Blockchain Technology and Applications

CS 731

Introduction to Smart Contracts and Vulnerabilities in Smart Contracts

IIT Kanpur

Chavan Sujeet Yashavant (sujeetc@cse.iitk.ac.in)

Part 1: Introduction to Smart Contracts

Blockchain beyond Finance

- Many people argued in 2013 that blockchains can be used for Healthcare, Supply Chain Management, Voting, Identity System, Prediction Markets, Crowd Funding and many more
- But How?

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Why Ethereum?

Bitcoin Blockchain

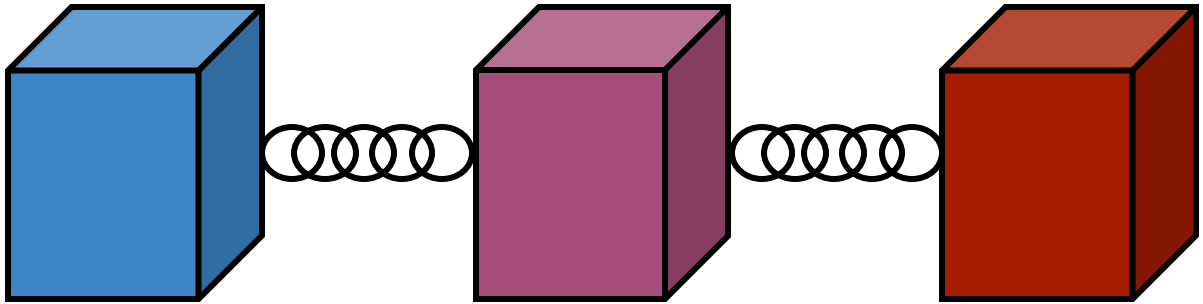
Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Why Ethereum?

Need something like this

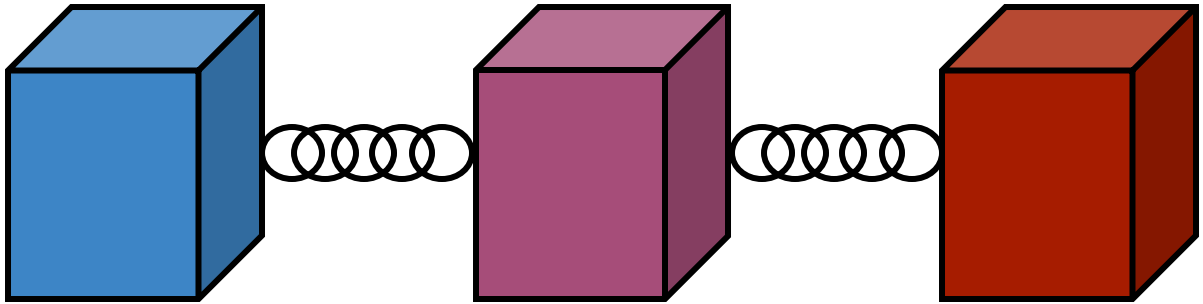
Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

What is Ethereum?



Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

What is Ethereum?



It’s a Blockchain ……. Boring Answer

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

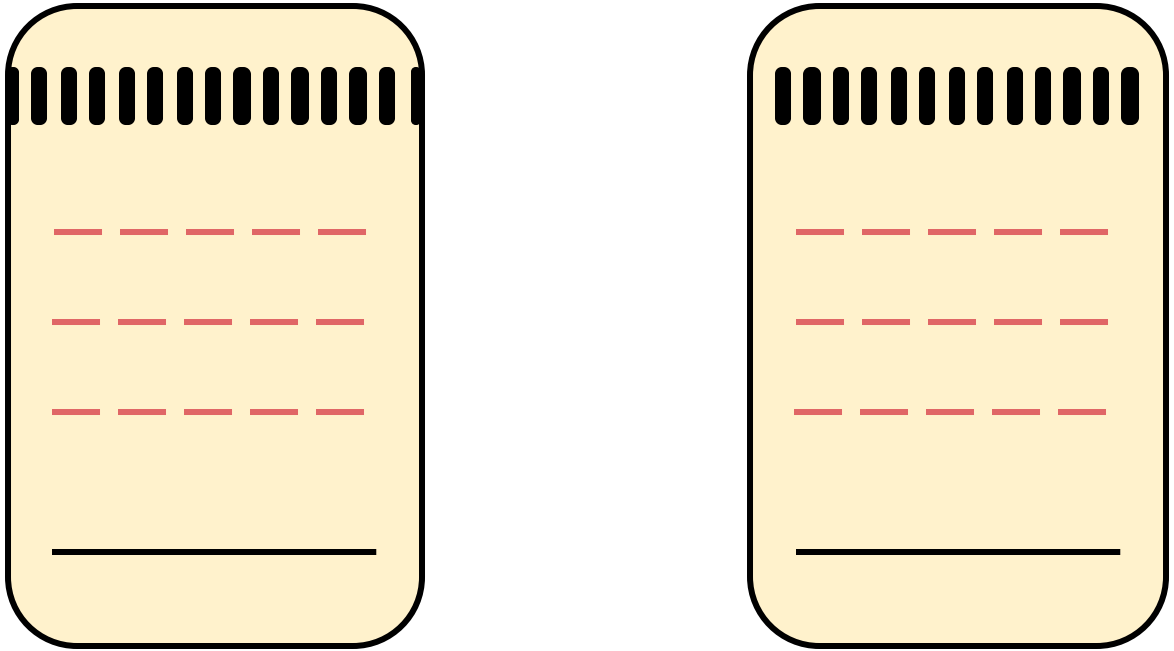
What is Ethereum?

