Tópicos em Segurança da Informação
Aula Primeiro Contrato

Ivan Sendin

FACOM - Universidade Federal de Uberlandia
ivansendin@yahoo.com,sendin@ufu.br

26 de setembro de 2024

"Does God want goodness or the choice of goodness?
Is a man who chooses to be bad perhaps in some way
better than a man who has the good imposed upon him?"

Laranja Mecânica - Anthony Burgess

Geral

- Contrato é autônomo
- As pessoas só vão se comportar se forem obrigadas
- Havendo oportunidade de roubar R$0.01, alguem cria um robo
- Havendo oportunidade de atrapalhar!

Geral

- self-executing
  O sistema **UFU** diz que voce tem direito a um diploma...mas não emite o diploma
- trustless
  Não preciso confiar nos atores...apenas no contrato(s)
  Para confiar no contrato voce precisa confiar no computador que ele roda!

O contrato

- Tomei algumas decisões de projeto
- Visando simplicidade
- Nem sempre boas!
- Posso fazer assim??
- Se voce imaginar um algoritmo...pode!
- Vai ser bom? vai funcionar?
- bugs 1 a 3 de proposito!
  Depois...

Uso do contrato

- Vc (desenvolvedor) faz o contrato, o deploy e uma propaganda
- O dono de um "carro tokenizado"
- Vou montar uma empresa e crio 10 tokens para pagar as contas
  Similar a uma chamada de capital
  Poderia ser esse contrato!!
- Jogador da NBA
  Nao sei se é trustless (link?)

Os bugs

- 3 bugs
- O local onde o problema “aparece” esta indicado
- Roubo?? Perdas?? DoS ??
- Voces devem (pelo menos) arquitetar uma solução
- Remix

```
  pragma solidity >=0.4.25 <0.6.0;


contract VerySimpleToken {
        string name;
        address tokenOwner;
        constructor(string memory  _n) public {
                tokenOwner = msg.sender;
               name = _n;
        }

        function transfer(address to) public {
                 require (msg.sender == tokenOwner);
                tokenOwner = to;
        }

        function isOwner(address d) public returns (bool) {
                return (d == tokenOwner);
        }

}
```

