NSFW

Smart Contract Vulnerabilities and Vyper

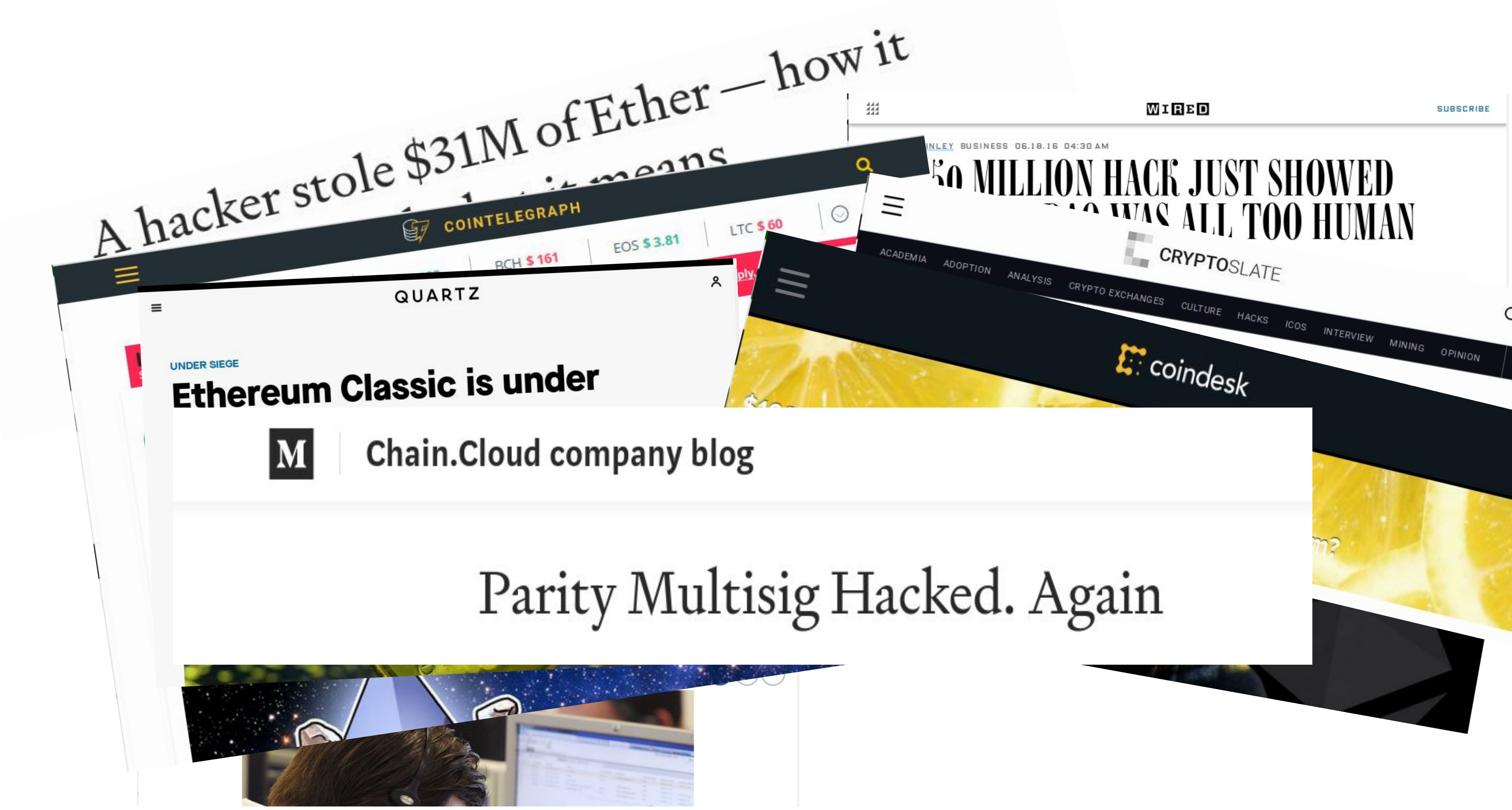
Souradeep Das
Next Tech Lab,
Ex- UC Berkeley Blockchain Lab

Hi!
I’m **Dave**!

How far has Ethereum come in 4 years?

Total addresses	Daily active users	24h transactions	Smart Contracts
>50m	**95.54k**	**1.58m**	**>1m**

Business logic coded as software had been automating and revolutionising the world around us, unless ...



A hacker stole $31M of Ether — how it

it means

COINTELEGRAPH

BCH $ 161

EOS $ 3.81

LTC $ 60

QUARTZ

UNDER SIEGE

Ethereum Classic is under

WIRED

SUBSCRIBE

INLEY BUSINESS 06.18.16 04:30 AM

50 MILLION HACK JUST SHOWED

DAO WAS ALL TOO HUMAN

CRYPTOSLATE

ACADEMIA ADOPTION ANALYSIS CRYPTO EXCHANGES CULTURE HACKS ICOS INTERVIEW MINING OPINION

coindesk

M Chain.Cloud company blog

Parity Multisig Hacked. Again

©Souradeep Das

Smart Contracts will radically change the world, but what tends to get lost in the noise is that **coding a smart contract is extremely challenging**

> *“One bad programmer can easily create two new jobs a year”*
>
> ~David Parnas

About

34,200

Ethereum Smart Contracts are vulnerable to hacking due to poor coding that contains bugs

1 in 20 Smart Contracts

Courtesy- Finding The Greedy, Prodigal, and Suicidal Contracts at Scale, NUS

A walk down the memory lane - Solidity Vulnerabilities

> *“We are products of our past, but we don't have to be prisoners of it.”*
>
> ~Rick Warren

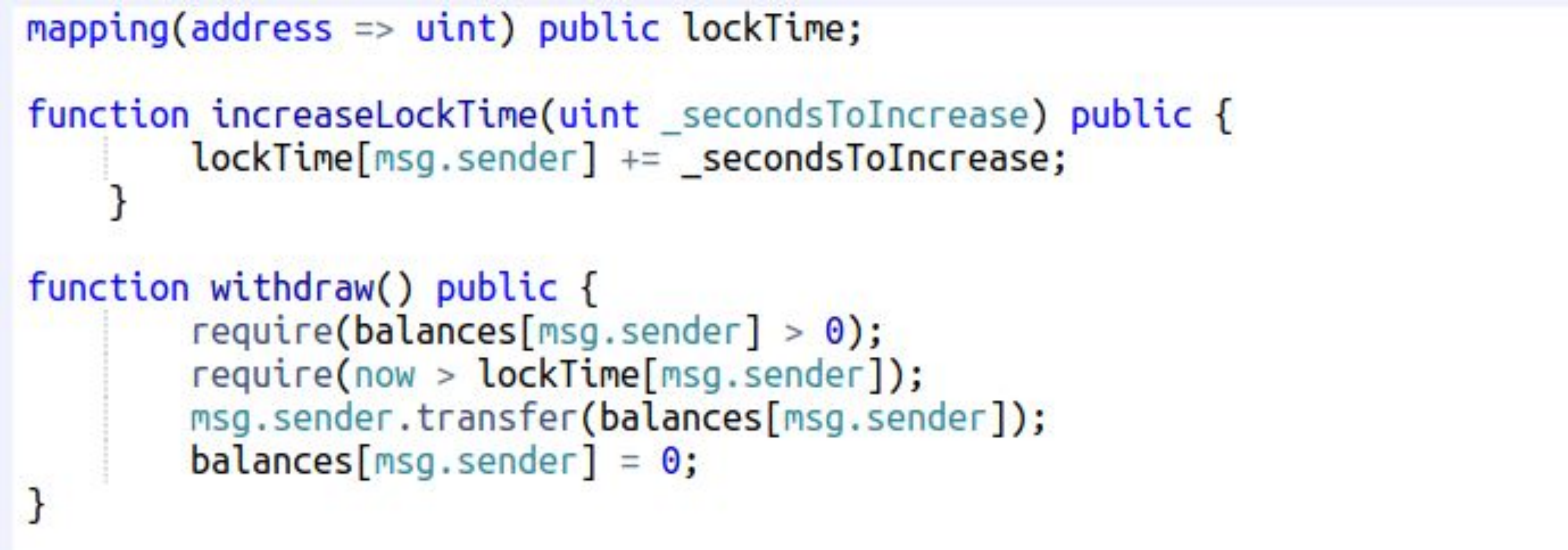
1. Arithmetic Overflows/ Underflows

PoWHC

Vulnerability

Can occur when a fixed size variable is required to store a number that is outside the range of the variable’s type.

