# Dussehra Audit Report

Version 1.0

*Lulox*

June 16, 2024

# Dussehra Audit Report

Lulox

June 16, 2024

Prepared by: Lulox Lead Auditors:

- Lulox

## Table of Contents

## Protocol Summary

The protocol is a metaphor for the Dussehra festival, including a pseudo-raffle where users attempt to be selected as Ram to win a prize.

According to the protocol's README:

- The `Dussehra` protocol allows users to participate in the event of Dussehra. The protocol is divided into three contracts: `ChoosingRam`, `Dussehra`, and `RamNFT`. The `ChoosingRam` contract allows users to increase their values and select Ram, but only if they have not selected Ram before. The `Dussehra` contract allows users to enter the people who like Ram, kill Ravana, and withdraw their rewards. The `RamNFT` contract allows the `Dussehra contract` to mint Ram NFTs, update the characteristics of the NFTs, and get the characteristics of the NFTs.

## Disclaimer

The Lulox team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for 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 |
| | High | H | H/M | M |
| Likelihood | Medium | H/M | M | M/L |
| | Low | M | M/L | L |

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

# Audit Details

## Scope

```
#-- src
#----- ChoosingRam.sol
#----- Dussehra.sol
#----- RamNFT.sol
```

- Solc Version: `0.8.20`
- Chain(s) to deploy contract to:
  - Ethereum
  - zksync
  - Arbitrum
  - BNB

## Roles

Organizer - Organiser of the event and Owner of RamNFT contract User - User who wants to participate in the event Ram - The user who has selected Ram for the event

# Executive Summary

## Issues found

| Severity | Number of issues found |
|---|---|
| High | 3 |
| Medium | 2 |
| Low | 0 |
| Info | 0 |
| Gas | 0 |
| Total | 5 |

# Findings

## High

### [H-01] A free participant could steal the prize because RamNFT::mintRamNFT has no restrictions, allowing minting outside the Dussehra rules

**Relevant GitHub Links** https://github.com/Cyfrin/2024-06-Dussehra/blob/main/src/RamNFT.sol#L49

**Tools Used** Manual review

**Description:** `RamNFT::mintRamNFT` has no restrictions and opens the window to a free participant to steal the prize if it gets selected by `ChoosingRam::selectRamIfNotSelected`. `RamNFT::mintRamNFT` should be only called by `Dussehra::enterPeopleWhoLikeRam`

The "free" RamNFT could be selected as Ram when `ChoosingRam::selectRamIfNotSelected` is called, thus allowing the owner of the "free" RamNFT to withdraw the prize after `Dussehra::killRavana` is called using `Dussehra::withdraw`.

**Impact:** A player that hasn't paid the entrance fee can get to steal the prize intended for paid participants.

**Proof of Concept:** Any number of NFTs can be minted by a player, thus giving it an unfair advantage in the randomness by which the Ram is selected by `ChoosingRam::selectRamIfNotSelected`. In the following code I show how a player can mint a RamNFT and then be selected as Ram, thus stealing the prize.

PoC Include this test in test/Dussehra.t.sol

```
function test_freeRamNFTStealsPrizeAfterkillRavana() public
    participants {
        vm.startPrank(player3);
        ramNFT.mintRamNFT(player3);
        vm.stopPrank();

        assertEq(player3.balance, 0); // Starting balance
        assertEq(ramNFT.ownerOf(2), player3);
        assertEq(ramNFT.getCharacteristics(2).ram, player3);

        vm.warp(1728691200 + 1);
        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected();
        vm.stopPrank();

        assertEq(choosingRam.selectedRam(), player3);

```

```
17          vm.startPrank(player3);
18          dussehra.killRavana();
19          dussehra.withdraw();
20          vm.stopPrank();
21
22          // Ending balance (note that the participants modifier makes
                player1 and player2 enter paying the entrance fee)
23          assertEq(player3.balance, 1 ether);
24      }
```

**Recommended Mitigation:** Add a way to check if the Dussehra contract is the one calling the `RamNFT::mintRamNFT`.

#### [H-02] Weak randomness allows organiser to select which player to be Ram

**Relevant GitHub Links** https://github.com/Cyfrin/2024-06-Dussehra/blob/main/src/ChoosingRam.sol#L90

**Tools Used** Manual review

**Description:** Hashing `block.timestamp` and `block.prevrandao` together creates a predictable number, and a predictable number is not a good random number. A malicious organiser could use the function `ChoosingRam::selectRamIfNotSelected` to choose which RamNFT becomes `ChoosingRam::selectedRam`.

**Impact:** This would make the game rigged, allowing the organiser to choose a player of its own to be Ram and withdraw the prize.

