox::claimDropBoxes`

Description: In `DropBox::claimDropBoxes` there is this check:

```
if (!(balance > 0 && balance >= amountToClaim)) revert InsufficientEarnmBalance();
```

However the `balance > 0` component is redundant since the function already reverted if `amountToClaim == 0`, hence the check can be simplified to:

```
if (!(balance >= amountToClaim)) revert InsufficientEarnmBalance();
```

This can be further simplified by removing the need for the `!` negation operation to:

```
if (balance < amountToClaim) revert InsufficientEarnmBalance();
```

Mode: Fixed in commit a1e6435.

Cyfrin: Verified.

7.5.15 Simplify checks to fail faster and remove negation operator on final boolean result

Description: Many checks in the code use a pattern of:

```
if( !(check_1 && check_2 && check_3) ) revert Error();
```

Often these checks can be simplified with improved efficiency by re-writting to:

- use `||` instead of `&&` which makes the check fail sooner, avoiding evaluation of subsequent components
- removes the need for the final `!` negation operator

File: `src/DropBox.sol`

```
// L218
- if (!(boxAmount > 0 && boxAmount <= MAX_BOX_AMOUNT_PER_REVEAL)) revert InvalidBoxAmount();
+ if(boxAmount == 0 || boxAmount > MAX_BOX_AMOUNT_PER_REVEAL) revert InvalidBoxAmount();

// L347
- if (!(oneTimeCodeData.boxAmount > 0 && oneTimeCodeData.boxAmount <= MAX_BOX_AMOUNT_PER_REVEAL)) {
+ if (oneTimeCodeData.boxAmount == 0 || oneTimeCodeData.boxAmount > MAX_BOX_AMOUNT_PER_REVEAL) {

// L417
- if (!(_boxIds.length > 0)) revert Unauthorized();
+ if (boxIds.length == 0) revert Unauthorized();

// L442
- if (!(ownerOf(_boxIds[i]) == msg.sender)) revert Unauthorized();
+ if (ownerOf(_boxIds[i]) != msg.sender) revert Unauthorized();
```

File: `src/DropBoxFractalProtocol.sol`

```
// L170
- if (!(boxTierId > 0 && boxTierId <= tierIdsLength)) revert InvalidTierId();
+ if (boxTierId == 0 || boxTierId > tierIdsLength) revert InvalidTierId();
```

```
// L173
- if (!(boxMintedTimestamp <= block.timestamp && boxMintedTimestamp > 0)) revert
↪  InvalidBlockTimestamp();
+ if (boxMintedTimestamp > block.timestamp || boxMintedTimestamp == 0) revert InvalidBlockTimestamp();
```

Mode: Fixed in commit 96773b9.

Cyfrin: Verified.

7.5.16 Use a simplified and more efficient implementation for `DropBoxFractalProtocol::_determineTier`

Description: Use a simplified and more efficient implementation for `DropBoxFractalProtocol::_determineTier` such as:

```
function _determineTier(
  uint256 randomNumber,
  uint256[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache
)
  internal
  view
  returns (uint128 /* tierId, function returns before the end of the execution */ )
{
  // if the randomNumber is smaller than TIER_1_MAX_BOXES, first attempt
  // tier selection which prioritizes the most valuable tiers for selection where:
  // 1) randomNumber falls within one or more tiers AND
  // 2) the tier(s) have not been exhausted
  if(randomNumber < TIER_1_MAX_BOXES) {
    if(randomNumber < TIER_6_MAX_BOXES && mintedTierAmountCache[6] < TIER_6_MAX_BOXES) return 6;
    if(randomNumber < TIER_5_MAX_BOXES && mintedTierAmountCache[5] < TIER_5_MAX_BOXES) return 5;
    if(randomNumber < TIER_4_MAX_BOXES && mintedTierAmountCache[4] < TIER_4_MAX_BOXES) return 4;
    if(randomNumber < TIER_3_MAX_BOXES && mintedTierAmountCache[3] < TIER_3_MAX_BOXES) return 3;
    if(randomNumber < TIER_2_MAX_BOXES && mintedTierAmountCache[2] < TIER_2_MAX_BOXES) return 2;
    if(randomNumber < TIER_1_MAX_BOXES && mintedTierAmountCache[1] < TIER_1_MAX_BOXES) return 1;
  }

  // if we get here it means that either:
  // 1) randomNumber >= TIER_1_MAX_BOXES OR
  // 2) the tier(s) randomNumber fell into had been exhausted
  //
  // in this case we attempt to allocate based on what is available
  // prioritizing from the least valuable tiers
  if(mintedTierAmountCache[1] < TIER_1_MAX_BOXES) return 1;
  if(mintedTierAmountCache[2] < TIER_2_MAX_BOXES) return 2;
  if(mintedTierAmountCache[3] < TIER_3_MAX_BOXES) return 3;
  if(mintedTierAmountCache[4] < TIER_4_MAX_BOXES) return 4;
  if(mintedTierAmountCache[5] < TIER_5_MAX_BOXES) return 5;
  if(mintedTierAmountCache[6] < TIER_6_MAX_BOXES) return 6;

  // if we get here it means that all tiers have been exhausted
  revert NoMoreBoxesToMint();
}
```

Mode: Fixed in commit 0e3410b.

Cyfrin: Verified.

7.5.17 Only read and write once for storage locations `boxIdCounter` and `mintedTierAmount` in `DropBox::_assignTierAndMint`

