Eggstravaganza - Findings Report

Table of contents

Contest Summary

Sponsor: First Flight #37

Dates: Apr 3rd, 2025 - Apr 10th, 2025

See more contest details here (https://codehawks.cyfrin.io/c/2025-04-eggstravaganza)

Results Summary

Number of findings:

- High: 3
- Medium: 0
- Low: 0

High Risk Findings

H-01. Unauthorized NFT Withdrawal

Summary

Any user can assign themselves as the depositor of an NFT they do not own, allowing them to later withdraw it and effectively steal the asset.

Vulnerability Details

The `depositEgg` function lets *any* caller provide an arbitrary depositor address, without verifying that the caller owns the NFT or has deposited it properly

```
function depositEgg(uint256 tokenId, address depositor) public {
    require(eggNFT.ownerOf(tokenId) == address(this), "NFT not transferred to vault");
    require(!storedEggs[tokenId], "Egg already deposited");
    storedEggs[tokenId] = true;
    eggDepositors[tokenId] = depositor;
    emit EggDeposited(depositor, tokenId);
}
```

- The attacker can front-run a legitimate transfer and call `depositEgg(tokenId, attackerAddress)` to claim ownership of the NFT.

Impact

An attacker could claim and withdraw any NFT that gets transferred to the contract, even if they never owned it.

Tools Used

Manual review

Recommendations

Require the caller to own and transfer the NFT within the function

```
function depositEgg(uint256 tokenId) public {
    require(eggNFT.ownerOf(tokenId) == msg.sender, "Caller must be the owner");
    require(eggNFT.getApproved(tokenId) == address(this) || eggNFT.isApprovedForAll(msg.sender, address(this)), "Contract not approve
    require(!storedEggs[tokenId], "Egg already deposited");

    eggNFT.transferFrom(msg.sender, address(this), tokenId);

    storedEggs[tokenId] = true;
    eggDepositors[tokenId] = msg.sender;
    emit EggDeposited(msg.sender, tokenId);
}
```

H-02. Insecure Randomness for Game Logic

Summary

The contract uses insecure, predictable pseudo-random number generation to determine if a player finds an egg, making it susceptible to manipulation.

Vulnerability Details

```
uint256 random = uint256(
    keccak256(abi.encodePacked(block.timestamp, block.prevrandao, msg.sender, eggCounter))
) % 100;

if (random < eggFindThreshold) {
    ...
}
```

- The random number is based on `block.timestamp`, `block.prevrandao`, and `msg.sender`, which are **predictable** by users and miners.

- This allows **manipulation of input parameters** to consistently win the egg.

Impact

Players (especially bots or miners) can **predict or brute-force winning attempts**, severely compromising fairness and integrity of the game.

Tools Used

Manual review

Recommendations

- For simple games, use Chainlink VRF (Verifiable Random Function) for secure randomness.
- If using pseudo-random logic for demonstration, clearly document its insecurity and restrict for testing only.

H-03. Missing Access Control in depositEggToVault

Summary

The `depositEggToVault` function allows users to deposit NFTs into the EggVault contract. However, it does not enforce internal access control on the vault side and relies on an external input for identifying the depositor, which can be unsafe if misused or front-run.

Vulnerability Details

```
function depositEggToVault(uint256 tokenId) external {
    require(eggNFT.ownerOf(tokenId) == msg.sender, "Not owner of this egg");
    // The player must first approve the transfer on the NFT contract.
    eggNFT.transferFrom(msg.sender, address(eggVault), tokenId);
    eggVault.depositEgg(tokenId, msg.sender);
}
```

- Although the function verifies that `msg.sender` owns the NFT **before** transferring it to the vault, it passes the depositor (`msg.sender`) to `depositEgg`, which is a `public` method on the vault contract.
- If the `depositEgg` function on the EggVault does not enforce access control or ownership checks itself, a malicious actor could call it directly or front-run legitimate deposits to claim others' NFTs.
- The security of this function relies on the assumption that the `depositEgg` function behaves securely and isn't externally callable or misused—which is **not enforced here**.

Impact

An attacker could potentially claim ownership of NFTs deposited into the vault if the vault contract allows setting depositor values from external callers without proper verification. This could lead to loss of assets for legitimate players.

Tools Used

Manual Review

Recommendations

- Do not rely on external input for the depositor. The vault should determine the depositor from `msg.sender`.

- Remove the `msg.sender` argument and enforce ownership inside the vault contract.
- Alternatively, perform the full NFT transfer and depositor registration **atomically within the vault**, so external calls like `depositEgg` cannot be exploited