EARN'M Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

0kage

December 14, 2023

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 is a unique rewards program where users receive reward codes which they can use to claim mystery boxes of varying tiers. The probability of receiving lower-tier boxes is relatively high, while the chances of obtaining higher-tier, more valuable boxes are significantly lower. Users are eligible to claim their boxes after a set vesting period, receiving a predetermined amount of EARN'M tokens corresponding to their box's tier. An early claim option is also available but it comes with a penalty; a reduction in the reward tokens received. To ensure fairness and transparency in the distribution of boxes, EARN'M integrates the Chainlink Verified Random Number Generator, guaranteeing verifiable randomness in each box allocation.

5 Audit Scope

Following contracts were included in the scope for this audit:

```
contracts/EARNM.sol
contracts/MysteryBox.sol
```

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the EARN'M smart contracts provided by MODE. In this period, a total of 20 issues were found.

The findings consist of 2 Critical, 3 High, 2 Medium & 1 Low severity issues with the remainder being informational and gas optimizations. One of the critical issues resulted in the mystery box tier changing during chain re-orgs which could also be exploited by validators who are minting mystery boxes, and the other critical bricked token redemption rewards when users called the standard `ERC1155` transfer functions to transfer their mystery boxes.

One high finding prevented the distribution of all 5B redemption tokens to users and another high identified a rug-pull vector by which the contract owner could rug-pull the redemption tokens, leaving the mystery box contract insolvent with users unable to redeem their mystery boxes. The last high identified a broken protection which could result in multiple fulfillments of randomness requests.

In addition to the above findings, Cyfrin provided significant advice on how to refactor the `MysteryBox` contract to reduce the complexity, reduce the storage requirements and eliminate many storage reads; these findings resulted in significant refactoring of the `MysteryBox` contract. Due to the significant changes the focus of the audit switched to the "audit" branch; issues that were identified in this branch during the mitigations and refactoring were reported to the client but not added as separate issues in the report.

Mitigation was made easier as all mitigations were committted to a separate "audit" branch and the later half of the audit time was spent verifying and hardening the "audit" branch. All serious issues were successfully mitigated and the project's unit tests were also improved.

Summary

Project Name	EARN'M
Repository	smart-contracts
Commit	43d3a8305dd6...
Audit Timeline	Nov 13th - Nov 17th
Methods	Manual Review

Issues Found

Critical Risk	2
High Risk	3
Medium Risk	2
Low Risk	1
Informational	4
Gas Optimizations	8
Total Issues	20

Summary of Findings

[C-1] Polygon chain reorgs will change mystery box tiers which can be gamed by validators	Resolved
[C-2] Transferring mystery boxes bricks token redemption	Resolved
[H-1] Broken check in `MysteryBox::fulfillRandomWords()` fails to prevent same request being fulfilled multiple times	Resolved
[H-2] Owner can rug-pull redemption tokens leaving mystery box contract insolvent and mystery box holders unable to redeem	Resolved

[H-3] Incorrect cap on `batchesAmount` results in 500M instead of 5B tokens distributed to mystery box holders	Resolved
[M-1] Excess eth not refunded to user in `MysteryBox::revealMysteryBoxes()`	Resolved
[M-2] Minting can be indefinitely stuck due to request timeout of external adapters when using Chainlink Any API	Acknowledged
[L-1] Use low level `call()` to prevent gas griefing attacks when returned data not required	Resolved
[I-1] Prevent duplicate `boxId` inputs to `MysteryBox::claimMysteryBoxes()`	Resolved
[I-2] `MysteryBox::claimMysteryBoxes()` should return custom error when reverting due to `amountToClaim == 0`	Resolved
[I-3] Potential Risk of Price Volatility in EarnNM Token Due to Concentrated Mystery Box Rewards	Acknowledged
[I-4] Centralisation risks as the reward code generator and the Chainlink node operator is the same entity	Acknowledged
[G-1] Remove from storage `baseMetadataURI` as already stored in `ERC1155` and `name` as never used	Resolved
[G-2] Standardize `tierId` to either `uint8` or `uint256` avoiding constant conversions back and forth	Resolved
[G-3] Simplify `boxId` storage mappings as `boxId` is unique to addresses and tiers	Resolved
[G-4] State variables should be cached in stack variables rather than re-reading them from storage	Resolved
[G-5] Loop backwards in `MysteryBox::_determineTier()` to avoid multiple variables and simplify code	Resolved
[G-6] Simplify calculation in `MysteryBox::_calculateAmountToClaim()`	Resolved
[G-7] Remove unused `category` from `MysteryBox::_calculateVestingPeriodPerBox()`	Resolved
[G-8] Check `boxAmount < 100` only once before loop in `MysteryBox::_assignTierAndMint()`	Resolved

