

## Создание Blockchain CTF: исследование уязвимости смарт-контрактов

---

*Автор публикации: Лукьянов Артур, младший исследователь*

Мы в CyberOK в ходе пентестов очень любим “взламывать” разнообразные инновационные и необычные вещи. Смарт-контракты на блокчейне давно появились на наших радарх, так как они не только предлагают прозрачность, надежность и автоматизацию, но и легко могут стать объектом атак и уязвимостей. В рамках кибербитвы Standoff 10 мы решили объединить наш опыт по анализу защищенности и расследованию инцидентов в блокчейне и представить его в игровой форме — в виде открытой платформы для проведения соревнований Capture The Flag (CTF). Мы развернули собственную блокчейн-сеть с помощью ganache, чтобы дать участникам возможность взаимодействовать со смарт-контрактами в наиболее реалистичной атмосфере.

Эта статья – подробный tutorial о том, как сделать такой же CTF для блокчейна своими руками. Я расскажу какие технологии могут помочь вам поиграться со смарт-контрактами у себя дома и устроить собственное соревнование из подручных средств.

### Создание задачи (уязвимого смарт-контракта)

---

Перед началом создания задачи необходимо определить сценарий и контрактную логику, на основе которых будет разработан уязвимый смарт-контракт. Это позволит создать реалистичное окружение — ведь ломать контракт банка с функциями кредитования куда интереснее (и понятнее), чем абстрактного коня в вакууме. Важно учесть различные типы уязвимостей, такие как уязвимости рекурсии, переполнения, проверки прав доступа и другие. Это дает возможность создать задачу с разными уровнями сложности, чтобы каждый участник мог найти что-то интересное и подходящее для своего уровня навыков.

*Например, вместо непонятной “хранилки флагов” можно создать сервис для “безопасного хранения заметок/информации”. Когда у сервиса есть легенда, участник CTF понимает, как пользователи могут использовать такое приложение. Так участник будет искать баги функционала и логики, а не играть в “угадайку” или заучивать уязвимые паттерны.*

В качестве вдохновения можно брать уже существующие сервисы как референсы, например [здесь](#).

При этом, важно не забывать про то, что это обучающая задача и не переусложнять её, превращая в полноценный аудит.

## Подготовка тестового блокчейна с использованием Ganache

Для проведения Blockchain CTF необходимо иметь тестовую блокчейн-среду, которая позволит участникам разрабатывать и тестировать свои решения без риска потери реальных средств. В этой статье мы будем использовать Ganache — легковесную блокчейн-среду, которая позволяет запускать локальный блокчейн для разработки и тестирования смарт-контрактов. Ganache предоставляет удобный интерфейс для создания аккаунтов, имитации различных сценариев и взаимодействия с контрактами.

Я запускал Ganache на Arch Linux, но и для других систем установить его не составит труда: [вот официальная инструкция](#).

Для начала установим ganache-cli:

```
yaq -S ganache-cli
```

Далее нам понадобится RPC провайдер для ноды. URL RPC провайдера можно получить на [alchemy.com](#) или [infura.io](#). Они имеют свои ограничения, но для запуска тестнета “для себя” их вполне хватает.

*RPC провайдер необходим только если вы хотите работать с данными реального блокчейна — например, взаимодействовать с уже существующими смарт-контрактами. Если ваши задачи не требуют такого, смело пропускайте этот шаг.*

