Smart Contracts Vulnerabilities

Cybersecurity lab 1 - 2022 / 2023

Khalil Abdellah Lemtaffah

Agenda

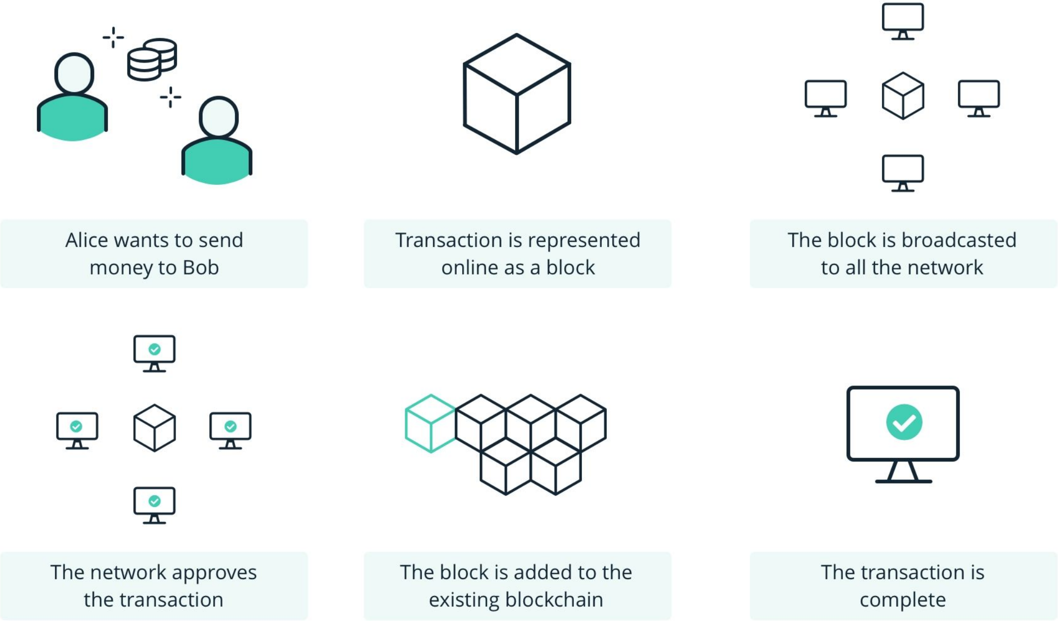
- Blockchain overview
- Ethereum
- Smart contracts
  - Definition
  - Security
- Timeframe
- Contract Development
- Security Issues
- Fixing
- Conclusion

What is blockchain?

What is blockchain?

- Distributed ledger / database, shared among the nodes of a computer network.
- Every node has the exact same shared information with the other nodes
- Since the network is decentralized, there are no owners in the blockchain database, and the data travels **through the peer-to-peer network secured by an immutable cryptographic signature**.

What is blockchain?



Alice wants to send money to Bob

Transaction is represented online as a block

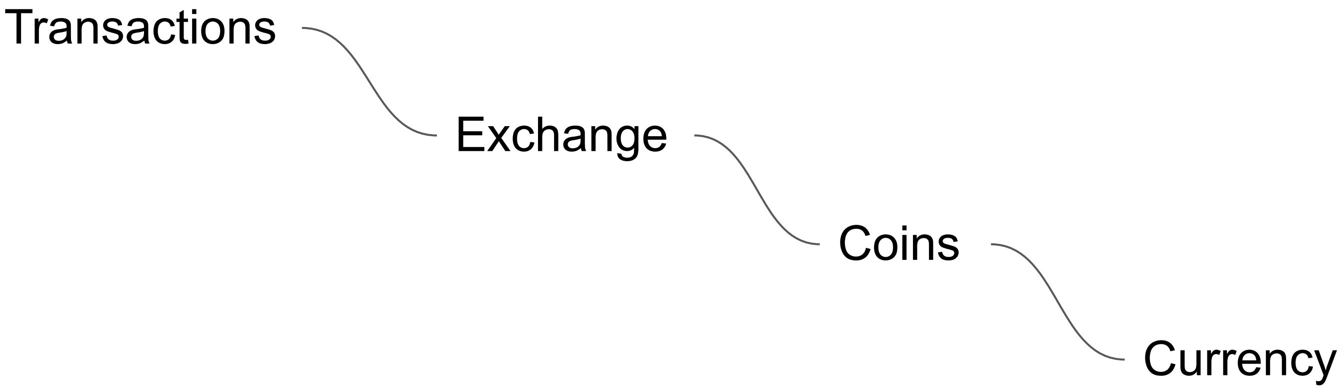
The block is broadcasted to all the network

The network approves the transaction

The block is added to the existing blockchain

The transaction is complete

What is blockchain?



Transactions

Exchange

Coins

Currency

What is blockchain?

CryptoCurrency

What is blockchain?

CryptoCurrency

Ethereum

Ethereum

- Ethereum (2015) is a decentralized and open source blockchain platform, that runs via smart contracts execution.