7 Findings

7.1 Critical Risk

7.1.1 Polygon chain reorgs will change mystery box tiers which can be gamed by validators

Description: `REQUEST_CONFIRMATIONS = 3` is too small for polygon, as chain re-orgs frequently have block-depth greater than 3.

Impact: Chain re-orgs re-order blocks and transactions changing randomness results. Someone who originally won a rare box could have that result changed into a common box and vice versa due to changing randomness result during the re-org.

This can also be exploited by validators who can intentionally rewrite the chain's history to force a randomness request into a different block, changing the randomness result. This allows validators to get a fresh random value which may be to their advantage if they are minting mystery boxes by moving the txn around to get a better randomness result to mint a rarer box.

Recommended Mitigation: `REQUEST_CONFIRMATIONS = 30` appears very safe for polygon as it is very rare for chain re-orgs to have block-depth greater than this. If this happens occasionally it isn't a big deal, but if it happens all the time ("3" ensures this) that is not good and potentially exploitable by validators.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.1.2 Transferring mystery boxes bricks token redemption

Description: `MysteryBox` is an `ERC1155` contract which users expect to be able to transfer to other addresses via the in-built transfer functions. But `MysteryBox::claimMysteryBoxes()` reverts unless the caller is the same address who minted the box since the internal mappings that track mystery box ownership are never updated when transfers occur.

Impact: Token redemption is bricked if users transfer their mystery box. Users reasonably expect to be able to transfer their mystery box from one address they control to another address (if for example their first address is compromised), or they may wish to sell their mystery box on platforms like OpenSea which support `ERC1155` sales.

Recommended Mitigation: Override `ERC1155` transfer hooks to either prevent transferring of mystery boxes, or to update the internal mappings such that when mystery boxes are transferred the new owner address can redeem their tokens. The second option may be more attractive for the protocol as it allows mystery box holders to access liquidity without putting sell pressure on the token, creating a "secondary market" for mystery boxes.

Mode: Fixed in commit a65a50c by overriding `ERC1155::_beforeTokenTransfer()` to prevent mystery boxes from being transferred.

Cyfrin: Verified.

7.2 High Risk

7.2.1 Broken check in `MysteryBox::fulfillRandomWords()` fails to prevent same request being fulfilled multiple times

Description: Consider the check which attempts to prevent the same request from being fulfilled multiple times:

```
if (vrfRequests[_requestId].fulfilled) revert InvalidVrfState();
```

The problem is that `vrfRequests[_requestId].fulfilled` is never set to `true` anywhere and `vrfRequests[_requestId]` is deleted at the end of the function.

Impact: The same request can be fulfilled multiple times which would override the previous randomly generated seed; a malicious provider who was also a mystery box minter could generate new randomness until they got a rare mystery box.

Recommended Mitigation: Set `vrfRequests[_requestId].fulfilled = true`.

Consider an optimized version which involves having 2 mappings `activeVrfRequests` and `fulfilledVrfRequests`:

- revert `if(fulfilledVrfRequests[_requestId])`
- else set `fulfilledVrfRequests[_requestId] = true`
- fetch the matching active request into memory from `activeVrfRequests[_requestId]` and continue processing as normal
- at the end `delete activeVrfRequests[_requestId]`

This only stores forever the `requestId : bool` pair in `fulfilledVrfRequests`.

