University of Zurich[UZH]

Challenge Task: Decentralized Lottery Application

Blockchain and Overlay Networks

Danijel Dordevic, Ratanak Hy, Claude Muller, Lucas Thorbecke
Student IDs: ..., 11-717-790, 11-067-675, 12-933-917

Date of Submission: May 22, 2019

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Contents

Chapter 1

Introduction

To get hands on experience with decentralized applications a Lottery system was implemented. The requirements were the following.

- The lottery system must run entirely within one or more Smart Contracts (SC).
- For determining a winner a random number generator oracle must be implemented an run in a separate SC than the lottery. Though, the random numbers can be generated either on the blockchain or externally.
- The lottery SC must automatically pay the prize to the winner, divide the prize in case of multiple winners, or accumulate the prize for the next lottery round in case of no winners.

The lottery was realized on the Ethereum blockchain with the Solidity programming language. Furthermore, a front-end for convenient interaction with the lottery was implemented using JavaScript and React[1].

The source code can be found at `https://github.com/ddanijel/bcoln-ct`.

[1] `https://reactjs.org/`

Chapter 2

Design and Implementation

The exact mechanisms of a lottery can be implemented in many ways. The following describes the design choices made in this work.

The most important concept is that there is not only a single lottery contract but rather one contract for each round of the lottery. One round consists of players betting and the picking of a winner. After a winner has been chosen the specific lottery is closed and a new lottery is deployed. The lotteries are controlled via a central lottery factory that spawns new lotteries, knows of the current active lottery, and is the interface by which the lottery issuer can manage the lottery flow and the players can set there bets. The advantage of this design is that each lottery round is retrievable by its own address and the factory can be queried for the list of all past lotteries. This seems more convenient than the case in which only one lottery is reused for each round. We can avoid having to scan through the blockchain to retrieve the transaction history of the lottery and reconstruct its intermediary states.

Other design decisions are:

- A player can bet multiple times on the same or different numbers in a single lottery round.
- The lottery issuer sets a ticket price which is the minimum price a player has to pay for his bet. The system does not prevent the player to pay more, but he does not get any benefits from it.
- The issuer of the lottery factory sets the number range by providing a maximum number that can be bet.

```
contract LotteryFactory {

    address manager;
    uint ticketPrice;
    uint maxGuessNumber;
    address[] allLotteries;
    Lottery currentLottery = Lottery(address(0x0));
    RandomNumberOracle randomNumberGenera-
tor = RandomNumberOracle(address(0x0));
```

```
constructor(uint _ticketPrice, uint _maxGuessNumber) public {
    manager = msg.sender;
    ticketPrice = _ticketPrice;
    maxGuessNumber = _maxGuessNumber;
    randomNumberGenerator = new RandomNumberOracle();
    currentLottery = new Lottery(_ticketPrice, address(this), ad-
dress(randomNumberGenerator), maxGuessNumber);
    allLotteries.push(address(currentLottery));
}

    function play(uint8 guess) public payable {
        require(currentLottery != Lot-
tery(address(0x0)), "There is no lottery running.");
        require(msg.value >= currentLot-
tery.getTicketPrice(), "You have to send enough money.");
        currentLottery.play(guess, msg.sender);
    }

    function pickWinner() public {
        require(msg.sender == manager, "You are not authorized.");
        address[] memory win-
ners = currentLottery.pickWinner(address(this));
        if (winners.length != 0) {
            uint prize = address(this).balance / winners.length;
            for (uint i = 0; i < winners.length; i++) {
                winners[i].transfer(prize);
            }
        }
        // creating a new round immediately after the winner is selected
        currentLottery = new Lottery(ticketPrice, address(this), ad-
dress(randomNumberGenerator), maxGuessNumber);
        allLotteries.push(address(currentLottery));
    }
```

Chapter 3

Random Number Generator