Dussehra Security Review

Version 1.0

July 4, 2024

Conducted by:
Antonio Iliev, Security Researcher

Table of Contents

1 Disclaimer

Antonio Iliev 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.

2 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

2.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

2.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

3 Executive summary

Overview

Project Name	Dussehra
Repository	https://github.com/Cyfrin/2024-06-Dussehra
Methods	Manual review & testing

Scope

contracts/ChoosingRam.sol
contracts/Dussehra.sol
contracts/RamNFT.sol

Issues Found

Critical risk	o
High risk	1
Medium risk	1
Low risk	0
Informational	0

4 Findings

4.1 High risk

4.1.1 Anyone can Mint Ram NFT

Severity: *High risk*

Description: In DOCS is stated that `mintRamNFT - Allows the Dussehra contract to mint Ram NFTs`. However, that is not true, because there is not modifier.

Proof of Concept:

1. Create the following test case:

Code

```
    function test_anyoneCanMintNft() public {
        vm.startPrank(player1);
        ramNFT.mintRamNFT(player1);
        vm.stopPrank();
        assertEq(ramNFT.balanceOf(player1), 1);
    }
```

2. Run the test:

```
forge test --match-test test_anyoneCanMintNft
```

And the results are the following:

```
Ran 1 test for test/Dussehra.t.sol:CounterTest
[PASS] test_anyoneCanMintNft() (gas: 107267)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.46ms (307.96ms CPU
    time)
```

Recommendation: Consider adding variable and modifier that will check if msg.sender is Dussehra contract.

1. Add variable:

```
address public dussehraContract;
```

2. Add modifier:

```
    modifier onlyDussehraContract() {
        if (msg.sender != dussehraContract) {
            revert RamNFT__NotDussehraContract();
        }
        _;
    }
```

4.2 Medium risk

4.2.1 killRavana can be called multiple times, which will lead to not receiving the payment for Ram.

Severity: *Medium risk*

Description: In DOCS is stated that `killRavana---Allows users to kill Ravana, and the Organizer will receive half of the total amount collected in the event. This function will only work after 12 October 2024 and before 13 October 2024.` However, users can kill Ravana multiple times.

If we take a look at `killRavana` function:

```
    function killRavana() public RamIsSelected {
        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");
    }
```

We can see that `IsRavanKilled` is set to true, but we don't check if it is killed after calling the function after first time. This means that killRavana can be called multiple times and the organizer will collect the money.

Proof of Concept:

1. Create the following test case:

Code

```
    function test_killRavanaMultipleTimes() public participants {
        vm.warp(1728691200 + 1);
        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected();
        vm.stopPrank();

        vm.startPrank(player1);
        dussehra.killRavana();
        vm.stopPrank();

        assertEq(dussehra.IsRavanKilled(), true);
        assertEq(organiser.balance, 1 ether);

        vm.startPrank(player2);
        dussehra.killRavana();
        vm.stopPrank();
```

```
        assertEq(organiser.balance, 2 ether);
    }
```

2. Run `forge test --match-test test_killRavanaMultipleTimes`

The results are the following:

```
[PASS] test_killRavanaMultipleTimes() (gas: 410448)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.63ms (537.25ms CPU
    time)
```

Recommendation:

1. Consider adding modifier that will check if Ravan is alive.

```
    modifier RavanIsAlive() {
        require(!IsRavanKilled, "Ravan is dead!");
        _;
    }
```

2. Add the modifier to the function:

```
 function killRavana() public RamIsSelected RavanIsAlive {
}
```