Dussehra Protocol Audit Report

Version 1.0

mahithchigurupati.me

June 12, 2024

Dussehra Protocol Audit Report

Mahith Chigurupati

June 12, 2023

Prepared by: Mahith Chigurupati

Lead Auditors: - Mahith

Table of Contents

    * [H-2] Weak randomness in `ChoosingRam.sol`, allows anyone to choose the randomness that favour's them
    * [H-3] Anyone can win and withdraw ETH without paying any fees
    * [H-4] `selectRamIfNotSelected` can be called even after ram is selected
    * [H-5] Challenger and participant can be same, results in challenger wins everytime
  - Medium
    * [M-1] Events are not indexed, makes it difficult to recover incase of some unexpected situations
    * [M-2] organiser may fail to call `ChoosingRam::selectRamIfNotSelected`, resulting in no one winning reward
    * [M-3] winner can be overwritten by another winner
    * [M-4] Player can join even after event ends, resulting in no chance of winning and funds locked in contract forever
  - Low
    * [L-1] Using long strings in require statements, will result in more gas
    * [L-2] setting value to false in constructor, might increase gas cost a bit
    * [L-3] visibility of public functions must be changed to external, to allow other contracts to interact
    * [L-4] Simplify if-else statements to save gas and improve readability
    * [L-5] zero check for `RamNFT::setChoosingRamContract`, results in unintended behavior
    * [L-6] Boolean constants can be used directly and do not need to be compare to true or false
    * [L-7] Unnecessary modifier, results in more gas
  - Informational / Non-Critical
    * [I-1] Absence of Natspec documentation, resulting in poor code readability for anyone reading to understand the code
    * [I-2] Unchanged variables should be constant or immutable
    * [I-3] Not using best practices in naming convention for state variables, resulting in lack of better readability for developers
    * [I-4] Low Test Coverage is bad for protocol and might cause unintended behavior of the protocol
    * [I-5] Typo in the error message and variable name, thereby causes confusion on how users interpret

Protocol Summary

- Dussehra, a major Hindu festival, commemorates the victory of Lord Rama, the seventh avatar of Vishnu, over the demon king Ravana. The festival symbolizes the victory of good over evil, righteousness over wickedness. According to the epic Ramayana, Ravana kidnaps Rama's wife, Sita, leading to a brutal battle between Rama and his allies against Ravana and his forces. After a ten-day battle, Rama emerged victorious by slaying Ravana, marking the victory of virtue and the restoration of dharma. Dussehra is celebrated with grand processions, reenactments of Rama's victory, and the burning of effigies of Ravana, symbolizing the destruction of evil forces. It signifies the enduring significance of courage, righteousness, and the eventual victory of light over darkness.
- 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.

ChoosingRam.sol

This contract allows users to increase their values and select Ram if all characteristics are true. If the user has not selected Ram before 12 October 2024, then the Organizer can select Ram if not selected.

- `increaseValuesOfParticipants` allows users to increase their values(or characteristics) and become Ram for the event. The values will never be updated again after 12 October 2024.
- `selectRamIfNotSelected` - Allows the organizer to select Ram if not selected by the user.

Dussehra.sol

This contract allows users to enter the people who like Ram, kill Ravana, and withdraw their rewards.

`enterPeopleWhoLikeRam` allows users to enter the event like Ram by paying an entry fee and receiving the ramNFT.

- `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.
- `withdraw` - Allows ram to withdraw their rewards.

RamNFT.sol

This contract allows the Dussehra contract to mint Ram NFTs, update the characteristics of the NFTs, and get the characteristics of the NFTs.

- `setChoosingRamContract` - Allows the organizer to set the choosingRam contract.
- `mintRamNFT` - Allows the Dussehra contract to mint Ram NFTs.
- `updateCharacteristics` - Allows the ChoosingRam contract to update the characteristics of the NFTs.
- `getCharacteristics` - Allows the user to get the characteristics of the NFTs.
- `getNextTokenId` - Allows the users to get the next token id.

NFTs are minted with the following characteristics:

- `ram`: address of user
- `isJitaKrodhah`: false // *JitaKrodhah means one who has conquered anger*
- `isDhyutimaan`: false // *Dhyutimaan means one who is intelligent*
- `isVidvaan`: false // *Vidvaan means one who is knowledgeable*
- `isAatmavan`: false // *Aatmavan means one who is self-controlled*
- `isSatyavaakyah`: false // *Satyavaakyah means one who speaks the truth*

Disclaimer

SaiMahith Chigurupati 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

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

Scope

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

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

None

Issues found

Severity	Number of issues found
High	5

Severity	Number of issues found
Medium	4
Low	7
Info	5
Total	21

Findings

High

[H-1] Anyone can call `RamNFT::mintRamNFT`, violating the protocol requirement

Description: As per the requirement in Readme, `mintRamNFT()` function must be called only by `Dussehra` contract to mint Ram NFTs to the users, but there isn't anything that's restricting the function being called by anyone which is not an expected behavior.