It's a Blockchain, with following additions

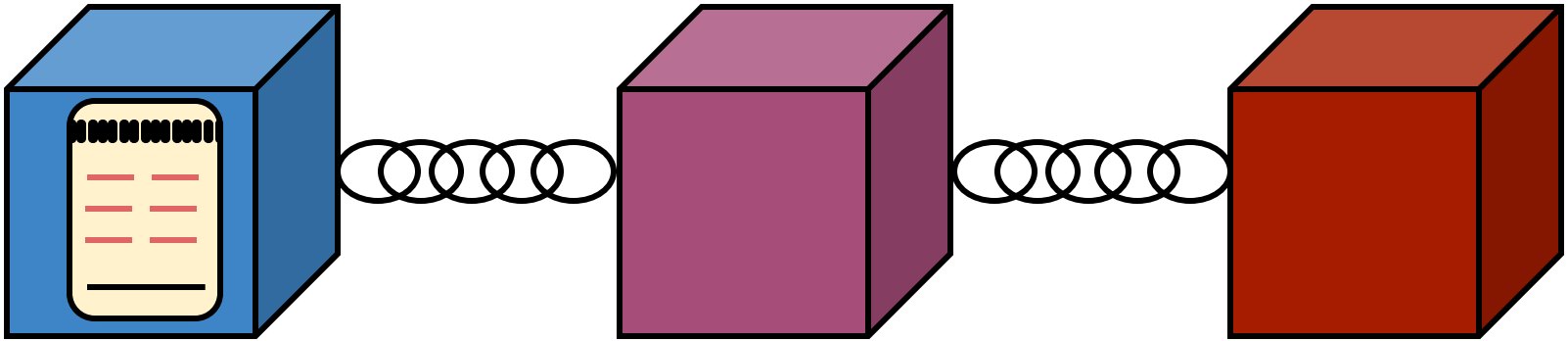
- A built-in programming Language
- Two types of accounts
  - User Accounts (Controlled by Private Keys)
  - Contract Accounts (Controlled by Code)
- Anyone can create an app by defining it as a Contract

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Smart Contracts



Smart contracts



- Tiny computer programs
- Stored inside a blockchain

Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Smart Contract

- A code that resides on blockchain
- Executes when certain predetermined conditions are satisfied

Smart Contract

- agreement between mutually distrusting participants
- automatically enforced by the consensus mechanism of the blockchain
- without relying on a trusted authority.

Ref: Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on ethereum smart contracts (sok)." *International conference on principles of security and trust*. Springer, Berlin, Heidelberg, 2017.

What a Contract can Do?

- Send ETH to other contracts

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

What a Contract can Do?

- Send ETH to other contracts
- Read/write Storage

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

What a Contract can Do?

- Send ETH to other contracts
- Read/write Storage
- Call (i.e. start execution in) other Contracts

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Smart Contract Execution

- Every node on Ethereum network processes every transaction

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Smart Contract Execution

```
1 pragma solidity ^0.4.17;
2 contract Inbox
3 {
4         string public message;
5         function Inbox(string initialMessage) public {
6                 message = initialMessage;
7         }
8         function SetMessage (string newMessage) public {
9                 message = newMessage;
10        }
11 }
```

Solidity Compiler

bytecode:

'6060604052341561000f57600080fd5b60405161038e38038061038e83398101604052808051909101905060008180516100 3d9291602001906100445 65b50506100df565b828054600181600116156101000203166002900480601f0160208091040260200160405190810160405280929190818152602001828054600181600116156101000203166002900480156101d15780601f106101a57610100808354040283529160200191610 1d1565b820191906000526020600020905b81548152906001019060200180831161 01b45782900360 1f 68201915b505050505081565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f1061021a5780516 0ff1916838001178555610247565b828001600101855582156102475791820 15b828111156102475782518255916020019190600101906 1022c565b506102539291506102575 65b5090565b6102719190 5b808211156102535760008155600101610 25d565b905600a165627a7a72305820 8f0f3e57d60457b58d09389d0da7d2d5375adecd20cfdb52a65ef21ee981d1440029',

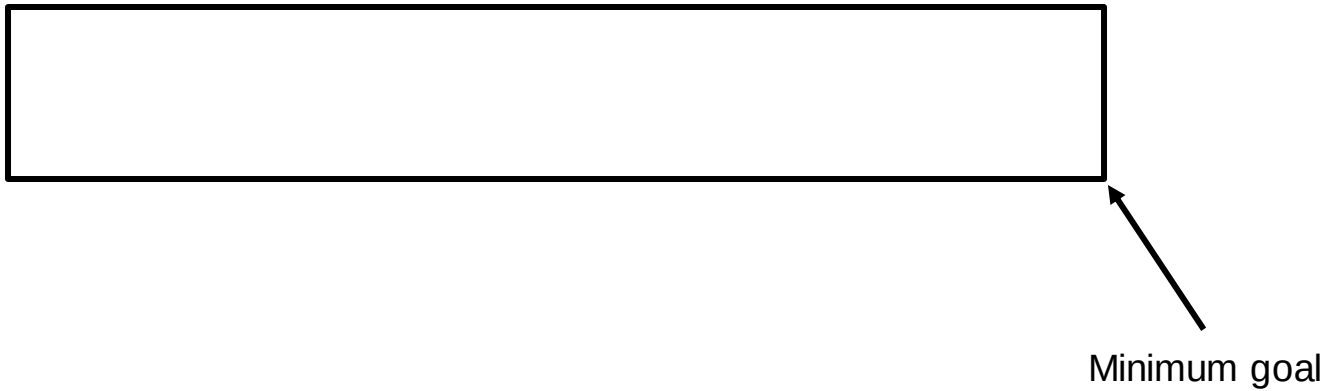
Smart Contract Execution

EVM
(Ethereum
Virtual
Machine)

Ethereum Virtual Machine (EVM)

- Global Singleton Computing Machine with a shared ledger of data

Crowdfunding platform



Ref: https://savjee.be/videos/simply-explained/smart-contracts/

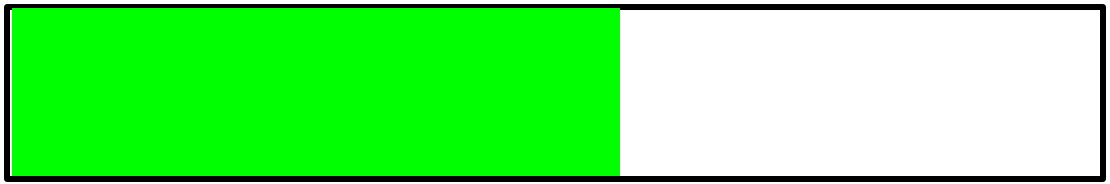
Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Crowdfunding platform