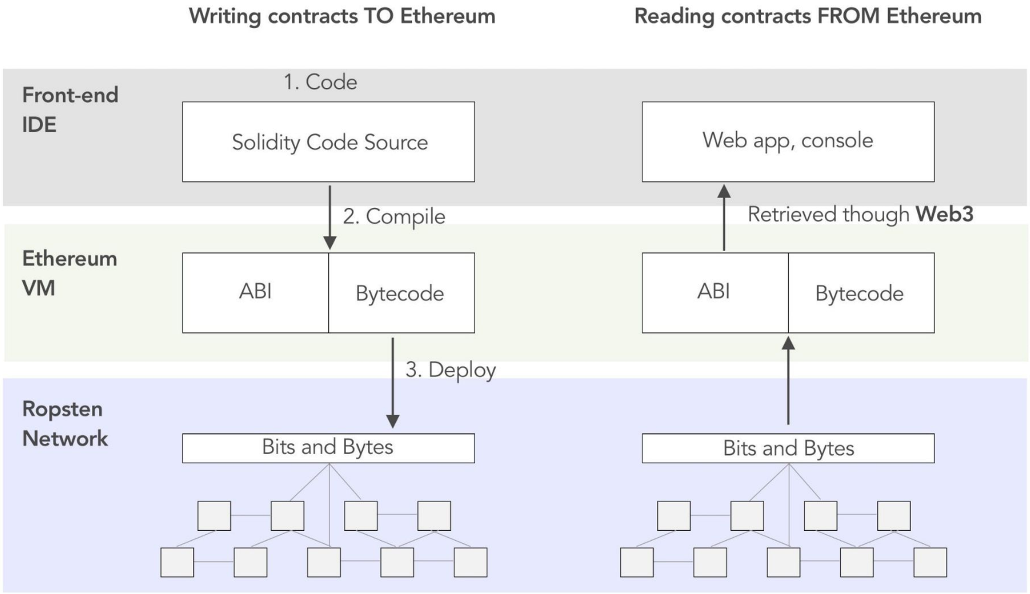
- The currency name of the blockchain is **Ether (ETH)**, at the moment it costs ~ **€1,247**.

Ethereum

Ethereum introduced the Smart Contracts technology to the crypto market, where 2 parties agree upon some transaction digitally.

Ethereum

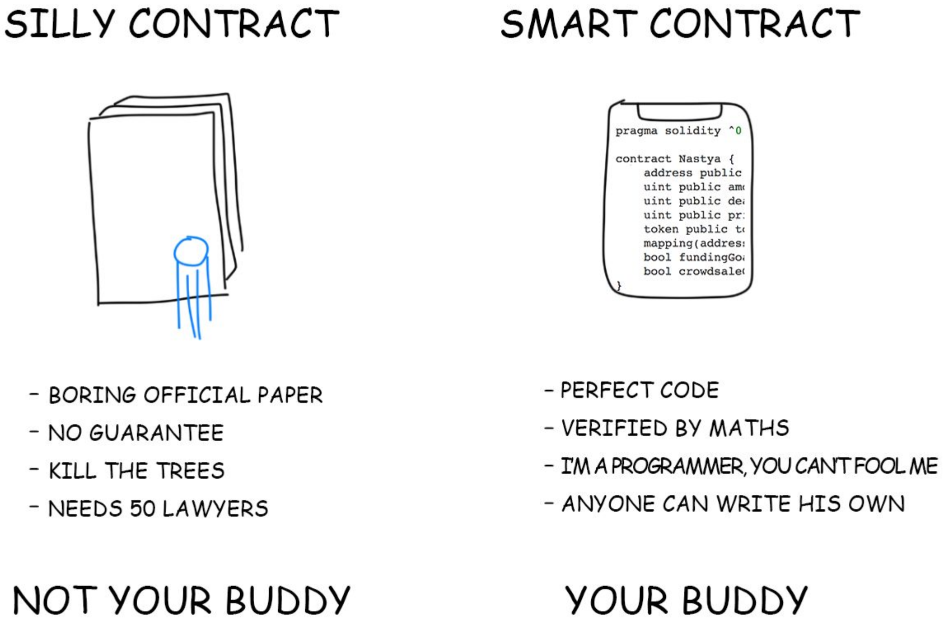
These smart contracts are compiled to a bytecode, and executed in a machine called the Ethereum Virtual Machine (EVM).



Smart contracts

Smart contracts - Definition

Imagine smart contracts as a program with a fix address on the Ethereum blockchain, that needs some fees to run instructions.