Impact: Anyone can call the function to mint NFT to anyone which is a serious vulnerability of the protocol

Proof of Concept:

Anyone can mint any number of NFT's directly to anyone from `RamNFT::mintRamNFT()` without paying any entranceFee

code

```
function test__AnyoneCanMintDirectly() public {
        vm.startPrank(player1);
        ramNFT.mintRamNFT(player1); // mint NFT directly without paying
             1 ether via `Dussehra.sol`
        vm.stopPrank();

        assertEq(ramNFT.ownerOf(0), player1);
        assertEq(ramNFT.getCharacteristics(0).ram, player1);

        vm.startPrank(player1);
        ramNFT.mintRamNFT(player2); // can mint n number of NFT's
            directly without paying entranceFee via `Dussehra.sol` to
            anyone
        vm.stopPrank();

        assertEq(ramNFT.ownerOf(1), player2);
        assertEq(ramNFT.getCharacteristics(1).ram, player2);

        assertEq(ramNFT.getNextTokenId(), 2);
    }
```

Recommended Mitigation:

So, to be able to check if caller is `Dussehra` contract, we need to create a modifier and apply it on `mintRamNFT()` function

Make below code changes in `RamNFT.sol`

code

```

+   error RamNFT__NotDussehra(); // creating new custom error

+   address immutable i_dussehra // creating a new immutable variable
    to store address of Dussehra.sol contract address

+   modifier onlyDussehra() { // creating a new modifier to check if
    caller is Dussehra contract
+         if (msg.sender != i_dussehra) {
+             revert RamNFT__NotDussehra();
+         }
+         _;
+     }

-   constructor(uint256 _entranceFee, address _choosingRamContract,
    address _ramNFT)
+   constructor(uint256 _entranceFee, address _choosingRamContract,
    address _ramNFT, address _dussehra) {
        ...
+       i_dussehra = _dussehra
    }

-   function mintRamNFT(address to) public
+   function mintRamNFT(address to) external onlyDussehra // applying
    the modifier to execute before function call
```

[H-2] Weak randomness in `ChoosingRam.sol`, allows anyone to choose the randomness that favour's them

Description: Hashing `msg.sender`, `block.timestamp`, `block.prevrandao` together creates a predictable final number. A predictable number is not a good random number. Malicious users can manipulate these values or know them ahead of time to choose the randomness that gives them a favorable NFT characteristics.

Impact: `ChoosingRam::selectRamIfNotSelected` and `ChoosingRam::increaseValuesOfPartici` can be manipulated easily in a way that miner can submit these call transactions that can benifit him by picking his NFT or characteristics that can make him a winner.

Proof of Concept:

There are a few attack vectors here.

1. Validators can know ahead of time the `block.timestamp` and `block.difficulty` and use that knowledge to predict when / how to participate. See the solidity blog on prevrando here. `block.difficulty` was recently replaced with `prevrandao`.
2. Users can manipulate the `msg.sender` value to result in their index being the winner.

Example proof of code:

Paste below code in `Dessehra.t.sol` and run command - `forge test --mt test__manipulateTimesta`

code

```
function test__manipulateTimestamp() public participants {
        assertEq(ramNFT.ownerOf(0), player1); //player1 owns the token
            0

        vm.warp(1728691198 + 1); // executing the transaction exactly
            at a particular timestamp to get a predictable outcome of 0
        vm.startPrank(player1);
        uint256 random = uint256(keccak256(abi.encodePacked(block.
            timestamp, block.prevrandao, msg.sender))) % 2;
        vm.stopPrank();

        assertEq(random, 0);
        assertEq(ramNFT.getCharacteristics(random).ram, player1);

        assertEq(ramNFT.getCharacteristics(random).isJitaKrodhah, false
            ); // isJitaKrodhah is false initially

        vm.startPrank(player1);
        choosingRam.increaseValuesOfParticipants(0, 1); // executed
            with predictable random value
        vm.stopPrank();

        assertEq(ramNFT.getCharacteristics(random).isJitaKrodhah, true)
            ; // updated the value of isJitaKrodhah to true in favor of
            caller/miner
}
```

Using on-chain values as a randomness seed is a well-known attack vector in the blockchain space.

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

[H-3] Anyone can win and withdraw ETH without paying any fees

Description: since `RamNFT::mintRamNFT()` isn't restricted to only be minted by `Dussehra` contract, anyone can mint and join the protocol by directly calling `RamNFT::mintRamNFT()` and still be eligible to be chosen as a winner and be able to withdraw the reward even though he paid zero fees.

Impact: Anyone can mint an NFT without paying any fee but still be eligible to win and withdraw the reward

Proof of Concept:

Paste below code to `Dussehra.t.sol` and run command: `forge test --mt test__anyoneCanWinAndWi`

code

```
    function test__anyoneCanWinAndWithdrawWithoutPayingFee() public participants {
        assertEq(address(dussehra).balance, 2 ether); // two players entered the game and now protocol has 2 ether

        vm.startPrank(player3);
        ramNFT.mintRamNFT(player3); // attacker mints NFT directly from RamNFT::mintRamNFT() without paying entranceFee
        vm.stopPrank();

        vm.warp(1728691200 + 1); // Now, time passed and no one is selected as Ram
        uint256 random = uint256(keccak256(abi.encodePacked(block.timestamp, block.prevrandao))) % ramNFT.tokenCounter();

        assertEq(random, 2);
        assertEq(ramNFT.getCharacteristics(random).ram, player3);

        // so organiser calls function to choose one randomly
        // possibly, organiser can enter game and choose his token id predictably
        // also miner can submit transaction in a block that can benifit him by picking his NFT as winner
        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected();
        vm.stopPrank();

        assertEq(choosingRam.isRamSelected(), true);

        vm.startPrank(player3);
        dussehra.killRavana(); // player3 will be chosen as winner and kills ram
        vm.stopPrank();

        uint256 RamwinningAmount = dussehra.totalAmountGivenToRam();
        assertEq(RamwinningAmount, 1 ether);

        vm.startPrank(player3);
        dussehra.withdraw(); // player3 despite paying fee, wins and
            withdraws amount
        vm.stopPrank();

        assertEq(ramNFT.getCharacteristics(random).ram.balance,
            RamwinningAmount);
    }
```

Recommended Mitigation:

1. Do not use block.timestamp for random selection instead use chainlink VRF or any other oracles
2. Restrict `RamNFT::mintRamNFT` to be only called by `Dussehra.sol`

[H-4] `selectRamIfNotSelected` can be called even after ram is selected

Description: Once ram is selected, organiser or no one should be able to change ram again. If it can be changed then game becomes unfair and people loses trust in the protocol thereby results in failure of protocol.

Impact: ram once selected shouldn't be changed later, to maintain transparency.

Proof of Concept:

place below code in `Dussehra.t.sol` and run `forge test --mt test__RamCanBeChangedAfterSelec`

code

```
    function test__RamCanBeChangedAfterSelection() public participants
        {
        vm.warp(1728691198 + 1); // executing the transaction exactly
            at a particular timestamp to get a predictable outcome of 0
            for simplicity
        vm.startPrank(player1);
        uint256 random = uint256(keccak256(abi.encodePacked(block.
            timestamp, block.prevrandao, msg.sender))) % 2;
        vm.stopPrank();

        assertEq(random, 0);
        assertEq(ramNFT.getCharacteristics(random).ram, player1);
        vm.startPrank(player1);

        for (uint64 i = 0; i < 5; i++) {
            choosingRam.increaseValuesOfParticipants(0, 1); // lets assume player 1 is selected as ram
        }

        vm.stopPrank();

        assertEq(choosingRam.selectedRam(), player1); // selected ram is player 1
        assertEq(choosingRam.isRamSelected(), false);

        vm.warp(1728691199 + 1);
        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected(); // though player 1 is already selected, organiser can again call this function to change the selected ram to another player
        vm.stopPrank();

        assertEq(choosingRam.isRamSelected(), true);
        assertNotEq(choosingRam.selectedRam(), player1); // now, selected ram is not player 1
    }
```

Recommended Mitigation:

Make below code change in `ChoosingRam::increaseValuesOfParticipants`

```
    function increaseValuesOfParticipants(uint256 tokenIdOfChallenger, uint256 tokenIdOfAnyPerticipent)
        public
        RamIsNotSelected{
        ...
        if(random == 0){

        }
        else if (ramNFT.getCharacteristics(tokenIdOfChallenger).isSatyavaakyah == false) {
            ramNFT.updateCharacteristics(tokenIdOfChallenger, true, true, true, true, true);
            selectedRam = ramNFT.getCharacteristics(tokenIdOfChallenger).ram;
+           isRamSelected = true;
        }
    }
```

[H-5] Challenger and participant can be same, results in challenger wins everytime

Description th expected nft charateristics update logic must happen randomly by choosing either challenger or participant, but there is a bug by which both participant and challenger can be same resulting in always winning of caller/players.

Impact: Unfair advantage for players to always win

Proof of Concept:

Place below code in `Dussehra.t.sol` anf run `forge test --mt test__ChallengerIsParticipant`

```
    function test__ChallengerIsParticipant() public participants {
        assertEq(ramNFT.getCharacteristics(0).isJitaKrodhah, false);

        vm.startPrank(player1);
        choosingRam.increaseValuesOfParticipants(0, 0);
        vm.stopPrank();

        assertEq(ramNFT.getCharacteristics(0).isJitaKrodhah, true);
    }
```

Recommended Mitigation:

Place below code changes in `ChoosingRam.sol`

```

+   error ChoosingRam__ChallengerAndParticipantCantBeSame

    function increaseValuesOfParticipants(uint256 tokenIdOfChallenger,
       uint256 tokenIdOfAnyPerticipent)
        public
        RamIsNotSelected
    {
+       if(tokenIdOfChallenger == tokenIdOfAnyPerticipent){
+           revert ChoosingRam__ChallengerAndParticipantCantBeSame();
        }
    }
```

Medium

[M-1] Events are not indexed, makes it difficult to recover incase of some unexpected situations

Description: Events are necessary to be emitted for front end applications to read on-chain data of a contract or for developers to recover lost contract data due to unexpected sutuations like serious bugs or rekt.

Impact: logging is important for any application. failure to do so, will result in not knowing what happened and why behind it hence difficult to debug.

Proof of Concept:

create and emit enough events for the functions as shown below.

