



# Protocol Audit Report

Version 1.0

*Jaime Ribeiro Barrancos*

January 24, 2024

# Protocol Audit Report

Jaime Ribeiro Barrancos

February 15, 2024

Prepared by: Jaime Ribeiro Barrancos

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
  - Issues found
- Findings
  - HIGH
    - \* [H-1]: Reentrancy in `PuppyRaffle::refund` can drain balance.
    - \* [H-2]: Weak randomness makes the winner be influenced / predicted.
    - \* [H-3]: A winner can revert the transaction if he does not like the prize.
  - MEDIUM
    - \* [M-1]: `PuppyRaffle::enterRaffle` Looping through array is a potential DOS, since array can be so big that calling `enterRaffle` becomes too gas expensive.
  - LOW
    - \* [L-1]: `PuppyRaffle::getActivePlayerIndex` returns 0 for first user, causing that player to think they are inactive.

#### – INFO

- \* [I-1]: Solidity pragma should be specific, not wide
- \* [I-2]: Using an outdated version of solidity is dangerous
- \* [I-3]: Should check 0 address
- \* [I-4]: `PuppyRaffle::selectWinner` does not follow CEI.
- \* [I-5]: missing several events.

#### – GAS

- \* [G-1]: Reading from storage variables that are never changed - can be constant/immutable.
- \* [G-2]: Result of `players.length` should be cached.

## Protocol Summary

This project is to enter a raffle to win a dog NFT.

## Disclaimer

Jaime Barrancos made 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
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

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

## Audit Details

### Scope

```
1 src/PuppyRaffle.sol
```

### Roles

Owner / Deployer

### Issues found

Severity	Number of issues found
High	3
Medium	1
Low	1
Info	5
Gas	2

## Findings

### HIGH

#### [H-1]: Reentrancy in `PuppyRaffle::refund` can drain balance.

**Description:** `PuppyRaffle::refund` does not follow CEI, meaning it can be drained. A malicious contract can call `PuppyRaffle::refund`, which triggers its `receive` function which then calls `PuppyRaffle::refund`. This process can be repeated until all funds are drained

**Impact:** The balance of the contract is drained by a single player.

**Proof Of Concept:** 1 - Attacker enters raffle with malicious contract 2 - Attacker calls `PuppyRaffle::refund` 3 - Fallback / Receive functions are triggered in malicious contract and subsequently call `PuppyRaffle::refund` 4 - Process is repeated until contract has no more balance

**[H-2]: Weak randomness makes the winner be influenced / predicted.**

**Description:** Hashing on-chain data like `msg.sender`, `block.timestamp` and `block.difficulty` is not truly random and can be manipulated by miners / malicious attackers.

(This could be front-run)

**Impact:** A malicious actor could win the lottery and/or know who will win it.

**Proof Of Concept:**

- An attacker tries multiple addresses as `msg.sender` until he wins.

**[H-3]: A winner can revert the transaction if he does not like the prize.**

**Description:** A winner of the raffle can see what puppy he won. He can then use a malicious contract to prevent the winner from being selected, because the external call `(bool success,) = winner.call{value: prizePool}("");` can be made to a malicious smart contract. He can then call `PuppyRaffle::selectWinner` repeatedly until he gets a satisfying prize.

**Impact:** The best possible rarity can always be won in a raffle.

**Proof Of Concept:**

- An attacker calls `PuppyRaffle::enterRaffle` using a malicious contract
- The attacker calls `PuppyRaffle::selectWinner` with the malicious contract
- The metadata is available for anyone to see
- The attacker sees if the rarity is legendary, if not he reverts the `PuppyRaffle::selectWinner` call
- He tries again until it is legendary

**MEDIUM****[M-1]: `PuppyRaffle::enterRaffle` Looping through array is a potential DOS, since array can be so big that calling `enterRaffle` becomes too gas expensive.**

**Description:** The longer the `players` array is the more expensive becomes to call `enterRaffle`, so gas costs increase exponentially, benefitting whoever comes first.

**Impact:** An attacker can create an array so big, making it impossible to enter the raffle, guaranteeing his win.

**Proof Of Concept:**

```
1
2  function testAuditDOSEnterRaffle() public {
3      uint256 size = 400;
4      vm.txGasPrice(1);
5
6      address[] memory fakePlayerArray = new address[](size);
7      for (uint256 i = 0; i<size; i++) {
8          fakePlayerArray[i] = address(i);
9      }
10
11     uint256 gasStart = gasleft();
12     puppyRaffle.enterRaffle{value: entranceFee * fakePlayerArray.
        length}(fakePlayerArray);
13     uint256 gasEnd = gasleft();
14
15     uint256 gasUsed = gasStart - gasEnd;
16     console.log("first 400 users", gasUsed);
17
18
19     address[] memory newPlayer = new address[](1);
20     newPlayer[0] = address(401);
21     uint256 gasStart2 = gasleft();
22     puppyRaffle.enterRaffle{value: entranceFee}(newPlayer);
23     uint256 gasEnd2 = gasleft();
24
25     uint256 gasUsed2 = gasStart2 - gasEnd2;
26     console.log("next user", gasUsed2);
27
28
29     assert(gasUsed/400 < gasUsed2 );
30 }
```

**Recommended Mitigation:**

1. Consider allowing duplicates.
2. Use a mapping to check for duplicates, since it has a constant lookup time.

**LOW**

**[L-1]: PuppyRaffle::getActivePlayerIndex returns 0 for first user, causing that player to think they are inactive.**

**INFO**

**[I-1]: Solidity pragma should be specific, not wide**

Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.0;`, use `pragma solidity 0.8.0;`

- Found in src/PuppyRaffle.sol

**[I-2]: Using an outdated version of solidity is dangerous**

**Recommended Mitigation:**

Use solidity version 0.8.18.

**[I-3]: Should check 0 address**

Assigning values to address state variables without checking for `address(0)`.

**[I-4]: PuppyRaffle::selectWinner does not follow CEI.**

**[I-5]: missing several events.**

**GAS**

**[G-1]: Reading from storage variables that are never changed - can be constant/immutable.**

**[G-2]: Result of `players.length` should be cached.**