Ex- uint8 a = 0;
      a=a+257; // a=1



```
mapping(address => uint) public lockTime;

function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        msg.sender.transfer(balances[msg.sender]);
        balances[msg.sender] = 0;
}
```

PoWHC

Ponzi scheme smart contract called **Proof of Weak Hands Coin** by 4chan

866 ether was liberated due to the vulnerability

Preventive Techniques

Use **OpenZeppelin's SafeMath** Library which has functions to replace math operators like addition, subtraction and multiplication

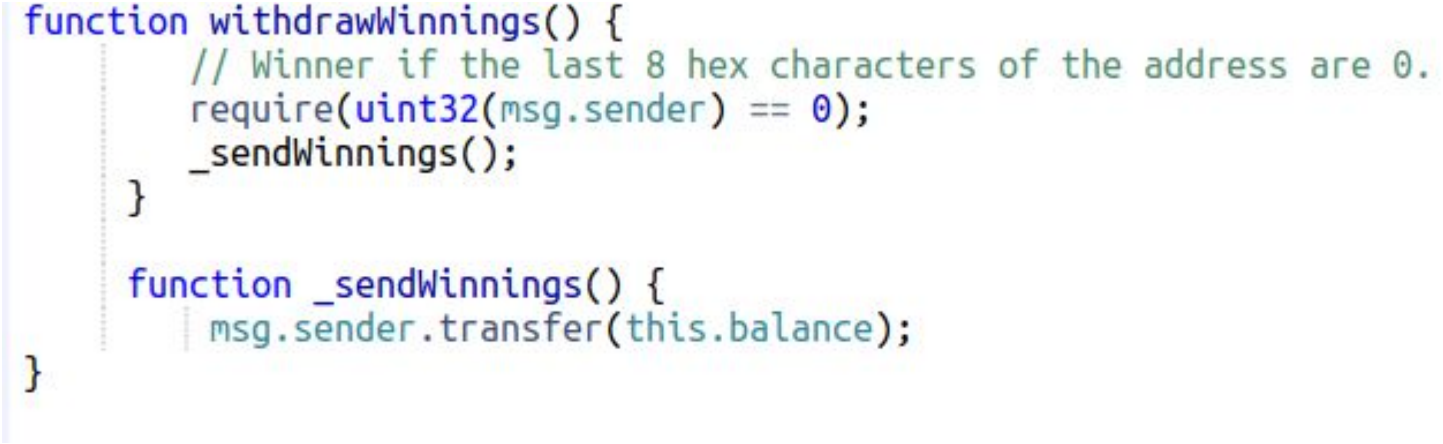
2. Default Visibilities

Parity Wallet First Hack

Vulnerability

The default visibility specifier for smart contracts are **‘public’**.

The issue comes when developers mistakenly **ignore visibility specifiers** on functions which should be private (or only callable within the contract itself)



```
function withdrawWinnings() {
        // Winner if the last 8 hex characters of the address are 0.
        require(uint32(msg.sender) == 0);
        _sendWinnings();
    }

    function _sendWinnings() {
        msg.sender.transfer(this.balance);
}
```

Source: Dr Adrian Manning

©Souradeep Das

Parity First Hack

Functions were accidentally left public, an attacker was a functions, **resetting the ownership** to the attacker

About **$31M worth of Ether** was stolen from primarily three wallets

Preventive Techniques

Always **specify the visibility** of all functions.

Solidity shows **warnings** for functions with no explicit visibility set

Lets create

Dave Token

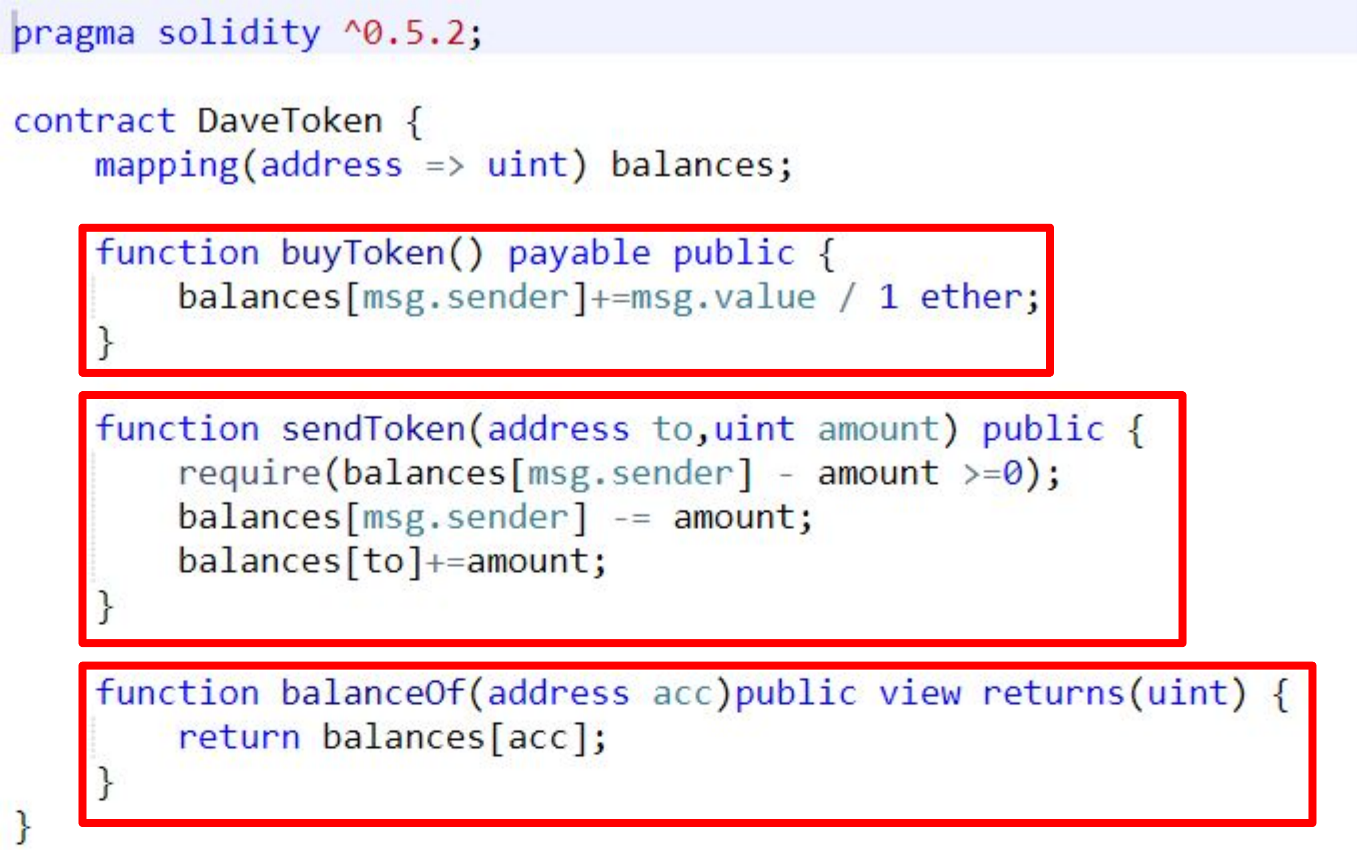
Buy Token

1 Ether = 1 DaveToken

Transfer Token

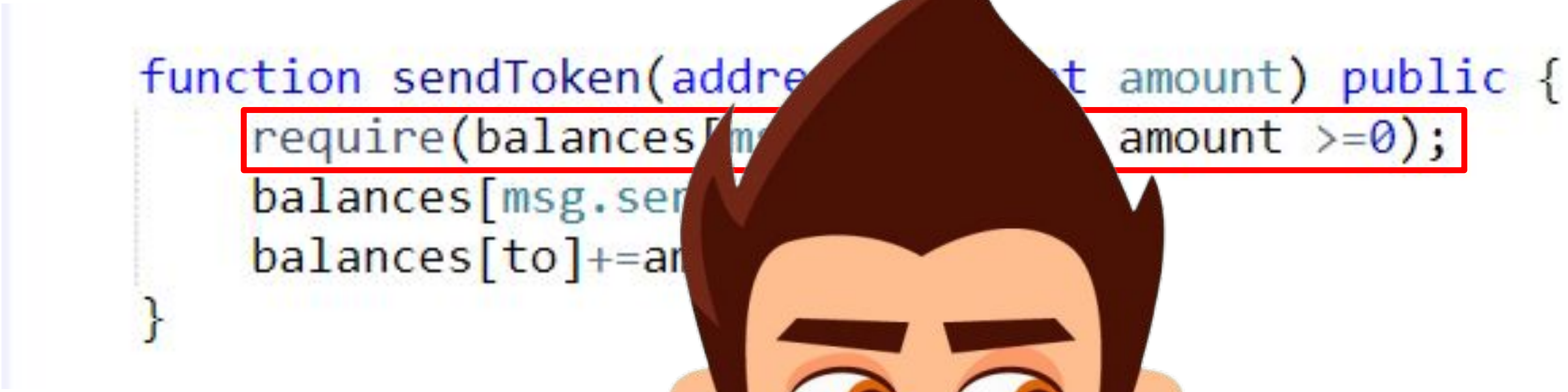
Check Balance

Challenge 1



```
pragma solidity ^0.5.2;

contract DaveToken {
    mapping(address => uint) balances;

    function buyToken() payable public {
        balances[msg.sender]+=msg.value / 1 ether;
    }

    function sendToken(address to,uint amount) public {
        require(balances[msg.sender] - amount >=0);
        balances[msg.sender] -= amount;
        balances[to]+=amount;
    }

    function balanceOf(address acc)public view returns(uint) {
        return balances[acc];
    }
}
```

http://tiny.cc/souradeep



```
function sendToken(addre          t amount) public {
    require(balances[m            amount >=0);
    balances[msg.sen
    balances[to]+=am
}
```

Balances[msg.sender] **ount** >= **0**

uint uint

Always greater than 0 !

