CSE5095-010: Blockchain Technology

Lecture 7

Ghada Almashaqbeh

UConn - Fall 2020

Outline

- More about Ethereum capabilities:
    - DApps.
    - Smart contracts; a hands-on example.
    - Ethereum tokens.

DApps

- A DApp consists of:
  - Smart contract(s) deployed on a blockchain (backend).
  - A web frontend user interface.
- May also need (based on its functionality):
  - Decentralized storage network/platform (e.g., IPFS).
  - Decentralized messages protocol (e.g., Whisper).
- Envisioned as the next evolution of the web.
  - The third generation, or Web 3.0
  - Web 2.0 is more about user-generated content and interactivity, Web 3.0 is more about decentralization and transparency.

Smart Contracts;
A Hands-on Example

Overview I

- A smart contract is simply a program written in Ethereum scripting language, deployed at the miners in the network (the EVM) who will execute this program.
- It is composed of two parts:
  - State: represented by the values of the variables defined inside the smart contract, the contract ether/gas balance, code, and other state variables.
  - API: a set of functions to interact with the contract. Two types:
    - Read-only or constant functions that do not change the contract state, thus, can be called for free.
    - Transactional functions that change the state of the contract and result in transactions logged on the blockchain. Need to the gas cost when calling any of these functions.

Overview II

- The full code of the smart contract and its current state are public on the blockchain.
- Once the contract owner deploys the contract he/she cannot change its code.
    - The owner can ask the miners to destruct the contract (if it contains a function to do that) and deploy a new contract.
    - However, this does not mean that it will disappear, it only means that it cannot be invoked and miners will not track its state.
- Deploying smart contracts is not like deploying software packages.
    - Contracts deal with currency all the time. Attackers are more motivated to hack them.
    - No patches to fix security vulnerabilities or some entity to complain to and get a refund/etc.

Developing Smart Contracts I

- You are about to write programs that deal with currency.
    - Deploying the contract by itself has a fee that must be paid to the miners.
        - This besides the gas fee for making function calls.
    - Even you, the contract owner/creator, has to pay if you want the miners to execute any of the code for you.
- Just like traditional coding, you need a programming language (we will use Solidity), an IDE, compiler, etc.

Developing Smart Contracts II

- Testing is an essential step before deploying a contract.
  - Blockchain-based systems usually have two networks:
    - Testnet: has fake tokens and allow experimenting with the code for free in a decentralized way.
    - Mainet, or production network (the real network).
  - Beside the testnet, you can test on some fake blockchain that is generated locally on your machine.
- Several tools are available to help with testing and deployment. We will examine two of them.

Our Example

- We will write a simple contract the does the following (a toy example just to clarify the concept):

“A market smart contract.

Each seller submits offers to the contract; an offer contains the ID of the item he/she wants to sell and its price.

Buyers can buy these items through the contract as well.”

Approaches

- There are different tools and approaches to help in writing and testing smart contracts.
- We will examine two of them:
    - Approach 1: online IDE/compiler tool called Remix (http://remix.ethereum.org).
    - Approach 2: offline tools on your local machine (detailed description is found in the next slides).
- In both we will use Solidity, the most popular scripting language for Ethereum.
    - The documentation is found at https://solidity.readthedocs.io/en/latest/

Approach 1

- An easy tool that allow you to write the contract code in Solidity, compile, deploy, and test its functionalities.
- GUI based, which serve as a fast way to do the aforementioned tasks.
- Documentation available at: https://remix.readthedocs.io/en/latest/
- We will try this tool together to implement the required smart contract functionality outlined earlier.

Market Smart Contract Code I

```
pragma solidity ^0.6.6;

contract Market {

    mapping(uint8 => uint) items;

    address payable market_owner;

    event itemSold(uint8 id);

    /// Initialize contract,

    // we assume that there is only one owner who can list items for sale.

    constructor() public {

        market_owner = msg.sender; }

/// list an item for sale

    function sell(uint8 item_id, uint price) public {

        if (item_id >= 0 && item_id <= 255 && price > 0)

            items[item_id] = price;

    }
```

Market Smart Contract Code II

```
   /// Buy an item.

    function buy(uint8 item_id) public payable {

        // check that the item exists and then sell

        if(items[item_id] > 0 && msg.value >= items[item_id]) {

            // mark the item as sold by setting its price to zero

            items[item_id] = 0;

            // transfer the paid currency to the market owner

            market_owner.transfer(msg.value);

        }

        else {

            emit itemSold(item_id);

        }

    }

}
```