`increaseValuesOfParticipants` - emit to log when a user challenges to increase nft characteristics and emit if ram is selected. `selectRamIfNotSelected` - emit when ram is selected

`killRavana` - emit when ravana is killed `withdraw` - emit when ram withdraws

`RamNFT::setChoosingRamContract` - emit when ChoosingRam contract address is set. `RamNFT::updateCharacteristics` - emit when characteristics of an nft are updated.

Recommended Mitigation:

for eg. you can write events like this and emit them. make sure to follow (CEI) design patter to check, emit and then interact

```
+   event ChoosingRam__RamSelected(address indexed);

    function selectRamIfNotSelected() public RamIsNotSelected
        OnlyOrganiser {
        ...

         uint256 random = uint256(keccak256(abi.encodePacked(block.
             timestamp, block.prevrandao))) % ramNFT.tokenCounter();
+        emit ChoosingRam__RamSelected(selectedRam);
         selectedRam = ramNFT.getCharacteristics(random).ram;
         isRamSelected = true;
    }
```

[M-2] organiser may fail to call `ChoosingRam::selectRamIfNotSelected`, resulting in no one winning reward

Description: So, `ChoosingRam::selectRamIfNotSelected`, only organiser can call this function between the event time. if no winner is selected by `increaseValuesOfParticipants` & event ends and for some reason, organiser decides to act malicious and not call the function, then, no one will win and money is locked into the contract forever.

Impact: possibility of funds being locked in contract forever

Proof of Concept:

place below code in `Dussehra.t.sol` and run command: `forge test --mt test__fundsLocked`

```
function test__fundsLocked() public participants {
        vm.warp(1728777600 + 1); // 2 participants are already there
            with event finished

        vm.expectRevert();
        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected(); // selecting ram after event ends is not possible
        vm.stopPrank();

        vm.expectRevert();
        dussehra.killRavana(); // killing ravana after event ends is not possible

        vm.expectRevert();
        dussehra.withdraw(); // withdrawing after event ends is not possible

        assertEq(address(dussehra).balance, 2 ether); // funds are locked
    }
```

Recommended Mitigation:

1. Allow anyone to call `ChoosingRam::selectRamIfNotSelected` to not worry about trusting single organiser/entity.
2. Allow `ChoosingRam::selectRamIfNotSelected` to be called even after event ended as there isn't any downside for allowing so.
3. Better way to handle this situation is to use something like chainlink Automation or keepers which triggers upkeep and executes function `ChoosingRam::selectRamIfNotSelected` based on CRON job called as time based upkeep. you can read more about it at chainlink

third method is decentralised way to handle the situation without relying on anyone such that `ChoosingRam::selectRamIfNotSelected` will be called and executed at a particular time interval irrespective of anything.

[M-3] winner can be overwritten by another winner

Description Once ram is selected, organiser or no one should be able to change ram again. If it can be changed then game becomes unfair and people loses trust in the protocol thereby results in failure of protocol.

Impact ram once selected shouldn't be changed later, to maintain transparency.

Proof of Concepts

Paste below code in `Dussehra.t.sol` and run `forge test --mt test__RamCanBeChangedAfterSelec`

```
    function test__RamCanBeChangedAfterSelection() public participants {
        vm.warp(1728691196); // executing the transaction exactly at a
            particular timestamp to get a predictable outcome of 0 for
            simplicity
        uint256 random = uint256(keccak256(abi.encodePacked(block.
            timestamp, block.prevrandao, msg.sender))) % 2;

        assertEq(random, 0);
        assertEq(ramNFT.getCharacteristics(random).ram, player1);
        vm.startPrank(player1);

        for (uint64 i = 0; i < 5; i++) {
            choosingRam.increaseValuesOfParticipants(0, 1); // lets
                assume player 1 is selected as ram
        }

        vm.stopPrank();

        assertEq(choosingRam.selectedRam(), player1); // selected ram
            is player 1

        vm.warp(1728691197);

        vm.startPrank(player2);
        for (uint64 i = 0; i < 5; i++) {
            choosingRam.increaseValuesOfParticipants(1, 0); // player2
                is tring at different tiime to overwrite player 1
        }
        vm.stopPrank();

        assertEq(choosingRam.selectedRam(), player2); // now, selected
            ram is not player 1 instead it player 2, so next player can
            overwrite initial winner
    }
```

Recommended mitigation

Make below code change in `ChoosingRam::increaseValuesOfParticipants`

```
    function increaseValuesOfParticipants(uint256 tokenIdOfChallenger,
        uint256 tokenIdOfAnyPerticipent)
        public
        RamIsNotSelected{
        ...
        if(random == 0){

        }
        else if (ramNFT.getCharacteristics(tokenIdOfChallenger).
            isSatyavaakyah == false) {
                ramNFT.updateCharacteristics(tokenIdOfChallenger, true,
                    true, true, true, true);
                selectedRam = ramNFT.getCharacteristics(
                        tokenIdOfChallenger).ram;
+                   isRamSelected = true;
            }
        }
```

[M-4] Player can join even after event ends, resulting in no chance of winning and funds locked in contract forever