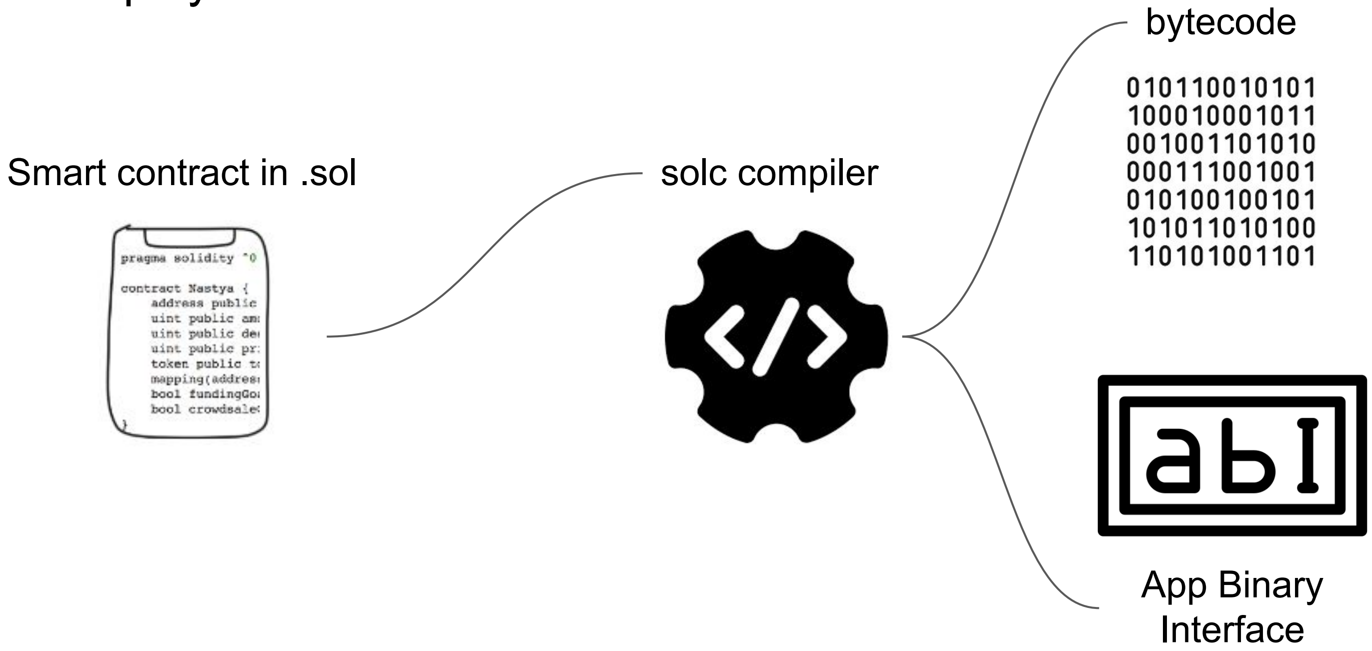
```
pragma solidity ^0

contract Nastya {
    address public
    uint public am
    uint public de
    uint public pr
    token public t
    mapping(addres
    bool fundingGo
    bool crowdsale
}
```

Smart contracts - Definition

The smart contract code is written in a programming language specified for that, Solidity is the widely used one.

Smart contracts - Definition

How is it deployed to the Blockchain then?



Smart contract in .sol

```
pragma solidity ^0

contract Nastya {
    address public
    uint public am
    uint public de
    uint public pr
    token public t
    mapping(addres
    bool fundingGo
    bool crowdsale
}
```

solc compiler

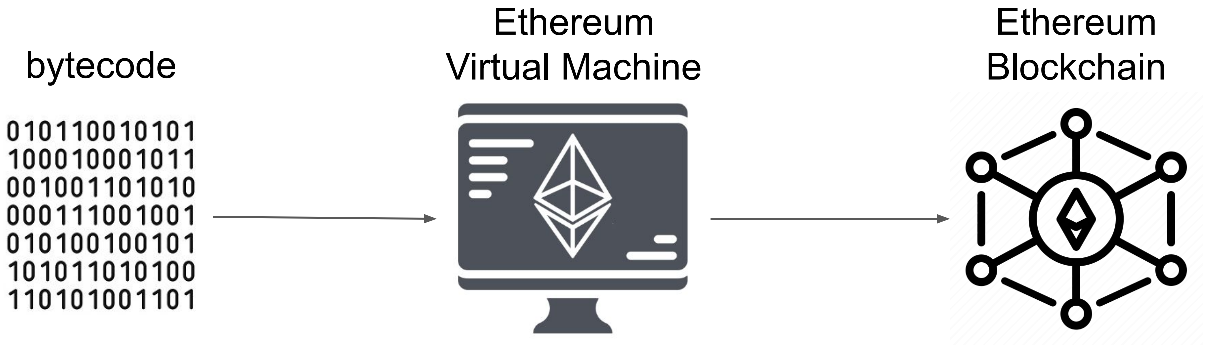
bytecode

```
010110010101
100010001011
001001101010
000111001001
010100100101
101011010100
110101001101
```

App Binary Interface

Smart contracts - Definition

How is it deployed to the Blockchain then?