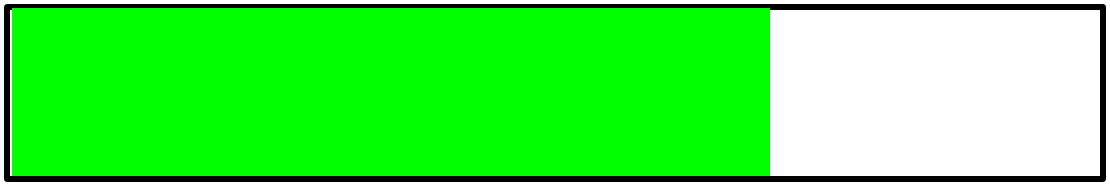
Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Crowdfunding platform



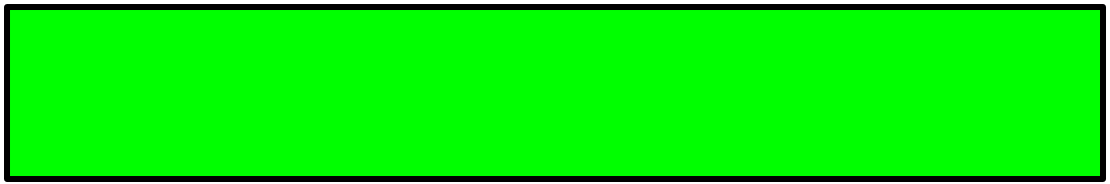
Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Crowdfunding platform



Ref: https://savjee.be/videos/simply-explained/smart-contracts/

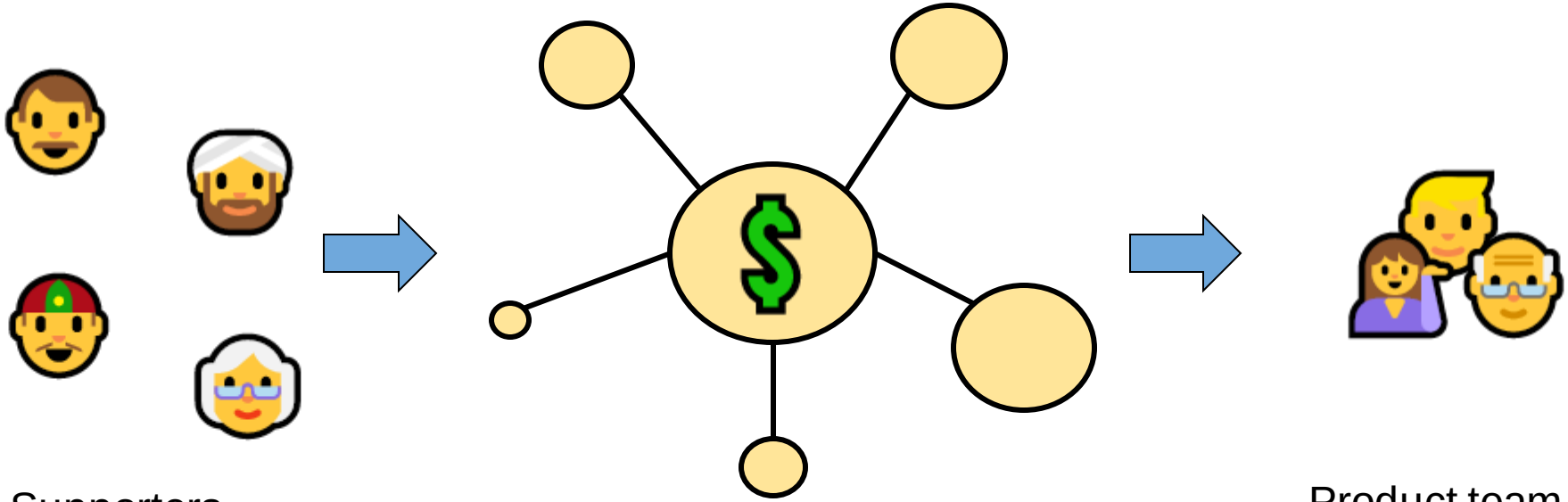
Crowdfunding platform



Funded!

Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Kickstarter for Crowdfunding platform



Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Kickstarter for Crowdfunding platform

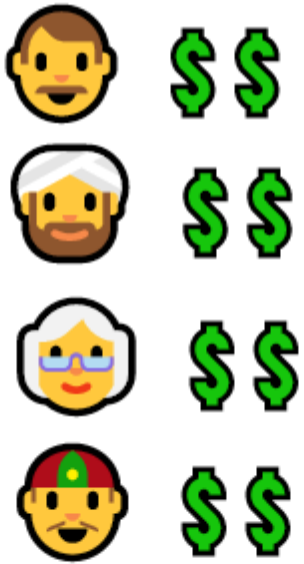
Trusting a third-party is required

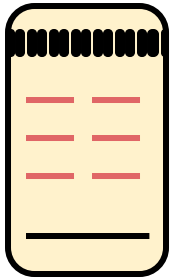
Ref: https://savjee.be/videos/simply-explained/smart-contracts, https://pngimg.com

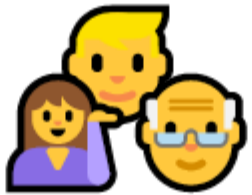
Smart contracts

We can build a similar system with a Smart Contracts without the requirement of any third party

Kickstarter with Smart Contract





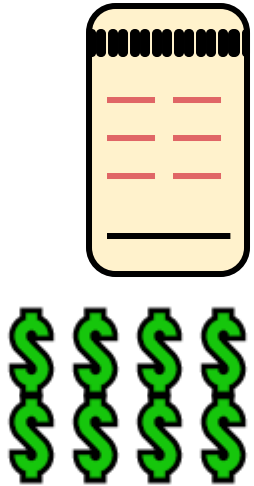


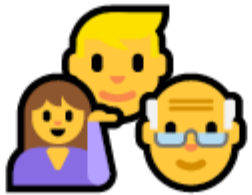
Supporters

Product team

Kickstarter with Smart Contract

Supporters



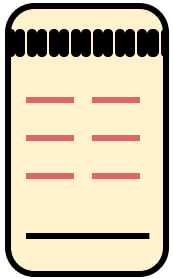


Product team

Kickstarter with Smart Contract

Supporters

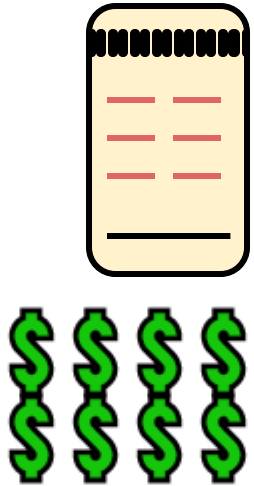
Funded!

