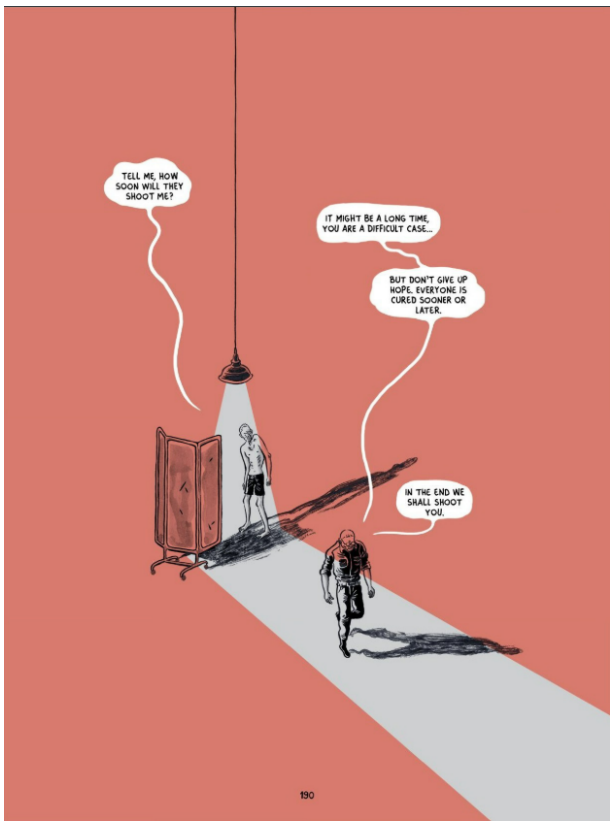

Protocol Audit Report

Version 1.0

Cyfrin.io

December 4, 2024

Protocol Audit Report

idir

November 29, 2024

Prepared by: idir badache Lead Auditors: idir badache (some day in the future) - xxxxxxx

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
  - Scope
  - Roles
  - Issues found
- Findings
  - High
    * [H-#] The `PuppyRaffle::refund()` is vulnerable to a reentrancy attack
    * [H-2] A weak randomness for selecting the raffle winner. The value can possibly be calculated before selection, breaking the Protocol.
    * M-3: `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`
    * [H-#] Unsafe casting, high probabilty of Loss of Funds.
  - Medium
    * [M-#] Probable DOS attack in `PuppyRaffle::enterRaffle()` function, exactly in the dublicate checking implementation.

        * [] There are no checks to ensure the receiver has a proper implementation of the fallback or receive functions. This could lead to wasted gas and disrupt the user experience.
        * [Info-2] A floating pragma version. Can lead to unexpected behaviour. And the version Used is unsafe.
        * [M-#]
    - Low
        * [L-#] The `PuppyRaffle::getActivePlayerIndex()` could be missleading to a user at index 0
    - Gas
        * There are no checks to ensure the receiver has a proper implementation of the fallback or receive functions. This could lead to wasted gas and disrupt the user experience.
        * Reading the length of the players array is from storage and consumes a lot of Gas.
- Low Issues
    - L-#: Missing checks for `address(0)` when assigning values to address state variables
    - L-5: Define and use `constant` variables instead of using literals
    - L-6: Event is missing `indexed` fields
    - L-7: Loop contains `require`/`revert` statements
    - G-9: Loop condition contains `state_variable.length` that could be cached outside.
    - L-10: Costly operations inside loops.
    - L-11: State variable could be declared constant
    - L-#: State variable changes but no event is emitted.

Protocol Summary

this protocol claims that it allow users to create and store there passwords completely onchain. creators of password have complete and only ownability and transparency.

Disclaimer

The IDIR dream 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
Likelihood	High	H	H/M	M
	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.

Scope

- In Scope:

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

- Solc Version: 0.7.6
- Chain(s) to deploy contract to: Ethereum

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	2
Medium	0

Severity	Number of issues found
Low	0
Total	2

Findings

High

[H-#] The `PuppyRaffle::refund()` is vulnerable to a reentrancy attack

Description: This is a very popular attack. The typical scenario is that the state is updated after a user has already received funds!!, which is wrong.

Impact: funds directly at risk!!, an attacker can drain the contract from its balance.

