Aderyn Analysis Report

This report was generated by Aderyn, a static analysis tool built by Cyfrin, a blockchain security company. This report is not a substitute for manual audit or security review. It should not be relied upon for any purpose other than to assist in the identification of potential security vulnerabilities.

Table of Contents

- Summary
  - Files Summary
  - Files Details
  - Issue Summary
- High Issues
  - H-1: `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`
- Low Issues
  - L-1: Centralization Risk for trusted owners
  - L-2: Solidity pragma should be specific, not wide
  - L-3: Missing checks for `address(0)` when assigning values to address state variables
  - L-4: `public` functions not used internally could be marked `external`
  - L-5: Define and use `constant` variables instead of using literals
  - L-6: Event is missing `indexed` fields

Summary

Files Summary

Key	Value
.sol Files	2
Total nSLOC	138

Files Details

Filepath	nSLOC
src/PuppyRaffle.sol	138
Total	**138**

Issue Summary

Category	No. of Issues
High	1

Category	No. of Issues
Low	6

High Issues

H-1: `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`

Use `abi.encode()` instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. `abi.encodePacked(0x123,0x456)` => `0x123456` => `abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456)` => `0x0...1230...456`). Unless there is a compelling reason, `abi.encode` should be preferred. If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` instead. If all arguments are strings and or bytes, `bytes.concat()` should be used instead.

- Found in src/PuppyRaffle.sol Line: 261

  ```
              abi.encodePacked(
  ```

- Found in src/PuppyRaffle.sol Line: 265

  ```
                      abi.encodePacked(
  ```

Low Issues

L-1: Centralization Risk for trusted owners

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

- Found in src/PuppyRaffle.sol Line: 20

  ```
  contract PuppyRaffle is ERC721, Ownable {
  ```

- Found in src/PuppyRaffle.sol Line: 226

  ```
      function changeFeeAddress(address newFeeAddress) external onlyOwner {
  ```

L-2: 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 Line: 2

  ```
  pragma solidity ^0.7.6;
  ```

L-3: 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/PuppyRaffle.sol Line: 70

  ```
          feeAddress = _feeAddress;
  ```

- Found in src/PuppyRaffle.sol Line: 227

  ```
          feeAddress = newFeeAddress;
  ```

L-4: `public` functions not used internally could be marked `external`

Instead of marking a function as `public`, consider marking it as `external` if it is not used internally.

- Found in script/DeployPuppyRaffle.sol Line: 12

  ```
      function run() public {
  ```

- Found in src/PuppyRaffle.sol Line: 87

  ```
      function enterRaffle(address[] memory newPlayers) public payable {
  ```

- Found in src/PuppyRaffle.sol Line: 109

  ```
      function refund(uint256 playerIndex) public {
  ```

- Found in src/PuppyRaffle.sol Line: 253

```
    function tokenURI(uint256 tokenId) public view virtual override returns
(string memory) {
```

L-5: Define and use `constant` variables instead of using literals

If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

- Found in src/PuppyRaffle.sol Line: 167

  ```
          uint256 prizePool = (totalAmountCollected * 80) / 100;
  ```

- Found in src/PuppyRaffle.sol Line: 168

  ```
          uint256 fee = (totalAmountCollected * 20) / 100;
  ```

- Found in src/PuppyRaffle.sol Line: 189

  ```
          uint256 rarity = uint256(keccak256(abi.encodePacked(msg.sender,
  block.difficulty))) % 100;
  ```

L-6: Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

- Found in src/PuppyRaffle.sol Line: 59

  ```
      event RaffleEnter(address[] newPlayers);
  ```

- Found in src/PuppyRaffle.sol Line: 60

  ```
      event RaffleRefunded(address player);
  ```

- Found in src/PuppyRaffle.sol Line: 61

```
    event FeeAddressChanged(address newFeeAddress);
```