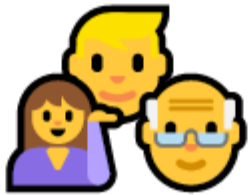


Product team

Kickstarter with Smart Contract

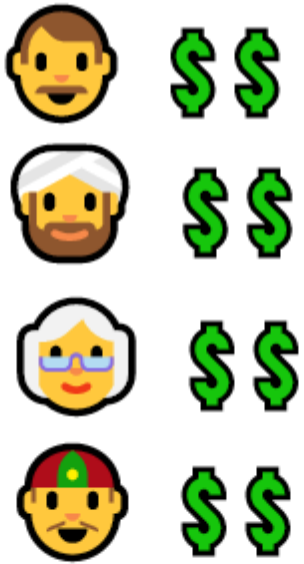
Supporters





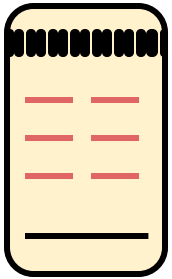
Product team

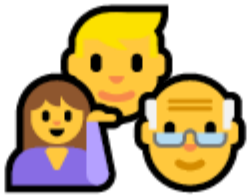
Kickstarter with Smart Contract



Supporters

Failed!





Product team

Introduction to Solidity

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

Version Pragma

```
1 pragma solidity ^0.4.17;
```

- Instructions to the compiler on how to treat the code.
- All solidity source code should start with a “version pragma” which is a declaration of the version of the solidity compiler this code should use.
- This helps the code from being incompatible with the future versions of the compiler which may bring changes.

Ref: https://www.geeksforgeeks.org

Contract keyword

```
1 pragma solidity ^0.4.17;
2 contract Inbox {
```

- It declares a contract under which the code is encapsulated.
- Contract is similar to classes in OOP
- Contains
  - state variables
  - Functions

Ref: https://www.geeksforgeeks.org

Address in Ethereum

- Externally Owned Address (EOA)
  - public account that holds the funds
  - accessible by private key pairs
- Contract Address
  - address hosting a collection of code

Types in Solidity

Boolean	bool	**True, False**
Integer	int/uint/uint8 to uint256 in steps of 8	`uint32` → `0` up to $2^{32}-1$ `int, uint` → `int256, uint256`
Address	address	Holds a 20 byte value (size of an Ethereum address)
String	Array of characters	string **public** str = "GeeksforGeeks";
Arrays	group of variables of the same data type	**uint[5] public array = [1, 2, 3, 4, 5] ;**

Ref: docs.soliditylang.org/en/latest/types.html

Types in Solidity

Struct	grouping together related data	`struct Todo { string text; bool completed; } // An array of 'Todo' structs Todo[] public todos;`
Mapping	• Dictionary • Key-value pair	// Mapping from address to uint **mapping**(**address** => **uint**) **public** myMap;
….	…….	……..

Ref: www.geeksforgeeks.org, solidity-by-example.org

State Variables

```
1 pragma solidity ^0.4.17;
2 contract Inbox {
3     string public message;
4     function Inbox(string initialMessage) public {
5         message = initialMessage;
6     }
```

- Permanently stored in contract storage → written to Ethereum Blockchain
- Declared inside a contract and outside of function
- Adding a slot to a Database

Ref: https://www.geeksforgeeks.org

State Variable Visibility

Public	can be accessed by any contract
Private	can be only accessed by the contract in which the variable is defined
Internal	can be accessed by contract in which the variable is defined or by its inherited contracts

Ref: https://docs.soliditylang.org/en/latest/contracts.html#visibility-and-getters

Function declaration

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

Constructor

Ref: https://www.geeksforgeeks.org

Constructor

- It invokes only once when the contract is deployed
- used to initialize the contract state
- optional to create a constructor
- Version < 0.4.22, constructors → the same name as the contract
- Version > 0.4.22 contractors → constructor() keyword

Ref: https://www.geeksforgeeks.org

Constructor


```
1 pragma solidity ^0.4.17;
2 contract Inbox {
3     string public message;
4     function Inbox(string initialMessage) public {
5         message = initialMessage;
6     }
```

- Initializes message variable with input passed while contract creation

Ref: https://www.geeksforgeeks.org

Other Functions

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

Ref: https://www.geeksforgeeks.org

Function Visibility

Function Visibility

Public	can be called by any contract
Private	can be only called by the contract in which the function resides
Internal	can be called contract in which the function is present or by its inherited contracts
External	can be called by external contracts only

Ref: www.geeksforgeeks.org

View and Pure functions

View	Read-only function
Pure	Does not read or modify the state variables

Ref: www.geeksforgeeks.org

msg Global Variables in Ethereum

- Special global variables
- Always exists globally

msg.sender	address where the current function call came from
msg.value	the amount of wei (money) sent
……..	……

Ref: https://medium.com/upstate-interactive/what-you-need-to-know-about-msg-global-variables-in-solidity-566f1e83cc69

Function Modifiers

- change the behavior of functions in a declarative way
- automatically check a condition prior to executing the function.
- The placeholder statement (_) → where the body of the function should be inserted.

Ref: docs.soliditylang.org

Function Modifiers

```
1  pragma solidity >=0.4.22 <0.9.0;
2
3  contract Purchase {
4      address public seller;
5
6      modifier onlySeller() { // Modifier
7          require(
8              msg.sender == seller,
9              "Only seller can call this."
10         );
11         _;
12     }
13
14     function abort() public view onlySeller { // Modifier usage
15         // ...
16     }
17 }
```

Ref: docs.soliditylang.org

Events

- events are a way to log and notify external entities
- emitting and recording data onto the blockchain
- similar to logs or records
- When an event is emitted it generates an event log that is stored on the blockchain.