Description: In `DropBox::_assignTierAndMint`, `boxIdCounter` and `mintedTierAmount` can be cached before the loop, use the cached version during the loop and finally update once at the end:

```
function _assignTierAndMint(
  uint32 boxAmount,
  uint256[] memory randomNumbers,
  address claimer,
  bytes memory code
)
  internal
{
  uint256[] memory boxIdsToEmit = new uint256[](boxAmount);

  // [Gas] Cache mintedTierAmount, update cache during the loop and record
  // which tiers were updated. At the end of the loop write once to storage
  // only for tiers which were updated. Also prevents `_determineTier` from
  // re-reading storage multiple times
  bool[TIER_IDS_ARRAY_LEN] memory tierChangedInd;
  uint256[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache = mintedTierAmount;

  // [Gas] cache the current boxId, use cache during the loop then write once
  // at the end; this results in only 1 storage read and 1 storage write for `boxIdCounter`
  uint256 boxIdCounterCache = boxIdCounter;

  // For each box to mint
  for (uint32 i; i < boxAmount; i++) {
    // Generate a new box id, starting from 1
    uint256 newBoxId = boxIdCounterCache++;

    // The box seed is directly derived from the random number without any
    // additional values like the box id, the claimer address, or the block timestamp.
    // This is made like this to ensure the tier assignation of the box is purely random
    // and the tier that the box is assigned to, won't change after the randomness is fulfilled.
    // The only way to predict the seed is to know the random number.
    // Random number validation is done in the _fulfillRandomWords function.
    uint256 boxSeed = uint256(keccak256(abi.encode(randomNumbers[i])));

    // Generate a random number between 0 and TOTAL_MAX_BOXES - 1 using the pure random seed
    // Modulo ensures that the random number is within the range of [0, TOTAL_MAX_BOXES) (upper bound
    ↪  exclusive),
    // providing a uniform distribution across the possible range. This ensures that each tier has a
    // probability of being selected proportional to its maximum box count. The randomness is
    ↪  normalized
    // to fit within the predefined total boxes, maintaining the intended probability distribution
    ↪  for each tier.
    //
    // Example with the following setup:
    // _tierMaxBoxes -> [1, 10, 100, 1000, 2000, 6889]
    // _tierNames -> ["Mythical Box", "Legendary Box", "Epic Box", "Rare Box", "Uncommon Box",
    ↪  "Common Box"]
    //
    // The `TOTAL_MAX_BOXES` is the sum of `_tierMaxBoxes`, which is 10_000.
    // The probability distribution is as follows:
    //
    // Mythical Box:   1    / 10_000  chance.
    // Legendary Box:  10   / 10_000  chance.
    // Epic Box:       100  / 10_000  chance.
    // Rare Box:       1000 / 10_000  chance.
    // Uncommon Box:   2000 / 10_000  chance.
```

```
      // Common Box:     6889 / 10_000  chance.
      //
      // By using the modulo operation with TOTAL_MAX_BOXES, the random number's probability
      // to land in any given tier is determined by the number of max boxes available for that tier
      uint256 randomNumber = boxSeed % TOTAL_MAX_BOXES;

      // Determine the tier of the box based on the random number
      uint128 tierId = _determineTier(randomNumber, mintedTierAmountCache);

      // Increment the cached amount for this tier and mark this tier as changed
      ++mintedTierAmountCache[tierId];
      tierChangedInd[tierId] = true;

      // [Memory] Add the box id to the boxes ids array
      boxIdsToEmit[i] = newBoxId;

      // Save the box to the boxes mappings
      _saveBox(newBoxId, tierId);

      // Mint the box; explicitly not using `_safeMint` to prevent
      // re-entrancy issues
      _mint(claimer, newBoxId);
    }

    // update storage for boxIdCounter
    boxIdCounter = boxIdCounterCache;

    // update storage for mintedTierAmount only for tiers which were changed
    // hence each tier from mintedTierAmount storage is read only once and only
    // tiers which changed are written to storage only once
    for(uint128 tierId = 1; tierId<TIER_IDS_ARRAY_LEN; tierId++) {
      if(tierChangedInd[tierId]) {
        mintedTierAmount[tierId] = mintedTierAmountCache[tierId];
      }
    }

    // Emit an event indicating the boxes have been minted
    emit BoxesMinted(claimer, boxIdsToEmit, code);
  }
```

The supplied code also has a couple of other useful changes inside the loop:

- `mintedTierAmountCache` and `boxIdsToEmit` are changed before calling `_saveBox` and `_mint` (Effects before Interactions)
- comment added to note that `_mint` is being explicitly used instead of `_safeMint`

We also found that the unit testing around `mintedTierAmount` was lacking in that we could comment out lines of code and all the tests would still pass, so we added more unit tests around this area:

File: `test/mocks/DropBoxMock.sol`

```
// couple of helper functions
  /// @dev [Test purposes] Test get the minted tier amounts
  function mock_getMintedTierAmount(uint128 tierId) public view returns (uint256) {
    return mintedTierAmount[tierId];
  }
  function mock_getMintedTierAmounts() public view returns (uint256[TIER_IDS_ARRAY_LEN] memory) {
    return mintedTierAmount;
  }
```

File: `test/DropBox/behaviors/revealDropBox/revealDropBoxes.t.sol`

```
// firstly over-write the `_validateMinting` function to perform additional verification
  function _validateMinting(
    uint256[TIER_IDS_ARRAY_LEN] memory prevMintedTierAmounts,
    uint32 boxAmount,
    string memory code) internal
  {
    uint256[] memory expectedBoxIds = new uint256[](boxAmount);
    for (uint256 i; i < boxAmount; i++) {
      expectedBoxIds[i] = i + 1;
    }

    vm.expectEmit(address(dropBox));
    emit DropBoxFractalProtocolLib.BoxesMinted(users.stranger, expectedBoxIds, abi.encode(code));
    _fundAndReveal(users.stranger, code, 1 ether);

    (uint256 blockTs, uint256 tier, address owner) = dropBox.getBox(1);
    assertEq(owner, users.stranger);
    assertGt(tier, 0);
    assertLt(tier, 7);

    // iterate over the boxes to get their types and update the
    // previous minted tier amounts
    for (uint256 i; i < expectedBoxIds.length; i++) {
      (,uint128 boxTierId, address owner) = dropBox.getBox(expectedBoxIds[i]);
      assertEq(owner, users.stranger);

      // increment previously minted tierId for this tier
      ++prevMintedTierAmounts[boxTierId];
    }

    // verify DropBox::mintedTierAmount has been correctly changed
    for(uint128 tierId = 1; tierId < TIER_IDS_ARRAY_LEN; tierId++) {
      assertEq(dropBox.mock_getMintedTierAmount(tierId), prevMintedTierAmounts[tierId]);
    }
  }

// next in the tests where _validateMinting is called, simply replace the call with this line
_validateMinting(dropBox.mock_getMintedTierAmounts(), boxAmount, code);
```

Mode: Fixed in commit 5badac3.

Cyfrin: Verified.