**Recommended Mitigation:** Implement Chainlink VRF for true randomness

#### [H-03] Ravana can be killed twice to steal the whole prize

**Relevant GitHub Links** https://github.com/Cyfrin/2024-06-Dussehra/blob/main/src/Dussehra.sol#L67

**Tools Used** Manual review and Foundry test suite

**Description:** The `Dussehra::killRavana` function can be called more than once, because it doesn't have a check for `Dussehra::IsRavanKilled` or something that prevents this behavior. Also, the function can be called by anyone. A bad organiser could use an anonymous wallet to call this function twice and get all the money in the prize, leaving the chosen Ram without prize to withdraw.

**Impact:** All the money that this contract is intended to handle is at risk, not to be stolen away, but as a rug pull by the organiser.

**Proof of Concept:** As simple as that. The `Dussehra::killRavana` function can get called twice to distribute the whole prize.

PoC Include this test in Dussehra.t.sol:

```
function test_RavanaCanBeKilledTwiceToStealPrize() public participants {
        vm.warp(1728691200 + 1);
        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected();
        uint256 startingOrganiserBalance = organiser.balance;
        vm.stopPrank();

        vm.startPrank(player2);
        dussehra.killRavana();
        uint256 RamwinningAmount = dussehra.totalAmountGivenToRam();
        dussehra.killRavana();
        vm.stopPrank();

        assertEq(organiser.balance, startingOrganiserBalance + (RamwinningAmount * 2));
```

**Recommended Mitigation:** Include a check for `Dussehra::IsRavanKilled` in `Dussehra::killRavana` to prevent the function being called twice

```
    function killRavana() public RamIsSelected {
+       require(IsRavanKilled == false, "Ravana is already killed!");
        if (block.timestamp < 1728691069) {
            revert Dussehra__MahuratIsNotStart();
        }
        if (block.timestamp > 1728777669) {
            revert Dussehra__MahuratIsFinished();
        }
        IsRavanKilled = true;
        uint256 totalAmountByThePeople = WantToBeLikeRam.length * entranceFee;
        totalAmountGivenToRam = (totalAmountByThePeople * 50) / 100;
        (bool success,) = organiser.call{value: totalAmountGivenToRam}("");
        require(success, "Failed to send money to organiser");
    }
```

## Medium

### [M-01] Weak randomness allows player to increase characteristics without risking to lose a challenge

**Relevant GitHub Links** https://github.com/Cyfrin/2024-06-Dussehra/blob/main/src/ChoosingRam.sol#L51

**Tools Used** Manual review

**Description:** Hashing `block.timestamp`, `block.prevrandao` and `msg.sender` together creates a predictable number, and a predictable number is not a good random number. A malicious player could call `ChoosingRam::increaseValuesOfParticipants` without risking to lose a challenge, and achieve easily the highest characteristics and become `ChoosingRam::selectedRam`

**Impact:** None really, because the `ChoosingRam::increaseValuesOfParticipants` function doesn't set the `ChoosingRam::isRamSelected` boolean as true, making the function `ChoosingRam::selectRamIfNotSelected` the only one that matters for selecting Ram. But if that was patched, it would be a severe vulnerability that allows any player to become Ram.

**Recommended Mitigation:** Implement Chainlink VRF for true randomness

### [M-02] `ChoosingRam::increaseValuesOfParticipants` doesn't turn `ChoosingRam::isRamSelected` as true when selecting Ram

**Relevant GitHub Links** https://github.com/Cyfrin/2024-06-Dussehra/blob/main/src/ChoosingRam.sol#L65

**Tools Used** Manual review

**Description:** Lack of updating `ChoosingRam::isRamSelected` variable in the `ChoosingRam::increaseValuesOfParticipants` function allows for overwriting players as Ram, because the `ChoosingRam::RamIsNotSelected` modifier is not triggered.

**Impact:** After updating all characteristics of a Ram NFT with `ChoosingRam::increaseValuesOfParticipants` and getting stored as `ChoosingRam::selectedRam`, there's no updating `ChoosingRam::isRamSelected` to true, which allows for other calls to `ChoosingRam::increaseValuesOfParticipants` to the highest update to overwrite the players address as `ChoosingRam::selectedRam`.

No player is assured its victory, even when being selected as Ram, which breaks the purpose of the protocol.

**Proof of Concept:**

**Recommended Mitigation:** Add `isRamSelected = true;` after both `selectedRam = ramNFT.getCharacteristics(tokenIdOfChallenger).ram;` lines in the `ChoosingRam::increaseValuesOfParticipants` function.