Protocol Audit Report

Version 1.0

Akshat

August 30, 2024

Protocol Audit Report

Akshat

August 30 , 2024

Prepared by: Akshat

Table of Contents

Protocol Summary

MyCut is a contest rewards distribution protocol which allows the set up and management of multiple rewards distributions, allowing authorized claimants 90 days to claim before the manager takes a cut of the remaining pool and the remainder is distributed equally to those who claimed in time!

Disclaimer

Akshat 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.

Details

- Blockchains: EVM Equivalent Chains Only
- Tokens: Standard ERC20 Tokens Only

Scope

All Contracts in `src` are in scope.

```
src/
#-- ContestManager.sol
#-- Pot.sol
```

Roles

- Owner/Admin (Trusted) - Is able to create new Pots, close old Pots when the claim period has elapsed and fund Pots
- User/Player - Can claim their cut of a Pot

Executive Summary

I had fun auditing this project. Being a Codehawks First Flight , I found it to be kinda easy and almost all the bugs were amongst the ones I have already seen. This was good practice of writing PoC's and report. It had around 100 nsloc and I did it in almost 1 day.

Issues found

Severity	Number of issues found
High	5
Medium	2
Low	2
Gas	1
Info	1
Total	11

Findings

High

[H-1] Owner might mistakenly fund a contest that has already ended , causing them to lose out on their funds

Description In the `ContestManager` contract , owner has functionality to close an existing pot. But after a pot closes , owner can still fund that contract (using `ContestManager::fundContest`). The owner would obviously not do this on purpose , but if they do , they have no way to get all those

funds back. Only thing they can do is call the `Pot::closePot` function. But this function only gives the owner 10% of the total rewards , so the owner will lose out on 90% of their funds.

One more problem is that `Pot::claimCut` has no controls to prevent users from claiming if the pot has ended. If the user didn't claim before owner called `closePot` , then this user shouldn't be able to claim afterwards. In normal circumstances when the pot is funded only once , this functionality is preserved as even if this user tries to claim , `claimCut` would revert as the contract wouldn't have funds(actually it would due to another bug in `closePot` , but let's ignore that for now). Now if the owner funds the pot again , this user can claim. Now the contract balance is less than `Pot::i_totalRewards` , and now if the owner calls `closePot` , they will get less than 10% if the total rewards , which is even worse

This description got a little messy since 3 bugs are into play , but to summarise: The owner may accidently fund a pot that has already closed , leading to them loosing 90% or more of the funded amount.

Impact Owner will loose money if they fund a closed pot.

Proof of Concepts 1. Owner creates and funds a pot 2. Player 1 claims his reward 3. Owner closes the pot 4. Owner funds it again 5. Player 2 claims 6. Owner closes the pot again but doesnt get all of their funds back

PoC

Place this into `TestMyCut.t.sol`

```
1     function test_FundingOfContractAfterItHasEnded() public
         mintAndApproveTokens{
2         vm.startPrank(user);
3         contest = ContestManager(conMan).createContest(players, rewards
              , IERC20(ERC20Mock(weth)), 4);
4         ContestManager(conMan).fundContest(0);
5         vm.stopPrank();
6
7         vm.startPrank(player1);
8         Pot(contest).claimCut();
9         vm.stopPrank();
10
11        vm.warp(91 days);
12
13        vm.startPrank(user);
14        ContestManager(conMan).closeContest(contest);
15        vm.stopPrank();
16
17        // now fund it again
18
19        vm.prank(user);
20        ContestManager(conMan).fundContest(0);
```

```
21
22          uint256 balanceBefore = IERC20(ERC20Mock(weth)).balanceOf(
                player2);
23
24          vm.startPrank(player2);
25          Pot(contest).claimCut();
26          vm.stopPrank();
27
28          uint256 balanceAfter = IERC20(ERC20Mock(weth)).balanceOf(
                player2);
29
30          assert(balanceAfter - balanceBefore == 1);
31
32
33          uint256 userBalanceBefore = IERC20(ERC20Mock(weth)).balanceOf(
                user);
34
35          vm.startPrank(user);
36          ContestManager(conMan).closeContest(contest);
37          vm.stopPrank();
38
39          uint256 userBalanceAfter = IERC20(ERC20Mock(weth)).balanceOf(
                user);
40
41          assert(userBalanceAfter - userBalanceBefore < totalRewards);
42
43      }
```