Ref: medium.com/coinmonks

Events

```
pragma solidity >=0.4.21 <0.9.0;

contract ClientReceipt {
    event Deposit(
        address indexed from,
        bytes32 indexed id,
        uint value
    );

    function deposit(bytes32 id) public payable {
        // Events are emitted using `emit`, followed by
        // the name of the event and the arguments
        // (if any) in parentheses. Any such invocation
        // (even deeply nested) can be detected from
        // the JavaScript API by filtering for `Deposit`.
        emit Deposit(msg.sender, id, msg.value);
    }
}
```

Ref: docs.soliditylang.org

Retrieving Events

```
const contract = new web3.eth.Contract(abi, contractAddress);
contract.getPastEvents('NewTransaction', {
  filter: { sender: '0x123abc' }, // Optional event filtering
  fromBlock: 0, // Start block number
  toBlock: 'latest' // End block number
})
.then(function(events) {
  // Process the retrieved events
  console.log(events);
})
.catch(function(error) {
  // Handle errors
  console.error(error);
});
```

Event Name

Ref: docs.soliditylang.org

Contract ABI

- The ABI (Application Binary Interface): JSON file that describes the
  - interface of the smart contract
    - functions that it exposes (with parameters)
    - Events from the Smart Contracts

Ref: https://blockchain.oodles.io/dev-blog/How-to-Use-a-Web3-JS-Call-Contract-Function.

Code Execution on Real Blockchain (Try this after success on Local Blockchain)

- Testnet:
  - Can use **Remix** and **Metamask**
  - Can use **hardhat** to deploy on **Goerli Testnet**
- Mainnet
  - Require real money
  - Do not try unless you become expert

Ref: https://www.geeksforgeeks.org

Code Execution on Local Blockchain

- Offline (Blockchain inside local machine): It takes time to setup.
  - Can use **Remix** and **Ganache**
- Online (Blockchain inside browser): Remix IDE
  - Simple one, first try this

Ref: https://www.geeksforgeeks.org

“Hello World” Smart Contract in Remix-IDE

```
1   pragma solidity ^0.4.17;
2
3   contract HelloWorld
4   {
5       function get() public pure returns (string memory)
6       {
7           return 'Hello Contracts';
8       }
9   }
```

Let’s see a Demo

Ref: https://www.geeksforgeeks.org

“Inbox” Smart Contract in Remix-IDE

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

Let’s see a Demo

Ref: www.geeksforgeeks.org

Crowdfunding Smart Contract

```
pragma solidity ^0.4.19;
contract Crowdfunding {
    address owner;
    uint256 deadline;
    uint256 goal;
    mapping(address => uint256) public pledgeOf;
    function Crowdfunding(uint256 numberOfDays, uint256 _goal) public {
        owner = msg.sender;
        deadline = now + (numberOfDays * 1 days);
        goal = _goal;
    }
    function pledge(uint256 amount) public payable {
        require(now < deadline);                // in the fundraising period
        require(msg.value == amount);
        pledgeOf[msg.sender] += amount;
    }
    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(now >= deadline);               // in the withdrawal period
        require(msg.sender == owner);

        msg.sender.transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal);  // funding goal not met
        require(now >= deadline);               // in the withdrawal period
        uint256 amount = pledgeOf[msg.sender];
        pledgeOf[msg.sender] = 0;
        msg.sender.transfer(amount);
    }
}
```

Currency Example

```solidity
pragma solidity ^0.8.4;
contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping(address => uint) public balances;
    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);
    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }
    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }
    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);
    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        if (amount > balances[msg.sender])
            revert InsufficientBalance({
                requested: amount,
                available: balances[msg.sender]
            });
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Let’s see a Demo

Part 2: Vulnerabilities in Smart Contracts

Smart Contract Security

- Correctness is ensured by the consensus mechanism
- Unfortunately, correctness is not sufficient to make Smart Contracts secure.

Ref: Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on ethereum smart contracts (sok)." *International conference on principles of security and trust*. Springer, Berlin, Heidelberg, 2017.

Classification of Blockchain based Attacks

- Malicious Acts
- Weak Protocol
- Defraud
- Application Bugs

Ref: Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Classification of Blockchain based Attacks

- Malicious Acts
- Weak Protocol
- Defraud
- Application Bugs

Ref: Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Application Bugs

- Programs are correct but may have loopholes
- People can take advantage of it by exploiting loophole
- Example: buffer overflow (in C, something like you can read/write outside an array)

Ref: Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Application Bugs

- Blockchains are new
- Smart contract developers may write buggy code

Application Bugs

- Loophole in the smart Contract Code
- The DAO Attack ($60 M Loss)
- Attacker can steal money, influences an application to function differently

Ref: Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Application Bugs

- Reentrancy
- Overflow, Underflow
- Default Visibilities
- Timestamp Dependence
- Transaction Ordering Dependence

Ref: Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

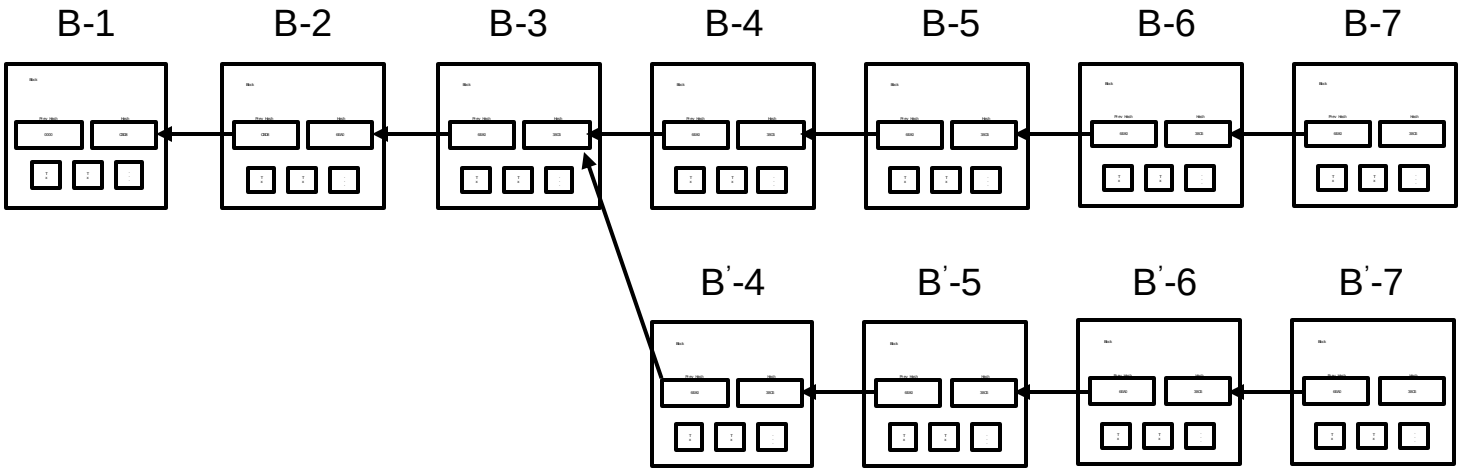
The DAO Hack on Ethereum

- DAO was crowdfunding platform
- It raised $150 Million
- Got hacked due to bug in Smart Contract (Reentrancy) and lost $60 Million
- Ethereum blockchain was hard forked to restore stolen funds

Hard Fork Example

- Demonetization (Govt banned 500 and 1000 Rupees notes)
- Let's say that one group don’t agree and accept old notes
- Normal people agreed to Govt decision
- Two currencies in existence

Hard Fork



Ethereum Classic

Ethereum

Smart Contract Vuln