Require tends to be usele

©Souradeep Das

3. DAO Hack

Re-entrancy

Vulnerability

Can to an unkn

A co n
the

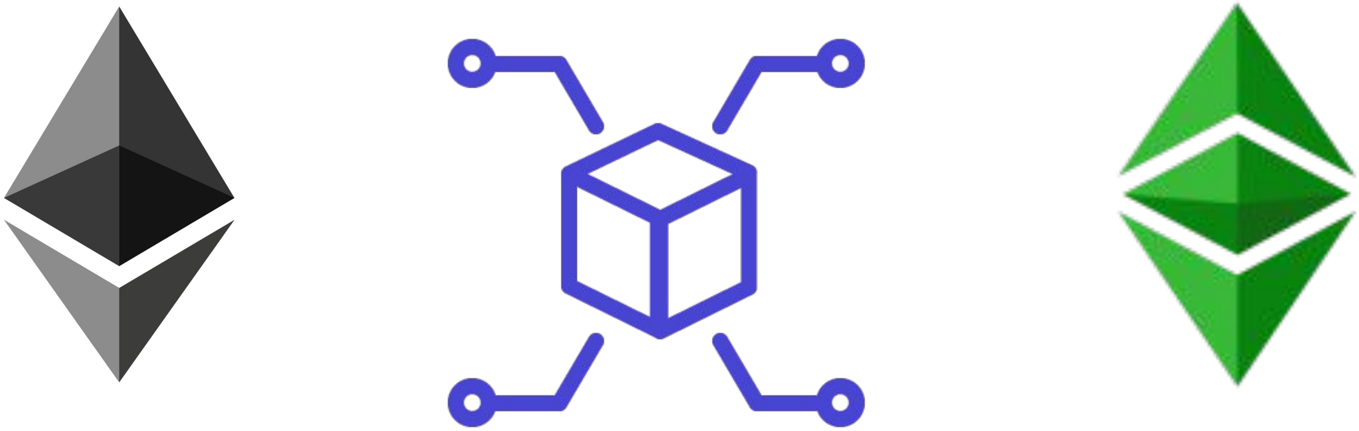
The

```
Vulnerable vul            vuladd)
function () pa
    vul.withdr
}
```

```
mapping (

function
    uint amountToWithdraw = userBalan
    require(msg.sender.call.value(amount ());
    // At this point, the caller's code is executed, and can call withdrawBalance again
    userBalances[msg.sender] = 0;
}
```

On June 17th 2016, The DAO was hacked and **3.6 million Ether ($50 Million)** were stolen using the reentrancy attack.

Ethereum Foundation issued a critical update to rollback the hack. This resulted in Ethereum being **forked into Ethereum Classic** and Ethereum.



Preventive Techniques

Use **transfer()** function instead of **call.value()** for sending ethers
- only sends **2300 gas** - not enough for re-entering

Ensure all logic that changes **state variables** happen before ether is sent out of contract

Adding **Mutex** as a state variable to lock the contract

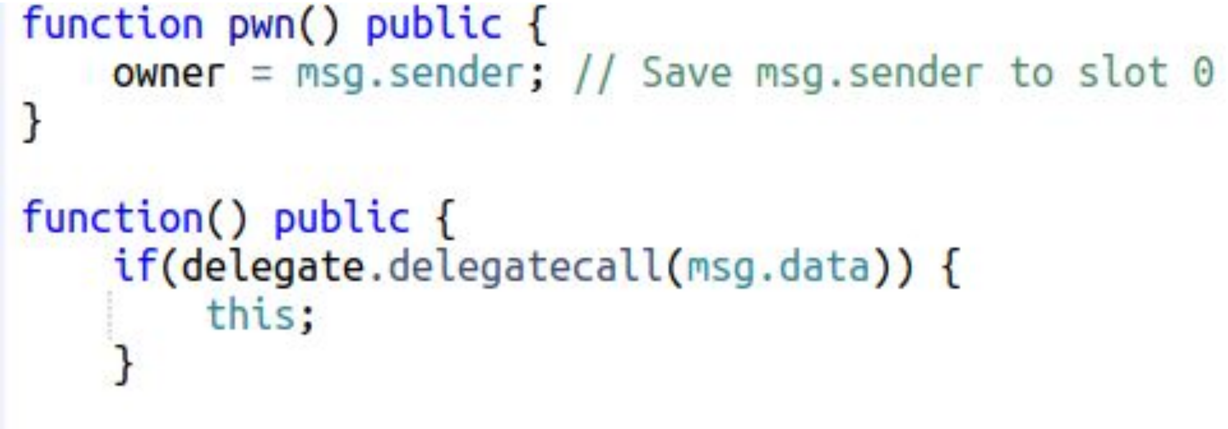
4. Delegatecall

Parity Wallet Second Hack

Vulnerability

Delegatecall **overrides** the second contract’s storage with the storage of the calling contract.

Can lead to changing the owner of the first contract by changing the **first contracts storage**.



```
function pwn() public {
    owner = msg.sender; // Save msg.sender to slot 0
}

function() public {
    if(delegate.delegatecall(msg.data)) {
        this;
    }
```

Source: Ethernaut

©Souradeep Das

Parity Second Hack

Library contract for multisig wallet had this vulnerability

User could get access to library contract and could call the **kill() function**

And the contract suicided

Preventive Techniques

Use **‘library’** keyword for implementing library contracts

Build state-less libraries so that contracts are **not self destructible**

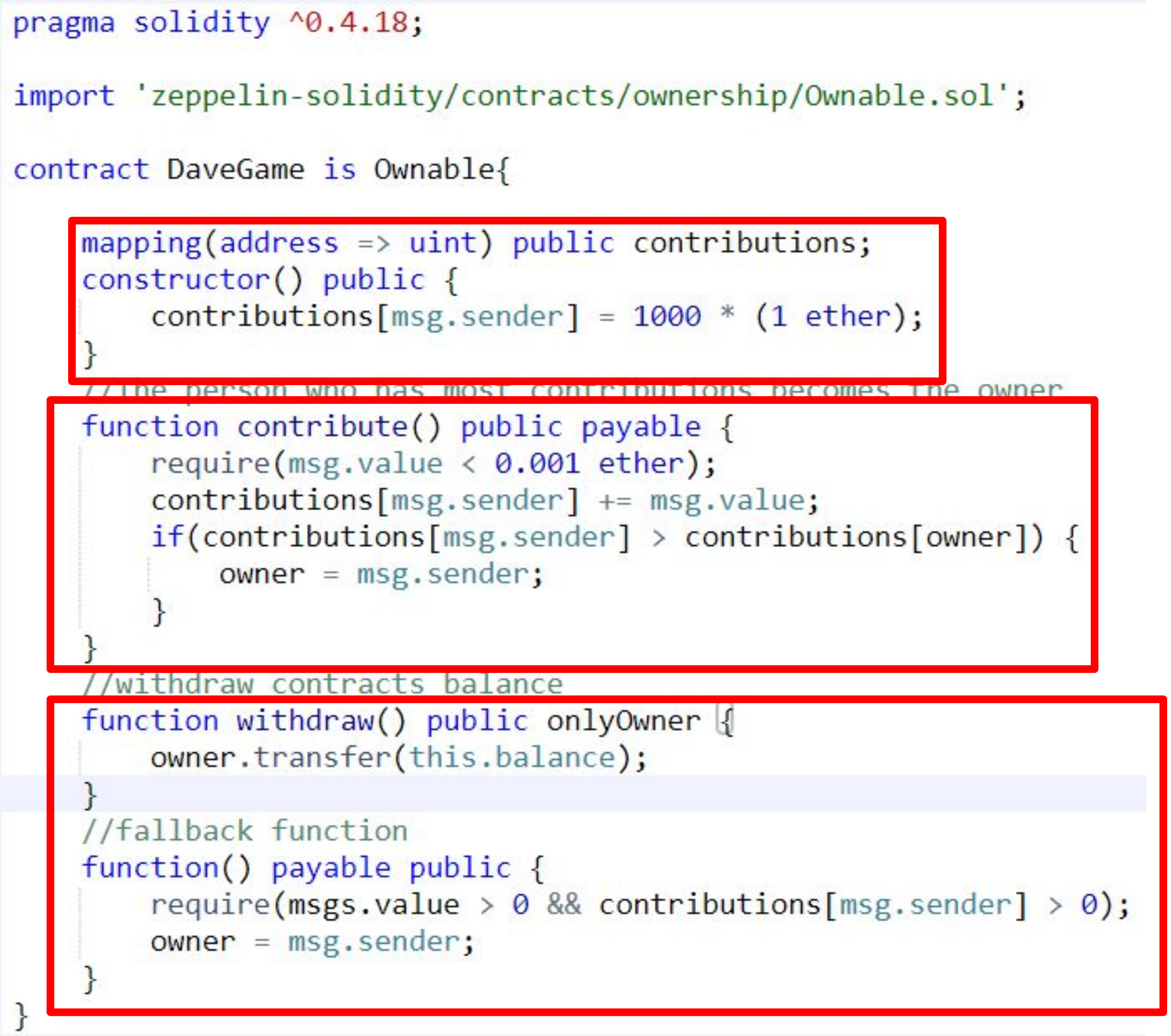
- Players will have to contribute to smart contract (Pool)
- The player with most contributions over time is chosen as the Leader
- The Leader has control of all the funds

Contribute

Withdraw All

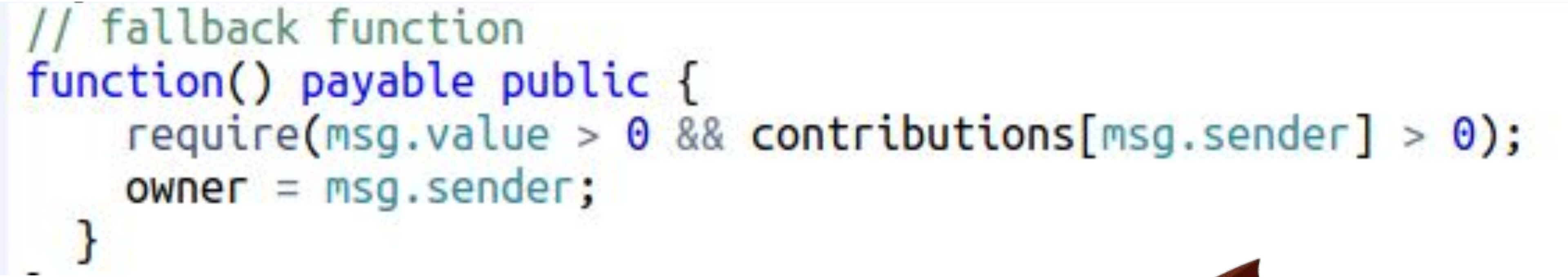
Only < 0.001 Eth

Challenge - 2



```
pragma solidity ^0.4.18;

import 'zeppelin-solidity/contracts/ownership/Ownable.sol';

contract DaveGame is Ownable{

    mapping(address => uint) public contributions;
    constructor() public {
        contributions[msg.sender] = 1000 * (1 ether);
    }
    //The person who has most contributions becomes the owner
    function contribute() public payable {
        require(msg.value < 0.001 ether);
        contributions[msg.sender] += msg.value;
        if(contributions[msg.sender] > contributions[owner]) {
            owner = msg.sender;
        }
    }
    //withdraw contracts balance
    function withdraw() public onlyOwner {
        owner.transfer(this.balance);
    }
    //fallback function
    function() payable public {
        require(msgs.value > 0 && contributions[msg.sender] > 0);
        owner = msg.sender;
    }
}
```

