LESSON SC 13 – Ready To Use Contracts

University of West Attica

Department of Electrical and Electronics Engineering

Ioannis Christidis

Christoforos Kachris

Support by Ethereum Foundation ESP

What will we accomplish!

In this lesson we will update our SC with ready to use SCs.

Specifically, we will use OpenZeppelin SCs.

OpenZeppelin smart contracts are a collection of modular, reusable, and secure open-source Solidity-based SCs that help developers build DApps on the Ethereum blockchain and other compatible networks.

They are widely used for their security and reliability in the development of blockchain projects.

The *OpenZeppelin SCs* we are going to use are:

- `Ownable`
- `ReentrancyGuard`

Ownable

The `Ownable` SC in OpenZeppelin is a Solidity SC that provides a simple and secure way to implement ownership-based access control.

It is used in blockchain applications to designate a privileged "owner" who has exclusive control over specific functions in the SC.

We already have a similar functionality in our `ICR` SC but let's see what OpenZeppelin has to offer.

Ownable Components:

- `owner` Variable: Stores the address of the current owner.
- `onlyOwner` Modifier: Restricts function execution to the owner.
- `transferOwnership` Function: Transfers ownership to a new address.
- `renounceOwnership` Function: Removes the owner, making the SC ownerless.

We will use `Ownable` to the `ICRFactory` SC to add the necessary access control so that only the IIoTC (`ICRFactory owner`) will be able to call the `deployICR` function.

`Ownable` Implementation

In Remix IDE, *you can add OpenZeppelin SCs to your files by just importing them* and Remix IDE will automatically add them to the `.deps` folder from `github` when you compile.

```
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
```

To use `Ownable`, you must inherit its properties to your SC with the `is` keyword.

```
contract ICRFactory is Ownable {}
```

To define who the `owner` is we need to we need to set it to the `Ownable constructor`.

We will make the `msg.sender` the `owner` of the SC (IIoTC's address).

```
    constructor() Ownable(msg.sender) {}
```

And finally, in the function `deployICR` we will add the `modifier` provided by the `Ownable` SC called `onlyOwner`.

```
    function deployICR() external onlyOwner {}
```

And with these changes we ensure that only the IIoTC's address will be able to deploy an `ICR` SC.

ReentrancyGuard

ReentrancyGuard is a utility SC provided by OpenZeppelin that *helps prevent reentrancy attacks in Ethereum SCs*.

Reentrancy is one of the most well-known vulnerabilities in Solidity and occurs when a malicious actor exploits the ability to repeatedly call a function in a SC before its previous execution is completed. This can lead to unexpected behaviors, such as draining funds.

The ReentrancyGuard SC prevents reentrancy by using a simple status flag.

It works as follows:

When a function with the nonReentrant modifier is called, the flag is set to indicate the function is in execution.

If another call tries to enter the same function (re-enter), it will fail because the flag is already set.

Once the execution completes, the flag is reset, allowing normal operation again.

ReentrancyGuard Implementation

Similar to `Ownable` you can just import the `ReentrancyGuard` SC to the files you want to use it.

You generally want to use `ReentrancyGuard` *when you make external calls or modify critical state variable and especially when you do not have access control.*

In our case we will use it on all our functions that require a transaction in both `ICR` and `ICRFactory`.

```
import {ReentrancyGuard} from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
```

To use `ReentrancyGuard`, you must inherit its properties to your SC with the `is` keyword.

```
contract ICRFactory is Ownable, ReentrancyGuard {}
```

In the function `deployICR` we will add the modifier provided by the `ReentrancyGuard` SC called `nonReentrant`.

```
    function deployICR() external onlyOwner nonReentrant {}
```

And with these changes we ensure that our SC cannot fall victim to reentrancy attacks.

Similarly, you can use it on the `ICR` SC by importing it to `ICRRegistry`.

Outro

Ok, now we learned how to use some ready to use SC.

There are more you can check but we will keep it simple.

In the next lesson we are going to make some adjustments, and we will also learn about external function calls.