bytecode

Ethereum
Virtual Machine

Ethereum
Blockchain

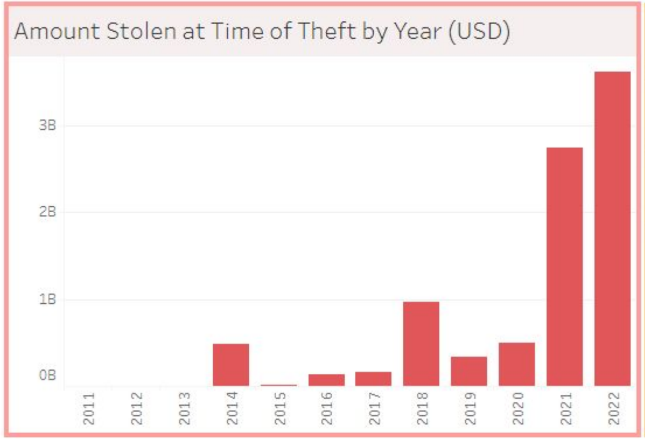
Smart contracts - Security

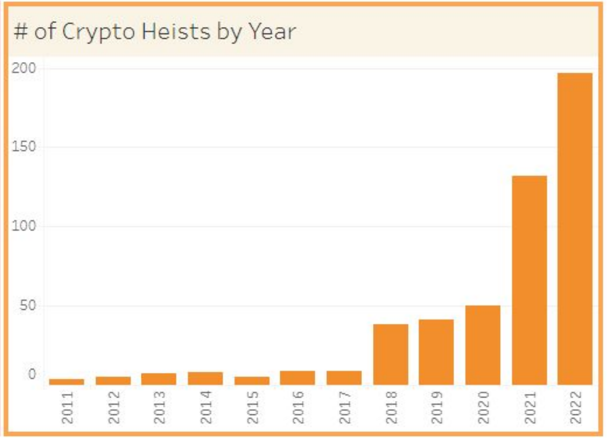
Everything is vulnerable, especially programming languages…

Smart contracts are kind of an “if… then… else” approach, meaning there are a lot of business logic bugs.