Recommended mitigation Keep track of which contests have ended , so the owner cannot fund a closed pot. Make a mapping of address to boolean for the same.

```
1  +    mapping(address pot => bool isClosed) public isClosed;
2  +    error ContestManager__CannotFundClosedContest();
3       .
4       .
5       .
6
7       function fundContest(uint256 index) public onlyOwner {
8           Pot pot = Pot(contests[index]);
9  +        if(isClosed[address(pot)]){
10 +            revert ContestManager__CannotFundClosedContest();
11 +        }
12          .
13          .
14          .
15      }
16      .
17      .
18      .
19
20      function closeContest(address contest) public onlyOwner {
```

```
21 +        if(!isClosed[contest]){
22 +            isClosed[address(pot)] = true;
23 +            _closeContest(contest);
24 +        }
25 -        _closeContest(contest);
26      }
```

[H-2] `ContestManager::createContest` takes in a `rewards` array and a `totalRewards` parameter , but doesn't check to see whether the rewards sum up to the total rewards. If sum is less , this makes some users unable to claim their rewards

Description `ContestManager::createContest` is used by owner to create a new contest . 2 of its parameters are : 1. `rewards` - array of rewards to be distributed to players 2. `totalRewards` - (should be ->) the sum of all the rewards in the `rewards` array

But this function doesn't check to see whether sum of all the rewards in the `rewards` array is actually equal to `totalRewards` or not. Consider 3 cases:

1. `totalRewards` > sum
    1. All users can claim
    2. Leftover rewards distributed via `Pot::claimPot` function
    3. But owner wouldn't wanna give away more rewards than what is specified in the `rewards` array . so this scenario is unwanted , even though it doesn't revert anywhere.
2. `totalRewards` == sum
    1. Everything works normally
3. `totalRewards` < sum
    1. Some users may face reverts while claiming since contract doesn't have as much balance as it is supposed to have
    2. If somebody doesn't claim and owner calls `claimPot` , this call will go through without reverts as it works on ratio calculation and not absolute values
    3. But obviously this scenario is unwanted as users weren't able to claim what they deserved.

The only case that the protocol intends to handle is case no. 2 , so we should only allow the owner to create a pot which corresponds to case 2 , i.e. `totalRewards` == sum

Impact If owner doesn't input `totalRewards` correctly , the owner or users may lose out on funds

