Blockchain-CTF-Docs

Smart Contracts in a nutshell

A smart contract is a program that is stored in the blockchain. Once it is deployed, it is **immutable**, meaning no one can change its code. Tho, the execution logic can not be changed.
Smart contracts are written using programming languages, the most popular is **Solidity** in case of Ethereum blockchain (the blockchain on this CTF is a fork of Ethereum blockchain).

What does it mean to interact with the blockchain?

Like any other system, to interact with the blockchain we first need an account (EOA). **An account in a blockchain is just an address**, which is a 20-bytes hex number. The address is generated from a **public key** which is in turn generated from a **private key**. So, **an account is controlled by a private key**. Smart contracts are accounts but they are not controlled by a private key, they are controlled by their code.

There are two types of interactions with the blockchain, based on what we are attempting to do:

- **If we are reading from blockchain**: we don't need to be authenticated as the blockchain is public and transparent
- **If we are writing to blockchain**: writing to blockchain can be in several cases such as sending ethereum from an account to another, changing the state of a smart contract, etc. In such cases, we are making a **transaction** which requires authentication. **Authentication simply means we need to sign the transaction using our private key**.

How to interact with the blockchain?

There are several tools that we can use to interact with the blockchain. Metamask (Wallet browser extension), ethers.js, web3.js (Js libraries), Hardhat, Foundry, Truffle (Blockchain development frameworks), Remix (online IDE), etc.

The simplest one to use to play this CTF is Foundry (Remix IDE is easy as well). Below are installation instructions for foundry

```
$ curl -L https://foundry.paradigm.xyz | bash
$ foundryup
```

NOTE: if you get error saying `foundryup command is not found`, just reload your terminal

You can get more information about using framework in the official docs: **https://book.getfoundry.sh/**

Examples of using Foundry

Cast is a powerful command provided by foundry to interact with the blockchain. Below an example of a basic smart contract:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract SimpleContract {
    bool public solved;


    function makeACall() external {
        solved = true;
    }

    function isSolved() public view returns (bool) {
        return solved == true;
    }

}
```

- If we were about to call the `*isSolved*` function, this function does not modify the state of the contract, tho we are only reading from the blockchain, so we don't

need to sign the transaction using a private key. Below the cast command that calls the function:
$ cast call <CONTRACT ADDRESS> “isSolved()” –rpc-url <RPC>

- If we were about to call the `*makeCall*` function, this function does modify the state of the contract, tho we are writing to the blockchain, so we need to sign the transaction using our private key. Below is the cast command:
$ cast send <CONTRACT ADDRESS> “makeACall()” –rpc-url <RPC> –private-key <PK>

Read more about available cast commands and how to use them:
https://book.getfoundry.sh/reference/cast/

NOTES about Blockchain Category on the CTF

- For each challenge, you need to create a ticket (one per team). The ticket is **TOP SECRET**, do not share it with other teams.
- After creating a ticket, you need to launch a new instance of a blockchain. This instance is isolated from other teams, only you can access the instance through your ticket. Once the instance is created, connection information to it will be displayed (rpc url, private key that you need to use, Setup contract address). The instance lasts only for 30 minutes, you can create other instances after that time.
- You do not need to launch the instance directly, the solidity files (if they exist in the challenge) are provided, take your time trying to solve the challenge locally, once you do, launch the instance and make the necessary transactions.
- To solve each challenge, the function `isSolved` in the `Setup.sol` file needs to return true. Read the Setup file code and try to make that function return true.
- Once the challenge is solved (`isSolved` function will return true), get your flag.

Happy Hacking

“Blockchain is not hard, it is different” - 0xbrivan