Source: Ethernaut



```
// fallback function
function() payable public {
    require(msg.value > 0 && contributions[msg.sender] > 0);
    owner = msg.sender;
  }
```

Two Conditions :

1) msg.value>0

2) senders contribution should be greater than zero

5. Denial of Service (DoS)

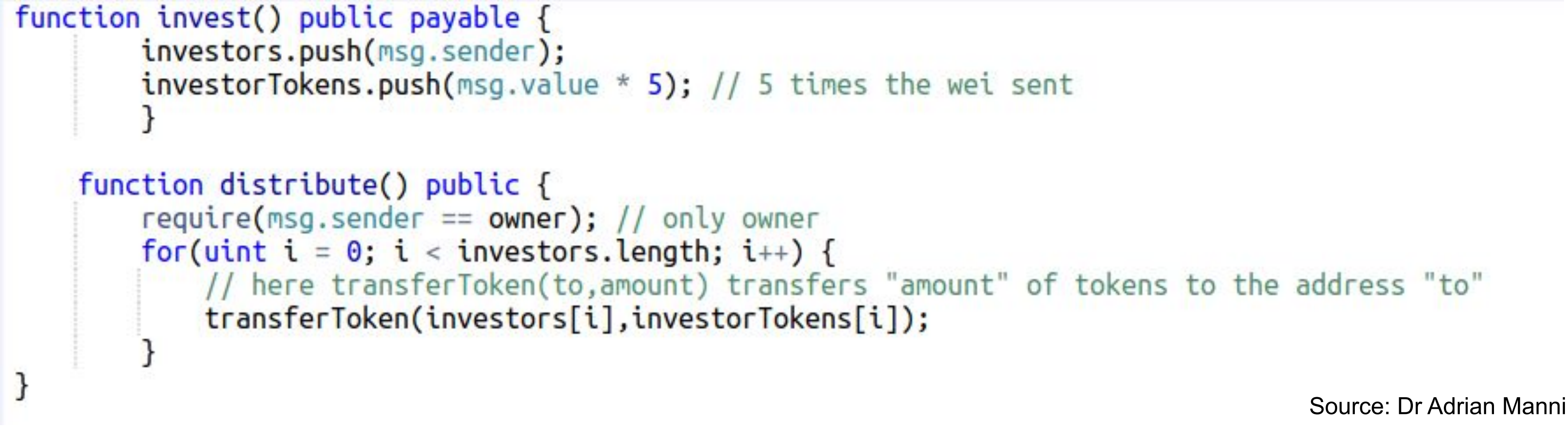
—

GovernMental

Vulnerability

Making the contract **inoperable** for some time or permanently

Attacker can **prevent other transactions** from being included by placing computationally intensive transactions with a high enough gas price



```
function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value * 5); // 5 times the wei sent
        }

    function distribute() public {
        require(msg.sender == owner); // only owner
        for(uint i = 0; i < investors.length; i++) {
            // here transferToken(to,amount) transfers "amount" of tokens to the address "to"
            transferToken(investors[i],investorTokens[i]);
        }
}
```

Source: Dr Adrian Manning

GovernMental Hack

Contract required the deletion of a large mapping in order to withdraw the ether. The deletion of this mapping had a **gas cost that exceeded** the block gas limit at the time, and thus was not possible to withdraw the 1100 ether. The ether was finally obtained with a transaction that used **2.5M gas**

Preventive Techniques

Avoid looping that can be **artificially manipulated** by external users

Favour Pull over Push Payments

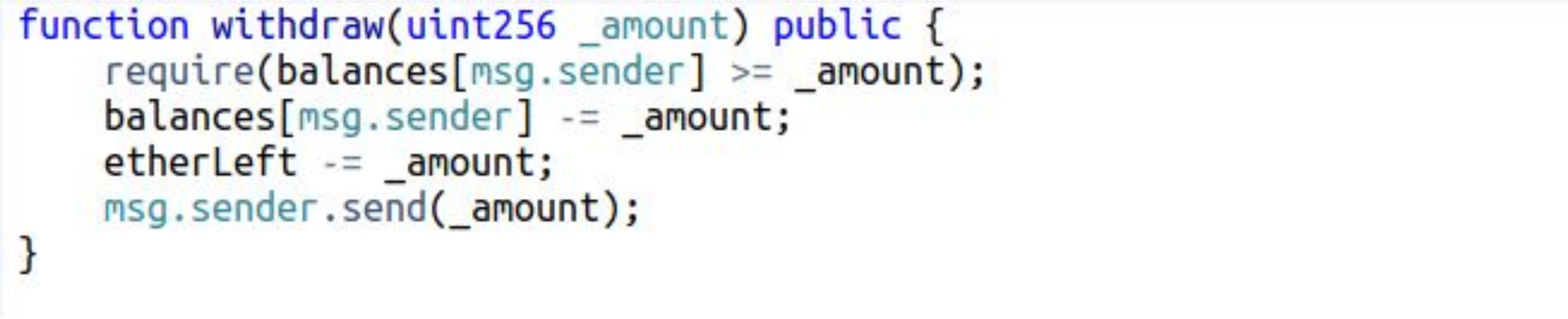
6. Unchecked CALL Return Values

Etherpot

Vulnerability

The state of the contract can have **inconsistencies** when the send() function fails and is used without checking the response

Doesn’t revert the state when send() fails



```
function withdraw(uint256 _amount) public {
    require(balances[msg.sender] >= _amount);
    balances[msg.sender] -= _amount;
    etherLeft -= _amount;
    msg.sender.send(_amount);
}
```

Etherpot contract

Smart contract lottery, send function was **unchecked**.

Could indicate the user has been sent funds even when the **send function fails**

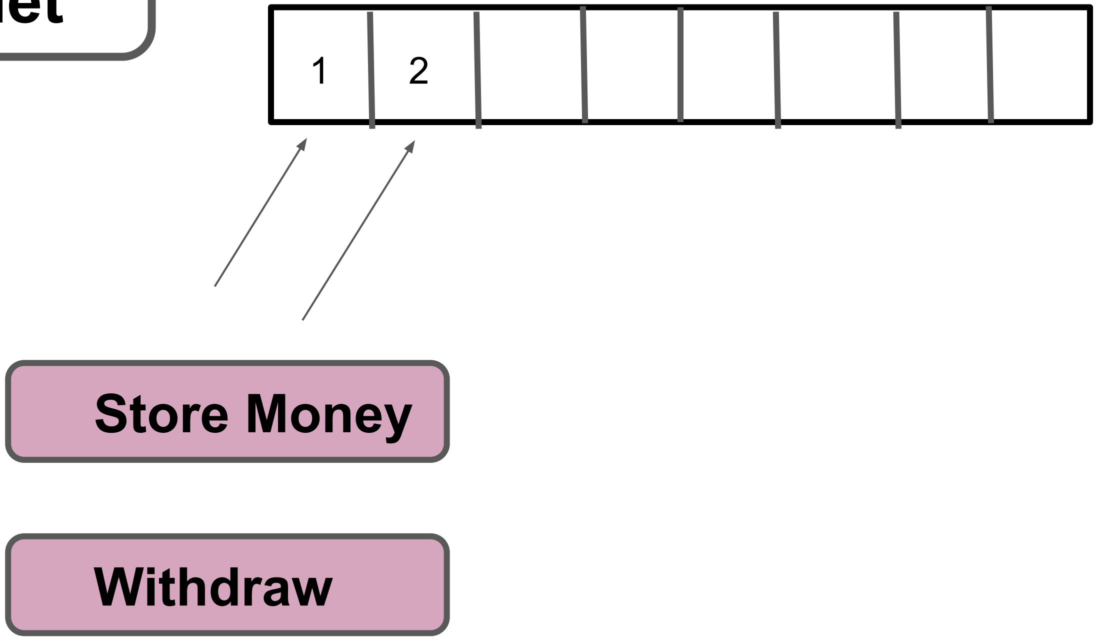
Primary downfall due to incorrect use of blockhashes

