P2P Finalterm - TRY

Geremia Pompei (MAT. 638432)

June 1, 2022

1 Introduction

In this work I have created a lottery using Ethereum smart contracts on a local blockchain simulated with *Ganache* tool. I used the online IDE *Remix* to deploy and test contracts. To develop this part I have created two contracts:

- `ERC721.sol`
- `Lottery.sol`

1.1 ERC721.sol

Extends interfaces `IERC721.sol` and `IERC721Metadata.sol` that provide methods and events to build a *Non-fungible token* (NFT). While the first interface is the one that has main features about all functions that an NFT contract should have the second has getter methods to retrieve metadata from contract. `ERC721.sol` is a contract able to handle NFTs with different methods. Most important are:

- **mint**: able to create new NFT (there is also SafeMint that provides further chucks to be sure that `_to` address can receive NFT)
- **transferFrom**: able to transfer an NFT from an owner to another (there is also SafeTransferFrom that provides further chucks to be sure that `_to` address can receive NFT)
- **balanceOf**: able to provide the number of NFT of an owner
- **ownerOf**: able to provide the owner of an NFT from his `_tokenId`

I used this contract to generate NFT that are lottery prizes that will be assigned to winning players. To do this I exploit only `SafeMint` and `SafeTransaction` functions.

1.2 Lottery.sol

Contract main methods are:

- **constructor**: creates a new lottery passing the address of `ERC721.sol` contract, the duration given to players to buy tickets, the parameter k used to generate random numbers and the ticket price
- **startNewRound**: start new round of lottery
- **buy**: with this method players can buy a ticket and play numbers
- **drawNumbers**: random number are extracting with this function to create a winning ticket. Here there is a check that change a numbers if in two or more of previous 5 there are equals
- **givePrizes**: function to give NFT prizes to who win lottery and transfer money of tickets to lottery operator. If more players win the same prize are generated two NFT with the same image url
- **mint**: to mine a new NFT prize that is owned from contract and then transferred to a player when win it
- **closeLottery**: function that disable the lottery and if there is an open round gives coin back to players

`Lottery.sol` inherits `IERC721TokenReceiver.sol` to receive an NFT from `ERC721.sol`. To manage the different functions in sequence I used a variable `state` and in all change of this is emitted an event to notify it.

2 Operations

To deploy and use contracts can be used *Remix* online IDE or *Truffle* tool on node js.

- *Remix*: import zip archive on `https://remix.ethereum.org/`, compile `Lottery.sol` contract and deploy to play with it. This can be done in different blockchain networks, I used `Ganache` in local environment because with default network the function `blockhash` doesn't work. I also implemented the test `Lottery_test.sol` to run automatically a simulation of lottery contract inside Remix
- *Truffle*: you need to have `node.js`, `npm` and `Ganache` installed. To install truffle the command is `npm install -g truffle`. To deploy contracts you should open `Ganache` and the terminal, navigate in the folder of the project and run `truffle migrate --reset`. To run truffle test run `truffle test`

To play with `Lottery.sol` contract a list of operations could be:

Method	Gas estimation	Gas spent
`constructor`	3183320	3183320
`buy`	140891	139880
`drawNumbers`	111937	111937
`givePrizes`	101781	41642
`mint`	183805	183805
`startNewRound`	44328	29328
`closeLottery`	48602	48602

Table 1: Comparison between gas estimated by remix and gas spent

1. Deploy `ERC721.sol` contract passing *name*, *symbol* and *baseUri* (e.g. `ERC721("NFT_TRY", "NFTRY", "")`)
2. Deploy `Lottery.sol` contract passing ERC721 contract address, *duration*, *k* and *ticketPrice* (e.g. `Lottery(0x..., 3, 2, 100)`)
3. Buy tickets from Lottery contract using other accounts different from owner of lottery and passing an array of first 5 numbers an powerball number (e.g. `lottery.buy([1, 2, 3, 4, 5, 1])`)
4. Mine an NFT for each class of prize passing the url of image associated to the NFT (e.g. `mint("http://img.com")`)
5. Using owner account call functions `drawNumbers` and `givePrizes` (function `drawNumbers` can be reverted the first times because to generate random number it waits the generation of block `startRoundBlockNumber + duration + k`)
6. To start a new round you can call `startNewRound` function, else to close lottery you can call `closeLottery` method

3 Gas estimation

In Table 1 we can find a comparison between gas estimated and gas spent according main methods of lottery contract. We can understand from this that methods like `startNewRound` and `closeLottery` that have few write operations cost much less than the others that are more complex. The `constructor` is the highest operation in term of gas because it is the responsible of the instantiation of the contract inside the blockchain. `givePrizes` function has a big difference between gas estimation and gas spent because when players win the lottery are activated lines inside the method that are able to give reward to them. It's really improbable that many players receives a reward for the rules of the lottery so the gas spent, like in this case, probably will be often lower than estimation.

4 Security analysis

4.1 Miner attack

To understand and exploit winning numbers can be done the *miner attack*. Random number used to create winning ticket is generated after that all players buy them tickets.

```
function __random() private view returns(uint256) {
    uint256 blockNumber = startRoundBlockNumber + duration + __k;
    require(blockNumber <= block.number, "Block not yet generated");
    return uint256(keccak256(abi.encode(
        blockhash(blockNumber), round, block.timestamp
    )));
}
```

This function takes the block that is created after the block number used to start current round plus the duration in term of block of the period that a player can buy a ticket plus a `__k` value setted at the beginning by the owner of the lottery. Given this block is computed his hash that is combined with current round and timestamp of current block. In the end is computed the hash of the combination and converted in a `uint256`.
A miner to attack the function `drawNumbers` that use `__random` function should manipulate timestamp of current block and the hash of `(startRoundBlockNumber + duration + __k)` block. To do this a miner should mine both blocks. Actually only the owner know __k parameter so the miner should be him. In the end:

- *miner* can attack only if mines both blocks and is the owner (very improbable becaues he should win each time the POW for the indicated blocks)
- *owner* can't attack because he knows `__k` parameter but can't know the hash and the timestamp of blocks used in random function
- *players* can't attack because they don't know `__k` parameter and blocks information

4.2 Phishing attack

This attack can be done exploiting the `tx.origin` parameter to steal sensible information from users. `Lottery.sol` contract store and check addresses of owner and players using `tx.origin` because I decide that only user accounts can play with lottery and not other smart contracts. In particular in this smart contract aren't use sensible information of users so this can't be exploited in a phishing attack.