Consider a similar approach in `MysteryBox::fulfillBoxAmount()`.

Mode: Fixed in commit 85b2012, c4c50ed, d5b14d8, 5df2b82.

Cyfrin: Verified.

7.2.2 Owner can rug-pull redemption tokens leaving mystery box contract insolvent and mystery box holders unable to redeem

Description: `MysteryBox::ownerWithdrawEarnm()` allows the owner to transfer the contract's total redemption token balance to themselves, rug-pulling the redemption tokens which mystery boxes are supposed to be redeemed for.

Impact: The contract becomes totally insolvent and mystery box owners are unable to redeem.

Recommended Mitigation: The contract should always have the necessary tokens to payout the maximum redemption liability on all currently minted and unclaimed mystery boxes. The owner should only be able to withdraw the surplus amount (the excess over the total liability).

When mystery boxes are minted the total liability increases and when mystery boxes are claimed the total liability decreases. Consider tracking the total liability as mystery boxes are minted & claimed and only allowing the owner to withdraw the surplus tokens above this value.

Mode: Fixed in commit db7b48e, edefb61, a65a50c.

Cyfrin: Verified.

7.2.3 Incorrect cap on `batchesAmount` results in 500M instead of 5B tokens distributed to mystery box holders

Description: `setBatchesAmount()` caps the maximum `batchesAmount` 100 but this is incorrect. Every batch releases mystery boxes which can be redeemed for ~5M tokens and there are 5B tokens in total so 1000 batches to distribute the entire supply.

Impact: Incorrectly capping to 100 batches results in never being able to distribute all 5B tokens, but only 500M tokens.

Recommended Mitigation: Cap `batchesAmount` to 1000 to allow full token distribution.

Mode: Fixed in commit ae3dc68.

Cyfrin: Verified.

7.3 Medium Risk

7.3.1 Excess eth not refunded to user in `MysteryBox::revealMysteryBoxes()`

Description: `MysteryBox::revealMysteryBoxes()` allows execution if `msg.value >= mintFee` but in the case where `msg.value > mintFee`, the extra eth gets sent to `operatorAddress` not refunded back to the user.

Impact: User loses excess eth above `mintFee`.

Recommended Mitigation: Either refund excess eth back to the user or revert if `msg.value != mintFee`.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.3.2 Minting can be indefinitely stuck due to request timeout of external adapters when using Chainlink Any API

Description: Mode has integrated Chainlink Any API to interact with external adapters, verifying user codes and wallet addresses to determine the number of boxes to mint. The system uses a `direct-request` job type, triggering actions based on the `ChainlinkRequested` event emission. However, there's a notable issue: if the initial GET request times out, such requests may remain pending indefinitely. Current design does not have a provision to cancel pending requests and create new ones.

Impact: If the external adapter doesn't respond promptly, users are unable to submit another minting request because their code is deleted after the initial request. This could result in users losing their codes and not receiving their mystery box rewards.

Recommended Mitigation: Consider implementing a function that code recipients can invoke in the event of a request timeout. This function should internally call `ChainlinkClient:cancelChainlinkRequest` and include a callback to the `MysteryBox` contract to initiate a new request using the same data as the original. Essentially, this means reusing the code/user address and the previously generated random number for the new request.

Mode: Acknowledged.

7.4 Low Risk

7.4.1 Use low level `call()` to prevent gas griefing attacks when returned data not required

Description: Using `call()` when the returned data is not required unnecessarily exposes to gas griefing attacks from huge returned data payload. For example:

```
(bool sent, ) = address(operatorAddress).call{value: msg.value}("");
if (!sent) revert Unauthorized();
```

Is the same as writing:

```
(bool sent, bytes memory data) = address(operatorAddress).call{value: msg.value}("");
if (!sent) revert Unauthorized();
```

In both cases the returned data will have to be copied into memory exposing the contract to gas griefing attacks, even though the returned data is not required at all.

Impact: Contracts unnecessarily expose themselves to gas griefing attacks.