```
1  contract Puzzle {
2    uint256 amount = 0.5; //0.5 ethers
3    function submit_answer (string answer) {
4      //Logic to check submitted answer
5      if (/*answer is correct*/)
6        _sendEther();
7    }
8    function _sendEther(){
9      msg.sender.tranfer(amount);
10   }
11 }
```

Ref: Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Smart Contract Vuln: Default Visibilities

Programmer forgot to add Visibility for the function _sendEther()

```
1  contract Puzzle {
2    uint256 amount = 0.5; //0.5 ethers
3    function submit_answer (string answer) {
4      //Logic to check submitted answer
5      if (/*answer is correct*/)
6        _sendEther();
7    }
8    function _sendEther(){
9      msg.sender.tranfer(amount);
10   }
11 }
```

Ref: Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Smart Contract Vuln: Default Visibilities

Let's see a Demo

```
1  contract Puzzle {
2    uint256 amount = 0.5; //0.5 ethers
3    function submit_answer (string answer) {
4      //Logic to check submitted answer
5      if (/*answer is correct*/)
6        _sendEther();
7    }
8    function _sendEther(){
9      msg.sender.tranfer(amount);
10   }
11 }
```

Ref: Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Smart Contract Vuln: Timestamp Dependence

- A smart contract that utilizes a current timestamp to produce random numbers in order to determine lottery results
- Miners can put a timestamp within 30 seconds of block validation
- Miners can alter outcome of random number generator

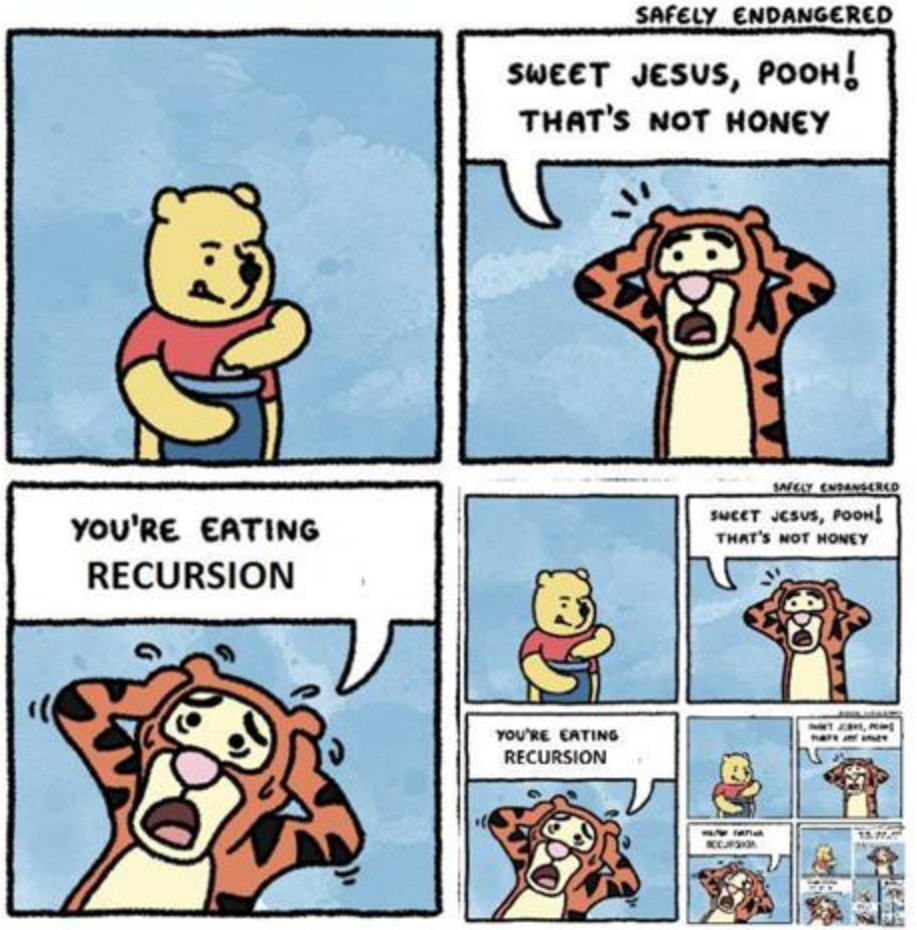
Ref: Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Smart Contract Vuln: Timestamp Dependence

