IERG 4360 / IEMS 5725 (Fall 2024)
Blockchain and Applications
Programming Assignment (10% of the coursework)
Deadline: 20 October 2024, 23:59 (Sunday).

Introduction

In this programming assignment, you need to implement a smart contract on Solidity that allows two users to play card-based "Jungle". To ease you from setting up a complicated Ethereum testing network, we recommend Remix, an online integrated development environment (IDE) for Solidity. Also, we offered a simple Remix tutorial on Blackboard.

Requirements (Part 1) [60%]

You need to write a small contract that provides the following functionalities and requirements:

1. Two players can register for a card-based Jungle game.
    a. This should be done by sending 1 ethers when invoking the **`register`** function.
    b. Return the received ethers if registration fails.
2. After registration, they can input the **TWO** choices via **`inputMove`** function. The choices represent the first and second round move for the player.
    a. Available choices (from the highest rank to the lowest): **Lion, Wolf, Cat, Rat**
    b. Rule:
        i. Lion captures Wolf;
        ii. Wolf captures Cat;
        iii. Cat captures Rat;
        iv. Rat captures Lion
    c. The choices must be **unique** for a player, e.g., (Lion, Lion) is not allowed. (use require to enforce it)
3. **`winning()`** function: anyone can check the winner after the players provide their choice.
    a. Both player must have inputted their choices.
    b. The player having the highest win count is the winner; she/he gets all the bets (2 ethers)
    c. Return 1 ether to both players if draw (no. of wins equal)
    d. Reset the game

Example:

- Alice inputs (Lion, Cat), and Bob inputs (Rat, Cat); Bob wins once, but Alice does not, so Bob gets 2 ethers.
- Both parties input (Rat, Cat), which draws the game; Alice and Bob get 1 ether.

Security Issues

You can assume all players are honest for Part 1.

Template

For your reference, here is a smart contract template (pw: blockchain). Fill in the code after the TODO comments.

Requirements (Part 2) [40%]
[You need to submit **1 file**; so, make a backup for part 1 in case you cannot finish part 2.]

Q1. However, there is a potential vulnerability/loophole: leave a comment in the code to explain briefly what could possibly go wrong. [**10**%]

> Recall from the DNS example (Lecture note 4A) and the rock-paper-scissor game (Lecture note 4B) that a transaction is first broadcasted before it is "mined" by the miners. How can the adversary attack the victim who inputs the choices first?

Q2. (trying to fix the game) You need to modify the contract to utilize the hash function `sha256()` (or `keccak256`) to "hide" the user's input during `inputMove()`, *i.e.*, `inputMove()` will take two "commitments", e.g.,(`sha256(Move1),sha256(Move2)`) as inputs for each player. The game enters the reveal phase if both players input their "hidden" move. Each player can only (successfully) call `inputMove()` once.

1. Players A and B will invoke `reveal()` (once only) to input his/her moves (`Move1, Move2`) in plaintext as commitment openings.
    a. The function will check if the commitment openings (`Move1, Move2`) are valid compared to the stored commitment during `inputMove()`.
    b. If the opening(s) is/are invalid, you can punish the player by slashing his/her bet. (In this case, you may distribute the bets and reset the game)

    Example: Both parties commit (Cat, Cat); And they are invalid after `reveal()`
2. [Optional, **5% Bonus**] If player A has revealed his move but player B has not done so for a period of time, player A can invoke `winning()` to win the game.
    (Hint: note the differences between timestamp and block number)

After both players reveal their input, (the owner) calls `winning()` to determine the winner. [**15**%]

Q3. (the real fix) It is still vulnerable to front-running attacks; why and how do you fix it? Give your explanation in the code. [**5**%]
Check the example code `commit_reveal.sol` and fix the game. [**10**%]

Submission: Submit your codes as a `[studentid].sol` file to Blackboard.

Deadline: 20 October 2024, 23:59 (Sunday).

Questions
If you have any questions, please feel free to raise them on Blackboard/Piazza or email us.