Recommended Mitigation: Any function that interacts with another contract by sending funds must ensure that the state is checked or updated before making the transfer.

Proof of Concept: will simulate an attack in a test. I've included comments to help clearify any ambiguities.

```
1  function test_reentrancyRefun() public {
2          address[] memory players = new address[](4);// The input array
               required by the enterRaffle function
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          ReentrancyAttacker attackContract = new ReentrancyAttacker(
               address(puppyRaffle));//this is the melicious contract the
               PuppyRaffle would send fund to.
10         address attackUser = makeAddr("attackUser");// probably the
               owner of the melicious contract
11         vm.deal(attackUser, 1 ether);//funding an address using foundry
                (just for testing)
12         uint256 startingAttackContractBalance = address(attackContract)
               .balance; //storing the blance of the attacker before the
               attack
13         uint256 startingContractBalance = address(puppyRaffle).balance;
               //same goes for the PuppyRaffle contract
14
```

```
15
16          vm.prank(attackUser);//initiating the attack
17          attackContract.refunding{value: entranceFee}();
18
19          console.log("starting attacker contract balance:",
                startingAttackContractBalance);
20          console.log("starting vulnerble contract balance: ",
                startingContractBalance);
21
22
23          console.log("after attack attacker contract balance:", address(
                attackContract).balance);
24          console.log("after attack vulnerble contract balance: ",
                address(puppyRaffle).balance);
25
26      }
```

[H-2] A weak randomness for selecting the raffle winner. The value can possibly be calculated before selection, breaking the Protocol.

Description: The randomness the game use to select a winner, is partially deterministic, meaning it can be predicted befor hand. for more details look for “pseudo random number generator”

Impact: A malicious user collaborating with a malicious validator can partially control the randomness, potentially manipulating the outcome and making the game unfair or unplayable for other participants.

Recommended Mittigation: Use chainLink VRF instead.

M-3: `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`

Use `abi.encode()` instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. `abi.encodePacked(0x123,0x456)` => `0x123456` => `abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456)` => `0x0...1230...456`). Unless there is a compelling reason, `abi.encode` should be preferred. If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` instead. If all arguments are strings and or bytes, `bytes.concat()` should be used instead.

2 Found Instances

- Found in src/PuppyRaffle.sol Line: 197

```
1               abi.encodePacked(
```

- Found in src/PuppyRaffle.sol Line: 201

```
1                                   abi.encodePacked(
```

[H-#] Unsafe casting, high probabilty of Loss of Funds.

Description: The range of values of a type uint64 bit is 0 to 18446744073709551615 this means that any number beyond tha maximum value woud cause an overflow so instead of 18446744073709551616, it would cycle back to 0.

Impact: In `PuppyRaffle::selectWinner()`, the assignment expression for totalFees contains an unsafe cast from a **uint256** to a **uint64** for the fee value. If the **fee** value before casting exceeds the maximum limit of **uint64** by even 1 bit, it would overflow during the cast and wrap back to 0. This can result in an incorrect or unintended value being assigned to **totalFees**. causing a loss of Funds.

Proof of Code: I've written a test to help clearify:

```
1     function test_arithmitic_overflow() public {
2         uint256 aFee = 20 ether; //2000000000000000000 wei this is how
              the value is represented
3         uint64 aFeeCastedTo64 = uint64(aFee); //casting
4         vm.expectRevert();
5         console.log("The uint256 value of the Fee is: ", aFee, "The
              uint64 value of the Fee is: ", aFeeCastedTo64);
6         assertEq(aFee, aFeeCastedTo64);
7     }
```

by running:

```
1 forge test --match-test test_arithmitic_overflow -vvv
```

if you can see the variable **aFee** after casting its value into **uint64**, it had become **1553255926290448384**.

Medium

[M-#] Probable DOS attack in `PuppyRaffle::enterRaffle()` function, exactly in the dublicate checking implementation.

Description: In the dublicate checking the nested `for loop` may cause fees to get extreamly expensive for users who enters late `PuppyRaffle::players`, because the nested loop would have to iterate through the entire `PuppyRaffle::players` for each element of that same array. the time complexity here is $O(n^2)$. you can read more here "https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/"

Impact & likelihood:: A melicious Hacker can input a large amount of data (as the input `newPlayers[]`), that would causing the protocol to reach the point where fees are too expensive for the normal user, thus DOS!. -The likelihood of this attack is probable, as adding a large number of users would be expensive, making the attack feasible only for powerful individuals.