Preventive Techniques

Use transfer(), which reverts the state if the external transaction fails

Favour Pull over Push Payments

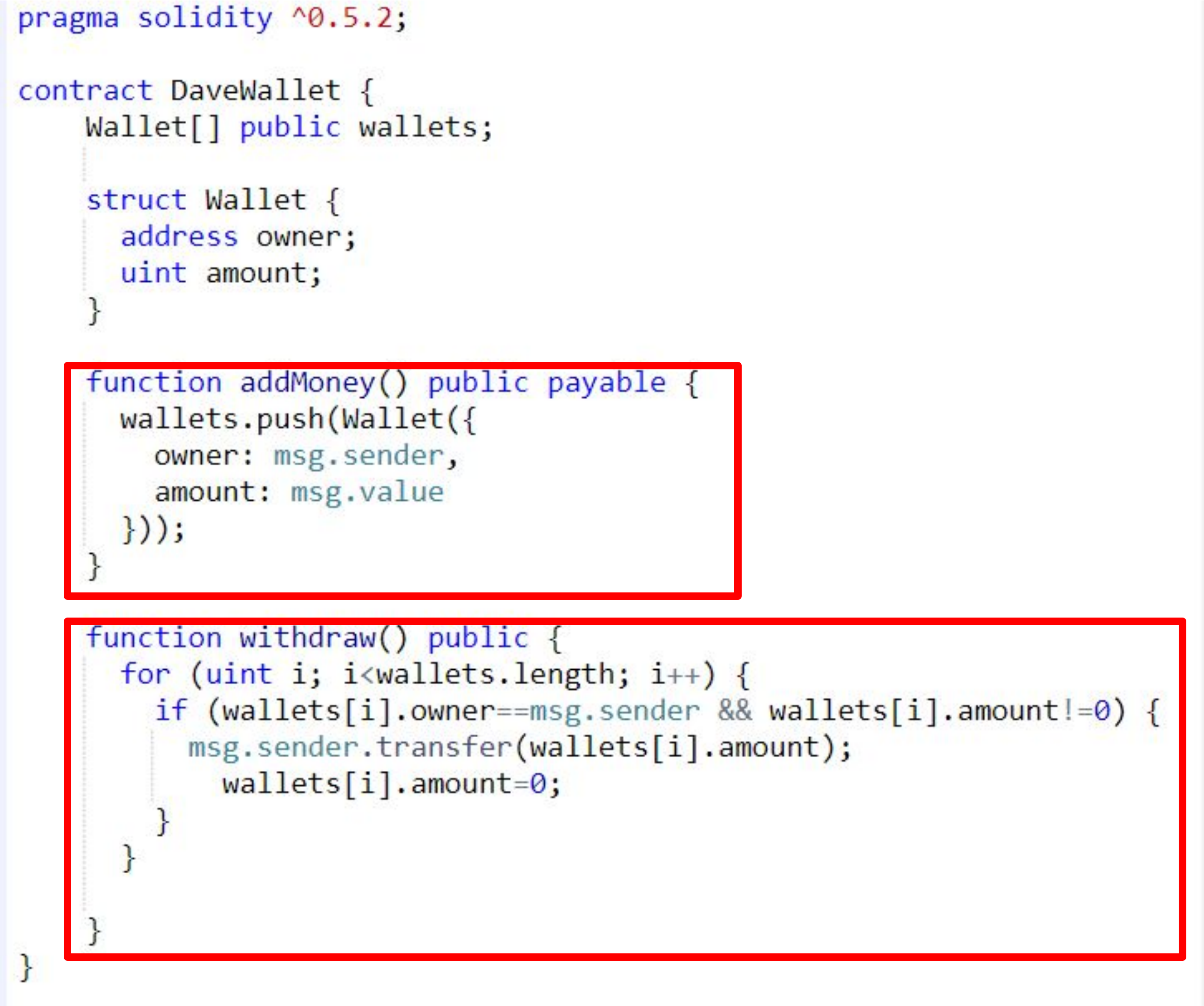
To create a **wallet**



1 2

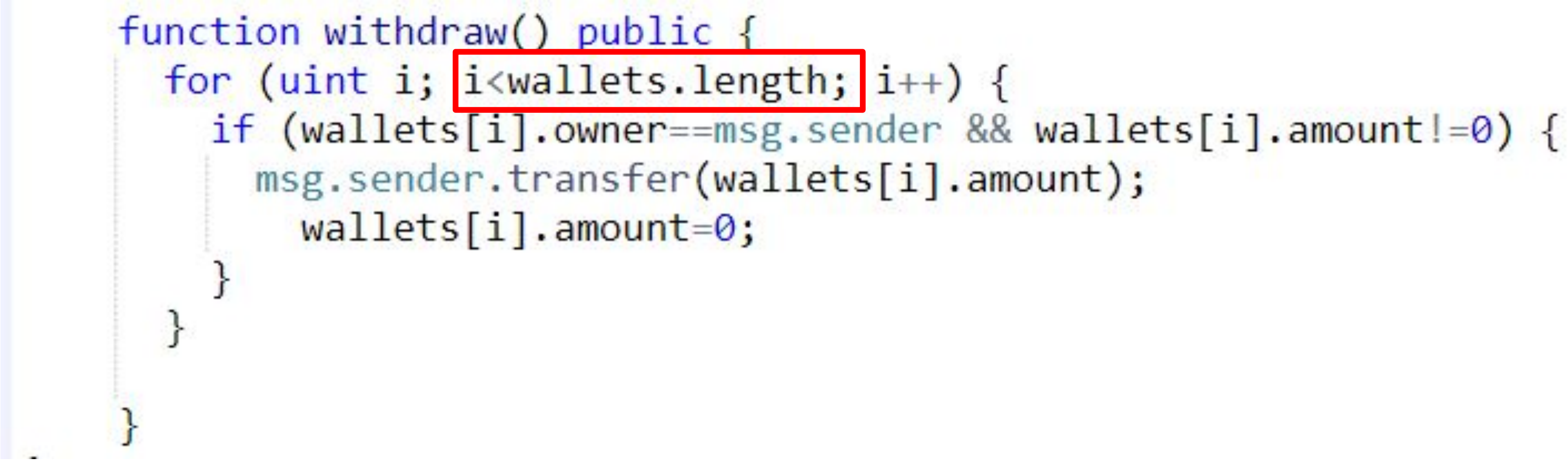
Store Money

Withdraw



```
pragma solidity ^0.5.2;

contract DaveWallet {
    Wallet[] public wallets;

    struct Wallet {
      address owner;
      uint amount;
    }

    function addMoney() public payable {
      wallets.push(Wallet({
        owner: msg.sender,
        amount: msg.value
      }));
    }

    function withdraw() public {
      for (uint i; i<wallets.length; i++) {
        if (wallets[i].owner==msg.sender && wallets[i].amount!=0) {
          msg.sender.transfer(wallets[i].amount);
            wallets[i].amount=0;
        }
      }

    }
}
```

Challenge 3



```
function withdraw() public {
  for (uint i; i<wallets.length; i++) {
    if (wallets[i].owner==msg.sender && wallets[i].amount!=0) {
      msg.sender.transfer(wallets[i].amount);
        wallets[i].amount=0;
    }
  }

}
```

The **length of the array** can be increased by **dummy transactions**

When the Block gas limit exceeds withdraw will not be possible

7. Time manipulation

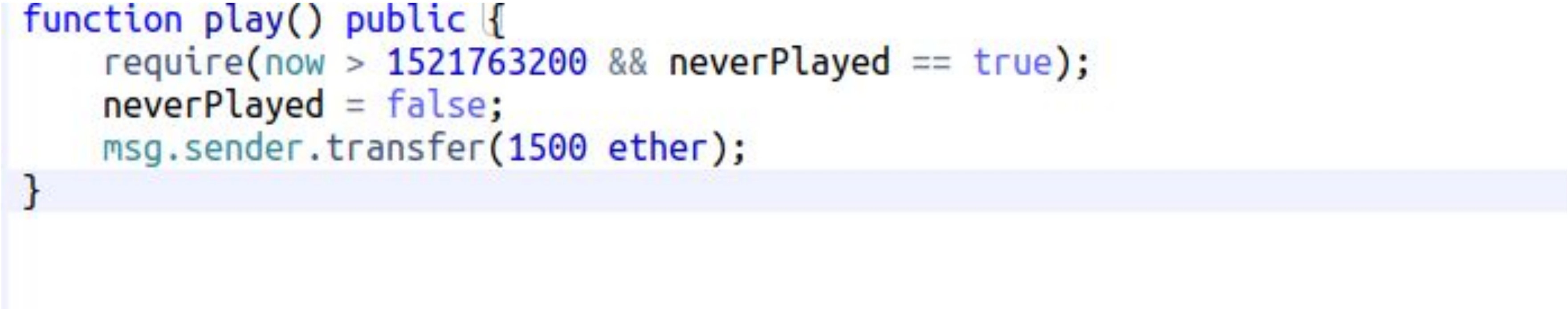
—

GovernMental

Vulnerability

block.timestamp or **now** can be manipulated by miners if they have some incentive to do so.

The timestamp **should not be** a base for the contract logic



```
function play() public {
    require(now > 1521763200 && neverPlayed == true);
    neverPlayed = false;
    msg.sender.transfer(1500 ether);
}
```