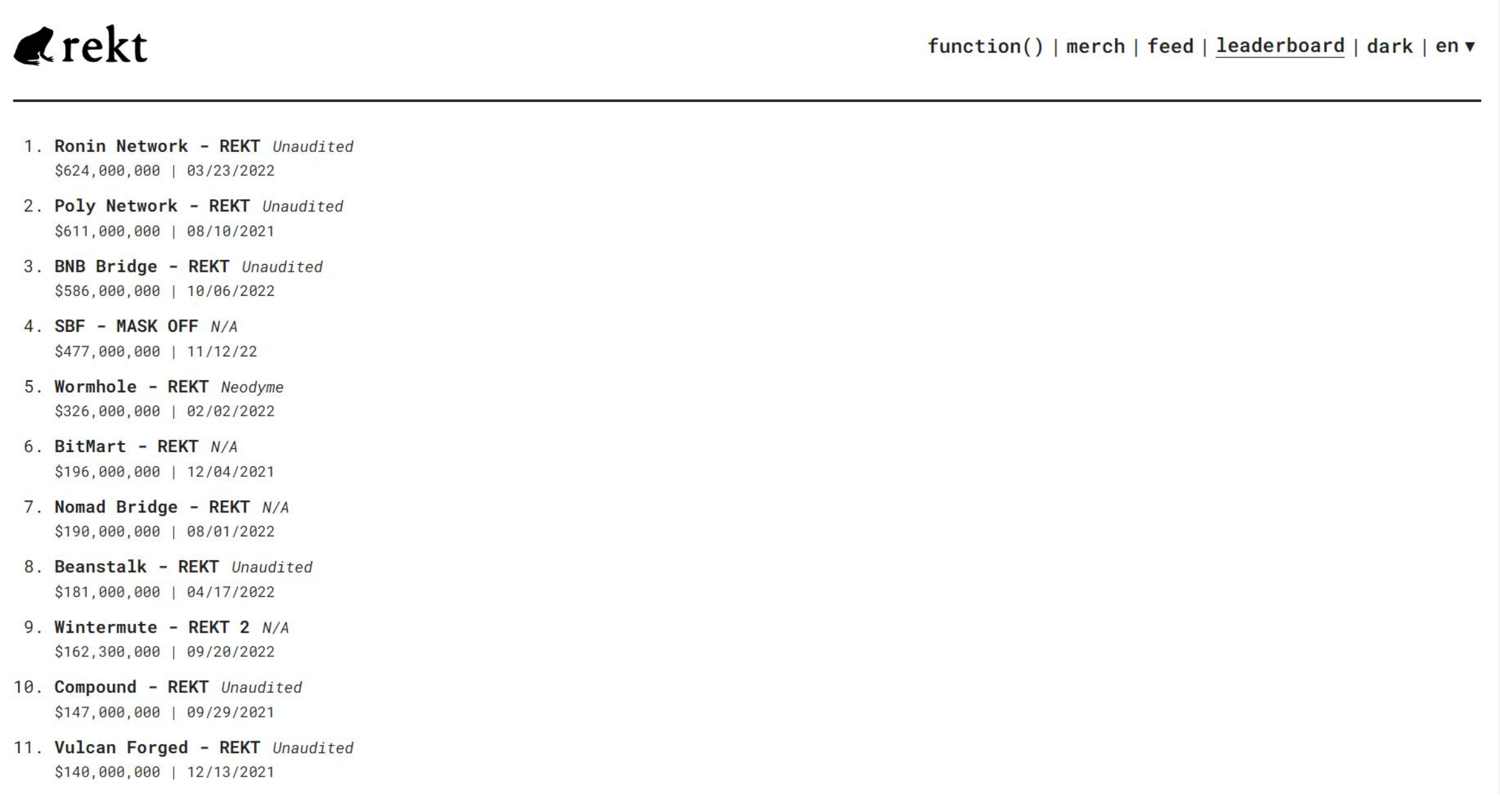
Smart contracts - Security

The blockchain vulns are known with the major loss consequences financially…





Smart contracts - Security



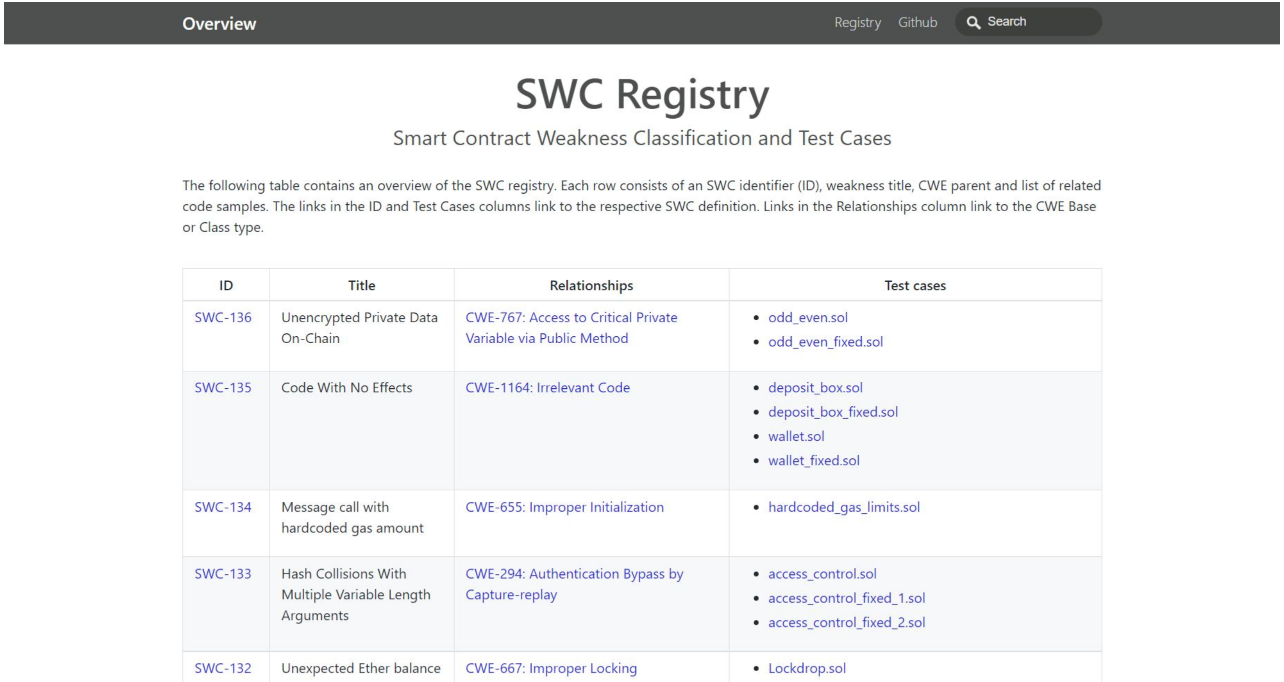
rekt

function() | merch | feed | leaderboard | dark | en ▾

1. **Ronin Network - REKT** *Unaudited*
   $624,000,000 | 03/23/2022
2. **Poly Network - REKT** *Unaudited*
   $611,000,000 | 08/10/2021
3. **BNB Bridge - REKT** *Unaudited*
   $586,000,000 | 10/06/2022
4. **SBF - MASK OFF** *N/A*
   $477,000,000 | 11/12/22
5. **Wormhole - REKT** *Neodyme*
   $326,000,000 | 02/02/2022
6. **BitMart - REKT** *N/A*
   $196,000,000 | 12/04/2021
7. **Nomad Bridge - REKT** *N/A*
   $190,000,000 | 08/01/2022
8. **Beanstalk - REKT** *Unaudited*
   $181,000,000 | 04/17/2022
9. **Wintermute - REKT 2** *N/A*
   $162,300,000 | 09/20/2022
10. **Compound - REKT** *Unaudited*
    $147,000,000 | 09/29/2021
11. **Vulcan Forged - REKT** *Unaudited*
    $140,000,000 | 12/13/2021

Smart contracts - Security

Web2 has OWASP top 10 project, and Web3 has the SWC registry.



