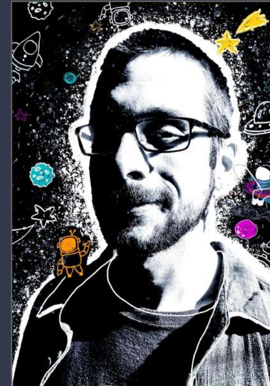
Blockchain and smart contracts

an intro

About me



José Carlos (JC) Ramírez

- ~10 years working on **offensive security**
- **Auditing** smart contracts since 2021
- Blockchain security engineering at

Teaching smart contract security in University of Malaga, IEBS, independent workshops...

What are we going to talk about?

- What is a blockchain?
- What is an Smart Contract?
- What causes Web3 incidents?
- Are there native vulnerabilities?

Smart Contracts

Smart Contracts

Application/program that "**runs on the blockchain**"

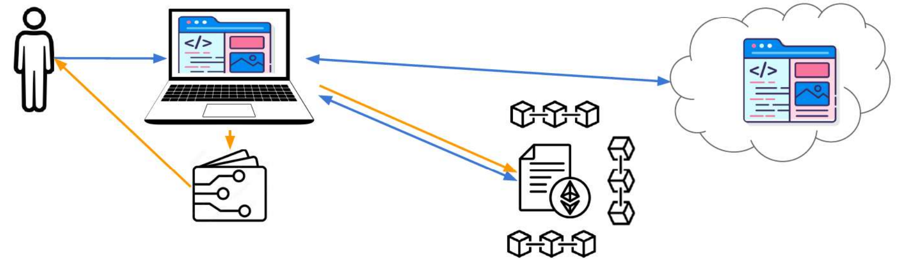
- Its code causes **state changes**
- Anyone can interact with their **exposed functions**
  - By sending a signed transaction
  - Can't initiate transactions on their own

Not every blockchain have smart contract support

Smart Contracts

We use the name **Dapps (decentralized apps)** for the webapps that use **blockchain**

- At least on their "backend" - **smart contract**
- The frontend can also be stored on **IPFS** o similar options



Smart Contracts

Just **ledger related data** is not enough now! We need to:

- Store their **code**
- Store memory **data/state variables**
- A **virtual machine** to execute the code

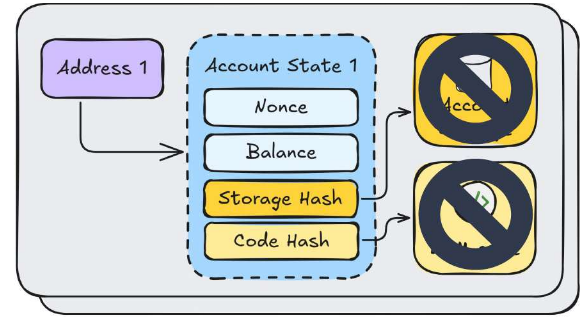
From now on, every example would refer to **Ethereum/EVM**

It is not just a "distributed ledger", but a **distributed turing-complete state machine**. Ethereum is sometimes referred as "the world computer"

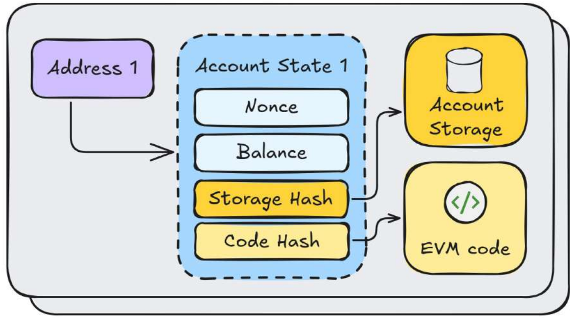
"A globally shared and replicated memory, where you can read and write from everywhere through smart contracts"

Smart Contracts

EOA (user generated keypair) → Can initiate transactions



Smart Contracts



Smart Contracts

Ethereum Virtual Machine (**EVM**) - running in every node

- Defines a **set of rules** that smart contracts uses to modify the state data
  - EVM OpCode set
- Execution is **deterministic** - allows consensus
  - Given a set of inputs and a state, you always reach the result
- **Mono-threaded**, one tx executed at a time