Proof of Concepts I have written 4 tests to prove cases 1 and 3

PoC

Place these tests into `TestMyCut.t.sol`

```
    function test_TotalRewardsBreaksContract() public
        mintAndApproveTokens{
        vm.startPrank(user);
        contest = ContestManager(conMan).createContest(players, rewards
            , IERC20(ERC20Mock(weth)), 3);
        ContestManager(conMan).fundContest(0);
        vm.stopPrank();

        vm.startPrank(player1);
        Pot(contest).claimCut();
        vm.stopPrank();
        vm.startPrank(player2);
        vm.expectRevert();
        Pot(contest).claimCut();
        vm.stopPrank();
    }

    function test_TotalRewardsBreaksContract_2() public
        mintAndApproveTokens{
        vm.startPrank(user);
        contest = ContestManager(conMan).createContest(players, rewards
            , IERC20(ERC20Mock(weth)), 2);
        ContestManager(conMan).fundContest(0);
        vm.stopPrank();

        vm.startPrank(player1);
        vm.expectRevert();
        Pot(contest).claimCut();
        vm.stopPrank();
    }

    function test_TotalRewards() public mintAndApproveTokens{
        vm.startPrank(user);
        contest = ContestManager(conMan).createContest(players, rewards
            , IERC20(ERC20Mock(weth)), 3);
        ContestManager(conMan).fundContest(0);
        vm.stopPrank();

        vm.startPrank(player2);
        Pot(contest).claimCut();
        vm.stopPrank();

        vm.warp(91 days);

        vm.startPrank(user);
        ContestManager(conMan).closeContest(contest); // doesnt break
            as works on ratio system , 10% of remaining balance to owner
             , rest to claimers.
        vm.stopPrank();
```

```
43      }
44
45      function test_TotalRewards_2() public mintAndApproveTokens{
46          vm.startPrank(user);
47          contest = ContestManager(conMan).createContest(players, rewards
                , IERC20(ERC20Mock(weth)), 100);
48          ContestManager(conMan).fundContest(0);
49          vm.stopPrank();
50
51          vm.startPrank(player1);
52          Pot(contest).claimCut();
53          vm.stopPrank();
54          vm.startPrank(player2);
55          Pot(contest).claimCut();
56          vm.stopPrank();
57
58          vm.warp(91 days);
59
60          vm.startPrank(user);
61          ContestManager(conMan).closeContest(contest); // doesnt break
                as works on ratio system , 10% of remaining balance to owner
                 , rest to claimers.
62          vm.stopPrank();
63      }
```

Recommended mitigation Add a check to see if sum of values of `rewards` array equals `totalRewards` in `ContestManager`

```
1  +    error ContestManager__TotalRewardsIncorrect();
2       .
3       .
4       .
5       function createContest(address[] memory players, uint256[] memory
            rewards, IERC20 token, uint256 totalRewards)
6           public
7           onlyOwner
8           returns (address)
9       {
10          Create a new Pot contract
11
12 +        uint256 sum = 0;
13 +        uint256 length = rewards.length;
14 +        for(uint i=0;i<length;i++){
15 +            sum+=rewards[i];
16 +        }
17 +        if(sum != totalRewards){
18 +            revert ContestManager__TotalRewardsIncorrect();
19 +        }
20          .
21          .
22          .
```

```
23      }
```

[H-3] `Pot::claimCut` should have a control to prevent users from claiming after pot has ended , else if pot gets some balance due to some reason , these users may claim if they didn't claim already

Description The documentation clearly states that users can claim before the 90 day deadline. After the deadline , the owner takes their cut , and distributes remaining funds to the people who claimed in time by calling the `Pot::closePot` function

But the players who didn't claim in time , can still call the `Pot::claimCut` function after pot has ended. If the contract has no balance then this call will revert , but if somehow contract gets some balance , then this call will go through and these users can get their rewards, which is obviously not intended.

Impact Players who didn't claim in time , can claim after pot has closed if the contract somehow contains some balance

Recommended mitigation Make a boolean variable which keeps track if `closePot` has been called , and this variable can be used to revert the `claimCut` call if pot has ended.

```
1  +    error Pot__CannotClaimAsPotHasEnded();
2  +    bool public hasEnded;
3       .
4       .
5       .
6       function claimCut() public {
7  +        if(hasEnded){
8  +            revert Pot__CannotClaimAsPotHasEnded();
9  +        }
10          .
11          .
12          .
13      }
14      function closePot() external onlyOwner {
15          .
16          .
17          .
18 +        hasEnded = true;
19      }
```

[H-4] `Pot::closePot` has erroneous math , causing claimants to get less rewards and some money to be left in the contract

Description The docs clearly state that when pot has to be closed and funds are left, manager takes his cut and remaining balance has to be distributed among those who claimed in time. Look at the following line from `closePot` function:

```
    function closePot() external onlyOwner {
        if (block.timestamp - i_deployedAt < 90 days) {
            revert Pot__StillOpenForClaim();
        }
        if (remainingRewards > 0) {
            uint256 managerCut = remainingRewards / managerCutPercent;
            i_token.transfer(msg.sender, managerCut);

=>          uint256 claimantCut = (remainingRewards - managerCut) /
    i_players.length;
            for (uint256 i = 0; i < claimants.length; i++) {
                _transferReward(claimants[i], claimantCut);
            }
        }
    }
```