Теперь запустим ganache-cli:

```
ganache-cli -f <RPC provider URL>
```

Ganache выводит набор приватных ключей для аккаунтов с балансами по 100 ETH. Скопируем один из этих ключей, он понадобится нам позже.

```
> ganache-cli -f https://eth-mainnet.g.alchemy.com/v2/-UGI0vL6A0V2sZ-GqMp6bbMHAjwqTTIo
Ganache CLI v6.12.2 (ganache-core: 2.13.2)

Available Accounts
=====
(0) 0xFE846448C5df73b00fe6C9AB63B8127830BCa970 (100 ETH)
(1) 0x14e14284f023F2eC74459cF01B80Ad246c57635C (100 ETH)
(2) 0x52F033d0ABD4dF482E7aC3d8c5892C69708Df1A8 (100 ETH)
(3) 0xbd092220bfEa86342B7e174fE429F12282Ac7999D (100 ETH)
(4) 0xea983461969D4310ad7582E95d9fa74C92E2C6b8 (100 ETH)
(5) 0xa48B0013C2072BbeD06E0aED6195Aa944bc63a94 (100 ETH)
(6) 0xf05b6AbF6Bc5981948Aba8b3271ff6c515CB47A7 (100 ETH)
(7) 0x4005959E3D0351772F71812c120A8775f9013532 (100 ETH)
(8) 0xd3Fb19F751ea4dF5F6EB4e663226ba6C00A79C44 (100 ETH)
(9) 0x3a19C7F22c70Ae190c055A24Bb3a3558Ef23Bca1 (100 ETH)

Private Keys
=====
(0) 0xe4e3c94a551e14eafb72ffbfe7cf1c904be3e0c9cebebef78a8e10cbd4a95c1a
(1) 0xd02671d2952667b13afc4dcfc2d9ccaf336d55350366de0ce6ac02585131c01f
(2) 0xf88bd0666aa3608331288bbda69492d185a63c05b873ed152c8a08ee2706f60
(3) 0x0dd2ac0bf3912ace7336da14e8fffb64350f0223d00d5cc38987946909301bc62
(4) 0x5b47fd24c00cbda2fb8514029f1210abd241831c59b1ceac9d36942173db39db
(5) 0xaa667bc583d5aefe1977c2a390dea8c1297ed0e663cb0423c575163cef1a110e
(6) 0x0446d7e837af29e777fd03ee2cd2e08af05cb5610a51a550eefe9809b7ea18d
(7) 0x2d97d57c734e98e09b1a1d37a49701f595d75628dc09132c64d7b50c52be8dd2
(8) 0x1fb982769ab358566e1b63b038d9fb2abebd661956e7acdd5eb7168b25eca5b2
(9) 0x6c6ba58e9e61bc50e4060a76c46265b0a118c195a6b8be243fa44fa664d33845

HD Wallet
=====
```

## Разработка смарт-контракта

Когда кейс для задачи уже придуман, дело за малым — реализовать уязвимый смарт-контракт.

Неплохая Web IDE для разработки: <http://remix.ethereum.org/>

Для примера далее мы будем использовать следующий смарт-контракт:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract SecretNoteKeeper {

    string private secretNote;
    address private owner;

    constructor(string memory newNote) {
        secretNote = newNote;
    }
}
```

```
function getSecret() public returns (string memory) {  
    require(msg.sender == owner);  
    return secretNote;  
}  
  
function changeOwner() public {  
    owner = msg.sender;  
}  
}
```

Без знания языка программирования выбранного блокчейна реализовать смарт-контракт будет проблематично, хотя ChatGPT, возможно, сможет помочь:

12/12 > RO Hello! Write a vulnerable smartcontract in educational purposes. I want it to be a NFT-exchange service which will allow users to swap their tokens from different ERC721 contracts with each other. User should be able to deposit, withdraw, list, suggest swap and confirm swap. It should keep track who owns NFTs in order to perform swaps and store it as mapping.

One main function of smart contract have logical mistake that allows to steal NFTs. Mark this function with comment "Vulnerable". Obfuscate the vulnerable part. Write code in Solidity language for Ethereum blockchain

2/2 > Certainly! I can provide you with an example of a vulnerable smart contract for educational purposes. However, please note that deploying or using vulnerable contracts on the Ethereum blockchain can lead to real-world consequences, such as loss of funds or assets. It is essential to thoroughly test and audit any smart contract before deploying it to a live network.

Here's an example of a vulnerable NFT exchange service smart contract:

```
solidity
Copy code

pragma solidity ^0.8.0;

import "./IERC721.sol";

contract NFTExchange {
    struct Swap {
        address sender;
        address receiver;
        address contractAddress;
        uint256 tokenId;
        bool isConfirmed;
    }

    mapping(address => mapping(uint256 => address)) public tokenOwners;
    mapping(bytes32 => Swap) public swaps;
    uint256 public swapCount;

    event SwapSuggested(bytes32 indexed swapId, address indexed sender, address indexed receiver, address indexed contractAddress, uint256 indexed tokenId);
    event SwapConfirmed(bytes32 indexed swapId);

    function deposit(address _contractAddress, uint256 _tokenId) external {
        IERC721 token = IERC721(_contractAddress);
        address tokenOwner = token.ownerOf(_tokenId);
        require(tokenOwner != address(0), "Invalid token");

        tokenOwners[_contractAddress][_tokenId] = msg.sender;
        token.transferFrom(msg.sender, address(this), _tokenId);
    }

    function withdraw(address _contractAddress, uint256 _tokenId) external {
        address tokenOwner = tokenOwners[_contractAddress][_tokenId];
        require(tokenOwner == msg.sender, "Not token owner");

