John Vitor da Silva Cunha
11821BCC005

Link do GitHub com o código: https://github.com/johnsigma/SmartContractsBugs

BUG 1:

No código original não havia verificação se o leilão de determinado nome já existia, permitindo que leilões existentes fossem sobrescritos. Para corrigir isso fiz uma verificação na criação do leilão, onde é verificado o valor do campo *blocklimit* do leilão com o nome passado como parâmetro. Se o valor de *blocklimit* deste leilão não for 0, quer dizer que outro leilão com este nome já foi inicializado anteriormente, proibindo a criação do novo leilão. Isso também limita para que não seja possível iniciar um leilão com o *blocklimit* como 0. Porém na prática um leilão que não dura nem um bloco não faz muito sentido, além de que, a menos que uma *blockchain* seja criada do zero para abranger este contrato, essa situação nunca vai ocorrer, pois qualquer *blockchain* usual já tem seu número de blocos maiores que 0.

BUG 2:

O segundo bug permitia que um usuário mal-intencionado observasse a *blockchain* para visualizar o valor do lance vencedor. Com isso ele próprio poderia invocar a função *claimToken*() e reivindicar o *token* para si, passando como valor (*msg.value*) o valor da aposta vencedora, mesmo não participante honestamente do leilão. Para corrigir isto basta verificar se o endereço do chamador da função (*msg.sender*) é de fato o endereço do ganhador do leilão (*myAuctions[name].winner*). Com isso somente o vencedor do leilão poderá reivindicar o *token*.

BUG 3:

Neste bug, a função *getFee*() permitia que o dono do contrato sacasse todo o saldo do contrato, que poderia incluir colateral e lances que não são destinados a ele, já que o dono do contrato deve receber apenas as taxas do contrato. Para corrigir isto foi criado uma variável para guardar as taxas acumuladas do contrato (*contractFeeBalance*), essa variável será incrementada com o valor da taxa de contrato (*contractFee*) toda vez que o dono do *token*, após finalizado o leilão, reivindica o valor do lance vencedor. Com isso o dono do *token* (ou ex-dono) recebe o valor do lance descontando a taxa do contrato. Na função *getFee*() é colocado duas verificações, se quem está chamando-a é o dono do contrato (essa verificação faz sentido, mas não é necessária para a correção do bug) e se o saldo das taxas é maior que 0

(*contractFeeBalance*). Depois das verificações, em vez de transferir pro dono do contrato todo o saldo, é transferido somente o valor acumulado das taxas.

CÓDIGO:

```
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;

    uint contractFeeBalance;

    //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;
        contractFeeBalance = 0;
    }
```

```
function createAuction(
    string memory name,
    uint time,
    VerySimpleToken t
) public {
    require(
        t.isOwner(msg.sender),
        "You must own the token to create one auction!"
    );

    require(
        myAuctions[name].blocklimit == 0,
        "An auction with this name has already started"
    );

    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 endere¸co mandar o collateral mais de uma vez??
function sendCollateral(string memory name) public payable {
    OneAuction storage a = myAuctions[name];

    require(
        a.myState == AuctionStates.Bid,
        "The auction should be in Bid state!"
    );
    require(
        msg.value == collateralValue,
        "You should send the corretc value!"
    );
    a.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
    require(
        msg.sender == myAuctions[name].winner,
        "Only the winner can claim the token!"
    );
```

```
        OneAuction storage a = myAuctions[name];
        verifyFinished(a);
        require(
            a.myState == AuctionStates.Finished,
            "The auction should be in Finished state!"
        );
        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,
            "The auction should be in Finished state!"
        );
        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);
        contractFeeBalance += contractFee;
        a.payment = true;
    }

    function getFee() public {
        //Bug3
        require(msg.sender == owner, "Only the owner can get the fee!");
        require(contractFeeBalance > 0, "No fee to get!");
        uint amount = contractFeeBalance;
        contractFeeBalance = 0;
        owner.transfer(amount);
    }
}
```