`claimantCut` is being found out by divinding the remaining balance ((`remainingRewards - managerCut`)) by the total number of players (`i_players.length`) . This is contradictory to the docs , as if the remaining balance has to distributed equally among the `claimants` , then remaining balance should be divided by the total number of claimants , which is `claimants.length`

Due to this wrong calculation , claimants get less rewards than they should and also some funds are left in the contract

Impact Due to this wrong calculation , claimants get less rewards than they should and also some funds are left in the contract

Proof of Concepts 1. Owner creates and funds the pool 2. Player 1 claims 3. Deadline passes 4. Owner calls `closePot` 5. Owner gets his 10% (no bug here) 6. There was only 1 claimer , and he should have gotten all the remaining balance of `450` , but since this got divided by 2 (num of players) , he only got 225 7. The contract , which should’ve been empty , still contains some money.

PoC

Paste this into `TestMyCut.t.sol`

```
    function test_ClosePotHasWrongMath() public mintAndApproveTokens {
        vm.startPrank(user);
        rewards = [500, 500];
        totalRewards = 1000;
```

```
5           contest = ContestManager(conMan).createContest(players, rewards
                , IERC20(ERC20Mock(weth)), totalRewards);
6           ContestManager(conMan).fundContest(0);
7           vm.stopPrank();
8
9           vm.startPrank(player1);
10          Pot(contest).claimCut();
11          vm.stopPrank();
12
13          uint256 claimantBalanceBefore = ERC20Mock(weth).balanceOf(
                player1);
14          uint256 ownerBalanceBefore = ERC20Mock(weth).balanceOf(conMan);
15
16          vm.warp(91 days);
17
18          vm.startPrank(user);
19          ContestManager(conMan).closeContest(contest);
20          vm.stopPrank();
21
22          uint256 claimantBalanceAfter = ERC20Mock(weth).balanceOf(
                player1);
23          uint256 ownerBalanceAfter = ERC20Mock(weth).balanceOf(conMan);
24
25          assert(ownerBalanceAfter - ownerBalanceBefore == 50); // no bug
                here
26
27          // assert(claimantBalanceAfter > claimantBalanceBefore);
28          // assert(claimantBalanceAfter - claimantBalanceBefore == 450);
                --> fails due to bug
29          assert(claimantBalanceAfter - claimantBalanceBefore == 225);
30
31          assert(ERC20Mock(weth).balanceOf(contest) == 225 ); // has non
                zero balance left
32      }
```

Recommended mitigation Change the way `claimantCut` is calculated:

```
1       function closePot() external onlyOwner {
2           if (block.timestamp - i_deployedAt < 90 days) {
3               revert Pot__StillOpenForClaim();
4           }
5           if (remainingRewards > 0) {
6               uint256 managerCut = remainingRewards / managerCutPercent;
7               i_token.transfer(msg.sender, managerCut);
8
9   -           uint256 claimantCut = (remainingRewards - managerCut) /
        i_players.length;
10  +           uint256 claimantCut = (remainingRewards - managerCut) /
        claimants.length;
11              for (uint256 i = 0; i < claimants.length; i++) {
12                  _transferReward(claimants[i], claimantCut);
```

```
13              }
14          }
15      }
```

[H-5] `ContestManager` is the owner of `Pot` , so `managerCut` goes to `ContestManager` , and the person who deployed `ContestManager` has no way getting these funds from the `ContestManager` contract and these funds are stuck here.

Description A person (let, Sam) deploys the `ContestManager` contract. Now , `ContestManager` deploys the `Pot` contract , so `ContestManager` is the owner of `Pot` . Whenever `Pot::closePot` is called , `managerCut` is sent to the owner of the pot , i.e. , `ContestManager`. But there is no function in `ContestManager` which lets it's owner (Sam) take out the funds.