Proof of Concept: In your `*.t.test` file add this test:

```
    function test_DosVulnerbility() public {
        address[] memory players = new address[](100);
        for (uint256 i = 0; i < 100; i++) {
            players[i] = address(i);
        }
        uint256 gasbefore = gasleft();
        puppyRaffle.enterRaffle{value: entranceFee * players.length}(
            players);
        uint256 gasAfter = gasleft();
        uint256 gasused = gasbefore - gasAfter;
        console.log("this is how much it cost to add first 100 players:
            ", gasused * 1);

        //the second 100 players
        address[] memory playerstwo = new address[](100);
        for (uint256 i = 0; i < 100; i++) {
            playerstwo[i] = address(i + 400);
        }
        uint256 gasbeforeSecondCall = gasleft();
        puppyRaffle.enterRaffle{value: entranceFee * players.length}(
            playerstwo);
        uint256 gasAfterSecondCall = gasleft();
        uint256 gasusedSecondCall = gasbeforeSecondCall -
            gasAfterSecondCall;
        console.log("this is how much it costs to add second 100
            players:", gasusedSecondCall * 1);
        assert(gasused < gasusedSecondCall);
    }
```

run the follwing command

```
forge test --match-test test_DosVulnerbility
```

Note: The output will show you that gas usage of the function call `PuppyRaffle::enterRaffle` before and after the second 100 players.

Rocommended mittigation: Use “mapping” data structure for both simplicity and gas efficiency.

The code:

```
contract example {

    mapping (address => bool) s_playersToMembership;
```

```
4
5       function enterRaffle(address[] memory newplayers) public {
6           for (uint256 i = 0; i <newplayers.length; i++) {
7               require(!s_playersToMembership[newplayers[i]], "No
                    duplicates allowed!!");// a checker to see if the player
                     is added already.
8               s_playersToMembership[newplayers[i]] = true;// adding the
                    player to the mapping if the his not.
9           }
10      }
11  }
```

[] There are no checks to ensure the receiver has a proper implementation of the fallback or receive functions. This could lead to wasted gas and disrupt the user experience.

Recommended Mitigation: use an initial call with low gas usage to make sure the receiver is set. or make rewards available through pull method (withdrawal).

[Info-2] A floating pragma version. Can lead to unexpected behaviour. And the version Used is unsafe.

Recommended Mitigation: Try removing the carrot key word instead of:

```
1 ^0.7.6
2 #use instead
3 0.7.6 #the compiler then only uses this version of solidity.
```

> *Regarding the Used Version:* The 0.7.6 version of Solidity is considered unsafe and not recommended for deployment. Solidity's security policy generally suggests using the latest version. Use 0.8.18 or higher instead.

[M-#]

Low

[L-#] The `PuppyRaffle::getActivePlayerIndex()` could be missleading to a user at index 0

Description: The function `PuppyRaffle::getActivePlayerIndex()` takes a player address as input and returns the player index in the `players` array. the problem here is that the function

returns `0` if the player is not active. this would lead to confusing since the user would not be able to determine wether his not active or active at index 0.

Impact & Mitigation: this would effect the user experience. To avoid this, an error message would suffice. example:

```
function getActivePlayerIndex(address player) external view returns (uint256) {
        for (uint256 i = 0; i < players.length; i++) {
            if (players[i] == player) {
                return i;
            }
        }
        revert("Error: You are not currently active");

}
```

Gas

There are no checks to ensure the receiver has a proper implementation of the fallback or receive functions. This could lead to wasted gas and disrupt the user experience.