7.5.18 Cache result of `_getOneTimeCodeHash` to prevent repeated identical calculation in `DropBox::associateOneTimeCodeToAddress`

Description: `DropBox::associateOneTimeCodeToAddress` currently calculates `_getOneTimeCodeHash(code, allowedAddress)` twice:

```
    // [Safety check] Validate that the one-time code is not already associated to an address
    // @audit 1) calculating _getOneTimeCodeHash for first time
    address claimerAddress = oneTimeCode[_getOneTimeCodeHash(code, allowedAddress)].claimer;
    if (claimerAddress != address(0x0)) revert OneTimeCodeAlreadyAssociated(claimerAddress);

    // [Safety check] Validate that the one-time code has not been used
    if (oneTimeCodeUsed[keccak256(bytes(code))]) revert OneTimeCodeAlreadyUsed();

    uint256 remainingBoxesAmountCache = remainingBoxesAmount;
    // [Safety check] Validate that there are still balance of boxes to mint
    if (remainingBoxesAmountCache == 0) revert NoMoreBoxesToMint();

    // [Safety check] Validate that the remaining boxes to mint are greater or equal to the box amount
    if (remainingBoxesAmountCache < boxAmount) revert NoMoreBoxesToMint();
    // ---------------------------------------------------------------------------------------------

    // Decrease the amount of boxes remaining to mint, since they are now associated to be minted
    remainingBoxesAmount -= boxAmount;

    // Generate the hash of the code and the sender address
    // @audit 2) calculating _getOneTimeCodeHash again with identical inputs
    bytes32 otpHash = _getOneTimeCodeHash(code, allowedAddress);
```

Since `code` and `allowedAddress` don't change there is no point recalculating again; simply do it once and used the cached result:

```diff
+   // Generate the hash of the code and the sender address
+   bytes32 otpHash = _getOneTimeCodeHash(code, allowedAddress);

    // [Safety check] Validate that the one-time code is not already associated to an address
-   address claimerAddress = oneTimeCode[_getOneTimeCodeHash(code, allowedAddress)].claimer;
+   address claimerAddress = oneTimeCode[otpHash].claimer;
    if (claimerAddress != address(0x0)) revert OneTimeCodeAlreadyAssociated(claimerAddress);

    // [Safety check] Validate that the one-time code has not been used
    if (oneTimeCodeUsed[keccak256(bytes(code))]) revert OneTimeCodeAlreadyUsed();

    uint256 remainingBoxesAmountCache = remainingBoxesAmount;

    // [Safety check] Validate that there are still balance of boxes to mint
    if (remainingBoxesAmountCache == 0) revert NoMoreBoxesToMint();

    // [Safety check] Validate that the remaining boxes to mint are greater or equal to the box amount
    if (remainingBoxesAmountCache < boxAmount) revert NoMoreBoxesToMint();
    // ---------------------------------------------------------------------------------------------

    // Decrease the amount of boxes remaining to mint, since they are now associated to be minted
    remainingBoxesAmount -= boxAmount;

-   // Generate the hash of the code and the sender address
-   bytes32 otpHash = _getOneTimeCodeHash(code, allowedAddress);
```

Mode: Fixed in commit 1c983c2.

Cyfrin: Verified.

7.5.19 Remove second `for` loop in `DropBox::claimDropBoxes` and reduce 1 storage read per deleted box

Description: The first `for` loop inside `DropBox::claimDropBoxes` can be written like this such that the second `for` loop can be removed, meaning there is only need to iterate once over the input `_boxIds`:

```
    for (uint256 i; i < _boxIds.length; i++) {
      // [Safety check] Duplicate values are not allowed
      if (i > 0 && _boxIds[i - 1] >= _boxIds[i]) revert BadRequest();

      // [Safety check] Validate that the box owner is the sender
      if (ownerOf(_boxIds[i]) != msg.sender) revert Unauthorized();

      // Copy the box from the boxes mapping
      Box memory box = boxIdToBox[_boxIds[i]];

      // Increment the amount of Earnm tokens to send to the sender
      amountToClaim += _calculateVestingPeriodPerBox(box.tier, box.blockTs, TIER_IDS_LENGTH);

      // Delete the box from the boxes mappings
      _deleteBox(_boxIds[i], box.tier);

      // Burn the ERC721 token corresponding to the box
      _burn(_boxIds[i]);
    }
```

The above code requires a modified `_deleteBox` function which saves 1 storage read per deleted box:

```
  /// @param _boxTierId the tier id of the box to delete
  function _deleteBox(uint256 _boxId, uint128 _boxTierId) internal {
    // Delete from address -> _boxId mapping
    delete boxIdToBox[_boxId];

    // Decrease the unclaimed amount of boxes minted per tier
    unclaimedTierAmount[_boxTierId]--;
  }
```

This in turn requires 2 changes to the test suite to pass in the additional `_boxTierId` parameter:

```
// mocks/DropBoxMock.sol
  function mock_deleteBox(uint256 boxId, uint128 tierId) public {
    _deleteBox(boxId, tierId);
  }

// DropBox/behaviors/boxStorage/deleteBox.t.sol
dropBox.mock_deleteBox(boxId, tierId);
```

Mode: Fixed in commit 541f929.

Cyfrin: Verified.