```
1  pragma solidity ^0.5.0;
2  contract TimedCrowdsale {
3    event Finished();
4    event notFinished();
5    // Sale should finish exactly at January 1, 2019
6    function isSaleFinished() private returns (bool) {
7      return block.timestamp >= 1546300800;
8    }
9    function run() public {
10     if (isSaleFinished()) {
11         emit Finished();
12     } else {
13         emit notFinished();
14     }
15   }
16 }
```

Ref: https://swcregistry.io/docs/SWC-116

Recursion



Ref: CS 410b, Blockchain Development and Security, Instructor: Wu-chang Feng, Portland State University

Fallback Function in Solidity

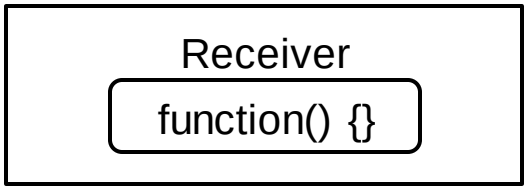
- Declare with fallback() and have no arguments.
- If it is not marked **payable**, the contract will throw an exception on receiving Ether without data
- No Return value, Once per contract
- Executed → if caller meant to call a non-available function or receive() does not exist
- Visibility: external.

Ref: https://www.geeksforgeeks.org/solidity-fall-back-function

Reentrancy Example (Expected Behavior)

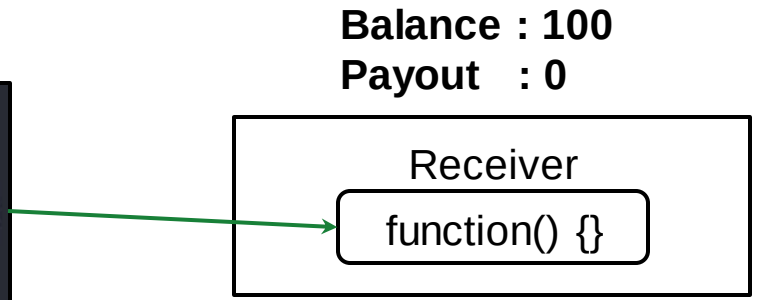
```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```

Balance : 100
Payout : 0



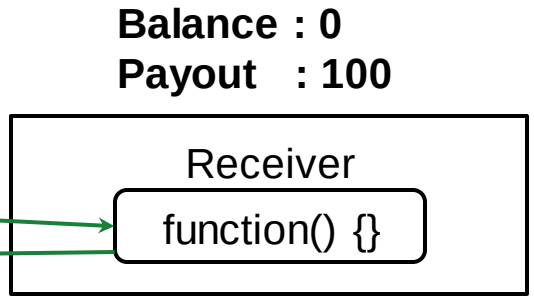
Reentrancy Example (Expected Behavior)

```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



Reentrancy Example (Expected Behavior)

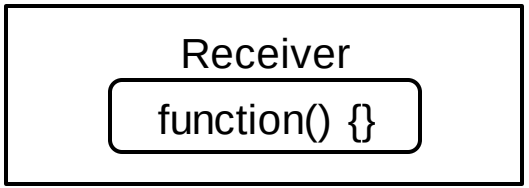
```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



Reentrancy Example (Exploit)

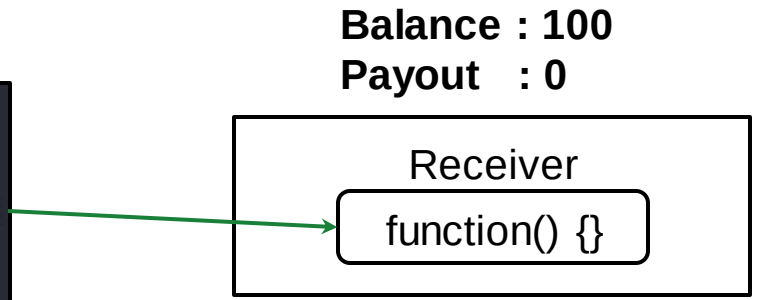
```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```

Balance : 100
Payout : 0



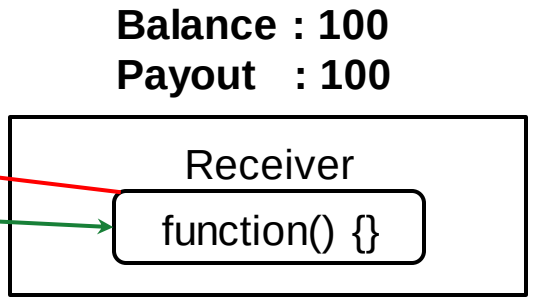
Reentrancy Example (Exploit)

```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



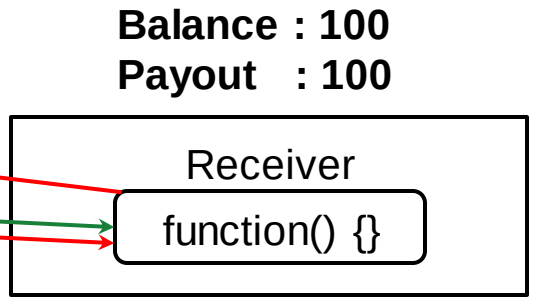
Reentrancy Example (Exploit)

```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



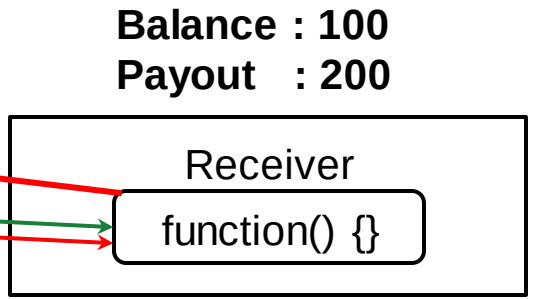
Reentrancy Example (Exploit)

