Eggstravaganza - Findings Report

Table of contents

Contest Summary

Sponsor: First Flight #37

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

See more contest details here

Results Summary

Number of findings:

- High: 1
- Medium: 0
- Low: 1

High Risk Findings

H-01. The searchForEgg() Function in the EggHuntGame Contract is Vulnerable to Random Number Manipulation, Allowing an Attacker to Gain an Unfair Advantage

Summery : The `searchForEgg` function in the `EggHuntGame` contract is vulnerable to random number manipulation. An attacker can change the `block.timestamp` to control the random number, making it meet the condition `random < eggFindThreshold`. This allows the attacker to easily get a unique Egg NFT, giving them an unfair advantage in the game. This weakness in the random number generation system allows the game to be unfairly manipulated

Proof Of Code

Demonstrating the Random Number Manipulation Attack on `EggHuntGame` Contract This test case demonstrates how an attacker can manipulate the random number generation in the `EggHuntGame` contract to gain an unfair advantage. The attack takes advantage of the fact that the game uses `block.timestamp` to generate random numbers, which can be manipulated.

Attack Scenario:

1. **Owner Starts the Game**: The game is started by the owner with a certain Duration (in this case, `80`).
2. **Attacker Manipulates the Timestamp**: The attacker (using a different address, `user1`) can then manipulate the `block.timestamp` using the `vm.warp()` function.
3. **Search for Egg**: The attacker then calls the `searchForEgg()` function. Due to the manipulated timestamp, the random number generated will meet the condition `random < eggFindThreshold`, allowing the attacker to get the unique Egg NFT.

```
function testmanipulatetherandom() external{

vm.prank(owner);
game.startGame(80);
address user1 = address(2);
vm.prank(user1);
vm.warp(10);
game.searchForEgg();
assert(game.eggsFound(user1) == 1);

}
```

Test Execution:

After running the test case, the attacker successfully receives a unique Egg NFT.

```
root@LAPTOP-6DCGCU3B:~/2025-04-eggstravaganza# forge test --mt
testmanipulatetherandom
[⠒] Compiling...
[⠃] Compiling 1 files with Solc 0.8.28
[⠊] Solc 0.8.28 finished in 1.30s
Compiler run successful!

Ran 1 test for test/EggHuntGameTest.t.sol:EggGameTest
[PASS] testmanipulatetherandom() (gas: 194829)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.76ms
(346.52µs CPU time)

Ran 1 test suite in 19.55ms (2.76ms CPU time): 1 tests passed, 0
failed, 0 skipped (1 total tests)
```

This proof of concept shows how easily an attacker can manipulate the game's random number generation and gain an unfair advantage by simply manipulating the timestamp. It highlights a critical flaw in the contract's design, which needs to be addressed to ensure fair gameplay

Tools Used

1) VS Code 2) Manual Review

Recommendations

Use Chainlink VRF for Secure Random Number Generation. To prevent manipulation and ensure fair gameplay, we recommend using Chainlink VRF (Verifiable Random Function) for generating random numbers in the `EggHuntGame` contract. Chainlink VRF provides a secure and tamper-proof way to generate random numbers, ensuring fairness for all players.

For more details on how to implement Chainlink VRF, you can find the official documentation here : [Link] (https://docs.chain.link/vrf)

Low Risk Findings

L-01. The mintEgg() function in the EggstravaganzaNFT contract has an unused return value, causing unnecessary gas consumption

Description

The `mintEgg()` function in the `EggstravaganzaNFT` contract is vulnerable due to an unused return value, which unnecessarily increases gas fees. To optimize, the return value should either be used or explicitly discarded

```
    function mintEgg(address to, uint256 tokenId) external returns
(bool) {
        require(msg.sender == gameContract, "Unauthorized minter");
        _mint(to, tokenId);
        totalSupply += 1;
       -> return true;
    }
```

Impact

The unused return value in the `mintEgg()` function increases gas fees.

Proof of code

Below is the test case that shows how gas fees are consumed by the unused return value:

- **Without Return Value**: Gas fees consumed: 80,913
- **With Return Value**: Gas fees consumed: 80,921

Tools Used

1) VS Code

2) Aderyn

Recommendations : If you don’t use the return value, either remove it or utilize it