7.5.20 Only read and write storage location `unclaimedTierAmount` once per changed tier in `DropBox::claimDropBoxes`

Description: `DropBox::claimDropBoxes` can be re-written to drastically reduce the amount of storage read/writes to `unclaimedTierAmount` storage location like this:

```
  function claimDropBoxes(uint256[] calldata _boxIds) external payable nonReentrant {
    //
    //
    //
    //  Request Checks
    // ---------------------------------------------------------------------------------
    // [Safety check] Only allowed to claim if the isEarnmClaimAllowed variable is true
    // Can be modified by the owner as a safety measure
    if (!(isEarnmClaimAllowed)) revert Unauthorized();

    // [Safety check] Validate that the boxIds array is not empty
    if (_boxIds.length == 0) revert Unauthorized();

    // Require that the user sends a fee of "claimFee" amount
    if (msg.value != claimFee) revert InvalidClaimFee();
```

```
// -------------------------------------------------------------------------------------------

// Amount of Earnm tokens to claim
uint256 amountToClaim;

// [Gas] Cache unclaimedTierAmount and update cache during the loop then
// write to storage at the end to prevent multiple update storage writes
uint256[TIER_IDS_ARRAY_LEN] memory unclaimedTierAmountCache;

// [Gas] Cache the tier changed indicator to update storage only for tiers which changed
bool[TIER_IDS_ARRAY_LEN] memory tierChangedInd;

//
//
//
//  Earnm Calculation and Safety Checks
// -------------------------------------------------------------------------------------------
// Iterate over box IDs to calculate rewards and validate ownership and uniqueness.
// - Validate that the sender is the owner of the boxes
// - Validate that the sender has enough boxes to claim
// - Validate that the box ids are valid
// - Validate that the box ids are not duplicated
// - Calculate the amount of Earnm tokens to claim given the box ids
for (uint256 i; i < _boxIds.length; i++) {
  // [Safety check] Duplicate values are not allowed
  if (i > 0 && _boxIds[i - 1] >= _boxIds[i]) revert BadRequest();

  // [Safety check] Validate that the box owner is the sender
  if (ownerOf(_boxIds[i]) != msg.sender) revert Unauthorized();

  // Copy the box from the boxes mappings
  Box memory box = boxIdToBox[_boxIds[i]];

  // Increment the amount of Earnm tokens to send to the sender
  amountToClaim += _calculateVestingPeriodPerBox(box.tier, box.blockTs, TIER_IDS_LENGTH);

  // if the unclaimed cache hasn't been updated for this tier type
  if(!tierChangedInd[box.tier]) {
      // then set it as updated
      tierChangedInd[box.tier] = true;

      // cache the current amount once from storage
      unclaimedTierAmountCache[box.tier] = unclaimedTierAmount[box.tier];
  }

  // decrement the unclaimed cached amount for this tier
  --unclaimedTierAmountCache[box.tier];

  // Delete the box from the boxIdToBox mapping
  delete boxIdToBox[_boxIds[i]];

  // Burn the ERC721 token corresponding to the deleted box
  _burn(_boxIds[i]);
}

// [Safety check] In case the amount to claim is 0, revert
if (amountToClaim == 0) revert AmountOfEarnmToClaimIsZero();

// Update tier storage only for tiers which were changed
for (uint128 tierId = 1; tierId < TIER_IDS_ARRAY_LEN; tierId++) {
  if (tierChangedInd[tierId]) unclaimedTierAmount[tierId] = unclaimedTierAmountCache[tierId];
}
```

```
    // ---------------------------------------------------------------------------------------------

    //
    //
    //
    //  Contract Earnm Balance and Safety Checks
    // ---------------------------------------------------------------------------------------------
    // Get the balance of Earnm ERC20 tokens of the contract
    uint256 balance = EARNM.balanceOf(address(this));

    // [Safety check] Validate that the contract has enough Earnm tokens to cover the claim
    if (balance < amountToClaim) revert InsufficientEarnmBalance();
    // ---------------------------------------------------------------------------------------------

    // Emit an event indicating the boxes have been claimed
    emit BoxesClaimed(msg.sender, _boxIds, amountToClaim);

    // Transfer the tokens to the sender
    EARNM.safeTransfer(msg.sender, amountToClaim);
    // ---------------------------------------------------------------------------------------------

    //
    //
    //
    //  Claim Fee Transfer
    // ---------------------------------------------------------------------------------------------
    _sendFee(msg.value);
    // ---------------------------------------------------------------------------------------------
  }
```

With this version the `DropBox::_deleteBox` function is no longer required and should be deleted along with its associated test file and helper function in `DropBoxMock.sol`.

We also noticed through mutation testing that updates to storage for `unclaimedTierAmount` and `boxIdToBox` which occur during the claim process were not being verified by the existing test suite, so added new features to the test suite to cover this:

File: `test/mocks/DropBoxMock.sol`

```
// new helper function
  /// @dev [Test purposes] Test get all the unclaimed tier amounts
  function mock_getUnclaimedTierAmounts() public view returns (uint256[TIER_IDS_ARRAY_LEN] memory) {
    return unclaimedTierAmount;
  }
```

File: `test/DropBox/behaviors/claimDropBox/claimDropBox.t.sol`

```
// updated this helper function to return the expected unclaimed amounts
  function _calculateExpectedClaimAmount(uint256[] memory boxIds) internal view
  returns (uint256 totalClaimAmount, uint256[TIER_IDS_ARRAY_LEN] memory expectedUnclaimedTierAmounts) {
    // first load the existing unclaimed tier amounts
    expectedUnclaimedTierAmounts = dropBox.mock_getUnclaimedTierAmounts();

    // iterate through the boxes that will be claimed
    for (uint256 i; i < boxIds.length; i++) {
      // get timestamp & tier
      (uint128 blockTs, uint128 tier,) = dropBox.getBox(boxIds[i]);

      // calculate expected token claim amount
      totalClaimAmount += dropBox.mock_calculateVestingPeriodPerBox(tier, blockTs, TIER_IDS_LENGTH);

      // update expected unclaimed tier amount
      --expectedUnclaimedTierAmounts[tier];
    }
  }

// updated this helper function to validate the expected amounts
  function _validateClaimedBoxes(
    uint256[TIER_IDS_ARRAY_LEN] memory expectedUnclaimedTierAmounts,
    uint256[] memory boxIds,
    uint256 expectedAmount,
    address claimer) internal
  {
    for (uint256 i; i < boxIds.length; i++) {
      // get info for deleted box
      (uint128 blockTs, uint128 tier, address owner) = dropBox.getBox(boxIds[i]);

      // box should no longer be owned
      assertEq(owner, address(0));

      // box should be deleted from boxIdToBox mapping
      assertEq(blockTs, 0);
      assertEq(tier, 0);
    }

    // verify DropBox::unclaimedTierAmount has been correctly decreased
    for(uint128 tierId = 1; tierId < TIER_IDS_ARRAY_LEN; tierId++) {
      assertEq(dropBox.mock_getUnclaimedTierAmount(tierId), expectedUnclaimedTierAmounts[tierId]);
    }

    // verify expected earnm contract balance after claims processed
    uint256 earnmBalance = earnmERC20Token.balanceOf(claimer);
    assertEq(earnmBalance, expectedAmount);

    emit DropBoxFractalProtocolLib.BoxesClaimed(claimer, boxIds, expectedAmount);
  }

// updated 2 test functions to use the new updated helper functions
  function test_claimDropBoxes_valid() public {
    string memory code = "validCode";
    address allowedAddress = users.stranger;
    uint32 boxAmount = 5;
    uint256[] memory boxIds = new uint256[](boxAmount);
    uint256 earnmAmount = 1000 * 1e18;

    _setupClaimEnvironment(code, allowedAddress, boxAmount, boxIds, earnmAmount);

    (uint256 expectedAmount, uint256[TIER_IDS_ARRAY_LEN] memory expectedUnclaimedTierAmounts)
      = _calculateExpectedClaimAmount(boxIds);

    _fundAndClaim(allowedAddress, boxIds, 1 ether);
    _validateClaimedBoxes(expectedUnclaimedTierAmounts, boxIds, expectedAmount, allowedAddress);
  }

  function test_claimDropBoxes_ClaimAndBurn() public {
    string memory code = "validCode";
    address allowedAddress = users.stranger;
    uint32 boxAmount = 5;
    uint256[] memory boxIds = new uint256[](boxAmount);
    uint256 earnmAmount = 1000 * 1e18;
    _setupClaimEnvironment(code, allowedAddress, boxAmount, boxIds, earnmAmount);
    (uint256 expectedAmount, uint256[TIER_IDS_ARRAY_LEN] memory expectedUnclaimedTierAmounts)
      = _calculateExpectedClaimAmount(boxIds);

    _fundAndClaim(allowedAddress, boxIds, 1 ether);
    _validateClaimedBoxes(expectedUnclaimedTierAmounts, boxIds, expectedAmount, allowedAddress);
  }
```