Smart Contracts

```
/**
    @notice Distributes a percentage of the total vested to the
        rewarded with a percentage of the distributed amount as
    @param percentage The percentage of the vested tokens to dis
 */
function distributeBenefits(uint percentage)
    external
    returnRewards(percentage)
{
    //  Checks
    require(total_invested >= MIN_INVESTED, "Not big enough to a
    require(percentage < MAX_PERCENTAGE, "Not big enough to avoi
    require(block.number - latest_distribution >=

    // Effects
    latest_distribution = block.number;
    // Calculate the amount to distribute as a per
    uint amount = total_invested * percentage / PERCENT;
    // Subsctract the distributed amount from the total vested
    total_invested -= amount;

    //Interactions
    doDistribute(amount);

    emit Benefits(amount);
}
```

```
0B: 57      JUMPI
0C: 6000    PUSH1 0x00
0E: 80      DUP1
0F: FD      REVERT
10: 5B      JUMPDEST
11: 50      POP
12: 60C7    PUSH1 0xc7
14: 80      DUP1
15: 61001F  PUSH2 0x001f
18: 60      PUSH1 0x00
1A: 39      CODECOPY
1B: 60      PUSH1 0x00
1D: F3      RETURN
1E: 00      STOP
```

Smart Contracts

The **Gas** in EVM is the payment for **computational efforts**

- It is not Eth, but it **costs Eth**
  - Price fluctuates depending on demand!
- Avoids **spam** and the "**halting-problem**" of touring-complete machines

A **max gas limit** per transacción exists!

Each tx requires a min amount of gas to be paid... but **you can send an extra tip!**

- Validators are **incentivized** to pick our tx sooner

Smart Contracts

Exploring the blocks

Solidity

Solidity

High level language syntactically similar to JavaScript (+-)

```
273        /**
274            @notice Endpoint to set the vacation mode of a seller. If the seller is in vacation
275            @param _vacationMode The new vacation mode of the seller
276         */
277        function setVacationMode(bool _vacationMode) external {
278            for (uint i = 0; i < offerIndex; i++) {
279                if (offered_items[i].seller == msg.sender) {
280
281                    if (_vacationMode && offered_items[i].state == State.Selling) {
282                        offered_items[i].state = State.Vacation;
283
284                    } else if (!_vacationMode && offered_items[i].state == State.Vacation) {
285                        offered_items[i].state = State.Selling;
286
287                    }
288                }
289            }
290        }
```

Solidity

Data types and structure commonly found in other languages:

- `int8…int256, uint8…uint256`
- `Strings, booleans, byte`
- `Arrays, mappings, enums, structs`

Loops and **statements**:

- `If - else if - else`
- `For, while`
- `+ - * / %`
- `==, !=, >, >=, <, <=, &&, ||...`

Solidity

- "**Contract oriented**": the unit that stores the functionalities is "a contract"
- **Typed** language
- Supports **inheritance** and external libraries

```
pragma solidity ^0.8.13;

import "./my_library/counter.sol";

contract HelloWorld is Counter {

    string public name;
    uint256 public age;

    ...

}
```

Solidity

Functions can be **exposed** (or not) through **visibility modifiers**

```
function setName(string _name) public {
    name = _name;
}

function getName () public view returns(string) {
    return name;
}

function doSomething() internal {
 // ...
}
```

Solidity

Let's explore and deploy a contract!

1. Code inspection
    a. Special variables, payability, special functions...
2. Get testnet ether
3. Deploy our contract
4. Interact with our contract
5. Explore the chain!

Exercise

Thanks

@jcsec_audits

/jcsec-security

/in/jcramirezv/

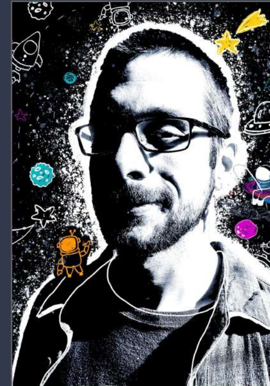
jc@jcsec.io

Questions?

Smart contract security

an intro

About me



José Carlos (JC) Ramírez

- ~10 years working on **offensive security**
- **Auditing** smart contracts since 2021
- Blockchain security engineering at

Teaching smart contract security in University of Malaga, IEBS, independent workshops...

What are we going to talk about?

What is a blockchain?

What is an Smart Contract?

What causes Web3 incidents?

Are there native vulnerabilities?