GovernMental

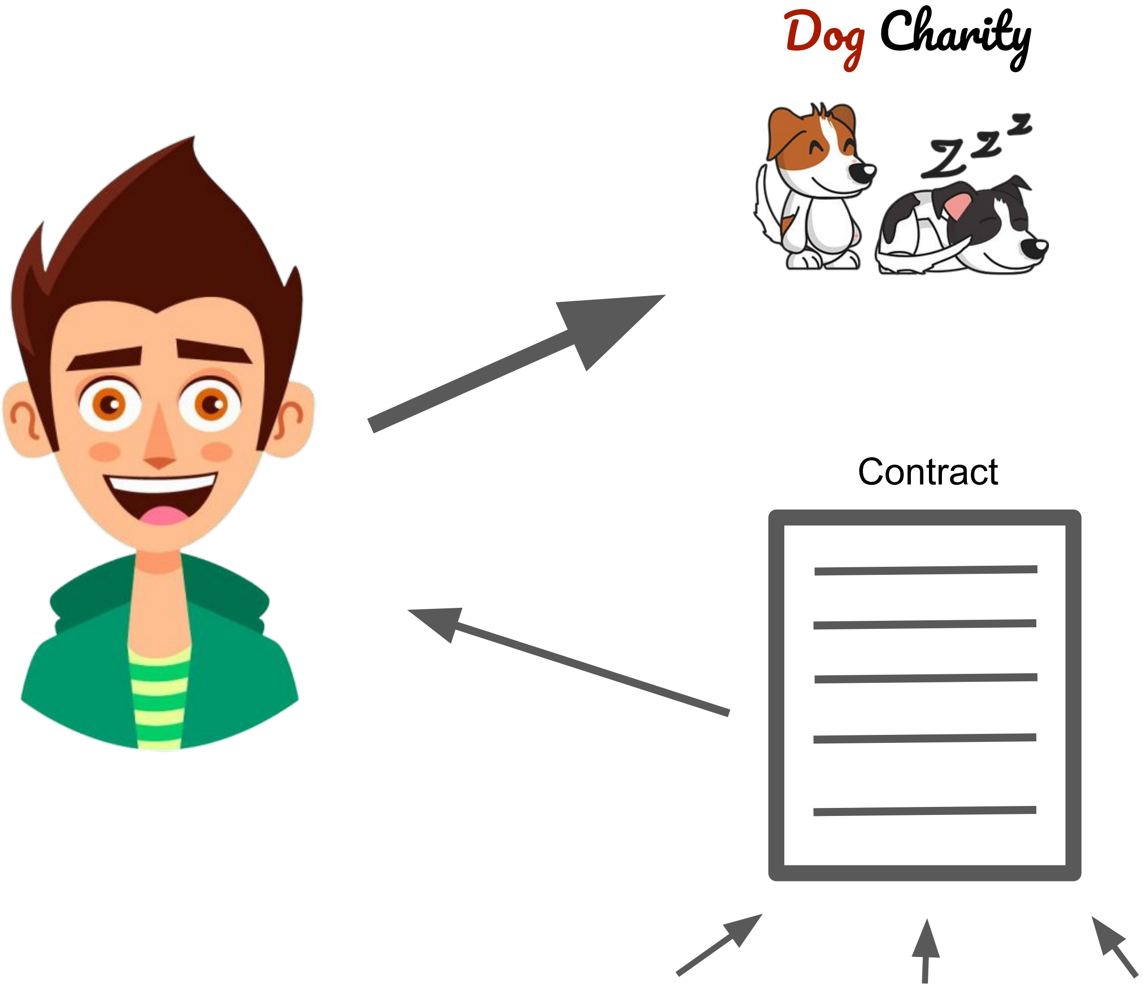
Also prone to the timestamp vulnerability

Contract **paid out** to player that joined last. **Miners** could manipulate the time slightly to break the game.

Preventive Techniques

Block.timestamp **should not be** used for generating random numbers

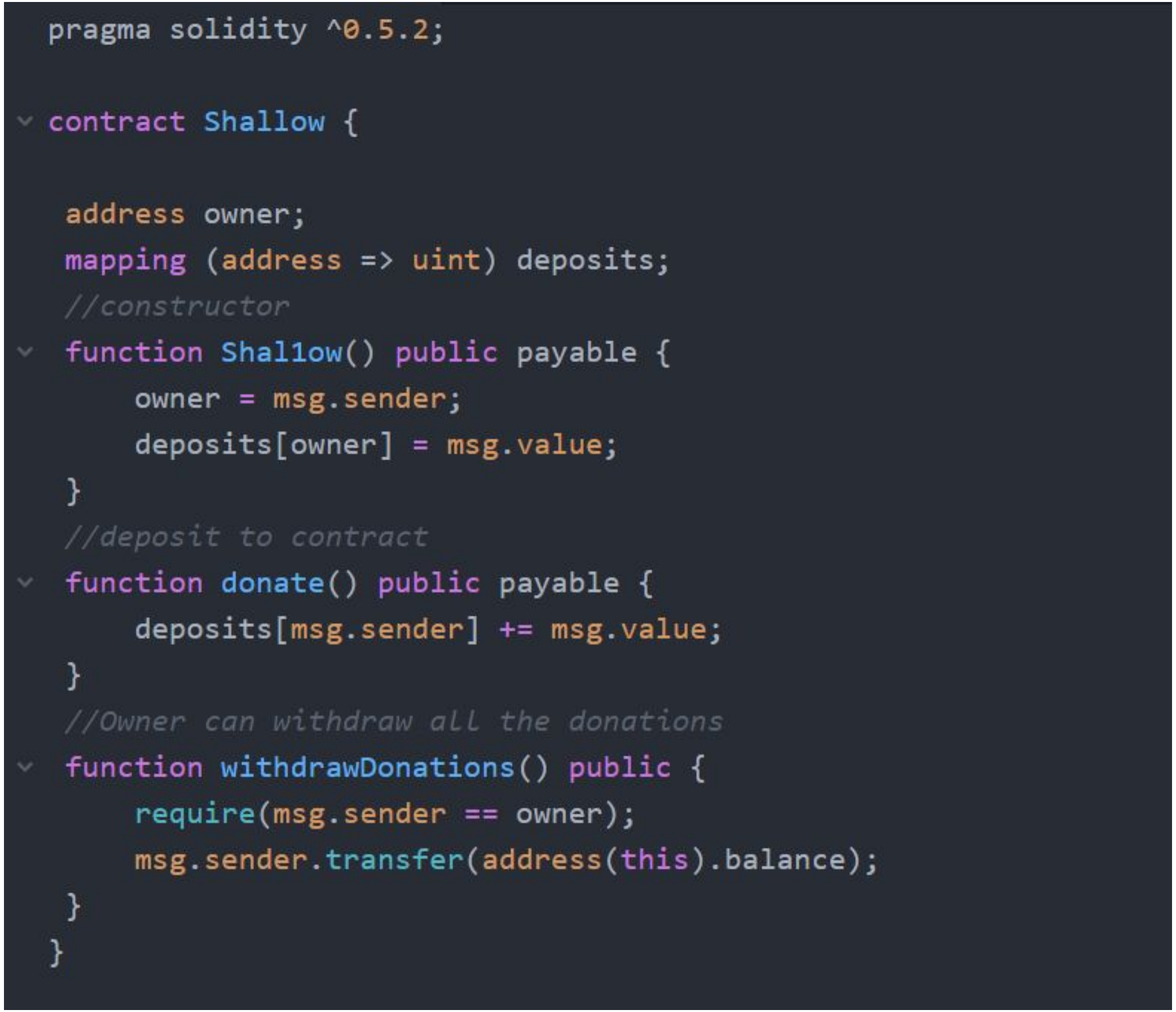
Block.number could be used instead for time-sensitive logic indirectly



Dog Charity

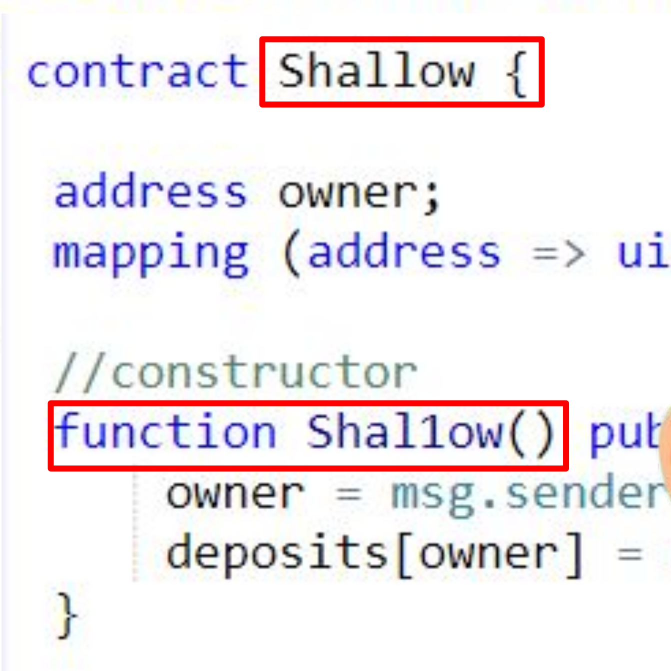
Contract

Challenge 4



```
pragma solidity ^0.5.2;

contract Shallow {

  address owner;
  mapping (address => uint) deposits;
  //constructor
  function Shal1ow() public payable {
      owner = msg.sender;
      deposits[owner] = msg.value;
  }
  //deposit to contract
  function donate() public payable {
      deposits[msg.sender] += msg.value;
  }
  //Owner can withdraw all the donations
  function withdrawDonations() public {
      require(msg.sender == owner);
      msg.sender.transfer(address(this).balance);
  }
}
```

©Souradeep Das



```
contract Shallow {

 address owner;
 mapping (address => ui

 //constructor
 function Shal1ow() pub
     owner = msg.sender
     deposits[owner] = msg
}
```

This actually happened!

Dynamic Pyramid changed its na somehow didn't rename the constructor

References

- OpenZeppelin
- ConsenSys
- Loom Network
- Dr Adrian Manning

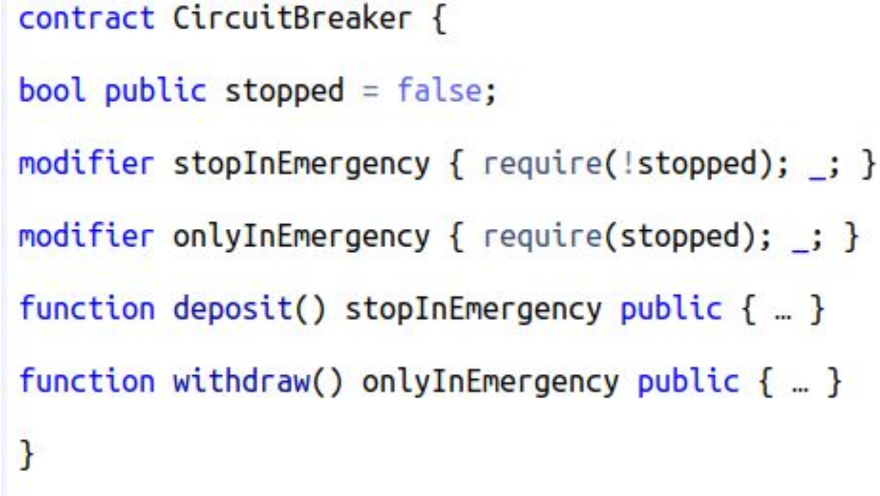
Best Practices and Design Patterns

> *“Every great design begins with an even better story”*
>
> *~Lorinda Mamo*

Circuit Breakers

Circuit Breakers are design patterns that allow contract functionality to be stopped.**Freezing the contract** would be beneficial for reducing harm before a fix can be implemented.