Mode: Fixed in commit 82a2365.

Cyfrin: Verified.

7.5.21 Only read and write storage location `unclaimedTierAmount` once per changed tier in `DropBox::_assignTierAndMint`

Description: `DropBox::_assignTierAndMint` can be re-written to drastically reduce the amount of storage read/writes to `unclaimedTierAmount` storage location like this:

```
function _assignTierAndMint(
  uint32 boxAmount,
  uint256[] memory randomNumbers,
  address claimer,
  bytes memory code
)
  internal
{
  uint256[] memory boxIdsToEmit = new uint256[](boxAmount);

  // [Gas] Cache mintedTierAmount and update cache during the loop to prevent
  // _determineTier() from re-reading it from storage multiple times and
  // to prevent multiple update storage writes
  uint256[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache = mintedTierAmount;

  // [Gas] Cache unclaimedTierAmount and update cache during the loop then
  // write to storage at the end to prevent multiple update storage writes
  uint256[TIER_IDS_ARRAY_LEN] memory unclaimedTierAmountCache;

  // [Gas] Cache the tier changed indicator to update storage only for tiers which changed
  bool[TIER_IDS_ARRAY_LEN] memory tierChangedInd;

  // [Gas] cache the current boxId, use cache during the loop then write once
  // at the end; this results in only 1 storage read and 1 storage write for `boxIdCounter`
  uint256 boxIdCounterCache = boxIdCounter;

  // For each box to mint
  for (uint32 i; i < boxAmount; i++) {
    // [Memory] Add the box id to the boxes ids array
    boxIdsToEmit[i] = boxIdCounterCache++;

    // The box seed is directly derived from the random number without any
    // additional values like the box id, the claimer address, or the block timestamp.
    // This is made like this to ensure the tier assignation of the box is purely random
    // and the tier that the box is assigned to, won't change after the randomness is fulfilled.
    // The only way to predict the seed is to know the random number.
    // Random number validation is done in the _fulfillRandomWords function.
    uint256 boxSeed = uint256(keccak256(abi.encode(randomNumbers[i])));

    // Generate a random number between 0 and TOTAL_MAX_BOXES - 1 using the pure random seed
    // Modulo ensures that the random number is within the range of [0, TOTAL_MAX_BOXES) (upper bound
    ↪  exclusive),
    // providing a uniform distribution across the possible range. This ensures that each tier has a
    // probability of being selected proportional to its maximum box count. The randomness is
        ↪  normalized
    // to fit within the predefined total boxes, maintaining the intended probability distribution
        ↪  for each tier.
    //
    // Example with the following setup:
    // _tierMaxBoxes -> [1, 10, 100, 1000, 2000, 6889]
    // _tierNames -> ["Mythical Box", "Legendary Box", "Epic Box", "Rare Box", "Uncommon Box",
        ↪  "Common Box"]
    //
    // The `TOTAL_MAX_BOXES` is the sum of `_tierMaxBoxes`, which is 10_000.
    // The probability distribution is as follows:
    //
    // Mythical Box:   1    / 10_000  chance.
    // Legendary Box:  10   / 10_000  chance.
    // Epic Box:       100  / 10_000  chance.
    // Rare Box:       1000 / 10_000  chance.
    // Uncommon Box:   2000 / 10_000  chance.
    // Common Box:     6889 / 10_000  chance.
    //
    // By using the modulo operation with TOTAL_MAX_BOXES, the random number's probability
    // to land in any given tier is determined by the number of max boxes available for that tier
    uint256 randomNumber = boxSeed % TOTAL_MAX_BOXES;

    // Determine the tier of the box based on the random number
    uint128 tierId = _determineTier(randomNumber, mintedTierAmountCache);

    // if the tier amount cache hasn't been updated for this tier type
    if(!tierChangedInd[tierId]) {
        // then set it as updated
        tierChangedInd[tierId] = true;

        // cache the current amounts once from storage
        unclaimedTierAmountCache[tierId] = unclaimedTierAmount[tierId];
    }

    // [Gas] Adjust the cached amounts for this tier
    ++mintedTierAmountCache[tierId];
    ++unclaimedTierAmountCache[tierId];

    // Save the box to the boxIdToBox mapping
    boxIdToBox[boxIdsToEmit[i]] = Box({ blockTs: uint128(block.timestamp), tier: tierId });

    // Mint NFT for this box; explicitly not using `_safeMint` to prevent re-entrancy issues
    _mint(claimer, boxIdsToEmit[i]);
  }

  // Update storage for boxIdCounter
  boxIdCounter = boxIdCounterCache;

  // Update tier storage only for tiers which were changed
  for (uint128 tierId = 1; tierId < TIER_IDS_ARRAY_LEN; tierId++) {
    if (tierChangedInd[tierId]) {
      mintedTierAmount[tierId] = mintedTierAmountCache[tierId];
      unclaimedTierAmount[tierId] = unclaimedTierAmountCache[tierId];
    }
  }

  // Emit an event indicating the boxes have been minted
  emit BoxesMinted(claimer, boxIdsToEmit, code);
}
```

When we made this change we noticed that `test_assignTierAndMint_InvalidBoxAmount` started to fail but this test doesn't seem to be valid since the `boxAmount` input is validated prior to `_assignTierAndMint` being called, hence we believe this test should be deleted as it is not relevant to the current codebase.