Impact Owner of `ContestManager` contract gets no funds and the `managerCut` from all contests is stuck inside `ContestManager`

Proof of Concepts 1. Owner(of `Pot` , i.e. , `ContestManager`) creates and funds the pool 2. Player 1 claims 3. Deadline passes 4. Owner calls `closePot` 5. Owner(Contest Manager) gets his 10% 6. Owner of Contest Manager (here , user) gets nothing

PoC

Place this in `TestMyCut.t.sol`

```
1     function test_ManagerCutGoesToContestManager() public
          mintAndApproveTokens{
2         vm.startPrank(user);
3         rewards = [500, 500];
4         totalRewards = 1000;
5         contest = ContestManager(conMan).createContest(players, rewards
              , IERC20(ERC20Mock(weth)), totalRewards);
6         ContestManager(conMan).fundContest(0);
7         vm.stopPrank();
8
9         vm.startPrank(player1);
10        Pot(contest).claimCut();
11        vm.stopPrank();
12
13        uint256 userBalanceBefore = ERC20Mock(weth).balanceOf(user);
14        uint256 ownerBalanceBefore = ERC20Mock(weth).balanceOf(conMan);
15
16        vm.warp(91 days);
17
18        vm.startPrank(user);
19        ContestManager(conMan).closeContest(contest);
20        vm.stopPrank();
21
```

```
22          uint256 userBalanceAfter = ERC20Mock(weth).balanceOf(user);
23          uint256 ownerBalanceAfter = ERC20Mock(weth).balanceOf(conMan);
24
25          assert(ownerBalanceAfter - ownerBalanceBefore == 50); //
               contest manager gets the manager cut
26          assert (userBalanceBefore == userBalanceAfter); // owner of
               contest manager doesn't get anything
27      }
```

Recommended mitigation 1. Make functions which owner of Contest Manager can use to pull out the funds corresponding to a particular token

- Add these functions to `ContestManager.sol`

```
1       function getToken(address _pot) public view returns(IERC20 token) {
2           Pot pot = Pot(_pot);
3           token = pot.getToken();
4       }
5
6       function receiveCut(IERC20 token) public onlyOwner{
7           token.transfer(msg.sender , token.balanceOf(address(this)));
8       }
```

- Owner can input address of the contest in `getToken()` to get the token corresponding to that contest , then use `receiveCut()` to pull out the funds.

2. In the `Pot::closePot` , instead of transferring `managerCut` to `msg.sender` (which is `ContestManager`) , transfer it to `tx.origin` (which is owner of `ContestManager`) . Make the following change :

```
1       function closePot() external onlyOwner {
2           if (block.timestamp - i_deployedAt < 90 days) {
3               revert Pot__StillOpenForClaim();
4           }
5           if (remainingRewards > 0) {
6               uint256 managerCut = remainingRewards / managerCutPercent;
7   -           i_token.transfer(msg.sender, managerCut);
8   +           i_token.transfer(tx.origin, managerCut);
9
10              uint256 claimantCut = (remainingRewards - managerCut) /
                   i_players.length;
11              for (uint256 i = 0; i < claimants.length; i++) {
12                  _transferReward(claimants[i], claimantCut);
13              }
14          }
15      }
```

Medium

[M-1] `ContestManager::fundContest` takes index of the contest as input , but there is no way to determine the index of a contest as `ContestManager::createContest` returns address of the contest , making it difficult to fund a contest

Description `ContestManager::fundContest` is to be called after creating a contest . The contest is created by `ContestManager::createContest` , but this returns the address instead of the index of the contest. Also , there is no other method to get the index of a contest if we know the address of a contest. So , the owner may mistakenly fund a contract they don't want to . Basically using index as param causes difficulties in funding contests.