        IERC721 token = IERC721(_contractAddress);
        tokenOwners[_contractAddress][_tokenId] = address(0);
        token.transferFrom(address(this), msg.sender, _tokenId);
    }

    function suggestSwap(address _receiver, address _contractAddress, uint256 _tokenId) external {
        IERC721 token = IERC721(_contractAddress);
        require(token.ownerOf(_tokenId) == msg.sender, "Not token owner");

        bytes32 swapId = keccak256(abi.encodePacked(msg.sender, _receiver, _contractAddress, _tokenId));
        Swap storage swap = swaps[swapId];
        require(swap.sender == address(0), "Swap already exists");

        swap.sender = msg.sender;
        swap.receiver = _receiver;
        swap.contractAddress = _contractAddress;
        swap.tokenId = _tokenId;
        swap.isConfirmed = false;

        emit SwapSuggested(swapId, msg.sender, _receiver, _contractAddress, _tokenId);
    }

    // Vulnerable function - This function allows an attacker to steal NFTs
    function confirmSwap(bytes32 _swapId) external {
        Swap storage swap = swaps[_swapId];
        require(swap.receiver == msg.sender, "Not receiver of the swap");
        require(!swap.isConfirmed, "Swap already confirmed");

        IERC721 token = IERC721(swap.contractAddress);
        require(token.ownerOf(swap.tokenId) == msg.sender, "Not token owner");

        swap.isConfirmed = true;
        token.transferFrom(address(this), msg.sender, swap.tokenId);

        emit SwapConfirmed(_swapId);
    }
}
```

In this contract, the `confirmSwap` function has a logical flaw that allows an attacker to steal NFTs. The vulnerable part has been marked with a comment.

Неплохо, как отправная точка, но уязвимости всё же лучше добавлять руками.

## Загрузка заданий на тестовый блокчейн

После создания уязвимого смарт-контракта необходимо загрузить задание на тестовый блокчейн с помощью Ganache. Это позволит участникам взаимодействовать с контрактом и исследовать его уязвимости.

Можно залить смарт-контракт через тот же [remix](#). Но лучше всё же создать скрипт деплоя — тогда повторная заливка и разворачивание стенда на сервере будет куда проще.

Для деплоя смарт-контракта будем использовать hardhat: <https://hardhat.org/>

Создаём hardhat проект:

```
npx hardhat
```

Выбираем 'create JS project'.

Далее, в папку contracts помещаем написанный нами контракт, назовём его SecretNoteKeeper.sol.

Контракт-пример (Lock.sol), нужно удалить.

В hardhat.config.js надо указать правильную версию solidity:

```
1 require("@nomicfoundation/hardhat-toolbox");
2
3 /** @type import('hardhat/config').HardhatUserConfig */
4 module.exports = {
5   solidity: "0.8.7",
6 };
7
```

Установим ganache для hardhat:

```
npm install --save-dev @nomiclabs/hardhat-ganache
```

Добавим эту строчку в начало hardhat.config.js

```
require("@nomiclabs/hardhat-ganache");
```

Изменим scripts/deploy.js

```
// We require the Hardhat Runtime Environment explicitly here. This is optional
// but useful for running the script in a standalone fashion through `node <script>`.
//
// You can also run a script with `npx hardhat run <script>`. If you do that, Hardhat
// will compile your contracts, add the Hardhat Runtime Environment's members to the
// global scope, and execute the script.
const hre = require("hardhat");

async function main() {
  // Тут надо поменять SecretNoteKeeper на имя контракта - чтобы можно было создать
  // инстанс
  const Task = await hre.ethers.getContractFactory("SecretNoteKeeper");
  // А тут можно передать необходимые аргументы в конструктор
  const task = await Task.deploy('ctf{flag}');

  await task.deployed();

  console.log(
    `Task deployed to ${task.address}`
  );
}

// We recommend this pattern to be able to use async/await everywhere
// and properly handle errors.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

И задеплоим на ganache:

```
npx hardhat run --network ganache scripts/deploy.js
```

### (Дополнительно) Создание сайта для web3

Задеплоенного контракта уже достаточно для задачи, но будет ещё круче оформить всё как Web3 сайт.

Создать базовую страничку для взаимодействия с контрактом не слишком сложно, но перед тем, как начать оформлять её, оцените, готовы ли вы поддерживать ещё и фронт? Браузерные кошельки (Metamask, Trust wallet, etc) могут менять свои API, да и добавление тестнета в них иногда может оказаться совсем не очевидным.