Recommended Mitigation: Use a low-level call when the returned data is not required, eg:

```
bool sent;
assembly {
    sent := call(gas(), receiver, amount, 0, 0, 0, 0)
}
if (!sent) revert Unauthorized();
```

Consider using ExcessivelySafeCall.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.5 Informational

7.5.1 Prevent duplicate `boxId` inputs to `MysteryBox::claimMysteryBoxes()`

Description: Consider preventing duplicate `boxId` inputs to `MysteryBox::claimMysteryBoxes()` as this may be exploitable under certain circumstances.

Impact: Attackers could use duplicate inputs to exploit token claiming.

Recommended Mitigation: Revert if duplicate inputs occur; `boxId` is unique so duplicate inputs are an obvious sign of a malicious attack.

Mode: Fixed in commit 85b2012, 3713107.

Cyfrin: Verified.

7.5.2 `MysteryBox::claimMysteryBoxes()` should return custom error when reverting due to `amountToClaim == 0`

Description: `MysteryBox::claimMysteryBoxes()` should return custom error when reverting due to `amountToClaim == 0`. Currently it returns `InsufficientEarnmBalance` which is the same error as if the contract had insufficient token balance for the mystery box being redeemed.

Impact: Misleading error is returned.

Recommended Mitigation: Return a custom error.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.5.3 Potential Risk of Price Volatility in EarnNM Token Due to Concentrated Mystery Box Rewards

Description: The current mechanism for distributing mystery box rewards in the system is based on randomness, which carries the risk of a large influx of tokens entering circulation within a short span. In particular, unusual situations might arise where a substantial number of high-value boxes (such as 1 mythical, 2 legendary, and 10 epic) are allocated over a brief period, like 1-2 days. Additionally, there's a possibility of minting a significant volume of boxes in a short duration. As a result, there's a possibility that all these boxes might release EarnM tokens simultaneously when their vesting period ends.

EarnM tokens are not fee-based tokens (e.g., token value linked to protocol fees) or any staking mechanisms to encourage token retention. In effect, there are no demand drivers and no supply dampeners in the current design.

Impact: Intense sell pressure, especially during a market downturn, may lead to price manipulation risks in liquidity pools. Such a significant price drop could incite panic among users, prompting them to redeem their mystery boxes notwithstanding the 50/90% haircut. This action could amplify the sell-off, potentially spiralling into a severe scenario akin to previous market collapses seen with tokens like Terra Luna.

Recommended Mitigation: Given the uncertainty surrounding the scale and reach of EarnM token liquidity pools, we recommend the team ensures sufficient liquidity to counterbalance potential sell pressure post-vesting. Proactive liquidity management could be crucial in stabilising token value during critical periods.

Mode: Acknowledged.

7.5.4 Centralisation risks as the reward code generator and the Chainlink node operator is the same entity

Description: The current system architecture for managing reward codes in MODE is centralized, with both code generation and Chainlink node operations controlled by the MODE team. The endpoint tracking and managing these codes is not public. Using Chainlink Any API under this setup adds limited value, as it's managed by a single node operator – the MODE team itself. This centralization undermines the potential benefits of a decentralized oracle network.

Impact: This setup leads to unnecessary complications and expenses, including LINK fees, without offering the decentralization benefits typically associated with Chainlink's infrastructure.

Recommended Mitigation: Two potential alternatives could be considered to address this issue:

1. **Engage an External Node Operator:** Delegate the reward code verification tasks to an external node operator. This approach would involve creating a function to call `Chainlink:setChainlinkOracle`, allowing future updates to the oracle. Making the endpoint public in the future would empower MODE to appoint new operators as needed.
2. **Simplify with In-House Tracking:** If the node operator remains the same as the code generation entity, consider simplifying the process. Maintain an on-chain mapping linking codes and addresses to their respective box amounts. Update this mapping each time `apiAddress` triggers `MysteryBox::associateOneTimeCodeToAddress` with the permissible box amount. This streamlined approach would bypass the need for Chainlink oracles and external adapters, reducing LINK fees and complexity while maintaining the current level of centralisation.