Impact Owner finds it difficult to fund the intended contest

Recommended mitigation 1. Use address of contest as param as input in `fundContest` instead of the index.

```
-   function fundContest(uint256 index) public onlyOwner {
+   function fundContest(address _contest) public onlyOwner {
-       Pot pot = Pot(contests[index]);
+       Pot pot = Pot(_contest)
        IERC20 token = pot.getToken();
-       uint256 totalRewards = contestToTotalRewards[address(pot)];
+       uint256 totalRewards = contestToTotalRewards[_contest];

        if (token.balanceOf(msg.sender) < totalRewards) {
            revert ContestManager__InsufficientFunds();
        }

-       token.transferFrom(msg.sender, address(pot), totalRewards);
+       token.transferFrom(msg.sender, _contest, totalRewards);
    }
```

2. Make a function which takes in the address of a contest , loops through the `contests` array to find the index. But all this is just extra useless stuff , and this isn't recommended. Also if the array becomes really large then this'll be a DoS attack .

[M-2] Potential erroneous calculation of the Manager's cut in `Pot::closePot` , causing the manager to lose some funds

Description The calculation of the manager's cut in `closePot` is as follows:

```
    uint256 managerCut = remainingRewards / managerCutPercent;
```

Now , `managerCutPercent` is intended to be 'how much pecent of the remaining rewards should the manager get' . This value is hardcoded to be `10` in the `Pot` contract. Now see , 10% means 1/10 so `remainingRewards/10` gives the cut of the manager. But if the developers decide to change this fee percentage to, say `15` , then this formula will not work.

The owner will expect `(15 * remainingRewards)/100` as his cut , but he will get `remainingRewards/15` ~ 6.67% of the remainingRewards. Clearly the owner will lose out on his cut and get way less (or way more , depending on value of `managerCutPercent`) than expected

Impact Owner will get less cut in some cases (`managerCutPercent` > 10)

Proof of Concepts 1. (Change `managerCutPercent` to 15 for this test) 2. Owner creates and funds the pool 3. Player 1 claims 4. Deadline passes 5. Owner calls `closePot` 6. Owner expects 15 % of `remainingRewards` 7. Owner gets 6.67% of `remainingRewards`

PoC

Change `managerCutPercent` to 15 for this test

Place this into `TestMyCut.t.sol`

```
    function test_ClosePotHasWrongMath_2() public mintAndApproveTokens
        {
        vm.startPrank(user);
        rewards = [500, 500];
        totalRewards = 1000;
        contest = ContestManager(conMan).createContest(players, rewards
            , IERC20(ERC20Mock(weth)), totalRewards);
        ContestManager(conMan).fundContest(0);
        vm.stopPrank();

        vm.startPrank(player1);
        Pot(contest).claimCut();
        vm.stopPrank();

        uint256 ownerBalanceBefore = ERC20Mock(weth).balanceOf(conMan);

        vm.warp(91 days);

        vm.startPrank(user);
        ContestManager(conMan).closeContest(contest);
        vm.stopPrank();

        uint256 ownerBalanceAfter = ERC20Mock(weth).balanceOf(conMan);

        // assert(ownerBalanceAfter - ownerBalanceBefore == 75); //
            expects 15 % of 500
        assert(ownerBalanceAfter - ownerBalanceBefore == 33); // gets
            6.67% of 500 (1/15 == 0.0667)
```

```
25      }
```

Recommended mitigation Change the formula as follows

```
1      function closePot() external onlyOwner {
2          if (block.timestamp - i_deployedAt < 90 days) {
3              revert Pot__StillOpenForClaim();
4          }
5          if (remainingRewards > 0) {
6  -           uint256 managerCut = remainingRewards / managerCutPercent;
7  +           uint256 managerCut = (managerCutPercent * remainingRewards)
       /100;
8              i_token.transfer(msg.sender, managerCut);
9
10             uint256 claimantCut = (remainingRewards - managerCut) /
                   i_players.length;
11             for (uint256 i = 0; i < claimants.length; i++) {
12                 _transferReward(claimants[i], claimantCut);
13             }
14         }
15     }
```

Low

[L-1] `ContestManager::fundContest` is not called instantly after `ContestManager::createContest` , making the pot unusable till it is funded

Description `createContest` function is used to create a new contest/pot. The main functionality of the pot is that users can collect their rewards. But for this, the pot must have the necessary funds. To give the pot these funds , the owner/creater/manager must call the `fundContest` function after which the pot functions normally. The problem being the time after the pot is deployed but not funded . Users see their transactions getting reverted . Also the '90 day deadline' starts when the pot is created , not when it is funded. So there is no point in having 2 specific functions , rather fund the deployed contest in the same function.