Description Players should only be allowed to join between a particular period to avoid players from joining after the event hence losing funds to be locked in contract forever

Impact: Funds locked in contract forever

Proof of Concept:

place below code in `Dussehra.t.sol` and run `forge test --mt test__PlayerCanJoinAfterEvent`

code

```
    function test__PlayerCanJoinAfterEvent() public {
        vm.startPrank(player1);
        vm.deal(player1, 1 ether);
        dussehra.enterPeopleWhoLikeRam{value: 1 ether}();

        vm.warp(1728777600);

        vm.startPrank(organiser);
        choosingRam.selectRamIfNotSelected(); // selecting ram
        vm.stopPrank();

        dussehra.killRavana(); // killing ravana

        vm.startPrank(player1);
        dussehra.withdraw(); // withdrawing by ram

        vm.startPrank(player2);
        vm.deal(player2, 1 ether);
        dussehra.enterPeopleWhoLikeRam{value: 1 ether}(); // joining
            after event ends

        assertEq(address(dussehra).balance, 1 ether);

        vm.warp(1728777669 + 1);

        vm.expectRevert();
        dussehra.killRavana(); // killing ravana after event ends is
            not possible

        vm.expectRevert();
        vm.startPrank(player1);
        dussehra.withdraw(); // ram cant withdraw as reward is claimed

        vm.startPrank(player2);
        vm.expectRevert();
        dussehra.withdraw(); // even player2 cant withdraw hence funds
            are locked
    }
```

Recommended Mitigation:

make below changes in `Dussehra.sol`

```
+   error Dussehra__EventEnded();

    function enterPeopleWhoLikeRam() external payable {

+       if(block.timestamp > 1728777669){
+           revert Dussehra__EventEnded();
+       }

    ...

    }
```

Low

[L-1] Using long strings in require statements, will result in more gas

Description: from solidity 0.8.4, we can use custom errors instead of long strings in require statements as custom errors takes less gas than the long strings in require statements.

Impact:

Proof of Concept:

Proof of code to show the difference in gas cost is as follows:

Place below code in `ChoosingRam.sol`

code

```

    error ChoosingRam__RamIsSelected();
    error ChoosingRam__NotOrganiser();

    modifier RamIsNotSelectedCustom() {
        if (!isRamSelected) {
            revert ChoosingRam__RamIsSelected();
        }
        _;
    }

    modifier OnlyOrganiserCustom() {
        if (ramNFT.organiser() == msg.sender) {
            revert ChoosingRam__NotOrganiser();
        }
        _;
    }

    function selectRamIfNotSelectedCustom() public
        RamIsNotSelectedCustom OnlyOrganiserCustom {
        if (block.timestamp < 1728691200) {
            revert ChoosingRam__TimeToBeLikeRamIsNotFinish();
        }
        if (block.timestamp > 1728777600) {
            revert ChoosingRam__EventIsFinished();
        }

        uint256 random = uint256(keccak256(abi.encodePacked(block.
            timestamp, block.prevrandao))) % ramNFT.tokenCounter();
        selectedRam = ramNFT.getCharacteristics(random).ram;
        isRamSelected = true;
    }
```

Place below code in `Dussehra.t.sol` and run the command - `forge test --mt test__GasCostIsLowForCustomErrors -vvv`

code

```
        function test__GasCostIsLowForCustomErrors() public {
            vm.txGasPrice(1);

            uint256 gasStart = gasleft();

            vm.expectRevert();
            choosingRam.selectRamIfNotSelected();

            uint256 gasEnd = gasleft();
            uint256 gasUsedFirst = (gasStart - gasEnd) * tx.gasprice;
            console.log("Gas cost of the long string errors", gasUsedFirst)
                ;

            uint256 gasStartCustom = gasleft();

            vm.expectRevert();
            choosingRam.selectRamIfNotSelectedCustom();

            uint256 gasEndCustom = gasleft();
            uint256 gasUsedCustom = (gasStartCustom - gasEndCustom) * tx.gasprice;
            console.log("Gas cost of custom errors", gasUsedCustom);

            assert(gasUsedFirst > gasUsedCustom);
        }
```

Recommended Mitigation:

Make below code changes in `ChoosingRam.sol`

code

```
+   error ChoosingRam__RamIsSelected();
+   error ChoosingRam__NotOrganiser();

    modifier RamIsNotSelected() {
-       require(!isRamSelected, "Ram is selected!");
+       if(!isRamSelected){
+           ChoosingRam__RamIsSelected();
+       }
        _;
    }

    modifier OnlyOrganiser() {
-       require(ramNFT.organiser() == msg.sender, "Only organiser can call this function!");
+       if(ramNFT.organiser() == msg.sender){
+           ChoosingRam__NotOrganiser();
+       }
        _;
    }
```

[L-2] setting value to false in constructor, might increase gas cost a bit

Description: In `ChoosingRam.sol::isRamSelected` is being set as false. Though this is fine, we are just setting the value to false which is already false by default. so, this might cost a little bit of more gas during deployment

Impact: increase in gas cost during deployment

Proof of Concept:

Proof to show that value of `isRamSelected` is **`false`** by default:

