Puppy Raffle Security Review

Prepared by: Fabriziogianni7
Lead Auditors:

- Fabriziogianni7

Table of Contents

Protocol Summary

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

Call the enterRaffle function with the following parameters:
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.
Duplicate addresses are not allowed
Users are allowed to get a refund of their ticket & value if they call the refund function
Every X seconds, the raffle will be able to draw a winner and be minted a random puppy
The owner of the protocol will set a feeAddress to take a cut of the value, and the rest of the funds will be sent to the winner of the puppy.

[gh repo containing the code](https://github.com/Cyfrin/4-puppy-raffle-audit/tree/main

Disclaimer

Fabriziogianni7 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.

This security review is done for learning purpose following Cyfrin Updraft course.

This review has PuppyRaffle.sol contract as object. Please find here the gh repo containing the code.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L

	Impact		
Low	M	M/L	L

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

Audit Details

The findings described in this document correspond the following commit hash:

> 22bbbb2c47f3f2b78c1b134590baf41383fd354f

Scope

> ./src/PuppyRaffle.sol

Protocol Summary

Puppy Rafle is a protocol dedicated to raffling off puppy NFTs with variying rarities. A portion of entrance fees go to the winner, and a fee is taken by another address decided by the protocol owner.

Roles

- Owner: Deployer of the protocol, can change the address to which the fees are direted using changeFeeAddress
- Player: participant of raffle. can get into or get off the raffle

Executive Summary

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	1
Info	4
Gas	1
Total	11

Findings

High

[H-1] `PuppyRaffle::refund` and `PuppyRaffle::selectWinner` are a subject to reentrancy attacks, attackers can drain the contract

Impact: High
Likelyhood: High

Description:
in `PuppyRaffle::refund`:

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

        players[playerIndex] = address(0);
```

`PuppyRaffle::refund` change the state of the contract after making the external call to refund the player. this can allow a reentrancy attack

Impact:
A reentrancy attack can let the attacker drain the contract funds

Proof Of Code:

▶ Poc

```
    function test_ReentrancyAttack() public {
        // let some players enter
        address[] memory players = new address[](3);
        players[0] = address(1);
        players[1] = address(2);
        players[2] = address(3);

        puppyRaffle.enterRaffle{value: entranceFee * 3}(players);

        // create attacker
        ReentrancyAttacker reentrancyAttacker = new
ReentrancyAttacker(puppyRaffle);

        uint256 attackerBalanceBeforeAttack =
address(reentrancyAttacker).balance;
        uint256 raffleBalanceBeforeAttack = address(puppyRaffle).balance;

        // attack!
        reentrancyAttacker.attack{value: entranceFee}();

        uint256 attackerBalanceAfterAttack =
address(reentrancyAttacker).balance;
        uint256 raffleBalanceAfterAttack = address(puppyRaffle).balance;

        console.log("attacker Balance Before Attack %",
attackerBalanceBeforeAttack);
        console.log("raffle Balance Before Attack %",
```

```
raffleBalanceBeforeAttack);

        console.log("attacker balance after attack %",
attackerBalanceAfterAttack);
        console.log("raffle balance after attack %",
raffleBalanceAfterAttack);

        assert(raffleBalanceAfterAttack == 0);
    }
```

The execution of the above function gives the following logs, showing how the raffle contract can be drained

```
  Logs:
  attacker Balance Before Attack % 0
  raffle Balance Before Attack % 3000000000000000000
  attacker balance after attack % 4000000000000000000
  raffle balance after attack % 0
```

same happens here:

```
(bool success,) = winner.call{value: prizePool}("");
        require(success, "PuppyRaffle: Failed to send prize pool to
winner");
        _safeMint(winner, tokenId);
```

Recommended Mitigation:

- consider adding a `mutex lock`
- follow the CEI pattern, change the state before the external call

```
- payable(msg.sender).sendValue(entranceFee);
- players[playerIndex] = address(0);
+ players[playerIndex] = address(0);
+ payable(msg.sender).sendValue(entranceFee);
```

[H-2] `PuppyRaffle::selectWinner` select winners and rarity of nft with a weak randomness, attackers can repeat the ransaction until they get the number they want, (their player id)

Impact: High
Likelyhood: High

Description:
in `PuppyRaffle::selectWinner`:

```
    uint256 winnerIndex = uint256(keccak256(abi.encodePacked(msg.sender,
block.timestamp, block.difficulty))) % players.length;
```

and

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

the way the randomness is computed in those functions can be easily manipulated by other actors

Impact:
The Attacker can cheat and always win the raffle

Proof Of Concept:
see this article about randomnes: https://chain.link/education-hub/randomness-web3

Recommended Mitigation:

- consider using a service like Chainlink VRF

[H-3] `PuppyRaffle::selectWinner` summing up the fees can cause an overflow, making the fees returning back to 0

Impact: High
Likelyhood: High

Description:
in `PuppyRaffle::selectWinner`:

```
  totalFees = totalFees + uint64(fee);
```

fees will sum up until `totalFees` wil be 2^64, and setting fees to 2^64+1 will make the total count be 0

Impact:
It's a bug that can impeed the owner to properly keep the fee accounting

Proof Of Concept:

```
function test_overflow() public {
        // run 2 times the raffle and demo how fees decrease
        // let some players enter and
        address[] memory players = new address[](6);
        players[0] = address(1);
        players[1] = address(2);
        players[2] = address(3);
        players[3] = address(4);
```

```
        players[4] = address(5);
        players[5] = address(6);

        puppyRaffle.enterRaffle{value: entranceFee * players.length}
(players);

        vm.warp(block.timestamp + duration + 1);
        vm.roll(block.number + 1);

        puppyRaffle.selectWinner();

        uint64 totFeesStart = puppyRaffle.totalFees();

        // vm.warp(block.timestamp + duration + 1);
        // vm.roll(block.number + 1);
        // // second round
        uint256 totalPlayers = 89;
        address[] memory players2 = new address[](totalPlayers);
        for (uint256 i = 0; i < totalPlayers; i++) {
            address nthPlayer = address(i);
            players2[i] = nthPlayer;
        }
        puppyRaffle.enterRaffle{value: entranceFee * players2.length}
(players2);

        vm.warp(block.timestamp + duration + 1);
        vm.roll(block.number + 1);

        puppyRaffle.selectWinner();

        uint64 totFeesEnd = puppyRaffle.totalFees();
        console.log("totFeesEnd %s", totFeesEnd);
        console.log("totFeesStart %s", totFeesStart);
        assert(totFeesEnd < totFeesStart);
    }
```

this test logs:

```
Logs:
  totFeesEnd 553255926290448384
  totFeesStart 1200000000000000000
```

Recommended Mitigation:

- consider using uint256 also for fees

[H-4] `PuppyRaffle::enterRaffle` loop to check for duplicate can be a mean to make a Dos attack

Impact: High
Likelyhood: Medium

Description:
in `PuppyRaffle::enterRaffle`:

```
    for (uint256 i = 0; i < players.length - 1; i++) {
            for (uint256 j = i + 1; j < players.length; j++) {
                require(players[i] != players[j], "PuppyRaffle: Duplicate
player");
            }
        }
```

this nested loop causes the cost of the function to increase exponentially as the number of player increase

Impact:
The impact depends on the number of players joining the Raffle. the last players will be highly disadvantaged when joining the raffle because of high gas fees.

Proof Of Concept:

```
  function test_DosEnterRaffle() public {
        vm.txGasPrice(1);
        uint256 gasStart = gasleft();
        uint256 totalPlayers = 100;
        address[] memory players = new address[](totalPlayers);
        for (uint256 i = 0; i < totalPlayers; i++) {
            address nthPlayer = address(i);
            players[i] = nthPlayer;
        }
        puppyRaffle.enterRaffle{value: entranceFee * totalPlayers}
(players);
        uint256 gasEnd = gasleft();

        uint256 gasStart2 = gasleft();
        address[] memory players2 = new address[](totalPlayers);
        for (uint256 i = 0; i < totalPlayers; i++) {
            address nthPlayer = address(i + totalPlayers);
            players2[i] = nthPlayer;
        }
        puppyRaffle.enterRaffle{value: entranceFee * totalPlayers}
(players2);
        uint256 gasEnd2 = gasleft();
        console.log("gas after the 1st batch of user entered %s", gasStart
- gasEnd);
        console.log("gas after the 1st batch of user entered %s",
gasStart2 - gasEnd2);
        console.log(
            "the huge difference demonstrate that it cost exponentially
more gas to enter the raffle everytime there is a new user"
```