Mode: Fixed in commit 79eea07.

Cyfrin: Verified.

7.5.22 Use `uint32` for more efficient storage packing when tracking minted and unclaimed tier box amounts

Description: Firstly change `DropBox` storage like this for more efficient storage slot packing:

- move `vrfHandler` up to the start of the public variables
- change some non-constants to `uint32`
- group all the `uint32` non-constants together just after the `bool` non-constants:

```
contract DropBox is
  IVRFHandlerReceiver,
  DropBoxFractalProtocol,
  ReentrancyGuard,
  OwnableBasic,
  BasicRoyalties,
  ERC721C
{
  using SafeERC20 for IERC20;

  /// @notice - EARNM ERC20 token
  IERC20 internal immutable EARNM;

  /// @notice Public variables
  IVRFHandler public vrfHandler; // VRF Handler to handle the Chainlink VRF requests
  string public _contractURI; // OpenSea Contract-level metadata
  string public _baseTokenURI; // ERC721 Base Token metadata
  address public apiAddress;
  address public feesReceiverAddress;
  uint128 public revealFee = 1 ether;
  uint128 public claimFee = 1 ether;
  bool public isEarnmClaimAllowed;
  bool public isRevealAllowed;

  /// @notice Internal variables
  uint32 internal boxIdCounter = 1; // Counter for the box ids
  uint32 internal remainingBoxesAmount; // Remaining boxes amount to mint
  /// @notice this array stores index: tierId, value: total amount minted
  /// element [0] never used since tierId : [1,6]
  uint32[TIER_IDS_ARRAY_LEN] internal mintedTierAmount;
  uint32[TIER_IDS_ARRAY_LEN] internal unclaimedTierAmount;

  /// @notice Mappings
  mapping(uint256 => Box) internal boxIdToBox; // boxId -> Box
  mapping(bytes32 => OneTimeCode) internal oneTimeCode; // code+address hash -> one-time code data;
  ↪  ensure integrity of
    // one-time code data
  mapping(bytes32 => bool) internal oneTimeCodeUsed; // plain text code -> used (true/false); ensure
  ↪  uniqueness of
    // one-time codes
  mapping(bytes32 => uint256[]) internal oneTimeCodeRandomWords; // code+address hash -> random words
  /// [Chainlink VRF]
  mapping(uint256 => bytes32) internal activeVrfRequests; // vrf request -> code+address hash
  mapping(uint256 => bool) internal fulfilledVrfRequests; // vrf request -> fulfilled

  /// @notice Constants - DropBox
  uint32 internal constant MAX_BOX_AMOUNT_PER_REVEAL = 100;
  uint128 internal constant MAX_MINT_FEE = 1000 ether;
  uint128 internal constant MAX_CLAIM_FEE = 1000 ether;
```

Then change the appropriate types throughout `DropBox.sol`:

```
// function associateOneTimeCodeToAddress,
uint32 remainingBoxesAmountCache = remainingBoxesAmount;

// function claimDropBoxes - see optimized code at end of this issue

// function _assignTierAndMint - see optimized code at end of this issue

// view function return parameters
  function getTotalMaxBoxes() external view returns (uint32 totalMaxBoxes) {
```

And inside `DropBoxFractalProtocol.sol`:

```
// events
event OneTimeCodeAssociated(address indexed claimerAddress, bytes code, uint32 boxAmount);

  /// @notice Constants - Tier max boxes
  uint32 internal immutable TIER_6_MAX_BOXES;
  uint32 internal immutable TIER_5_MAX_BOXES;
  uint32 internal immutable TIER_4_MAX_BOXES;
  uint32 internal immutable TIER_3_MAX_BOXES;
  uint32 internal immutable TIER_2_MAX_BOXES;
  uint32 internal immutable TIER_1_MAX_BOXES;
  uint32 internal immutable TOTAL_MAX_BOXES;

  function _determineTier(
    uint256 randomNumber,
    uint32[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache
```

The test suite also needs some changes:

File: `test/mocks/DropBoxMock.sol`

```
function mock_setUnclaimedTierAmount(uint128 tierId, uint32 amount) public {
function mock_getUnclaimedTierAmount(uint128 tierId) public view returns (uint32) {
function mock_setAllBoxesMinted(uint32 amount) public {

function mock_remainingBoxesAmount() public view returns (uint32) {

  function mock_determineTier(
    uint256 randomNumber,
    uint32[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache

function mock_getMintedTierAmount(uint128 tierId) public view returns (uint32) {
function mock_getMintedTierAmounts() public view returns (uint32[TIER_IDS_ARRAY_LEN] memory) {
function mock_getUnclaimedTierAmounts() public view returns (uint32[TIER_IDS_ARRAY_LEN] memory) {

  function mock_getMaxBoxesPerTier(uint128 tierId) external view returns (uint32 maxBoxesPerTier) {
```

File: `test/DropBox/behaviours/claimDropBox/claimDropBox.t.sol`

```
  function _calculateExpectedClaimAmount(uint256[] memory boxIds)
    internal
    view
    returns (uint256 totalClaimAmount, uint32[TIER_IDS_ARRAY_LEN] memory expectedUnclaimedTierAmounts)

  function _validateClaimedBoxes(
    uint32[TIER_IDS_ARRAY_LEN] memory expectedUnclaimedTierAmounts,

// occurs in 2 few places
    (uint256 expectedAmount, uint32[TIER_IDS_ARRAY_LEN] memory expectedUnclaimedTierAmounts) =
      _calculateExpectedClaimAmount(boxIds);

// near the bottom
    for (uint32 i; i < boxAmount; i++) {
```

