Lab 03:
Defending your Đapp

Luke Strgar

LAB OUTLINE

1. REVIEW
2. EXERCISE: SMART CONTRACT SECURITY

1 REVIEW

REVIEW

ARITHMETIC OPERATION

Unsigned Integer Overflow

```
  0xFFFFFFFFF……FFFFFFFFFFF
+ 0x000000000……000000000001
-----------------------------
= 0x000000000……000000000000
```

Unsigned Integer Underflow

```
  0x000000000……000000000000
- 0x000000000……000000000001
-----------------------------
= 0xFFFFFFFFF……FFFFFFFFFFF
```

REVIEW

MODIFIERS

Tool for restricting function access to conditions you specify

Who can call a function?

Is there a certain time a function can be called?

Is there a condition on the internal contract state under which a function can be called?

```
modifier onlyBy(address _account)
{
    require(msg.sender == _account);
    // Do not forget the "_;"! It will
    // be replaced by the actual function
    // body when the modifier is used.
    _;
}

/// Make `_newOwner` the new owner of this
/// contract.
function changeOwner(address _newOwner)
    public
    onlyBy(owner)
{
    owner = _newOwner;
}

modifier onlyAfter(uint _time) {
    require(now >= _time);
    _;
}
```

REVIEW

FALLBACK FUNCTIONS

Executed When

1) Contract receives value without a specific function being invoked (except when value is sent because of a contract self destruct)

2) Different user or contract attempts to invoke function that does not exist

```
pragma solidity ^0.4.0;

contract Test {
    // This function is called for all messages sent to
    // this contract (there is no other function).
    // Sending Ether to this contract will cause an exception,
    // because the fallback function does not have the `payable`
    // modifier.
    function() public { x = 1; }
    uint x;
}
```


2 EXERCISE

THE LAB

External Calls, Inheritance, Smart Contract Security

Implement two different “auction” contracts, GoodAuction and BadAuction

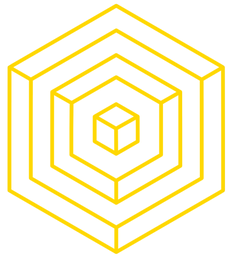
BadAuction suffers from 3 security vulnerabilities

1. Unsigned integer arithmetic
2. Failure to limit function access
3. Vulnerable to fallback function lack of payability

GoodAuction is secure from these vulnerabilities

- We want to eliminate conditionals that depend on success or failure of a payment
- Force users to “pull” their funds from our contract

More specifics in skeleton code comments



github.com/Blockchain-for-Developers/sp18-lab03

README.md

Attack the Auction

Welcome to lab03. In this assignment you'll be getting acclimated with external calls, inheritance, and smart contract security.

Homework Instructions

Your task is to complete both `BadAuction.sol` and `GoodAuction.sol` such that they are each, respectively, vulnerable and protected from three specific security vulnerabilities. Notice that both contracts inherit from `AuctionInterface.sol`.

The contracts `Poisoned.sol` and `NotPoisoned.sol` are players in an adversarial system. Both can conduct external calls to the auctions in order to place bids and lower their bid if they are the current highest bidder. While `NotPoisoned.sol` will act as expected, `Poisoned.sol` will attempt to take advantage of one of `BadAuction.sol`'s security vulnerabilities. **You Should NOT Change These Contracts**

Implement both auctions in order to demonstrate your understanding of what makes smart contracts vulnerable and how to secure them.

All work should be done in *contracts/BadAuction.sol* and *contracts/GoodAuction.sol*

QUESTIONS?

SEE YOU NEXT TIME

Testing and Web3.js Integration

Javascript Syntax

Mocha.js

Chai.js

Asynchronous Calls

Web3.js

Front-End Integration