```
        );
        assert(gasStart - gasEnd < gasStart2 - gasEnd2);
    }
```

Recommended Mitigation:

- consider allowing duplicate addresses
- consider using enumerableSet from openZeppelin

Medium

// todo - unsafe cast is missing

[M-1] `PuppyRaffle::withdrawFees` requires the balance to be exactly the same amount of totalfees causing possible revert if the balance is more due to a selfdestructed contract with puppyraffle as target

Impact: Medium
Likelyhood: Medium

Description:
in `PuppyRaffle::withdrawFees`:

```
  function withdrawFees() external {
        // @AUDIT: mishandling of ETH -> if this contract balance is
bigger than total fees, you won't be able to withdraw the funds (eg
selfdistruct)
        require(address(this).balance == uint256(totalFees), "PuppyRaffle:
There are currently players active!");
```

if `address(this).balance` is more that `totalFees`, this function will revert

Impact:
This can impeed to withdraw the fees from the contract

Proof Of Concept:

```
function test_mishandling_of_eth() public {
        // make a contract with some eth, call selfdistruct and impeed
user from calling `withdrawFees`
        SuicideContract suicideContract = new
SuicideContract(puppyRaffle);

        address[] memory players = new address[](6);
        players[0] = address(1);
        players[1] = address(2);
        players[2] = address(3);
```

```
        players[3] = address(4);
        players[4] = address(5);
        players[5] = address(6);

        puppyRaffle.enterRaffle{value: entranceFee * players.length}
(players);

        suicideContract.attack();
        vm.expectRevert();
        puppyRaffle.withdrawFees();
    }
```

Recommended Mitigation:

- consider adding a fallback/receive function to manage ETH sent to the contract
- consider wrapping the ETH received

Low

[L-1] `PuppyRaffle::enterRaffle` does not check for zero address pontentially causing the winner assign to fail

Impact: Low
Likelyhood: Low

Description:
in `PuppyRaffle::enterRaffle`:
there is no check for `address(0)`. that means that if in `PuppyRaffle::selectWinner` this address is selected as winner, the contract will try to send ETH to 0 address, causing the call to revert.

Recommended Mitigation:

- add a check for 0 addresses in `PuppyRaffle::enterRaffle

Informational

[I-1] Floating pragmas

Description: Contracts should use strict versions of solidity. Locking the version ensures that contracts are not deployed with a different version of solidity than they were tested with. An incorrect version could lead to uninteded results.

https://swcregistry.io/docs/SWC-103/

Recommended Mitigation: Lock up pragma versions.

```
```

```
- pragma solidity ^0.7.6;
+ pragma solidity 0.7.6;
```

[I-2] Potentially erroneous active player index

Description: The getActivePlayerIndex function is intended to return zero when the given address is not active. However, it could also return zero for an active address stored in the first slot of the players array. This may cause confusions for users querying the function to obtain the index of an active player.

Recommended Mitigation: Return 2**256-1 (or any other sufficiently high number) to signal that the given player is inactive, so as to avoid collision with indices of active players.

[I-3] `PuppyRaffle:_isActivePlayer` is never used

`PuppyRaffle:_isActivePlayer` is never used. Consider removing it to make the contract more gas eficient and readable

[I-4] `PuppyRaffle:enterRaffle` consider add a check if the array sent by the function caller is empty

`PuppyRaffle:enterRaffle` doesn't check for empty arrays when user is calling the function, this does not cause the call to revert but will cause some waste of gas

Gas

[G-1] reading from storage variable is expensive

Description: Reading from storage variable is expensive. try minimizing that.

Example:

```
 for (uint256 i = 0; i < players.length - 1; i++) {
            for (uint256 j = i + 1; j < players.length; j++) {
                require(players[i] != players[j], "PuppyRaffle: Duplicate
player");
            }
        }
```

`players` is a storage variable and it's readed every time there is a new loop thick.

Mitigation:

```
+uint256 cachedplayers = players;
+for (uint256 i = 0; i < cachedplayers.length - 1; i++) {
+            for (uint256 j = i + 1; j < cachedplayers.+length; j++) {
+                require(cachedplayers[i] != cachedplayers+[j],
```

```
"PuppyRaffle: Duplicate player");
+            }
+        }
```

there are other similar cases accross the code. consider marking the storage variables with s_ prefix to identify the cases these are used.