Place below code in `Dussehra.t.sol` and run command - `forge test --mt test__isRamSelectedValu`

code

```
    function test__isRamSelectedValueIsFalseByDefault() public {

        ramNFT = new RamNFT();
        choosingRam = new ChoosingRam(address(ramNFT));

        assertEq(choosingRam.isRamSelected(), false);
    }
```

Recommended Mitigation:

```diff
    constructor(address _ramNFT) {
-       isRamSelected = false;
        ramNFT = RamNFT(_ramNFT);
    }
```

[L-3] visibility of public functions must be changed to external, to allow other contracts to interact

Description: In `RamNFT.sol::mintRamNFT()` and `RamNFT.sol::updateCharacteristics()` are marked with visibility as `public` which costs more gas when called by external contracts. Since, these functions are not being accessed internally, its advisable to change the visibility to external to save gas

Also `RamNFT.sol::getCharacteristics` view function is being used by `ChoosingRam::increaseValuesOfParticipants()` function so it need to be external. Do note that view functions cost gas when a contract makes a call to read its storage.

Also, there is a possibility that users might use smart contract wallets/external smart contracts to interact with the protocol. Using public functions will cost more gas than external visibility of function.

Impact:

This will cost more gas to call these functions

Recommended Mitigation:

Make sure to do below code changes in `RamNFT.sol`

code

```diff
-   function updateCharacteristics(...) public
+   function updateCharacteristics(...) external

-   function mintRamNFT(address to) public
+   function mintRamNFT(address to) external

```

```diff
-   function setChoosingRamContract(address _choosingRamContract)
    public onlyOrganiser
+   function setChoosingRamContract(address _choosingRamContract)
    external onlyOrganiser

-   function getCharacteristics(uint256 tokenId) public view returns (
    CharacteristicsOfRam memory)
+   function getCharacteristics(uint256 tokenId) external view returns
    (CharacteristicsOfRam memory)
```

Make sure to do below code changes in `ChoosingRam.sol`

code

```diff
-   function selectRamIfNotSelected() public RamIsNotSelected
    OnlyOrganiser
+   function selectRamIfNotSelected() external RamIsNotSelected
    OnlyOrganiser

- function increaseValuesOfParticipants(...) public RamIsNotSelected
+ function increaseValuesOfParticipants(...) public RamIsNotSelected
```

Make sure to do below code changes in `Dussehra.sol`

code

```diff
-   enterPeopleWhoLikeRam () public payable
+   enterPeopleWhoLikeRam () external payable

- function killRavana() public RamIsSelected
+ function killRavana() external RamIsSelected

- withdraw() public RamIsSelected OnlyRam RavanKilled
+ withdraw() external RamIsSelected OnlyRam RavanKilled
```

[L-4] Simplify if-else statements to save gas and improve readability

Description: Any code is written once, but read a lot of times. so, we need to ensure that our code readability is very high such that anyone can easily understand our code.

Apart from it, We need to keep our code simple such that it consumes less gas. Its ideal to removed code thats used multiple times or code thats not called any time to simplify gas costs.

Impact: Save gas cost and improve readability

Proof of Concept:

Place below code in `ChoosingRam.sol`

Code

```
    function increaseValuesOfParticipants_Simplified(uint256
       tokenIdOfChallenger, uint256 tokenIdOfAnyParticipant)
        public
        RamIsNotSelected
    {
        if (tokenIdOfChallenger > ramNFT.tokenCounter()) {
            revert ChoosingRam__InvalidTokenIdOfChallenger();
        }
        if (tokenIdOfAnyParticipant > ramNFT.tokenCounter()) {
            revert ChoosingRam__InvalidTokenIdOfPerticipent();
        }
        if (ramNFT.getCharacteristics(tokenIdOfChallenger).ram != msg.
           sender) {
            revert ChoosingRam__CallerIsNotChallenger();
        }

        if (block.timestamp > 1728691200) {
            revert ChoosingRam__TimeToBeLikeRamFinish();
        }

        uint256 random = uint256(keccak256(abi.encodePacked(block.
           timestamp, block.prevrandao, msg.sender))) % 2;

        uint256 tokenId = random == 0 ? tokenIdOfChallenger :
           tokenIdOfAnyParticipant;
        RamNFT.CharacteristicsOfRam memory characteristics = ramNFT.
           getCharacteristics(tokenId);

        if (!characteristics.isJitaKrodhah) {
            ramNFT.updateCharacteristics(tokenId, true, false, false,
                false, false);
        } else if (!characteristics.isDhyutimaan) {
            ramNFT.updateCharacteristics(tokenId, true, true, false,
                false, false);
        } else if (!characteristics.isVidvaan) {
            ramNFT.updateCharacteristics(tokenId, true, true, true,
                false, false);
        } else if (!characteristics.isAatmavan) {
            ramNFT.updateCharacteristics(tokenId, true, true, true,
                true, false);
        } else if (!characteristics.isSatyavaakyah) {
            ramNFT.updateCharacteristics(tokenId, true, true, true,
                true, true);
            selectedRam = characteristics.ram;
        }
    }
```