SWC Registry

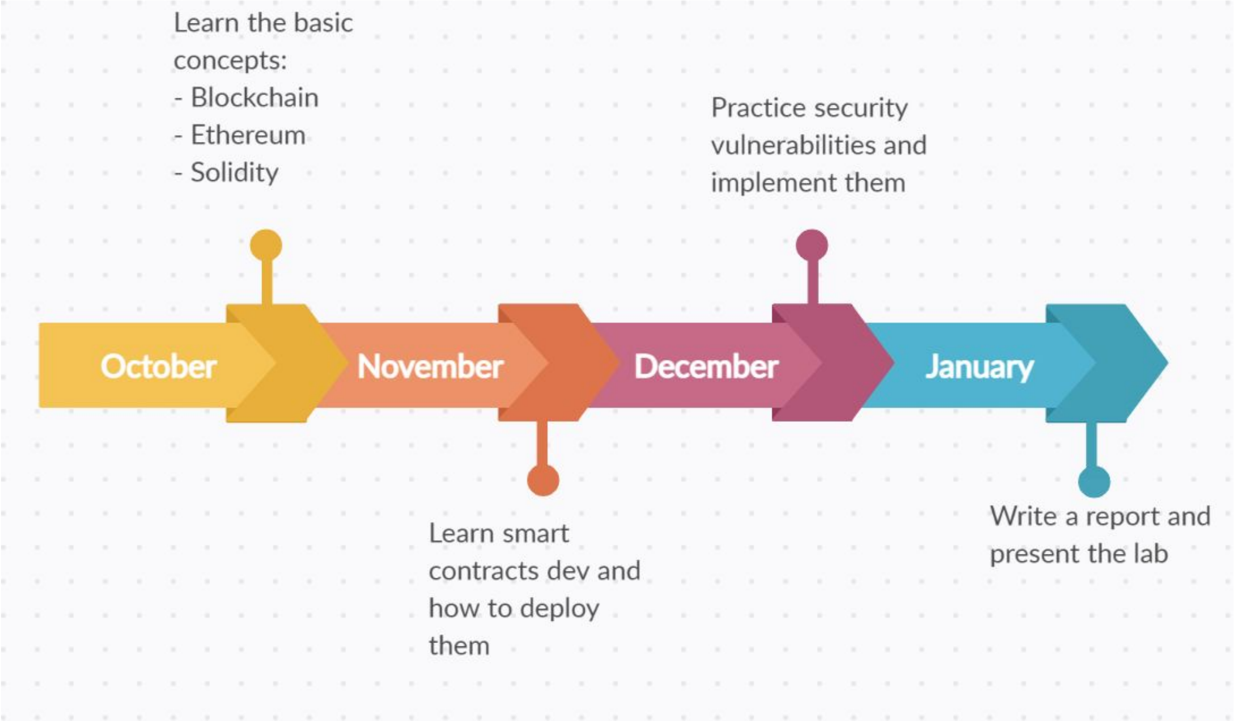
Smart Contract Weakness Classification and Test Cases

The following table contains an overview of the SWC registry. Each row consists of an SWC identifier (ID), weakness title, CWE parent and list of related code samples. The links in the ID and Test Cases columns link to the respective SWC definition. Links in the Relationships column link to the CWE Base or Class type.

ID	Title	Relationships	Test cases
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	• odd_even.sol • odd_even_fixed.sol
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	• deposit_box.sol • deposit_box_fixed.sol • wallet.sol • wallet_fixed.sol
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	• hardcoded_gas_limits.sol
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	• access_control.sol • access_control_fixed_1.sol • access_control_fixed_2.sol
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	• Lockdrop.sol

Timeframe

Timeframe



Learn the basic concepts:
- Blockchain
- Ethereum
- Solidity

Learn smart contracts dev and how to deploy them

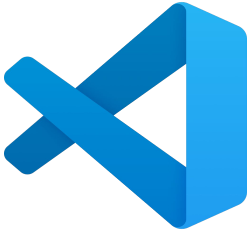
Practice security vulnerabilities and implement them

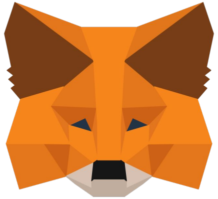
Write a report and present the lab

Contract development

Contract development

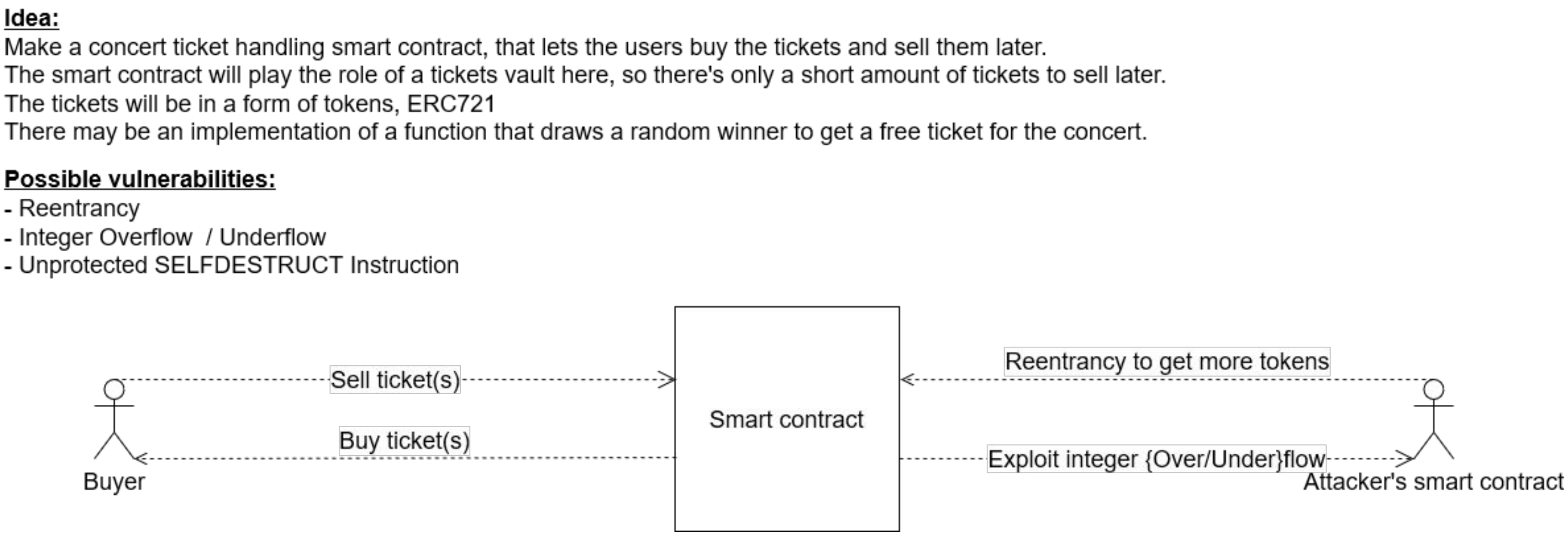
Tools used in this phase:





Contract development

Post-brainstorm overview:



Idea:

Make a concert ticket handling smart contract, that lets the users buy the tickets and sell them later.
The smart contract will play the role of a tickets vault here, so there's only a short amount of tickets to sell later.
The tickets will be in a form of tokens, ERC721
There may be an implementation of a function that draws a random winner to get a free ticket for the concert.

Possible vulnerabilities:

- Reentrancy
- Integer Overflow / Underflow
- Unprotected SELFDESTRUCT Instruction

Sell ticket(s)
Buy ticket(s)
Buyer
Smart contract
Reentrancy to get more tokens
Exploit integer {Over/Under}flow
Attacker's smart contract

Contract development

Chosen vulns from the SWC Registry:

Reentrancy (SWC-107)

It’s a vulnerability that lets a smart contract to collect funds from another smart contract in an infinite loop.

Integer overflow / underflow (SWC-101)

Either adding a number or subtracting it from a variable that already reached its maximum or minimum.

Unprotected SELFDESTRUCT (SWC-106)

When a contract has somehow a direct access to a selfdestruct function, can be executed by anyone.

Contract development

The code

Security issues

Security issues

In this phase, two methods were taken in place to security assess the contract:

- Static analysis
- Dynamic analysis

Security issues - Dynamic analysis

Dynamically analyse the contract by executing it / predicting the outcomes of a certain function.

Security issues - Static analysis

Using the help of the following auditing tools:

SLITHER

TRAIL OF BITS

Mythril

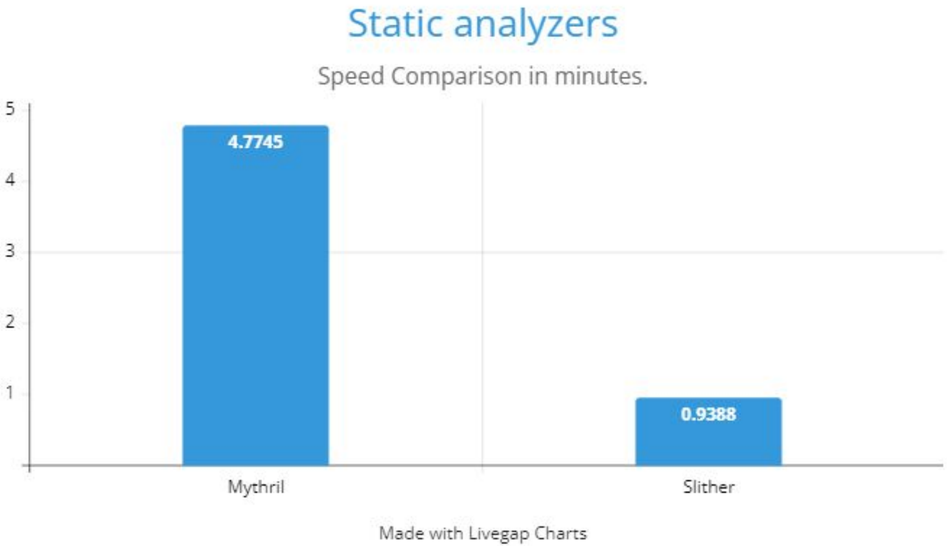
CONSENSYS

Security issues - Static analysis

Demo

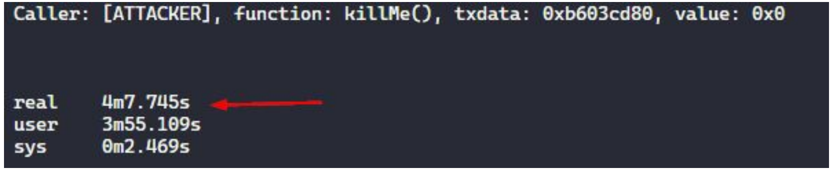
Security issues - Static analysis

Results and comparison - Speed:

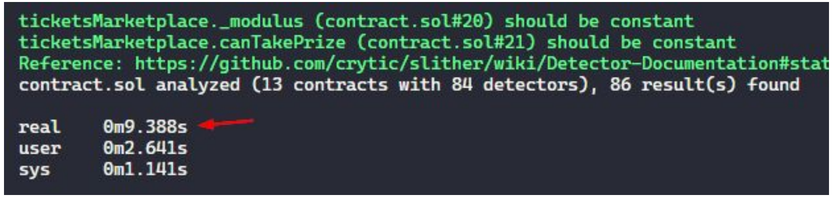

Static analyzers

Speed Comparison in minutes.