File: `test/DropBox/behaviours/earnmInVesting/liability.t.sol`

```
  function _setUnclaimedTierAmounts(uint32[] memory unclaimedAmounts) internal {
  function _calculateExpectedLiability(uint32[] memory unclaimedAmounts) internal view returns (uint256)

    // occurs in a few places
    uint32[] memory unclaimedAmounts = new uint32[](TIER_IDS_LENGTH);
    for (uint32 i = 1; i <= TIER_IDS_LENGTH; i++) {
```

File: `test/DropBox/behaviours/revealDropBox/revealDropBoxes.t.sol`

```
  function _validateMinting(
    address allowedAddress,
    uint32[TIER_IDS_ARRAY_LEN] memory prevMintedTierAmounts,

    for (uint32 i; i < boxAmount; i++) {
```

File: `test/FractalProtocol/FractalProtocol.sol`

```
// occurs many times
    uint32[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache;
```

File: `test/DropBoxFractalProtocolLib.sol`

```
  event OneTimeCodeAssociated(address indexed claimerAddress, bytes code, uint32 boxAmount);
```

After making the above changes everything compiles and all the tests pass when using `gas_limit = "18446744073709551615" # u64::MAX` in `foundry.toml` to raise gas limit for tests which attempt to reveal/claim all boxes.

Using `forge inspect DropBox storage` shows that the regular code used 46 storage slots while the version modified with the above changes uses only 33 storage slots.

Another benefit of this is that the entire arrays `unclaimedTierAmount` and `mintedTierAmount` are now stored in 1 storage slot each as opposed to using 1 storage slot per array element; this means that `claimDropBoxes` and `_assignTierAndMint` can be further simplified and optimized to read and write the arrays once without any conditional tier-based logic:

```
  function claimDropBoxes(uint256[] calldata _boxIds) external payable nonReentrant {
    //
```

```
//  Request Checks
// ---------------------------------------------------------------------------------------------
// [Safety check] Only allowed to claim if the isEarnmClaimAllowed variable is true
// Can be modified by the owner as a safety measure
if (!(isEarnmClaimAllowed)) revert Unauthorized();

// [Safety check] Validate that the boxIds array is not empty
if (_boxIds.length == 0) revert Unauthorized();

// Require that the user sends a fee of "claimFee" amount
if (msg.value != claimFee) revert InvalidClaimFee();
// ---------------------------------------------------------------------------------------------

// Amount of Earnm tokens to claim
uint256 amountToClaim;

// [Gas] Cache unclaimedTierAmount and update cache during the loop then
// write to storage at the end to prevent multiple update storage writes
uint32[TIER_IDS_ARRAY_LEN] memory unclaimedTierAmountCache = unclaimedTierAmount;

//  Earnm Calculation and Safety Checks
// ---------------------------------------------------------------------------------------------
// Iterate over box IDs to calculate rewards and validate ownership and uniqueness.
// - Validate that the sender is the owner of the boxes
// - Validate that the sender has enough boxes to claim
// - Validate that the box ids are valid
// - Validate that the box ids are not duplicated
// - Calculate the amount of Earnm tokens to claim given the box ids
for (uint256 i; i < _boxIds.length; i++) {
  // [Safety check] Duplicate values are not allowed
  if (i > 0 && _boxIds[i - 1] >= _boxIds[i]) revert BadRequest();

  // [Safety check] Validate that the box owner is the sender
  if (ownerOf(_boxIds[i]) != msg.sender) revert Unauthorized();

  // Copy the box from the boxes mappings
  Box memory box = boxIdToBox[_boxIds[i]];

  // Increment the amount of Earnm tokens to send to the sender
  amountToClaim += _calculateVestingPeriodPerBox(box.tier, box.blockTs, TIER_IDS_LENGTH);

  // Decrement the unclaimed cached amount for this tier
  --unclaimedTierAmountCache[box.tier];

  // Delete the box from the boxIdToBox mapping
  delete boxIdToBox[_boxIds[i]];

  // Burn the ERC721 token corresponding to the box
  _burn(_boxIds[i]);
}

// [Safety check] In case the amount to claim is 0, revert
if (amountToClaim == 0) revert AmountOfEarnmToClaimIsZero();

// Update unclaimed tier storage
unclaimedTierAmount = unclaimedTierAmountCache;

// ---------------------------------------------------------------------------------------------
//  Contract Earnm Balance and Safety Checks
// ---------------------------------------------------------------------------------------------
// Get the balance of Earnm ERC20 tokens of the contract
uint256 balance = EARNM.balanceOf(address(this));
```

```
  // [Safety check] Validate that the contract has enough Earnm tokens to cover the claim
  if (balance < amountToClaim) revert InsufficientEarnmBalance();
  // ---------------------------------------------------------------------------------------------

  // Emit an event indicating the boxes have been claimed
  emit BoxesClaimed(msg.sender, _boxIds, amountToClaim);

  // Transfer the tokens to the sender
  EARNM.safeTransfer(msg.sender, amountToClaim);
  // ---------------------------------------------------------------------------------------------

  //  Claim Fee Transfer
  // ---------------------------------------------------------------------------------------------
  _sendFee(msg.value);
  // ---------------------------------------------------------------------------------------------
}

function _assignTierAndMint(
  uint32 boxAmount,
  uint256[] memory randomNumbers,
  address claimer,
  bytes memory code
)
  internal
{
  uint256[] memory boxIdsToEmit = new uint256[](boxAmount);

  // [Gas] Cache mintedTierAmount and update cache during the loop to prevent
  // _determineTier() from re-reading it from storage multiple times
  uint32[TIER_IDS_ARRAY_LEN] memory mintedTierAmountCache = mintedTierAmount;

  // [Gas] Cache unclaimedTierAmount and update cache during the loop then
  // write to storage at the end to prevent multiple update storage writes
  uint32[TIER_IDS_ARRAY_LEN] memory unclaimedTierAmountCache = unclaimedTierAmount;

  // [Gas] cache the current boxId, use cache during the loop then write once
  // at the end; this results in only 1 storage read and 1 storage write for `boxIdCounter`
  uint32 boxIdCounterCache = boxIdCounter;

  // For each box to mint
  for (uint32 i; i < boxAmount; i++) {
    // [Memory] Add the box id to the boxes ids array
    boxIdsToEmit[i] = boxIdCounterCache++;

    // The box seed is directly derived from the random number without any
    // additional values like the box id, the claimer address, or the block timestamp.
    // This is made like this to ensure the tier assignation of the box is purely random
    // and the tier that the box is assigned to, won't change after the randomness is fulfilled.
    // The only way to predict the seed is to know the random number.
    // Random number validation is done in the _fulfillRandomWords function.
    uint256 boxSeed = uint256(keccak256(abi.encode(randomNumbers[i])));

    // Generate a random number between 0 and TOTAL_MAX_BOXES - 1 using the pure random seed
    // Modulo ensures that the random number is within the range of [0, TOTAL_MAX_BOXES) (upper bound
    ↪  exclusive),
    // providing a uniform distribution across the possible range. This ensures that each tier has a
    // probability of being selected proportional to its maximum box count. The randomness is
    ↪  normalized
    // to fit within the predefined total boxes, maintaining the intended probability distribution
    ↪  for each tier.
    //
```

```
            // Example with the following setup:
            // _tierMaxBoxes -> [1, 10, 100, 1000, 2000, 6889]
            // _tierNames -> ["Mythical Box", "Legendary Box", "Epic Box", "Rare Box", "Uncommon Box",
            ↪  "Common Box"]
            //
            // The `TOTAL_MAX_BOXES` is the sum of `_tierMaxBoxes`, which is 10_000.
            // The probability distribution is as follows:
            //
            // Mythical Box:   1    / 10_000  chance.
            // Legendary Box:  10   / 10_000  chance.
            // Epic Box:       100  / 10_000  chance.
            // Rare Box:       1000 / 10_000  chance.
            // Uncommon Box:   2000 / 10_000  chance.
            // Common Box:     6889 / 10_000  chance.
            //
            // By using the modulo operation with TOTAL_MAX_BOXES, the random number's probability
            // to land in any given tier is determined by the number of max boxes available for that tier
            uint256 randomNumber = boxSeed % TOTAL_MAX_BOXES;

            // Determine the tier of the box based on the random number
            uint128 tierId = _determineTier(randomNumber, mintedTierAmountCache);

            // [Gas] Adjust the cached amounts for this tier
            ++mintedTierAmountCache[tierId];
            ++unclaimedTierAmountCache[tierId];

            // Save the box to the boxIdToBox mapping
            boxIdToBox[boxIdsToEmit[i]] = Box({ blockTs: uint128(block.timestamp), tier: tierId });

            // Mint the box; explicitly not using `_safeMint` to prevent re-entrancy issues
            _mint(claimer, boxIdsToEmit[i]);
        }

        // Update storage for boxIdCounter
        boxIdCounter = boxIdCounterCache;

        // Update unclaimed & minted tier amounts
        mintedTierAmount = mintedTierAmountCache;
        unclaimedTierAmount = unclaimedTierAmountCache;

        // Emit an event indicating the boxes have been minted
        emit BoxesMinted(claimer, boxIdsToEmit, code);
    }
```