Place below code in `Dussehra.t.sol` and run - `forge test --mt test__gasCostSimplified`

code

```
    function test__gasCostSimplified() public participants {
        vm.txGasPrice(1);

        uint256 gasStart = gasleft();

        vm.startPrank(player1);
        choosingRam.increaseValuesOfParticipants(0, 1);

        uint256 gasEnd = gasleft();
        uint256 gasUsedFirst = (gasStart - gasEnd) * tx.gasprice;
        console.log("Gas cost of complex if-else statements",
            gasUsedFirst);

        uint256 gasStartCustom = gasleft();

        vm.startPrank(player2);
        choosingRam.increaseValuesOfParticipants_Simplified(1, 0);

        uint256 gasEndCustom = gasleft();
        uint256 gasUsedCustom = (gasStartCustom - gasEndCustom) * tx.
            gasprice;
        console.log("Gas cost of simplified if-else", gasUsedCustom);

        assert(gasUsedFirst > gasUsedCustom);
    }
```

Recommended Mitigation:

Make below code changes in `ChoossingRam.sol`

Code

```
    function increaseValuesOfParticipants(uint256 tokenIdOfChallenger,
        uint256 tokenIdOfAnyParticipant)
        public
        RamIsNotSelected
    {
        ...

+        uint256 tokenId = random == 0 ? tokenIdOfChallenger :
tokenIdOfAnyParticipant;
+        RamNFT.CharacteristicsOfRam memory characteristics = ramNFT.
getCharacteristics(tokenId);

-       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);
-            } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).isVidvaan == false) {
-                ramNFT.updateCharacteristics(tokenIdOfChallenger, true, true, true, false, false);
-            } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).isAatmavan == false) {
-                ramNFT.updateCharacteristics(tokenIdOfChallenger, true, true, true, true, false);
-            } else if (ramNFT.getCharacteristics(tokenIdOfChallenger).isSatyavaakyah == false) {
-                ramNFT.updateCharacteristics(tokenIdOfChallenger, true, true, true, true, true);
-                selectedRam = ramNFT.getCharacteristics(tokenIdOfChallenger).ram;
-            }
-        } else {
-            if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).isJitaKrodhah == false) {
-                ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent, true, false, false, false, false);
-            } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).isDhyutimaan == false) {
-                ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent, true, true, false, false, false);
-            } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).isVidvaan == false) {
-                ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent, true, true, true, false, false);
-            } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).isAatmavan == false) {
-                ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent, true, true, true, true, false);
-            } else if (ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).isSatyavaakyah == false) {
-                ramNFT.updateCharacteristics(tokenIdOfAnyPerticipent, true, true, true, true, true);
-                selectedRam = ramNFT.getCharacteristics(tokenIdOfAnyPerticipent).ram;
-            }
-        }


+       if (!characteristics.isJitaKrodhah) {
+            ramNFT.updateCharacteristics(tokenId, true, false, false, false, false);
+        } else if (!characteristics.isDhyutimaan) {
+            ramNFT.updateCharacteristics(tokenId, true, true, false, false, false);
+        } else if (!characteristics.isVidvaan) {
+            ramNFT.updateCharacteristics(tokenId, true, true, true,
    false, false);
+        } else if (!characteristics.isAatmavan) {
+            ramNFT.updateCharacteristics(tokenId, true, true, true,
    true, false);
+        } else if (!characteristics.isSatyavaakyah) {
+            ramNFT.updateCharacteristics(tokenId, true, true, true,
    true, true);
+            selectedRam = characteristics.ram;
+        }
+    }
```

[L-5] zero check for `RamNFT::setChoosingRamContract`, results in unintended behavior

Description As mentioned, making `choosingRamContract` as immutable is ideal. Hence, it becomes impossible to update the contract address later if it was set as wrong address initially.

Impact it's a best practice to check if the contract address being set is valid address. for eg. not a zero address to ensure any unintended behavior of protocol like loosing funds or assets.

Proof of Concepts Place below code in`Dussehra.t.sol` and run - `forge test --mt test_zeroAddress`

```
    function test_zeroAddress() public participants {
        assertEq(address(dussehra).balance, 2 ether); // two players
            entered the game and now protocol has 2 ether

        vm.startPrank(organiser);
        ramNFT.setChoosingRamContract(address(0)); // setting zero
            address
        vm.stopPrank();

        vm.startPrank(player1);
        vm.expectRevert();
        choosingRam.increaseValuesOfParticipants(0, 1); // should
            revert as zero address is set as a result nft
            characteristics can't be updated
        vm.stopPrank();
    }
```

Recommended mitigation make below code changes in `ChoosingRam.sol`

```
+   error ChoosingRamContract__InvalidchoosingRamContractAddress();

    function setChoosingRamContract(address _choosingRamContract)
        public onlyOrganiser {
+         if(_choosingRamContract == address(0)){
```

```
+             revert
    ChoosingRamContract__InvalidchoosingRamContractAddress();
+         }

          choosingRamContract = _choosingRamContract;
      }
```

[L-6] Boolean constants can be used directly and do not need to be compare to true or false

Description Boolean constants can be used directly and do not need to be compare to true or false.