Impact Users can't claim their rewards till the pot is funded.

Proof of Concepts Here is a test which shows what happens when a pot is deployed but not funded

1. Owner creates the pot
2. Player tries to claim their reward but cannot.

PoC

Place this test into `TestMyCut.t.sol`

```
function test_LateFundingOfContractIsBad() public mintAndApproveTokens {
        vm.startPrank(user);
        contest = ContestManager(conMan).createContest(players, rewards, IERC20(ERC20Mock(weth)), 4);
        // ContestManager(conMan).fundContest(0); --> DIDNT FUND
        vm.stopPrank();

        vm.startPrank(player1);
        vm.expectRevert();
        Pot(contest).claimCut();
        vm.stopPrank();
    }
```

Recommended mitigation Fund the pot inside the `createContest` function itself and remove the `fundContest` completely

```
    function createContest(address[] memory players, uint256[] memory rewards, IERC20 token, uint256 totalRewards)
        public
        onlyOwner
        returns (address)
    {
        // Create a new Pot contract
        Pot pot = new Pot(players, rewards, token, totalRewards);
        contests.push(address(pot));
        contestToTotalRewards[address(pot)] = totalRewards;
+       if (token.balanceOf(msg.sender) < totalRewards) {
+           revert ContestManager__InsufficientFunds();
+       }
+       token.transferFrom(msg.sender, address(pot), totalRewards);
        return address(pot);
    }

-   function fundContest(uint256 index) public onlyOwner {
-       Pot pot = Pot(contests[index]);
-       IERC20 token = pot.getToken();
-       uint256 totalRewards = contestToTotalRewards[address(pot)];
-        if (token.balanceOf(msg.sender) < totalRewards) {
-           revert ContestManager__InsufficientFunds();
-       }
-       token.transferFrom(msg.sender, address(pot), totalRewards);
    }
```

[L-2] 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;`

2 Found Instances

- Found in src/ContestManager.sol Line: 2

```
1 pragma solidity ^0.8.20;
```

- Found in src/Pot.sol Line: 2

```
1 pragma solidity ^0.8.20;
```

Gas

[G-1] `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

10 Found Instances

- Found in src/ContestManager.sol Line: 16

```
1     function createContest(address[] memory players, uint256[]
          memory rewards, IERC20 token, uint256 totalRewards)
```

- Found in src/ContestManager.sol Line: 28

```
1     function fundContest(uint256 index) public onlyOwner {
```

- Found in src/ContestManager.sol Line: 40

```
1     function getContests() public view returns (address[] memory)
          {
```

- Found in src/ContestManager.sol Line: 44

```
1     function getContestTotalRewards(address contest) public view
          returns (uint256) {
```

- Found in src/ContestManager.sol Line: 48

```
1     function getContestRemainingRewards(address contest) public
          view returns (uint256) {
```

- Found in src/ContestManager.sol Line: 53

```
1     function closeContest(address contest) public onlyOwner {
```

- Found in src/Pot.sol Line: 38

```
1       function claimCut() public {
```

- Found in src/Pot.sol Line: 71

```
1       function getToken() public view returns (IERC20) {
```

- Found in src/Pot.sol Line: 75

```
1       function checkCut(address player) public view returns (uint256
            ) {
```

- Found in src/Pot.sol Line: 79

```
1       function getRemainingRewards() public view returns (uint256) {
```

Informational

[I-1] Unused Custom Error

it is recommended that the definition be removed when custom error is unused

1 Found Instances

- Found in src/Pot.sol Line: 9

```
1       error Pot__InsufficientFunds();
```