$650M

Biggest smart contract related hack to date

$1400M

Biggest hack to the day (phishing, Feb 25), close to the 2023 total

Web3 security incidents

"My cryptos got stolen!" → 99% social engineering or scams

- **Physical compromise**
  - Written down seed phrase, keys in a USB, laptop…
- **Traditional phishing**: email, messaging apps, discord, etc.
  - "Could you run this tests for me in your machine?"
- **Malicious Dapps** to coerce you into signing transactions without validating the contents
  - "Please sign this tx due to whatever reason"

Web3 security incidents

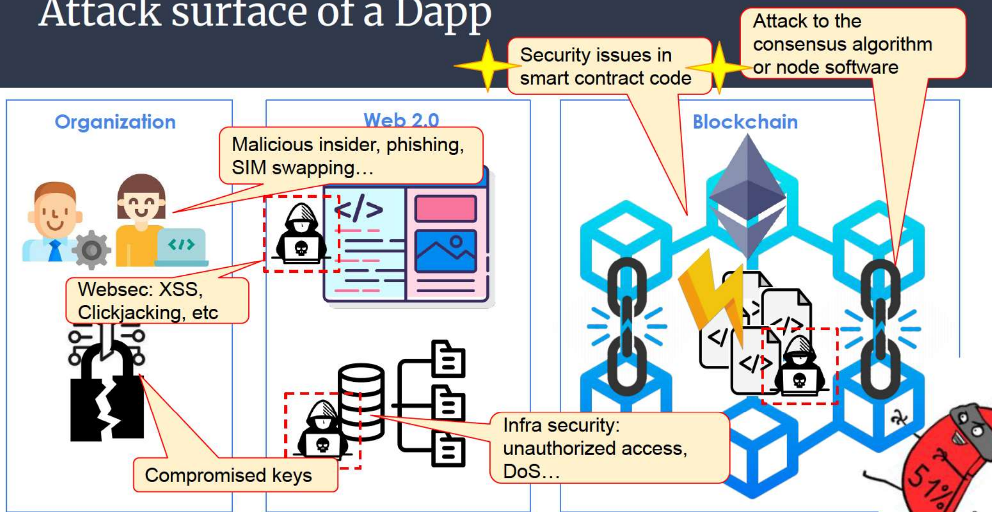
Most are related to **Phishing**, **Rug pulls and similar scams** and not to vulns

Attractive environment for exploitation:
- **"New" technology** that directly deals with **high value assets**

Traditional tech security should not be forgotten either, as it is part of most Web3 projects:
- Key and credentials storage
- Webapp security
- DNS
- …

Not every Web3/Crypto incident is due to a DApp vulnerability

Attack surface of a Dapp



Vulnerabilities in smart contracts

Blockchain "native" traits

- Pending TXs, the mempool
  - Front-running
- Everything is public
  - Clear-text secrets
- TX atomicity
  - Push vs Pull
- When ETH splitted
  - Reentrancy

Repo!

https://github.com/jcsec-security/smart-contract-and-hacking-101

Front-running

In Ethereum the computational costs for validators are paid through "**Gas**"

If someone pays **more** gas (*tip*), validators are incentivized to pick that transaction **before others** as they will get additional funds

Broadcasted transactions that have not been added to a block yet: **pending transactions**

Before being added into a block, transactions are held in **public (most of the time) mempool** where validators can select them as part of the next forged block

Exercise

Case study: "crack-the-hash" competition

Clear-text secrets

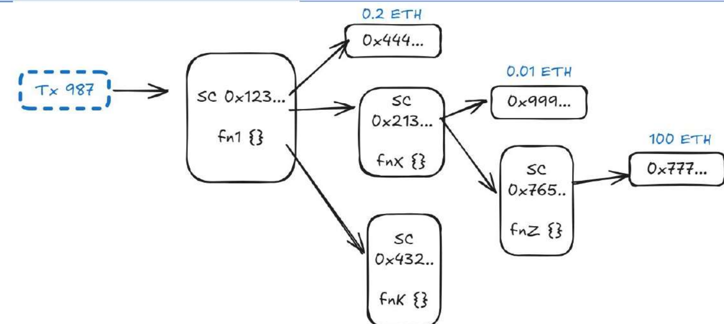
All the **data** contained inside transactions that have been forged into a **block** is **publicly** available

All.