```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



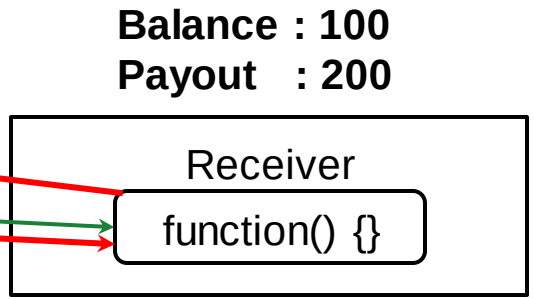
Reentrancy Example (Exploit)

```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



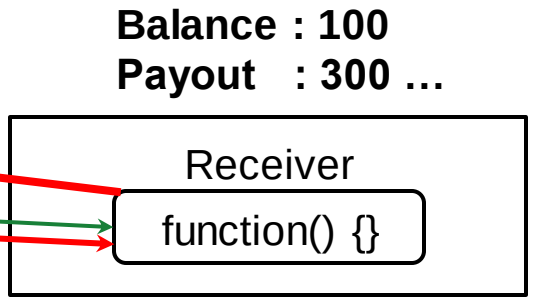
Reentrancy Example (Exploit)

```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



Reentrancy Example (Exploit)

```
1  function revoke() remote{
2    uint256 value = balances[msg.sender];
3    require(msg.sender.call.value(value) ());
4    balances[msg.sender] = 0;
5  }
```



Reentrancy Bug Fixed

```
1  function revoke() remote{ //Insecure
2      uint256 value = balances[msg.sender];
3      require(msg.sender.call.value(value) ());
4      balances[msg.sender] = 0;
5  }
6
7  function revoke() remote{ //Secure
8      uint256 value = balances[msg.sender];
9      balances[msg.sender] = 0;
10     require(msg.sender.call.value(value) ());
11 }
```


Ref: Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Reentrancy Example-2

```
15      function Collect(uint _am)
16      public
17      payable
18      {
19          var acc = Acc[msg.sender];
20          if( acc.balance>=MinSum && acc.balance>=_am && now>acc.unlockTime)
21          {
22              if(msg.sender.call.value(_am)())
23              {
24                  acc.balance-=_am;
25                  LogFile.AddMessage(msg.sender,_am,"Collect");
26              }
27          }
28      }
```

```
47      uint public MinSum = 2 ether;
```

```
37      struct Holder
38      {
39          uint unlockTime;
40          uint balance;
41      }
42
43      mapping (address => Holder) public Acc;
```

Reentrancy Example-3 (False Positive)

```
1  modifier onlyOwner{
2      require(msg.sender == owner);
3      _;
4  }
5  ...
6  function execute( address _to, uint _value, bytes
       _data) external onlyOwner {
7      ...
8      _to.call.value(_value)(data);
9  }
```

Code example: Identify control based permission control

Ref: Zibin Zheng, Neng Zhang, Jianzhong Su, Zhijie Zhong, Mingxi Ye, and Jiachi Chen. 2023. Turn the Rudder: A Beacon of Reentrancy Detection for Smart Contracts on Ethereum. In Proceedings of the 45th International Conference on Software Engineering (ICSE'23). IEEE Press, 295–306. https://doi.org/10.1109/ICSE48619.2023.00036

Reentrancy Example-4 (False Positive)

```
contract ReentrancyGuard {
    bool private _notEntered;
    constructor () internal {
        _notEntered = true;
    }
  modifier nonReentrant() {
      require(_notEntered);
      _notEntered = false;
      _;
      _notEntered = true;
    }
}
contract GovernanceVesting is ReentrancyGuard {
    ...
  function withdraw() public nonReentrant {
      ...
      IERC20(Token).transfer(governanceAddress,
          governanceFunds);
      Withdrawn = true;
    }
}
```

Code example: Reentrancy lock based permission control

Ref: Zibin Zheng, Neng Zhang, Jianzhong Su, Zhijie Zhong, Mingxi Ye, and Jiachi Chen. 2023. Turn the Rudder: A Beacon of Reentrancy Detection for Smart Contracts on Ethereum. In Proceedings of the 45th International Conference on Software Engineering (ICSE'23). IEEE Press, 295–306. https://doi.org/10.1109/ICSE48619.2023.00036

Reentrancy Example-5 (False Positive)

```
1  contract ForeignToken {
2      function balanceOf(address _owner) constant
           public returns (uint256);
3      ...
4    }
5  contract Bitcash {
6      ...
7      function getTokenBal(address tokenAddr,
           address who) constant public returns (uint
           ){
8          ForeignToken t = ForeignToken(tokenAddr);
9          uint bal = t.balanceOf(who);
10         return bal;
11       }
12   }
```

Code example: No state change after external call

Ref: Zibin Zheng, Neng Zhang, Jianzhong Su, Zhijie Zhong, Mingxi Ye, and Jiachi Chen. 2023. Turn the Rudder: A Beacon of Reentrancy Detection for Smart Contracts on Ethereum. In Proceedings of the 45th International Conference on Software Engineering (ICSE'23). IEEE Press, 295–306. https://doi.org/10.1109/ICSE48619.2023.00036

Unchecked Call Return Value

- Return value of message call → not checked.
- If
  - call fails accidentally
  - attacker forces the call to fail
    - may cause unexpected behaviour in the subsequent program logic.

Ref: https://swcregistry.io/docs/SWC-104/

Unchecked Call Return Value

```
pragma solidity 0.4.25;

contract ReturnValue {

  function callnotchecked(address callee) public {
    callee.call();
  }
```

Ref: https://swcregistry.io/docs/SWC-104/

Unchecked Call Return Value

```
pragma solidity 0.4.25;

contract ReturnValue {

  function callchecked(address callee) public {
    require(callee.call());
  }

  function callnotchecked(address callee) public {
    callee.call();
  }
```

Ref: https://swcregistry.io/docs/SWC-104/

Unchecked Call Return Value

```
// Bad Code:
function Transfer(address _addr) public {
    (bool success, bytes memory data) = _addr.call{value: msg.value, gas: 5000}();
```

Ref: https://sm4rty.medium.com/unchecked-call-return-value-solidity-security-1-fe794a7cdb6f

Unchecked Call Return Value

```
// Bad Code:
function Transfer(address _addr) public {
    (bool success, bytes memory data) = _addr.call{value: msg.value, gas: 5000}();


// Good Code
function Transfer(address _addr) public {
    (bool success, bytes memory data) = _addr.call{value: msg.value, gas: 5000}();
    require(success, "Transfer Failed")
```

Ref: https://sm4rty.medium.com/unchecked-call-return-value-solidity-security-1-fe794a7cdb6f

THE END