Recommended Mitigation: use an initial call with low gas usage to make sure the receiver is set. or make rewards available through pull method (withdrawal). - Table of Contents - Protocol Summary - Disclaimer - Risk Classification - Scope - Roles - Issues found - Findings - High - [H-#] The `PuppyRaffle::refund()` is vulnerable to a reentrancy attack - [H-2] A weak randomness for selecting the raffle winner. The value can possibly be calculated before selection, breaking the Protocol. - M-3: `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()` - [H-#] Unsafe casting, high probabilty of Loss of Funds. - Medium - [M-#] Probable DOS attack in `PuppyRaffle::enterRaffle()` function, exactly in the dublicate checking implementation. - [] There are no checks to ensure the receiver has a proper implementation of the fallback or receive functions. This could lead to wasted gas and disrupt the user experience. - [Info-2] A floating pragma version. Can lead to unexpected behaviour. And the version Used is unsafe. - [M-#] - Low - [L-#] The `PuppyRaffle::getActivePlayerIndex()` could be missleading to a user at index 0 - Gas - There are no checks to ensure the receiver has a proper implementation of the fallback or receive functions. This could lead to wasted gas and disrupt the user experience. - Reading the length of the players array is from storage and consumes a lot of Gas. - Low Issues - L-#: Missing checks for `address(0)` when assigning values to address state variables - L-5: Define and use `constant` variables instead of using literals - L-6: Event is missing `indexed` fields - L-7: Loop contains `require`/`revert` statements - G-9: Loop condition contains `state_variable.length` that could be cached outside. - L-10: Costly operations inside loops. - L-11: State variable could be

declared constant - L-#: State variable changes but no event is emitted.

Reading the length of the players array is from storage and consumes a lot of Gas.

Low Issues

E32 ### L-#: 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

```
1           feeAddress = _feeAddress;
```

- Found in src/PuppyRaffle.sol Line: 168

```
1           feeAddress = newFeeAddress;
```

L-5: Define and use `constant` variables instead of using literals

If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

3 Found Instances

- Found in src/PuppyRaffle.sol Line: 132

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

- Found in src/PuppyRaffle.sol Line: 133

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

- Found in src/PuppyRaffle.sol Line: 139

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

L-6: Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the

maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

3 Found Instances

- Found in src/PuppyRaffle.sol Line: 53

```
1      event RaffleEnter(address[] newPlayers);
```

- Found in src/PuppyRaffle.sol Line: 54

```
1      event RaffleRefunded(address player);
```

- Found in src/PuppyRaffle.sol Line: 55

```
1      event FeeAddressChanged(address newFeeAddress);
```

L-7: Loop contains `require`/`revert` statements

Avoid `require` / `revert` statements in a loop because a single bad item can cause the whole transaction to fail. It's better to forgive on fail and return failed elements post processing of the loop

1 Found Instances

- Found in src/PuppyRaffle.sol Line: 87

```
1              for (uint256 j = i + 1; j < players.length; j++) {
```

G-9: Loop condition contains `state_variable.length` that could be cached outside.

Cache the lengths of storage arrays if they are used and not modified in for loops.

4 Found Instances

- Found in src/PuppyRaffle.sol Line: 86

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

- Found in src/PuppyRaffle.sol Line: 87

```
1              for (uint256 j = i + 1; j < players.length; j++) {
```

- Found in src/PuppyRaffle.sol Line: 111

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

- Found in src/PuppyRaffle.sol Line: 174

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

L-10: Costly operations inside loops.

Invoking SSTOREoperations in loops may lead to Out-of-gas errors. Use a local variable to hold the loop computation result.

1 Found Instances

- Found in src/PuppyRaffle.sol Line: 81

```
1          for (uint256 i = 0; i < newPlayers.length; i++) {
```

L-11: State variable could be declared constant

State variables that are not updated following deployment should be declared constant to save gas. Add the `constant` attribute to state variables that never change.

3 Found Instances

- Found in src/PuppyRaffle.sol Line: 38

```
1      string private commonImageUri = "ipfs://
          QmSsYRx3LpDAb1GZQm7zZ1AuHZjfbPkD6J7s9r41xu1mf8";
```

- Found in src/PuppyRaffle.sol Line: 43

```
1      string private rareImageUri = "ipfs://
          QmUPjADFGEKmfohdTaNcWhp7VGk26h5jXDA7v3VtTnTLcW";
```

- Found in src/PuppyRaffle.sol Line: 48

```
1      string private legendaryImageUri = "ipfs://
          QmYx6GsYAKnNzZ9A6NvEKV9nf1VaDzJrqDR23Y8YSkebLU";
```

L-#: State variable changes but no event is emitted.

State variable changes in this function but no event is emitted.

2 Found Instances

- Found in src/PuppyRaffle.sol Line: 125

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

- Found in src/PuppyRaffle.sol Line: 157

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