For example, if a bug has been found, you may stop users from depositing while allowing people to withdraw

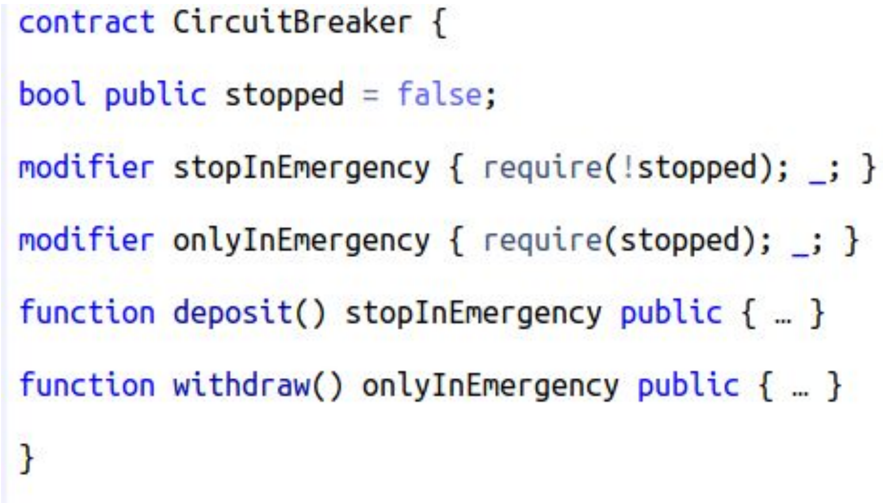


```
contract CircuitBreaker {

bool public stopped = false;

modifier stopInEmergency { require(!stopped); _; }

modifier onlyInEmergency { require(stopped); _; }

function deposit() stopInEmergency public { … }

function withdraw() onlyInEmergency public { … }

}
```

Speed Bump

Speed Bumps are useful when malicious events occur as it gives the owner time to act accordingly.

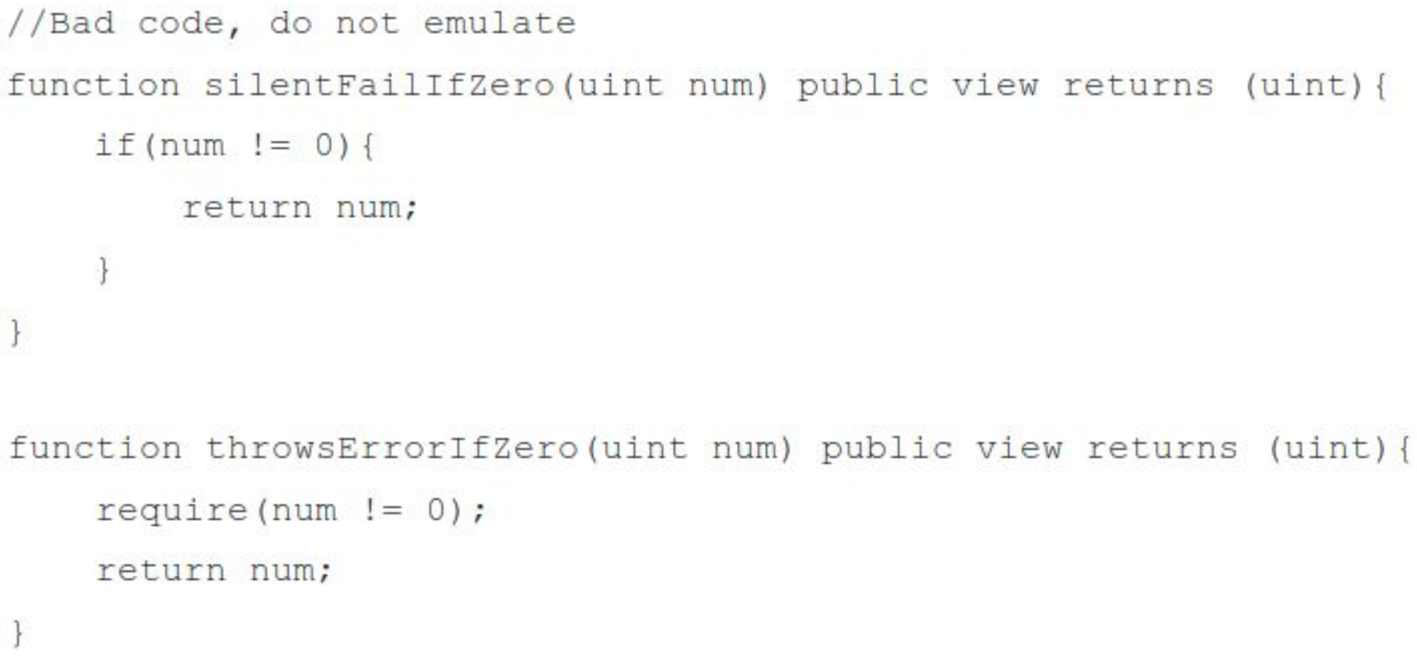
DAO had a speed bump, but no recovery options was present. Hence, speed bumps should be used with circuit breakers.



```
contract CircuitBreaker {

bool public stopped = false;

modifier stopInEmergency { require(!stopped); _; }

modifier onlyInEmergency { require(stopped); _; }

function deposit() stopInEmergency public { … }

function withdraw() onlyInEmergency public { … }

}
```

Fail Early Fail Loud

Check for errors in the beginning of the function



```
//Bad code, do not emulate
function silentFailIfZero(uint num) public view returns (uint){
    if(num != 0){
        return num;
    }
}

function throwsErrorIfZero(uint num) public view returns (uint){
    require(num != 0);
    return num;
}
```

require(), assert(), Or revert() ?

if(msg.sender != owner) { throw; }

Can be written as -

- if(msg.sender != owner) { revert(); }
- assert(msg.sender == owner);
- require(msg.sender == owner);

Difference between assert() and require()

assert() uses all of the gas sent with the transaction
require() return the gas if an error is encountered

Then, why should i use assert() ?

It should be considered a normal and healthy occurrence for a require() statement to fail. ✓

When an assert() statement fails, something very wrong and unexpected has happened, and you need to fix your code. ✗

revert()

1. Allows to return a value or an error message

```
revert(‘Something bad happened’);

require(condition, ‘Something bad happened’);
```

2. Refund the remaining gas to the caller

When a contract throws it uses up any remaining gas.
This can result in a very generous donation to miners, and often ends up costing users a lot of money.

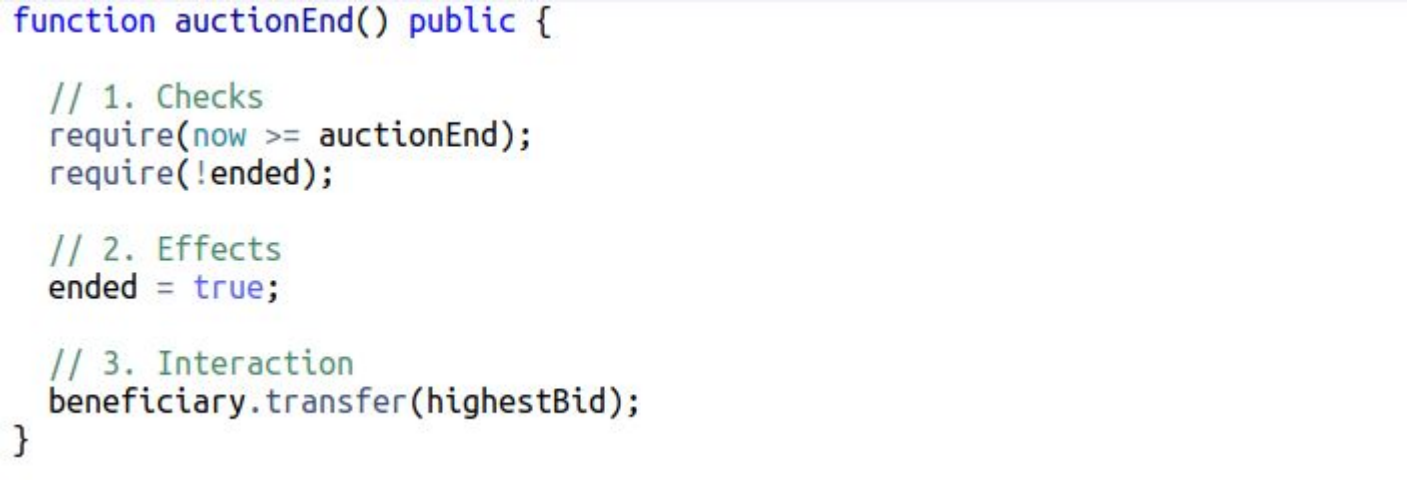
Checks-Effects-Interaction Pattern

1. Functions should start with checks in the beginning (if any)

    require(), assert(), revert()

2. Changes to state variables or In-contract execution

3. Interaction/ Calling functions of other contracts



```
function auctionEnd() public {

  // 1. Checks
  require(now >= auctionEnd);
  require(!ended);

  // 2. Effects
  ended = true;

  // 3. Interaction
  beneficiary.transfer(highestBid);
}
```

Auditing Tools

“Be sure you put your feet in the right place, then stand firm”

~Abraham Lincoln

SmartCheck

Online Static Code analyzer

Solgraph

Generates a DOT graph that visualizes function control flow of a Solidity contract and highlights potential security vulnerabilities

Mythrill

Reversing and bug hunting framework for Ethereum

Oyente

Static analysis tool for finding common vulnerabilities

Surya

Visual outputs and information on contract structure

Securify

Online Audit tool for static analysis

OpenZeppelin Libraries

OpenZeppelin is a framework of re-usable smart contracts for Ethereum
Tested, secure smart contract libraries, reduces the risk of vulnerabilities
Includes libraries for ERC-20, ERC-721, SafeMath etc

Ethereum Package Manager (EthPM)

EthPM is essentially npm for Ethereum contracts
Several secure smart contract packages

Better to use pre-written verified and secure code.