Impact gas cost

Recommended mitigation make changes similar to below one in `ChoosingRam.sol`

```
        ...
        ...

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

        ...
        ...
        ...
```

[L-7] Unnecessary modifier, results in more gas

description Ravan can only be killed after Ram is selected, so we dont need additional `RamIsSelected` modifier for `withdraw` function.

Impact: Gas cost

Recommended Mitigation:

```
-       function withdraw() public RamIsSelected OnlyRam RavanKilled
+       function withdraw() public OnlyRam RavanKilled
```

Informational / Non-Critical

[I-1] Absence of Natspec documentation, resulting in poor code readability for anyone reading to understand the code

Description: Any code is written once, but read a lot of times. so, we need to ensure that our code readability is very high such that anyone can easily understand our code.

Impact: No documentation may result in users not trusting the protocol

Recommended Mitigation: Write clear documentation everywhere is highly necessary

[I-2] Unchanged variables should be constant or immutable

Description: Variables that will need not have be changed later can be marked as immutable to improve security as well as save gas costs since immutable and constant variables are included in `bytecode` and does not occupy storage slots.

Recommended Mitigation: Make below changes to codebase to convert variables to immutable.

Immutable Instances:

for `ChoosingRam.sol`

```
    RamNFT public immutable i_ramNFT;
```

for `RamNFT.sol`

```
   address public immutable i_organiser;
```

for `Dussehra.sol`

```
    uint256 public immutable i_entranceFee;
    address public immutable i_organiser;
    RamNFT public immutable i_ramNFT;
    ChoosingRam public immutable i_choosingRamContract;
```

[I-3] Not using best practices in naming convention for state variables, resulting in lack of better readability for developers

Description Any code is written once, but read a lot of times. so, we need to ensure that our code readability is very high such that anyone can easily understand our code.

Hence, use `s_` prefix for state variables and `i_` for immutable variables for better readability and differentiating it from local variables

Recommended mitigation Make sure to improve the code as shown below:

for `RamNFT.sol`

code

```
-   uint256 public tokenCounter;
-   address public organiser;
-   address public choosingRamContract;
```

```diff

+   uint256 public s_tokenCounter;
+   address public i_organiser;
+   address public s_choosingRamContract;
```

for `ChoosingRam.sol`

code

```diff
-     bool public isRamSelected;
-     RamNFT public ramNFT;
-     address public selectedRam;

+     bool public s_isRamSelected;
+     RamNFT public i_ramNFT;
+     address public s_selectedRam;
```

for `Dussehra.sol`

code

```diff
-     address[] public WantToBeLikeRam;
-     uint256 public entranceFee;
-     address public organiser;
-     address public SelectedRam;
-     RamNFT public ramNFT;
-     bool public IsRavanKilled;
-     mapping(address competitor => bool isPresent) public peopleLikeRam
    ;
-     uint256 public totalAmountGivenToRam;
-     ChoosingRam public choosingRamContract;

+     address[] public s_WantToBeLikeRam;
+     uint256 public i_entranceFee;
+     address public i_organiser;
+     address public s_SelectedRam;
+     RamNFT public i_ramNFT;
+     bool public s_IsRavanKilled;
+     mapping(address competitor => bool isPresent) public
    s_peopleLikeRam;
+     uint256 public s_totalAmountGivenToRam;
+     ChoosingRam public i_choosingRamContract;
```

[I-4] Low Test Coverage is bad for protocol and might cause unintended behavior of the protocol

Description: The test coverage of the tests are below 90%. This often means that there are parts of the code that are not tested.

```
| File                 | % Lines         | % Statements    | % Branches
        | % Funcs         |
|----------------------|-----------------|-----------------|-----------------|---------

| src/ChoosingRam.sol | 61.36% (27/44) | 66.00% (33/50) | 39.47%
  (15/38) | 80.00% (4/5)    |
| src/Dussehra.sol    | 76.67% (23/30) | 78.79% (26/33) | 75.00%
  (15/20) | 85.71% (6/7)    |
| src/RamNFT.sol      | 84.62% (11/13) | 85.71% (12/14) | 50.00% (2/4)
      | 100.00% (8/8)   |
| Total               | 70.11% (61/87) | 73.20% (71/97) | 51.61%
  (32/62) | 90.00% (18/20) |
```

Recommended Mitigation: Increase test coverage to 90% or higher, especially for the `Branches` column.

[I-5] Typo in the error message and variable name, thereby causes confusion on how users interpret

Description: We need to make sure our protocol has no Typo's. If it isn't a Typo, it must be explained explicitly for users to understand it.

Impact: `ChoosingRam::tokenIdOfAnyPerticipent` is a misleading variable name, it should be changed to tokenIdOfAnyParticipant. Also, `error ChoosingRam__InvalidTokenIdOfParticipant()` must be corrected.

Recommended Mitigation:

Make below changes

```
-    error ChoosingRam__InvalidTokenIdOfPerticipent();
+    error ChoosingRam__InvalidTokenIdOfParticipant();
```

```
-    function increaseValuesOfParticipants(uint256 tokenIdOfChallenger,
    uint256 tokenIdOfAnyPerticipent)
+    function increaseValuesOfParticipants(uint256 tokenIdOfChallenger,
    uint256 tokenIdOfAnyParticipant)
```