- **Case study 1):** basic authentication for smart contracts
- **Case study 2):** hash-based authentication for smart contracts
- **Case study 3):** one-time hash-based authentication for smart contracts

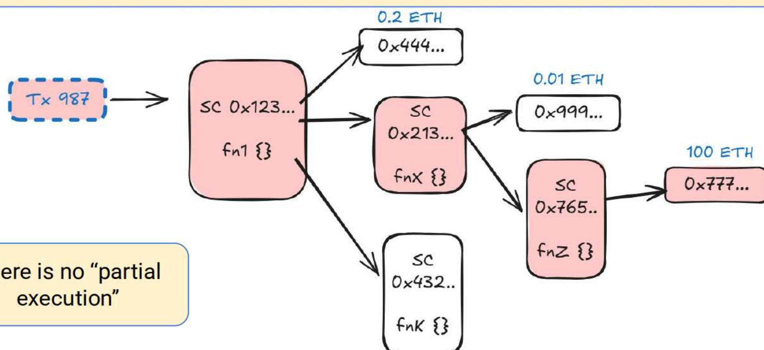
Push vs pull

Transactions in Ethereum can contain multiple calls to one or multiple addresses within



Push vs pull

Txs are **atomic**: when one call fails it propagates a `revert()`, making the whole TX fail



There is no "partial execution"

Push vs pull

Txs are **atomic**: when one call fails it propagates a `revert()`, making the whole TX fail

There is no "partial execution"

Bash script (no exc handling)

-> Read file A
-> Read file C
-> Write file A
-> Send HTTP request
-> Delete file B
-> Create user XYZ

Eth tx (no exc handling)

-> Call 1
-> Call 2
-> Call 3
-> Call 4
-> Call 5

Push vs pull

Txs are **atomic**: when one call fails it propagates a `revert()`, making the whole TX fail

Bash script (no exc handling)

-> Read file A
-> Read file C
-> Write file A
-> Send HTTP request
-> Delete file B
-> Create user XYZ

Eth tx (no exc handling)

-> Call 1
-> Call 2
-> Call 3
-> Call 4
-> Call 5

There is no "partial execution"

Push vs pull

Txs are **atomic**: when one call fails it propagates a `revert()`, making the whole TX fail

Push: calls done from a contract to a sequence of external accounts
E.g. *distribute all the prizes from a lottery*

Pull: calls done from the contract to a single account, in general the same one that is calling
E.g. *funds withdrawal*

Exercise

Case study: Funds redistribution

Reentrancy

Data stored in the blockchain is **immutable**

Smart contracts' code **too.** By default, an Ethereum smart contract code **can not be upgraded**

The code is law!

The DAO hack

The code... ***was*** law

Reentrancy

*"A program is called **reentrant** if it can be interrupted in the **middle of its execution** and **called again** before the previous call complete its execution."*

```
i = 2
```

```
function f1(){
  i++
  f2()
  i++
}
```

```
function f2(){
  if i == 1 {
     f1()
  }
}
```

Reentrancy

Victim: "MyVault" allows user to deposit funds and withdraw them later

Attacker: SC whose `receive()` calls `withdraw()` from the victim's contract

The special fns `receive()` and `fallback()` are automatically executed upon an ether tx

```
withdraw():
1.  Checks credit
2.  Transfers ETH
3.  Updates balance
```

(3. Updates balance — struck through)

```
receive():
1.  Checks if V still has funds
2.  withdraw()
```

21

Reentrancy

We have a reentrancy vuln when **within the same transaction** the following can be done:

1. **Contract A(ttacker)** calls **contract V(ictim)**

"I want to retrieve my deposit"

```
victim.withdraw();
```

For example, an ether transfer

2. During the execution, **contract V** interacts with **contract A** before updating some of its state variables

```
payable(msg.sender).call{value: balance[msg.sender]}("");

balance[msg.sender] = 0;
```

Reentrancy

3. At this point, **contract A gets control of the execution back** allowing it to call the vulnerable function again! Points 2 and 3 can now be **repeated**.

```
receive() external payable {

    victim.withdraw();

}
```

The special fns `receive()` and `fallback()` are automatically executed upon an ether tx

4. The pending state variables from **contract V** will be **updated** when the attacker decides to stop looping the reentrancy cycle

```
balance[msg.sender] = 0;
```

Reentrancy

Root cause: incorrect ordering of logic

```
// Checks
require(balance[msg.sender] > 0);

// Ext interactions :(
sender.call{value:
balance[msg.sender]}("");

// Effects
balance[msg.sender] = 0;
```

```
// Checks
require(balance[msg.sender] > 0);

// Effects
amount = balance[msg.sender];
balance[msg.sender] = 0;

// Interactions
sender.call{value: amount}("");
```

Conclusión

Bl0ckchain is unh4xkable!

Based on:

- Block's data **immutability**
- **Availability** of a globally distributed architecture
- Usage of battle-tested **cryptographic signing**

Actual risks exists:

- 51% or other **attacks on the consensus** algo
- **Supply chain** software vulnerability
- High node % running on **centralized infra**
- Key compromise / random **scams**
- **Smart contracts** vulnerabilities
- ...

Thanks

Questions?