Made with Livegap Charts



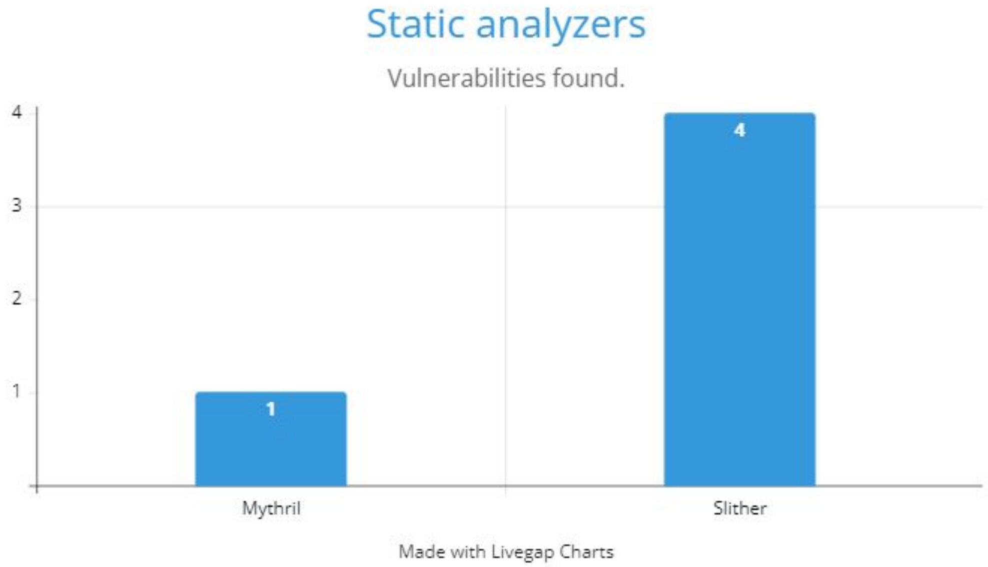
```
Caller: [ATTACKER], function: killMe(), txdata: 0xb603cd80, value: 0x0


real    4m7.745s
user    3m55.109s
sys     0m2.469s
```



```
ticketsMarketplace._modulus (contract.sol#20) should be constant
ticketsMarketplace.canTakePrize (contract.sol#21) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#stat
contract.sol analyzed (13 contracts with 84 detectors), 86 result(s) found

real    0m9.388s
user    0m2.641s
sys     0m1.141s
```

Security issues - Static analysis

Results and comparison - Efficiency:



Mythril	Slither
SWC-106	SWC-106 SWC-107 SWC-109 SWC-120

Security issues - Static analysis

Overall, slither was more efficient and faster than Mythril.

Fixing

Fixing

Finally, after discovering the bugs, the developer should fix his contract before pushing it to the mainnet network (production).

Fixing

We can follow these tips to fix our smart contract:

- Read the documentation of each bug class
- Always test and analyze before pushing the code to production
- Test locally and in a test network, such as Goerli
- Ask questions in forums, blockchain community is helpful.
- Stay up to date with the latest news.

Fixing

1. Unsecure PRNG

Fixing our mistakes:

```
function random() internal returns (uint) {
        randNonce += 1;
        return
uint(keccak256(abi.encodePacked(block.timestamp,
msg.sender, randNonce))) % _modulus;
    }
```

```
function random() internal returns (uint) {
        uint32 max = 100000;
        randNonce = randNonce.add(1);
        uint256 salt =  block.timestamp * randNonce;
        uint256 x = salt * 100 / max;
        uint256 y = salt * block.number / (salt % 5);
        uint256 seed = block.number / 3 + (salt % 300) + y;
        uint256 h = uint256(blockhash(seed));
        // Random number between 1 and max
        return uint256((h / x)) % max + 1;
    }
```

Takeaway:

Do not use **block.timestamp**, **now** or **blockhash** as a source of randomness.

2. Reentrancy

Fixing

Fixing our mistakes:

```
function withdraw(address payable
walletAddress) payable public {
        [...]
        (bool success, ) =
msg.sender.call{value:userBalance[msg
.sender]}("");
        userBalance[msg.sender] = 0;
    }
```

```
function withdraw(address payable walletAddress) payable
public {
        [...]
        userBalance[msg.sender] = 0;
        walletAddress.transfer(tickets[i].value);
    }
```

Takeaway:

Always update **before** transfer, and use the Checks-Effects-Interactions pattern.

Fixing

3. Uninitialized storage pointer

Fixing our mistakes:

```
Ticket[] public tickets;
```

→

```
Ticket admin;
Ticket[] public tickets;
function setAdmin() public {
    admin = Ticket(address(this), false, 100);
}
[...]
function getIndex(address walletAddress) private returns
(uint) {
    tickets[0] = admin;
    [...]
}
```

Takeaway:

Initialize all variables in their declaration to avoid value loss.

Fixing

3. **Unprotected SELFDESTRUCT**

Fixing our mistakes:

```
function killMe() public {
   selfdestruct(payable(msg.sender));
}
```

→

```
function killMe() public onlyOwner {
   selfdestruct(payable(msg.sender));
}
```

Takeaway:
Protect access to all sensitive functions.

Conclusion

Smart contracts vulnerabilities are real and they exist everywhere.
If you are a dev, pay attention to what you write. Test everything.
And if you are a bug hunter, there are many vulnerabilities waiting for you in the wild, to get caught and reported for some good $$$ revenue.

Thank you for the attention.