Mode: Fixed in commit 37d258c.

Cyfrin: Verified.

7.5.23 Use `uint128` for `revealFee` and `claimFee` for more efficient storage packing

Description: Currently `revealFee` and `claimFee` use `uint256` and are set to `1 ether` with a max value of `1000 ether`. Hence they each take up 1 storage slot but this is not that efficient since:

```
1 ether            = 1000000000000000000
1000 ether         = 1000000000000000000000
type(uint128.max)  = 340282366920938463463374607431768211455
```

Therefore use `uint128` for `revealFee` and `claimFee`:

```
// storage
  uint128 public revealFee = 1 ether;
  uint128 public claimFee = 1 ether;
  uint128 internal constant MAX_MINT_FEE = 1000 ether;
  uint128 internal constant MAX_CLAIM_FEE = 1000 ether;

// function updates
function setRevealFee(uint128 _mintFee) external onlyOwner {
  function setClaimFee(uint128 _claimFee) external onlyOwner {
```

Also update events in `DropBoxFractalProtocol`:

```
  event RevealFeeUpdated(uint128 revealFee);
  event ClaimFeeUpdated(uint128 claimFee);
```

Mode: Fixed in commit fb20301.

Cyfrin: Verified.

7.5.24 Remove redundant check from `VRFHandler::constructor`

Description: Remove redundant check from `VRFHandler::constructor`:

```
  constructor(
    address vrfCoordinator,
    bytes32 keyHash,
    uint256 subscriptionId
  )
    /**
     * ConfirmedOwner(msg.sender) is defined in
     ↪  @chainlink/contracts/src/v0.8/vrf/dev/VRFConsumerBaseV2Plus.sol:113:40
     * with "msg.sender".
     */
    VRFConsumerBaseV2Plus(vrfCoordinator)
  {
    if (keyHash == bytes32(0)) revert InvalidVrfKeyHash();
    if (subscriptionId == 0) revert InvalidVrfSubscriptionId();
-   if (vrfCoordinator == address(0)) revert InvalidVrfCoordinator();

    vrfKeyHash = keyHash;
    vrfSubscriptionId = subscriptionId;
  }
```

This check is redundant as `VRFHandler` inherits from `VRFConsumerBaseV2Plus` which already performs this check in its own constructor.

Mode: Fixed in commit cc45c9a.

Cyfrin: Verified.

7.5.25 Delete fulfilled vrf request from `vrfRequestIdToRequester` in `VRFHandler::fulfillRandomWords`

Description: Delete fulfilled vrf request from `vrfRequestIdToRequester` in `VRFHandler::fulfillRandomWords`:

```
  function fulfillRandomWords(uint256 _requestId, uint256[] calldata _randomWords) internal override {
    // Check if the request has already been fulfilled
    if (vrfFulfilledRequests[_requestId]) revert InvalidVrfState();
    // Cache the contract that requested the random numbers based on the request ID
    address contractRequestor = vrfRequestIdToRequester[_requestId];

    // Revert if the contract that requested the random numbers is not found
    if (contractRequestor == address(0)) revert InvalidVrfState();

+   // delete the active requestId -> requestor record
+   delete vrfRequestIdToRequester[_requestId];

    // Mark the request as fulfilled
    vrfFulfilledRequests[_requestId] = true;

    // Decrement the counter of outstanding requests
    activeRequests--;

    // Call the contract that requested the random numbers with the random numbers
    IVRFHandlerReceiver(contractRequestor).fulfillRandomWords(_requestId, _randomWords);
  }
```

Mode: Fixed in commit 08d7ce4.

Cyfrin: Verified.