Token de verdade: ERC721

- Vamos leiloar um token... o contrato consegue **entregar** o token
- Vamos exigir uma garantia (collateral) para os participantes
- Para deixar mais interessante o contrato poderá lidar com vários leiloes
- O contrato cobrará uma taxa pelo uso (monetização)

```
pragma solidity >=0.4.25 <0.6.0;

import "./VerySimpleToken.sol";

contract TokenAuction {
  enum AuctionStates { Prep, Bid, Finished}

  address payable owner;
  struct OneAuction {
          AuctionStates myState;
        mapping (address => bool) collateral;
          uint blocklimit;
          address winner;
          address payable tokenOwner;
          uint winnerBid;
          bool payment;
        VerySimpleToken token;
  }

  uint collateralValue;

  //Aqui eu decidi por uma taxa igual para todos
  //poderia ser diferente por leilao(colocar na struct)
  // poderia ser um porcentagem...ou...ou...
  uint contractFee;

  mapping (string => OneAuction) myAuctions;
```

```
    constructor(uint c,uint fee) public {
      //Qual a diferenca do owner para os demais usuarios??
    owner = msg.sender;
    collateralValue = c;
    contractFee = fee;
  }

function createAuction(string memory name, uint time, VerySimpleToken t) public {
    require (t.isOwner(msg.sender),"You must own the token to create one auction!");
    OneAuction memory l;
    l.blocklimit= block.number + time;
    l.myState = AuctionStates.Prep;
    l.winnerBid = 0;
    l.tokenOwner = msg.sender;
    l.payment = false;
    l.token = t;
    //Bug1
    myAuctions[name]=l;
  }

// Se o token for tranferido e o leilao nunca inicar...perda de token
// o blocklimit tambem seria melhor inicalizado aqui!
function initAuction(string memory name) public {
      require (myAuctions[name].myState == AuctionStates.Prep,
                                                "The auction should be in Prep state");
      require (myAuctions[name].token.isOwner(address(this)),
                                                "The contract should own the token");
          myAuctions[name].myState = AuctionStates.Bid;

  }
```

```
function verifyFinished(OneAuction storage a) private {
    if (block.number > a.blocklimit) {
        a.myState = AuctionStates.Finished;
      }
}

// E se o mesmo endereço mandar o collateral mais de uma vez??
  function sendCollateral(string memory name) public payable {
    require (myAuctions[name].myState == AuctionStates.Bid, "The auction should be in Bid sta
    require (msg.value == collateralValue,"You should send the corretc value!");
    myAuctions[name].collateral[msg.sender] = true;
  }

  function bid(string memory name, uint v) public {
    OneAuction storage a = myAuctions[name];
    verifyFinished(a);
    require (a.myState == AuctionStates.Bid, "The auction should be in Bid state");
    require (a.collateral[msg.sender],"Send the collateral value before bidding.");
    if (v>a.winnerBid) {
      a.winnerBid = v;
      a.winner = msg.sender;
    }
  }
```

```
function claimToken(string memory name) public payable {
    //Bug2
    OneAuction storage a = myAuctions[name];
    verifyFinished(a);
    require (a.myState == AuctionStates.Finished, "Wait a minute, boys, this one is not dea
    require (msg.value == a.winnerBid-collateralValue, "Pay First....");
    a.token.transfer(msg.sender);
    a.collateral[msg.sender] = false; //just to flag claimToken! DANGER!
  }

  function claimCollateral(string memory name) public {
    OneAuction storage a = myAuctions[name];
    verifyFinished(a);
    require (a.myState == AuctionStates.Finished, "Wait a minute, boys, this one is not dea
    require (a.collateral[msg.sender],"Nope");
    require (msg.sender != a.winner,"You cant claim the collateral");
    msg.sender.transfer(collateralValue);
    myAuctions[name].collateral[msg.sender] =false;

  }
```

```
function getProfit(string memory name) public {
  OneAuction storage a = myAuctions[name];
  verifyFinished(a);
  require (a.payment==false, "I will not pay twice!");
  require (a.collateral[a.winner] ==false,"Wait for payment");
  a.tokenOwner.transfer(a.winnerBid-contractFee);
  a.payment=true;
}

function getFee() public {
  //Bug3
  // O balance/saldo é uma propriedade de um endereco
  owner.transfer(address(this).balance);
}
```

```
from brownie import *
import brownie

def main():
    contractOwner = accounts[0]
    tokenOwner = accounts[1]

    bidder1 = accounts[2]
    bidder2 = accounts[3]
    bidder3 = accounts[4]
    bidderFake = accounts[5]

    token = VerySimpleToken.deploy("Token1",{'from': tokenOwner})
    auction = TokenAuction.deploy(100,50,{'from': contractOwner})

    auction.createAuction("Leilao1", 12, token,{'from':tokenOwner})

    token.transfer(auction,{'from':tokenOwner})

    auction.initAuction("Leilao1")

    auction.sendCollateral("Leilao1",{'from':bidder1,'value':100})
    auction.sendCollateral("Leilao1",{'from':bidder2,'value':100})
    auction.sendCollateral("Leilao1",{'from':bidder3,'value':100})
```

```
auction.bid("Leilao1",1000,{'from':bidder1})
auction.bid("Leilao1",1010,{'from':bidder2})
auction.bid("Leilao1",1020,{'from':bidder3})
auction.bid("Leilao1",1010,{'from':bidder1})
auction.bid("Leilao1",1050,{'from':bidder2})

try:
  auction.bid("Leilao1",1099,{'from':bidderFake})
except:
  print("Dont!!")
```

```
try:
    auction.bid("Leilao1",1000,{'from':bidder1})
    auction.bid("Leilao1",1000,{'from':bidder1})
    auction.bid("Leilao1",1000,{'from':bidder1})
    auction.bid("Leilao1",1000,{'from':bidder1})
    auction.bid("Leilao1",1000,{'from':bidder1})
except:
    print("Dont!!")

auction.claimToken("Leilao1",{'from':bidder2,'value':1050-100})

auction.claimCollateral("Leilao1",{'from':bidder1})
auction.claimCollateral("Leilao1",{'from':bidder3})
try:
    auction.claimCollateral("Leilao1",{'from':bidder2})
except:
    print('Tentei roubar...mas nao consegui...')

auction.getProfit("Leilao1",{'from':contractOwner})
auction.getFee({'from':tokenOwner})
```

```
print(bidder1.balance())
print(bidder2.balance())
print(token.isOwner.call(bidder2))
print(bidder3.balance())
print(contractOwner.balance())
print(tokenOwner.balance())
```

```
10000000000000000000
99999999999999998950
True
10000000000000000000
10000000000000000050
10000000000000001000
```

- import, struct, address payable
- memory (storage)
- this
- msg.value
- balance

1. Se o `claimToken` nunca for chamado o ganhador perde o collateral mas...o token fica preso no contrato, o dono do token nao ganha nada. Coloque um limite de blocos para o `claimToken` ser chamado, se ele nao for chamado o token volta para o dono e ele recebe o collateral.
2. Permitir que o dono do contrato altere a taxa (`contractFee`) para os novos leiloes
3. Identifique os problemas e arrume os 3 Bugs
4. Crie um token que recebe 10% das taxas. Esse token é recebido no construtor
5. Agora com 10 tokens....