Создадим отдельную папку для фронтенда.

Будем использовать следующий HTML шаблон:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi"
crossorigin="anonymous">
  </head>
  <body>
    <div class="container">
      <h1>CTF task #01</h1>
      <button id="button-getsecret" class="btn btn-outline-secondary my-3">Get
secret</button><br/>
      <button id="button-changeowner" class="btn btn-outline-secondary">Change
owner</button>
    </div>
  </body>
</html>
```



# CTF task #01

Get flag

Change owner

Остаётся подключить кнопки к контракту. Например, так (для Metamask):

```
const testnetServerAddress = 'https://your-testnet-rpc-url'; // Replace with your testnet server address
```

```
// Function to connect MetaMask
```

```
async function connectMetaMask() {
```

```
  // Check if MetaMask is installed
```

```
  if (typeof window.ethereum !== 'undefined') {
```

```
    try {
```

```
      // Request MetaMask to connect
```

```
      await window.ethereum.request({ method: 'eth_requestAccounts' });
```

```
      console.log('Connected to MetaMask');
```

```
    } catch (error) {
```

```
      console.error(error);
```

```
      alert('Failed to connect to MetaMask');
```

```
    }
```

```
  } else {
```

```
    alert('MetaMask is not installed');
```

```
  }
```

```
}
```

```
// Function to invoke getSecret() function
```

```
async function invokeGetSecret() {
```

```
  // Check if MetaMask is connected
```

```
  if (typeof window.ethereum !== 'undefined') {
```

```
    try {
```

```
      // Get the current selected account
```

```
      const accounts = await window.ethereum.request({ method: 'eth_accounts' });
```

```
// Get the contract instance
const contract = new window.ethereum.Contract(contractAbi, contractAddress);

// Call the getSecret() function
const secret = await contract.methods.getSecret().call({ from: accounts[0] });
console.log('Secret:', secret);
} catch (error) {
  console.error(error);
  alert('Failed to invoke getSecret()');
}
} else {
  alert('Please connect to MetaMask');
}
}

// Function to invoke changeOwner() function
async function invokeChangeOwner() {
  // Check if MetaMask is connected
  if (typeof window.ethereum !== 'undefined') {
    try {
      // Get the current selected account
      const accounts = await window.ethereum.request({ method: 'eth_accounts' });

      // Get the contract instance
      const contract = new window.ethereum.Contract(contractAbi, contractAddress);

      // Call the changeOwner() function
      await contract.methods.changeOwner().send({ from: accounts[0] });
      console.log('Owner changed successfully');
    } catch (error) {
      console.error(error);
      alert('Failed to invoke changeOwner()');
    }
  } else {
    alert('Please connect to MetaMask');
  }
}
```

```
}  
}  
  
// Event listener for the 'Get Secret' button  
document.querySelector('#button-getsecret').addEventListener('click', invokeGetSecret);  
  
// Event listener for the 'Change Owner' button  
document.querySelector('#button-changeowner').addEventListener('click',  
invokeChangeOwner);
```

## Игровой процесс

---

Task-based часть Blockchain CTF предлагает интересный игровой процесс для участников. Они получают задачу, которая представляет собой уязвимый смарт-контракт, и их задача состоит в том, чтобы найти и эксплуатировать уязвимость для получения доступа к защищенным ресурсам или выполнения определенного действия. Лучше подготовить несколько разных “испытаний” — так участники смогут проверить свои навыки в разных областях. Такой игровой процесс стимулирует участников к активному обучению и исследованию, а также позволяет им применить свои навыки на практике — “гугление” во время CTF не только развивает чуйку и эрудицию, но и совершенствует навыки.

## Подсчет баллов и объявление победителей

---

После завершения Blockchain CTF процесса необходимо подсчитать баллы участников и объявить победителей. В зависимости от сложности и успешности выполнения задачи участники получают определенное количество баллов. Интересный вариант подсчета баллов — выдача участнику монет на тестнете за выполнение заданий. Тогда побеждает самый “богатый” на конец соревнования участник.

Так, когда мы проводили CTF на Standoff 10, победителями соревнования стали: Сачивко Никита, Вячеслав Дмитриев, Греков Илья и Левчук Павел.

В следующей части статьи мы рассмотрим “Атаку на реальный смарт-контракт”, где мы расскажем, как использовали реальный контракт для CTF — реставрацию состояния смарт-контракта и подготовку контракта для участников.