Approach 2 - Used Tools I

- Several offline tools and compilers are available. We will see one setup in these slides.
- We will need the following tool:
  - **Truffle:** A development and testing environment of Ethereum smart contracts. It makes it easier to develop and interact with your contracts. Check the full documentation at http://truffleframework.com/docs/
  - **Testrpc:** local test network that creates a fake blockchain locally on your machine based on your contracts only.
  - **Geth:** Allows you to run an Ethereum node and interact with the network (in CLI). It can be used in the test mode (i.e. testnet) to experiment with your contract. You can explore this testnet on https://ropsten.etherscan.io/

Approach 2 - Used Tools II

- For testing, we will use:
    - **javascript:** In fact the syntax of solidity is very similar to javascript syntax. In addition, it will be used in Truffle to interact and test your contracts (you can use Python, Go, etc., as well). Check Truffle documentation.

Disclaimer: The installation instructions, coming next, are all done on Ubuntu 16.04. They may differ a little bit for different machines.

Installation - Prerequisites

- Install npm (package manager for javascript) and nodejs (runtime environment for javascript) using the following commands:

```
sudo apt-get install npm

sudo apt-get install python-software-properties

curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -

sudo apt-get install nodejs
```

- The second and third commands above are needed to enforce installing the latest version of nodejs to the installation of testrpc will not fail.
- Also use the following command to tie the name node with nodejs (in Ubuntu there is a conflict so it is called nodejs instead of node):

```
sudo apt install nodejs-legacy
```

Installation - testrpc and truffle

- Run the following commands:

```
sudo npm install -g ethereumjs-testrpc

sudo npm install -g truffle
```

Our First Smart Contract I

- We need to create a directory where our contract will be, this directory will be initialized by truffle. To do so, type the following commands:

```
mkdir example

cd example/

truffle init
```

- Open your favorite editor and write the contract code, save it inside example/contracts with extension .sol, or simply run:

```
truffle create contract market
```

- Then edit the code inside this new contract which is called market.

Our First Smart Contract II

- Compile your code using the following command which will compile all contracts inside example/contracts directory:

```
truffle compile
```

- To deploy your contract you need first to start testrpc on a separate terminal by running the command:

```
testrpc
```

- Then on the first terminal run the command to deploy your contract on the local test blockchain:

```
truffle migrate
```

Our First Smart Contract III

- Note that testrpc will create 10 accounts for you with fake ether inside them. You can use these accounts in your contract testing codes (accounts[0] is the address of the first account and so on).
- Every time you edit your contract, you need to recompile and redeploy as follows:

```
truffle compile

truffle migrate --reset
```

Testing the Code

- You can either interact with your contract using Truffle console or by writing test scripts (using javascript for example).
- Consult the tools’ documentation for more details.

Ethereum Tokens

Motivation

- A flexible way of creating new coins or digital assets on top of Ethereum ecosystem.
- Very similar to loyalty cards concept, you collect points and then you can use them to buy goods.
- Several advantages:
    - Utilize the underlying infrastructure of Ethereum such as the miners, the blockchain, the consensus protocol, etc.
        - Being tied to an existing, and known system.
    - No need to bootstrap a new cryptocurrency system. Just issue your token, i.e. contract, and start trading.
    - Allow the use of a token across multiple projects or platforms, which enhance liquidity.

Is not the Ether a token?

- Any cryptocurrency is a token.
- However, a token created on top of another cryptocurrency has several differences.
    - Ether is controlled by the ethereum protocol, a token is controlled by its smart contract code.
        - Account state, balance, ownership checking, etc.
    - The token is backed up by the original currency.
        - Minting a token is done by buying it using Ether (like depositing Ether to the token smart contract).
- You can write your own smart contract and control your token.
    - Does not need more than few tens of line of code in Solidity.
- However, to secure the contract and account for all possible vulnerabilities, as possible, the code is way complex.

Ethereum Token Standards

- Several standards to token smart contracts.
  - Goal: outline minimum specifications and encourage interoperability.
  - The most popular and widely used one is ERC20.
  - It specifies a set of functions and data structures that a token smart contract must implement, in addition to an optional function set.
    - Usually transfer tokens, approve, balance inquiry, etc.
  - A developer can add additional functions, etc.
- Used widely in ICOs (initial coin offering).

Ethereum Token Examples

- Widely used to create decentralized services.
  - Provide a digital service on top of Ethereum.
  - Clients pay token to obtain the service.
  - Thus, clients buy tokens in Ether servers can sell their tokens to obtain Ether in return.
- Example:
  - Livepeer; a decentralized video transcoding service.
  - Golum; a decentralized computation outsourcing service.
  - NuCypher; a decentralized access control service.