

[H-1] TITLE (Root + Impact) :

The `updateRegistry` function in the `GivingThanks` contract can be called by anyone, allowing anyone to update the registry. this one cause destory the contract

Description :

Allowing access to anyone to change the `registry` can lead to a loss of control over the contract. so we need to make sure `onlyowner` can update the registry.

Impact :

Anyone can accesss and change the registry address to any address they want.servery breaking the functionality of the protocol.

Proof of Concepts :

this below test case shows how anyone can access the `updateRegistry` function and change the registry address to any address they want.

```
function testregistrychangewithoutownercheck() external{

vm.prank(address(1));

address dummyregistryaddress = address(0);

charityContract.updateRegistry(dummyregistryaddress);

assert(address(charityContract.registry()) == dummyregistryaddress);

}
```

Recommended mitigation :

Add a modifier to the `updateRegistry` function to ensure that only the owner can update the registry.

below code shows how we can add accesss control so only owner can update the registry.

```
modifier onlyowner(){

    require(msg.sender == owner, "Only owner can call this function");
    _;
}

function updateRegistry(address _registry) public onlyowner {
    registry = _registry;
}
```

```
}
```

[H-2] TITLE (Root + Impact):

The `constructor` in the `GivingThanks` contract passes the wrong argument when setting `registry = CharityRegistry(msg.sender);`. This destroys the entire functionality of the contract.

Description :

In the `GivingThanks` contract, the `constructor` mistakenly assigns `msg.sender` as an argument when initializing the `CharityRegistry` instance. This is incorrect because `msg.sender` in the constructor refers to the deployer of the contract, not an actual `CharityRegistry` contract address. As a result, the `registry` variable is set to an invalid address, breaking all functionalities dependent on the `registry`. This prevents proper interaction with the `CharityRegistry` contract, making donation tracking, charity validation, or any registry-related operations non-functional

Impact :

The contract fails to interact with the intended `CharityRegistry`.

Proof of Concepts :

In the test case below, you can see that I print the original `registry` contract address and the address assigned by the line `registry = CharityRegistry(msg.sender);`.

```
vm.prank(admin);

registryContract = new CharityRegistry();

console.log("Original registry address" , address(registryContract));

vm.prank(admin);

charityContract = new GivingThanks(address(registryContract));

console.log("Charity contract assigned registry address" , address(charityContract.1
```

Result :

```
Original registry address 0x3Ede3eCa2a72B3aeCC820E955B36f38437D01395
Charity contract assigned registry address 0xaA10a84CE7d9AE517a52c6d5cA153b369Af99ecF
```

Recommended mitigation :

Simply pass the correct `registry` address as an argument, instead of setting it to `msg.sender`

```
constructor(address _registry) ERC721("DonationReceipt", "DRC") {  
  
    registry = CharityRegistry(_registry);  
    owner = msg.sender;  
    tokenCounter = 0;  
  
}
```

[L-1] TITLE (Root + Impact) :

In the `GivingThanks` contract, the `owner` variable is not declared as `immutable`, which increases gas fees

Description :

In the `GivingThanks` contract, the `owner` variable is stored in regular storage instead of being declared as `immutable`. This leads to higher gas costs because every access to the `owner` variable requires a storage read, which is more expensive than reading from bytecode. Since the `owner` value does not change after deployment, marking it as `immutable` would optimize gas efficiency by storing it directly in the contract's bytecode, reducing the cost of retrieval.

Impact :

Higher gas costs for every access to the `owner` variable, which can accumulate over time and increase the overall cost of contract interactions.

Proof of Concepts :

Impact of Using `immutable` on Gas Fees

```
function testgetownergascost_notimmutable() external {  
  
    charityContract.owner();  
    uint256 gasprice = gasleft();  
    console.log("Gas fees left before using the immutable keyword: " , gasprice);  
}  
  
function testgetownergascost_immutable() external {  
  
    charityContract.owner();
```

```

        uint256 gasprice = gasleft();
        console.log("Gas fees left after using the immutable keyword: " , gasprice);
    }

```

By comparing the gas fees left before and after using the immutable keyword, we can observe the following:

Gas fees left before using the immutable keyword: 1073712725

Gas fees left after using the immutable keyword: 1073714906

Before using immutable: The gas usage was 1073712725, indicating that the contract's operations consumed more gas.

After using immutable: The gas usage reduced to 1073714906, showing that the contract became more efficient and used less gas after declaring the owner variable as immutable.

Recommended mitigation :

Declare the **owner** variable as immutable to optimize gas efficiency and reduce gas costs for accessing the owner variable.

```

address public immutable owner;

```