Zeppelin Solutions

Vyper- The secure smart contract language

“Every sunset brings the promise of a new dawn”

~Ralph Waldo Emerson

Vyper is a Python 3 derived programming language for Ethereum Smart contracts, and an alternative to Solidity.

Principles

Security: It should be possible and natural to build secure smart-contracts in Vyper.

Language and compiler simplicity: The language and the compiler implementation should strive to be simple.

Auditability: Vyper code should be maximally human-readable.Simplicity for the reader is more important than simplicity for the writer.

Goals

Bounds and overflow checking

Support for signed integers and decimal fixed point numbers

Strong typing: support for units (e.g. timestamp, timedelta, seconds, wei, wei per second, meters per second squared).

Small and understandable compiler code

Vyper was not created to replace Solidity, it was created for having a secure smart contracting solution.

There are certain things which Vyper cannot do, that Solidity can!

Vyper vs Solidity

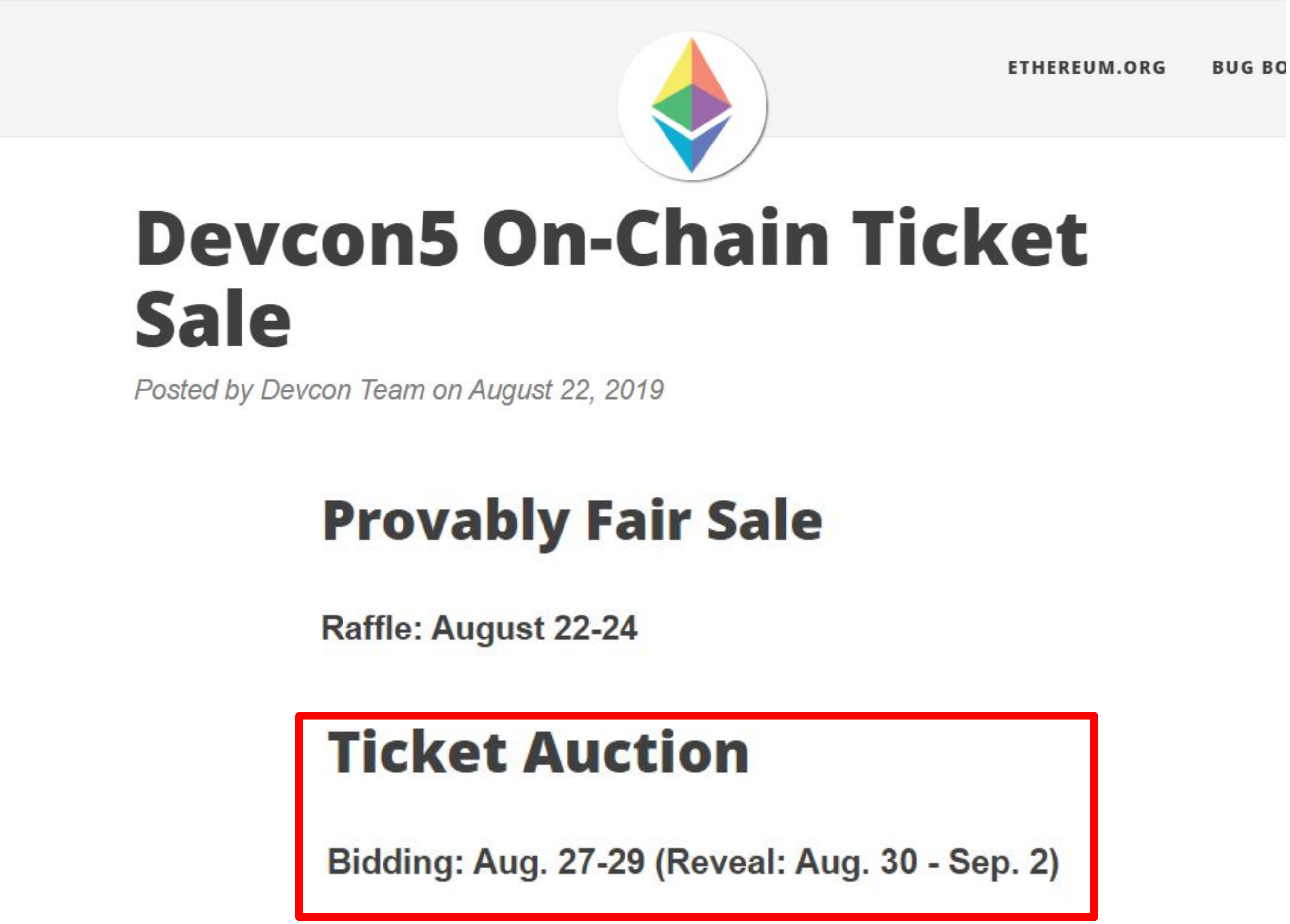
Vyper doesn't have-

- Modifiers
- Inheritance
- Inline assembly
- Function overloading
- Recursive calling
- Infinite-length types

Get started with Vyper at **https://vyper.online/**

Show time

Lets learn to write a smart contract in Vyper!



Devcon5 On-Chain Ticket Sale

Posted by Devcon Team on August 22, 2019

Provably Fair Sale

Raffle: August 22-24

Ticket Auction

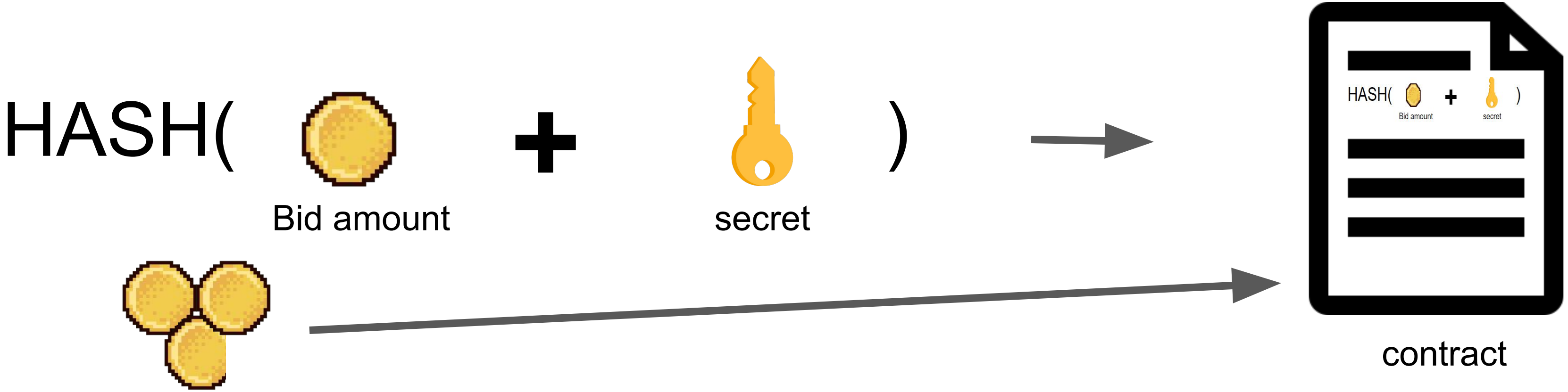
Bidding: Aug. 27-29 (Reveal: Aug. 30 - Sep. 2)

- **50 Tickets available for auction**

- **Your Bid is secret until everyone finishes bidding**

- **Top 50 bids win tickets**

1st Phase - Bid



HASH(+)

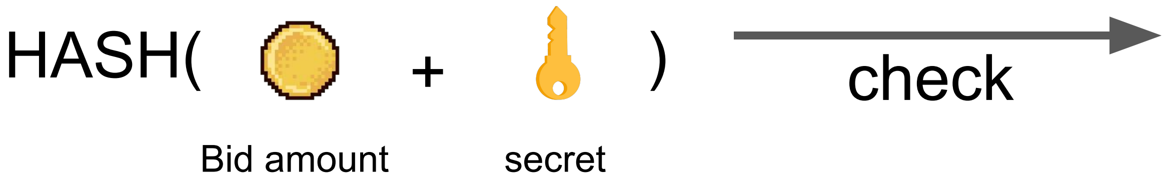
Bid amount

secret

contract

2nd Phase - Reveal

After Bidding is over -->

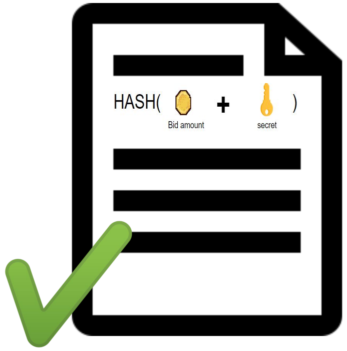


HASH(+)

Bid amount

secret

check



3rd Phase - Withdraw/ Refund

Excess money for Masking, refunded

Winners :

Refund extra money after deducting bid amount

Non-Winners :

Refund the whole amount

Thanks!

Contact Me:

souradeep.tech

dsouradeep2@gmail.com

in : souradeep-das

: souradeep-das

: thedeepdas

YOU FOUND A STAR

Speaker Track

dropparty.tech