We acknowledge that the chosen design was driven by the intent to facilitate the minting of mystery boxes in a single transaction, given the gas limitations associated with VRF (Verifiable Random Function) operations. MODE team's approach was reasonable under these constraints.

Mode: Acknowledged.

7.6 Gas Optimization

7.6.1 Remove from storage `baseMetadataURI` as already stored in `ERC1155` and `name` as never used

Description: Remove from storage `baseMetadataURI` as already stored in `ERC1155` & `name` as never used.

Impact: Extra storage costs and extra gas to write these unnecessary values to storage.

Recommended Mitigation: Remove both `baseMetadataURI` & `name` from storage.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.6.2 Standardize `tierId` to either `uint8` or `uint256` avoiding constant conversions back and forth

Description: Standardize `tierId` to either `uint8` or `uint256` avoiding constant conversions back and forth.

Impact: Having different types for `tierId` means it has to be converted but also increases complexity and confusion as to why it is different in some places to others.

Recommended Mitigation: Standardize `tierId` to either `uint8` or `uint256`.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.6.3 Simplify `boxId` storage mappings as `boxId` is unique to addresses and tiers

Description: Since `boxId` is unique such that multiple address or tiers can never have the same `boxId`, at least 2 storage mappings could potentially be simplified: `addressToTierToBoxIdToBlockTs` & `addressToBoxIdToTier`.

Consider refactoring the other nested mappings to simplify and reduce complexity.

Impact: The storage mappings are already quite complex which is error-prone and the way these 2 are implemented will require more gas to read/write.

Recommended Mitigation: Simplify these mappings by taking advantage of the fact that `boxId` is unique to addresses & tiers.

Mode: Fixed in commit 85b2012, efa8199, 9c5ac66.

Cyfrin: Verified.

7.6.4 State variables should be cached in stack variables rather than re-reading them from storage

Description: State variables should be cached in stack variables rather than re-reading them from storage.

- `MysteryBox::fulfillRandomWords()` reads `vrfRequests[_requestId]` 3 times; consider reading it once into memory then reading from memory to avoid multiple storage reads.
- `MysteryBox::fulfillBoxAmount()` could cache `eaRequestToAddress[_requestId]` and also `delete addressToRandomNumber[sender]`
- `MysteryBox::_assignTierAndMint()` should have `uint256 newBoxId = ++boxIdCounter;` then use `newBoxId` in the rest of the function.

Impact: Gas optimization

Recommended Mitigation: State variables should be cached in stack variables rather than re-reading them from storage.

Mode: Fixed in commit 85b2012, c4c50ed, d5b14d8, 5df2b82.

Cyfrin: Verified.

7.6.5 Loop backwards in `MysteryBox::_determineTier()` to avoid multiple variables and simplify code

Description: Loop backwards in `MysteryBox::_determineTier()` to avoid multiple variables and simplify code.

Impact: Gas optimization and simpler code.

Recommended Mitigation: See description.

Mode: Fixed in commit 4d56069.

Cyfrin: Verified.

7.6.6 Simplify calculation in `MysteryBox::_calculateAmountToClaim()`

Description: Execute this line every time `return (tokens * (10**EARNM_DECIMALS)) / divisor;` deleting the other branch and the useless % calculation.

Impact: Gas optimization and cleaner, simpler code.

Recommended Mitigation: See description.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.6.7 Remove unused `category` from `MysteryBox::_calculateVestingPeriodPerBox()`

Description: Remove unused `category` from `MysteryBox::_calculateVestingPeriodPerBox()`.

Impact: Gas optimization & simpler, cleaner code.

Recommended Mitigation: See description.

Mode: Fixed in commit 85b2012.

Cyfrin: Verified.

7.6.8 Check `boxAmount < 100` only once before loop in `MysteryBox::_assignTierAndMint()`

Description: As `boxAmount` input is static, check `boxAmount < 100` only once before loop in `MysteryBox::_assignTierAndMint()`.

Impact: Gas optimization.

Recommended Mitigation: See description.

Mode: Fixed in commit 06a6a4f.

Cyfrin: Verified.