THE UNIVERSITY OF TEXAS AT EL PASO

Smart Contracts & Blockchains

By: Adrian Urquizo

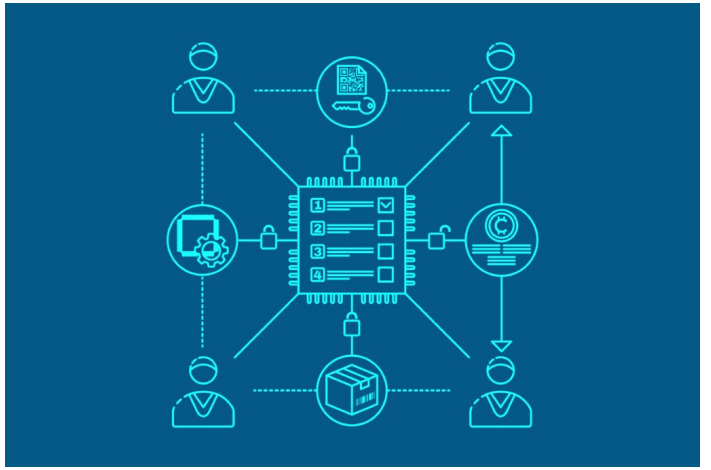
Smart Contracts

What is a Smart Contract

- Computer programs living on the Ethereum blockchain. These self automated programs are executed when triggered by a transaction from a user. These programs act as building blocks for decentralized apps and organizations.

Structure of Smart Contracts

- **State Variables =** permanently stored
- **Functions =** units of code
- **Function Modifiers =** amend semantics
- **Events =** interface with EVM
- **Errors =** define for failure situations
- **Struct Types =** group variables
- **Enum Types =** finite constant values



Example of Smart Contracts

Subcurrency in its simplest form

- This example lets the creator to create new coins, however, anyone can send coins to each other without registering for a username & password all you need is a Ethereum keypair.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.26;

// This will only compile via IR
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 {
        require(amount <= balances[msg.sender],
InsufficientBalance(amount, balances[msg.sender]));
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
          emit Sent(msg.sender, receiver, amount);
```

Faults in Smart Contracts

Importance of catching faults:

- Errors in smart contracts cannot be fixed due to their immutability.

- If a state in a blockchain generated a transaction by a faulty contract, the state cannot roll back. Therefore, the error is kept in the chain of history, which we do not want!

Faults in Smart Contracts

Best way to catch faults:

- Smart contract security audits check for any vulnerabilities in smart contracts.

Security teams when doing an audit check for:

- Fake token and dust attacks (Attacks contract edge cases)
- Timestamp dependence (Relies on timestamp value to execute an operation)

Faults in Smart Contracts

Example of a fault:

- Token Vault - Allows users to deposit and withdraw tokens. The smart contract keeps track of the balances of each user.
  - A *dust attack* can test the vulnerability of a smart contract by sending very small amounts (dust) of money to each address to test the contract's edge cases. If the smart contract cannot handle small amounts, this can lead to user de-anonymization and rounding errors in transactions.
  - To prevent this, one can deploy a new contract that rejects very small amounts.

Faults in Smart Contracts

Example of a fault:

- Lottery - Allows users to purchase tickets for a lottery. Winner is chosen based on the block timestamp.
    - A *timestamp dependence* in this example chooses a pseudo-random number to pick the winner, which can lead to attackers manipulating the timestamp to influence the outcome.
    - To prevent this, combining multiple unpredictable factors (block hash, previous block hash) can reduce the predictability of the outcome.

Blockchains

What is a Blockchain

- A distributed database that maintains a continuously growing list of ordered records, called **blocks**. Each block contains an address memory of the previous block (sort of like a list), a timestamp of process execution, and transaction data.

- A simplified definition from *Blockchains for Dummies* - Blockchain owes its name to the way it stores transaction data - in *blocks* linked together to form a chain.

Example of Blockchain

Blockchain for payment processing and money transfers

- Any transaction that is processed over a blockchain could be settled within a matter of seconds and reduce/eliminate banking transfer fees.

How They Interact With Each Other

Smart Contracts -> Blockchain

Smart contracts are ran on a decentralized blockchain instead of a centralized server. This means there are networks that distribute data and processing tasks across several nodes, with no single point of authority.

By doing so, smart contracts allow multiple parties/users to come to a shared result in an accurate and timely manner.

Solidity

object-oriented, high-level language for implementing smart contracts

voting, crowdfunding, blind auctions and multi-signature wallets

designed to target the Ethereum Virtual Machine (EVM)

Remix IDE

Remix IDE Instructions

Link to Remix IDE:

https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js

Step 1: Open Remix IDE - Select *New File* and click *Solidity.*

Step 2: Write your smart contract and click compile under the Solidity compile tab.

Step 3: Deploy your contract under the Deploy and Run Transactions tab.

Step 4: Click on *getResult* and look for “decoded output” for your contract’s result.

Remix IDE Example with Solidity

Home | test_code.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.26;
3  contract SolidityTest {
4      uint a = 10;
5      uint b = 12;
6      uint sum;
7      function getResult() public returns(uint){    infinite gas
8          sum = a + b;
9          return sum;
10     }
11 }
```

SOLIDITY COMPILER

COMPILER

latest local version - soljson...

Include nightly builds

Auto compile

Hide warnings

Advanced Configurations

Compile test_code.sol

Compile and Run script

DEPLOY & RUN TRANSACTIONS

evm version: cancun

Deploy

Publish to IPFS

At Address | Load contract from A

Transactions recorded 4

Pinned Contracts (network: vm-cancun)

No pinned contracts found for selected workspace & network

Deployed/Unpinned Contracts

SOLIDITYTEST AT 0XD

Balance: 0 ETH

getResult

Low level interactions

CALLDATA

Transact

```
decoded output {
    "0": "uint256: 22"
}
```

Ethereum

A network of computers that act as the foundation for communities, applications, organizations and digital assets you can build and use. For example…

- Networks
- Open Internet
- Commerce
- Banking

decentralized blockchain
= SECURE

NO single entity

References

https://consensys.io/knowledge-base/how-does-a-blockchain-work

https://docs.soliditylang.org/en/latest/structure-of-a-contract.html#

https://docs.soliditylang.org/en/latest/contracts.html

https://docs.soliditylang.org/en/v0.8.26/

https://ethereum.org/en/what-is-ethereum/

https://docs.soliditylang.org/en/v0.8.26/introduction-to-smart-contracts.html

https://chain.link/education/smart-contracts#:~:text=Each%20smart%20contract%20consists%20of,%2C%20and%20tamper%2Dproof%20manner.

https://www.synopsys.com/glossary/what-is-blockchain.html#:~:text=Definition,a%20timestamp%2C%20and%20transaction%20data.

https://blaize.tech/article-type/web3-security/9-most-common-smart-contract-vulnerabilities-found-by-blaize/

https://www.geeksforgeeks.org/steps-to-execute-solidity-smart-contract-using-remix-ide/