Protocol Audit Report

Prepared by: Bikalpa

Table of Contents

Protocol Summary

Protocol does X, Y, Z

Disclaimer

Bikalpa did 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.

Audit Details

Scope

Roles

Executive Summary

Issues found

Severity	Number of Issue Found
High	1
Medium	1
Low	0
gas	0
Info	0
Total	2

Findings

High

[H-1] `Registry::register` function does not refund the extra ethers deposited leading to invariant break.

Description : The `Registry::register` function was designed to accept exact 1 ethers and refund the extra ethers that user has inputed. However, the function doesn't include any refunding code of extra ethers that user might mistakenly enter. This may lead to permanent loss of some extra ethers of the user.

Impact : The user permanently loses the extra amount of ethers that he have mistakenly entered. This breaks the core invariant of this protocol.

Proof of Concept :

1. The user calls the *register*() function with example : 1.5 ethers.
2. He ends up loosing 0.5 extra ethers that he has entered.

Paste the following in `Registery.t.sol`

```
    function test_fuzz_register(uint256 amountToPay) public {
          vm.assume(amountToPay >= 1 ether);

          vm.deal(alice, amountToPay);
```

```
        vm.startPrank(alice);

        uint256 aliceBalanceBefore = address(alice).balance;

        registry.register{value: amountToPay}();

        uint256 aliceBalanceAfter = address(alice).balance;

        assertTrue(registry.isRegistered(alice), "Did not register user");
        assertEq(address(registry).balance, registry.PRICE(), "Unexpected registry
balance");
        assertEq(aliceBalanceAfter, aliceBalanceBefore - registry.PRICE(),
"Unexpected user balance");
    }
```

Recommended mitigation : Add the refunding functionality in your function `Registry::register`

```
   function register() external payable {
        if(msg.value < PRICE) {
            revert PaymentNotEnough(PRICE, msg.value);
        }

+   uint refundAmt = msg.value - PRICE ;

        registry[msg.sender] = true;

+  if (refundAmt > 0){
+    (bool success , ) = msg.sender.call{value:refundAmt}("");
+   require(success , "Transaction failed") ;
+ }

    }
```

Medium

[M-1] The User could enter again even if he has already registered in `Registry::register` function.

Description : Since there is no already entered check in the function `Registry::register` the user might forgot that he has already registered and he might try to register again wasting another 1 ether and gas fees. The user is charged multiple times for the same action.

Impact : The user will lose another 1 ether + gas fees even if their registration status was the same.

Proof of concept :

1. The user adds 1 ethers to the contract for registering.
2. He again comes back and registers and instead of reverting the contract accepts another 1 ethers.

Paste the following in `Registery.t.sol`

```
function test_theUserReRegisterAndLoosesMoney() external {
         uint256 amountToPay = registry.PRICE();

        vm.deal(alice, 2*amountToPay);
        vm.startPrank(alice);

        uint256 aliceBalanceBefore = address(alice).balance;

        registry.register{value: amountToPay}();
        registry.register{value: amountToPay}();

        uint256 aliceBalanceAfter = address(alice).balance;

        assertTrue(registry.isRegistered(alice), "Did not register user");
        assertEq(address(registry).balance, (2*registry.PRICE()), "Unexpected
registry balance");
        assertEq(aliceBalanceAfter, aliceBalanceBefore - (2*registry.PRICE()),
"Unexpected user balance");
    }
```

Recommended Mitigation :

```
function register() external payable {
        if(msg.value < PRICE) {
            revert PaymentNotEnough(PRICE, msg.value);
        }
+  require(!registry[msg.sender] , "User Already Registered") ;

   uint refundAmt = msg.value - PRICE ;

        registry[msg.sender] = true;

   if (refundAmt > 0){
     (bool success , ) = msg.sender.call{value:refundAmt}("");
     require(success , "Transaction failed") ;
   }

    }
```